HUMAN ACTION RECOGNITION USING LOCAL TWO-STREAM CNN FEATURES WITH SVMS

SCHOOL OF COMPUTER SCIENCE AND APPLIED MATHEMATICS
UNIVERSITY OF THE WITWATERSRAND

DAVID TORPEY
674425

SUPERVISED BY
PROFESSOR TURGAY CELIK

23 MAY 2019

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science.

Declaration

I, David Torpey, hereby declare that the contents of this Dissertation to be my own work. This report is submitted for the degree of Master of Science in Computer Science at the University of the Witwatersrand, Johannesburg. Additionally, this work has never been submitted to any other university or institution for any other degree.

Signed: 

Date: 23/05/2019

Abstract

This dissertation serves to survey existing methods for human action recognition; compare those existing methods on some of the publicly-available benchmark datasets; and to introduce a novel method to solve the problem of human action recognition. The proposed method separately extracts appearance and motion features using state-of-the-art three-dimensional convolutional neural networks from sampled snippets of a video. These local features are then concatenated to form global representations for the videos. These global feature vectors are then used to train a linear SVM to perform the action classification. Additionally, we show the benefit of performing two simple, intuitive pre-processing steps, termed *crop filling* and *optical flow scaling*. We test the method extensively, and report results on the KTH and HMDB51 datasets.

Dedication

This work is dedicated to my mother, Renee Torpey. Her memory will always live with us.

Acknowledgements

I would like to thank my supervisor, Prof. Turgay Celik, for his help during this research. His help, guidance on the content of this Dissertation, and the background in various concepts in this area of research is much appreciated. Further, his unwavering willingness to help at any time of any day is much appreciated.

Moreover, I would like to thank Shunmuga Pillay for his constant assistance with providing access and support with regards to all my computing needs. Lastly, I would like to thank my partner Alexandra, father Declan, and brother Joshua, and colleague Ziyad for their continued support and assistance with this research endeavour.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human action recognition is a very active area of study in the computer vision and machine learning research communities. The reason is likely two-fold: 1) it is an interesting, difficult problem that, as yet, has no robust, real-time, accurate solution; and 2) a solution would have a vast impact on society, including in the domains of surveillance, entertainment, and healthcare [15]. In the surveillance space, an effective solution would allow for detection of anomalous events in surveillance/CCTV footage, which could then trigger an alert to the relevant authorities and personnel. Further, automatic detection of illegal events, such as shoplifting and fighting, would be possible. In the entertainment space, human-computer interaction would reach new levels of effectiveness, since reliable detection of emotions and user behaviour/engagement would be possible. In the healthcare domain, assistance in the rehabilition of patients would be realisable [15].

In a video sequence, actions / events take place, captured via cameras of different sensors. Finding a reliable, general, and robust solution to recognise these actions is still an open problem. One possible reason is due to the difficulty of finding and extracting reliable, general features. Both motion and appearance information must be considered in order for the features to be rich enough to discriminate between actions reliably. One must also contend with the various issues that plague realistic videos. Some of these issues are listed below [15]:

1. *Occlusion*: Differing clothing; partial visibility of people in the video.
2. *Clutter*: Unrelated objects moving in the video that affect the discriminative power of motion features by confusing them for an object of interest.

(a)

(b)

Figure 1.1: Example frames showing the *complicated background* and *clutter* issues in human action recognition. (a) An example frame of the UCF Sports dataset [1] with a complicated background for the action *kick*. The background is very complicated and highly non-homogeneous. Further, there are many objects in the background completely unrelated to the action. This can directly affect the reliability and discriminative power of the extracted features. (b) An example frame of the KTH dataset [2] with a simple background for the action *walk*. The background is basically homogeneous, which means the actions are easier to discriminate between, as the background will not affect the temporal or spatial features negatively, or at all.

(a)

(b)

Figure 1.2: Illustration of the *occlusion*, *perspective changes* and *viewpoint variation* issues in human action recognition. (a) An example frame from the UCF Sports dataset [1] for the action *walk*. There is a viewpoint of the human from a low level (i.e. the ground), and a far away perspective. The whole human is visible in the frame. This makes detection or learning of reliable features comparatively easy. (b) An example frame from the HMDB51 dataset [3] for the action *walk*. The viewpoint is from a central position (i.e. at eye-level). Furthermore, the perspective is close-up. This means that, for the action of walking, there is large occlusion as the majority of the human is not visible. Thus, very little information is present in the frame to let an algorithm learn that this is an example of the 'walking' action. This in turn can negatively affect the performance of the extracted or learned features.

3. *Complicated background*: Backgrounds that are highly non-static, and also often introduce noise signal into motion features.

4. *Viewpoint variation*: Camera angle changes.

5. *Perspective changes*: Change in human pose and scale/zoom.

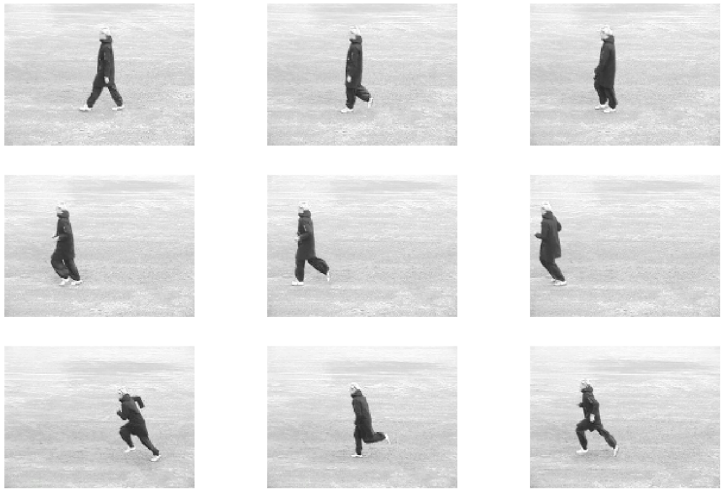


Figure 1.3: Sample frames from the KTH dataset [2].The rows from top to bottom show the following: the action *walking*; the action *jogging*; the action *running*. All frames are shown at five frames apart (due to the frame rate of the videos) for visualisation purposes. It is clear that these three actions are very temporally and spatially similar. This makes reliable features difficult to extract since it is difficult to encode the pertinent motion and appearance information robustly. However, a human can easily distinguish the action when watching the videos.

Another issue in the action recognition domain is extracting features that are able to differentiate between actions that are very spatially and temporally similar, *while* still remaining general and robust. Examples of such spatially and temporally similar situations are: 1) *washing hands* vs *rubbing hands together*; or 2) *jogging* vs *running*. Figure 1.3 depicts how similar the *walking*, *jogging* and *running* classes from the KTH dataset are. Finding features that generalise across various datasets, and to non-trivial, realistic settings, is very difficult. The difficulty of the problem is clearly demonstrated by the comparatively poor state-of-the-art performance on non-trivial, realistic datasets such as Hollywood2 or HMDB51 [16]. Example actions from these two datasets can be seen in Figures 1.4 and 1.5, respectively. These are settings in which complicated backgrounds, clutter, and massive intra-class variations are rife.

The problem of human action recognition can be formalised as attempting to estimate the mapping $f : \mathcal{S} \mapsto \mathcal{C}$, where $\mathcal{S}$ is the feature domain (commonly $\mathcal{S} = \mathbb{R}^n$, where n is the input feature vector dimensionality, or the number of extracted features); and $\mathcal{C}$ is the finite set of action classes under consideration. Solutions to human action recognition are typically separated into three main stages [15]:

1. Key-frame detection;
2. Feature extraction and representation; and
3. Classification.

Figure 1.4: Example actions from the Hollywood2 dataset [4].

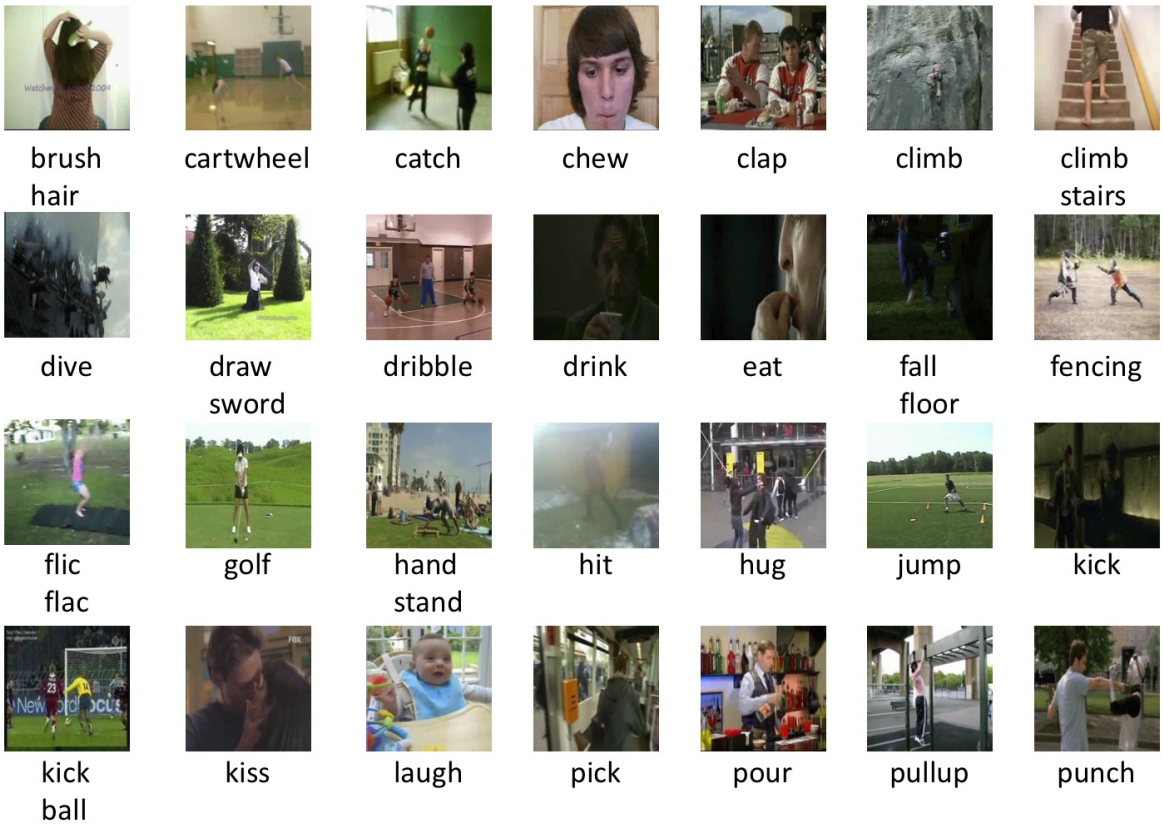


Figure 1.5: Example actions from the HMDB51 dataset [3].

The key-frame detection phase involves extracting *key frames* - frames in which actions of interest are occurring. These key frames can then be fed into the feature extraction phase, in which a robust, highly-discriminative set of features is extracted from each video sequence. Typically, the extracted feature vectors are encoded to incorporate both motion and appearance information, from separately- or jointly-

computed appearance and motion features. Appearance features refer to the features in a video sequence that relate to shape, whereas motion features refer to the features that relate to the temporal evolution of the action. That is, what occurs from frame to frame in time.

This feature extraction process is commonly performed by extracting many local features, and then encoded and quantised using a bag-of-words (BoW) approach with K-Means clustering or Fisher vectors to form a final, video-level descriptor [6; 17; 16; 18]. This encoding and quantisation step is important, since a video can be considered as a sequence of still frames where consecutive frames in this sequence have very high spatial correlations. Typically, features from each frame and sequence of frames are extracted to model appearance- and motion-based information. However, these features form a very large set, which makes it computationally challenging to process them further. Besides, not all features are informative. Thus, the methods using bag-of-words (BoW) with K-Means clustering or Fisher vectors are proposed to quantise these large sets of features. Moreover, some more modern approaches opt instead for automatic feature learning [13; 19; 20] as opposed to the former hand-crafted feature extraction method. These automatic feature learning approaches typically leverage deep learning and the plethora of associated deep neural network architectures. These are typically some type of convolutional net.

The local features are typically gradient-, or optical-flow-based descriptors that model appearance and motion information. They are computed around interest points such as space-time interest points [6] (a three-dimensional extension of the Harris interest point detector), SIFT [5], or utilising simple dense sampling. An interest point is simply a co-ordinate on an image that is defined to have some unique property, such as how corner-like the point is. These local features often include, but are not limited to, histograms of optical flow [8] and histograms of oriented gradients [21]. In order to obtain a final, video-level, uniform dimensionality descriptor, these many local features must be encoded to form a global feature. To achieve this, one of two approaches are typically taken, both of which are based on a bag-of-words framework: 1) k-means clustering; and 2) Fisher vectors. Fisher vectors often provide a more robust representation of the local features, while being less computationally expensive to compute. VLAD is a lesser-utilised alternative to Fisher vectors and BoW that is sometimes used instead [22].

Contrarily, deep architectures are tasked with automatically learning robust, spatio-temporal features that can reliably discriminate between the action classes $c \in C$. In this way, there is less of a need to hard-code quantisation techniques and/or hand-craft feature descriptors. However, there is a large computation cost as well as less domain knowledge incorporated into the features, which in turn can

sometimes decrease the performance of the system compared with hand-crafted approaches.

This work serves to introduce a novel method to solve the problem of human action recognition. This is achieved by using both deep learning, and classical computer vision techniques. Deep learning is leveraged for its adeptness at representation learning - deep learning is used to learn rich representations of our data. Classical computer vision methods such as optical flow are used in tandem with deep learning to embed some domain knowledge into the feature extraction process. Concretely, by separately learning local appearance and motion features directly from the RGB videos and optical flow videos using state-of-the-art three-dimensional convolutional neural network architectures. These local features are then concatenated together and fed into a support vector machine for classification. Further, we introduce some simple methods such as crop filling and optical flow scaling to improve recognition performance. It is important to obtain an accurate system for action recognition as the applications and possible societal impact of a robust, accurate, real-time system are huge. Furthermore, it may provide for a possible framework to build upon to solve general video analysis and recognition tasks.

This remainder of the dissertation is structured as follows. Chapter 2 introduces a background and related work section in which background concepts are explained in detail, and Chapter 3 introduces a thorough literature review of prior work in the field. Chapter 4 introduces the research methodology, and includes pertinent research questions and hypotheses, as well as full description of the proposed methodology. Chapter 5 then provides extensive experimental results of the proposed methodology on the benchmark datasets. Finally, Chapter 6 concludes the dissertation, and provide potential future work.

Chapter 2

Background Concepts

In the following we provide descriptions and explanations of all methods, concepts, and algorithms used in this dissertation. The structure of this chapter follows the natural flow of solving human action recognition, in an effort for clarity and ease of understanding.

2.1 General Concepts

2.1.1 Statistics

Random Variables and Probability Distributions

A random variable is a function that can take on any value at random. There are two main types of random variables - discrete and continuous. A discrete random variable X can take on any value from a set $\mathcal{X}$ that has a finite or countably-infinite cardinality [23]. Define $P(x)$ to be the probability of the event $X = x$. In this case, P is the *probability mass function* (PMF). A PMF satisfies the following properties [23]:

1. $0 \leq P(x) \leq 1$;
2. $\sum_{x \in \mathcal{X}} P(x) = 1$.

To develop the notion of a continuous random variable, consider a random variable X. Then define a function $F(q) := P(X \leq q)$ [23]. This is known as the cumulative distribution function (CDF) of

random variable X. Further, for any $a \leq b$, we have that [23]:

$$P(a \leq X \leq b) = F(b) - F(a) \tag{2.1}$$

We can then define the probability density function (PDF) f of random variable X: $f(x) = \frac{d}{dx}F(x)$. This PDF can then be used to calculate the probability of a continuous random variable being in a finite interval [23]:

$$P(a \leq X \leq b) = \int_a^b f(x)dx \tag{2.2}$$

Normal Distribution

The normal distribution (otherwise known as the Gaussian distribution) is the most widely-used distribution in statistics and machine learning. Its PDF is defined as [23]:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\text{exp}(-\frac{1}{2\sigma^2}(x-\mu)^2) \tag{2.3}$$

where $\mu = E[X]$ is the mean (and mode); $\sigma^2 = \text{var}[X]$ is the variance; and $\sqrt{2\pi\sigma^2}$ is a normalisation term ensuring the density integrates to 1. We write $X \sim \mathcal{N}(\mu, \sigma^2)$ to denote $P(X = x) = \mathcal{N}(x|\mu, \sigma^2)$, where X is some random variable [23]. We say that if $X \sim \mathcal{N}(0, 1)$, then random variable X is distributed according to the *standard normal* distribution. One interesting result involving the Normal distribution is the *Central Limit Theorem* (CLT). Consider N random variables with PDFs $P(x_i)$, each with mean μ and variance σ^2. We assume these random variables are independent and identically distributed (i.i.d.). Define $S_N = \sum_{i=1}^N X_i$ to be the sum of the random variables. Then, as $N \to \infty$ [23]:

$$P(S_N = s) = \frac{1}{\sqrt{2\pi N\sigma^2}}\text{exp}(-\frac{(s-N\mu)^2}{2N\sigma^2}) \tag{2.4}$$

This means that the distribution of the quantity Z_N [23]:

$$Z_N := \frac{S_N - N_\mu}{\sigma\sqrt{N}} = \frac{\bar{X} - \mu}{\sigma/\sqrt{N}} \tag{2.5}$$

converges to the standard normal distribution [23] (i.e. $Z_N \sim \mathcal{N}(0, 1)$); where $\bar{X} = (1/N)\sum_{i=1}^N x_i$ is the sample mean. This is the CLT. Thus, for sufficiently many i.i.d. samples from any distribution P, we have that the distribution can be approximated by a Gaussian. We can generalise the Gaussian

distribution to multiple dimensions by defining it over a vector $\mathbf{x}$ of D continuous random variables. The formula for the general, multivariate Gaussian distribution is [24]:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\} \tag{2.6}$$

where $\boldsymbol{\mu} \in \mathbb{R}^D$ is the mean vector; $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D}$ is the covariance matrix; and $|\cdot|$ is the determinant operator (defined later).

2.1.2 Validation

Cross Validation

Cross validation is a technique used to estimate prediction error of a machine learning algorithm or statistical model [25]. The extra-sample error e is directly estimated by $e = E[L(Y, \hat{f}(X))]$, where L is some loss function, X is the input; Y is the target value; $\hat{f}$ is the estimated function mapping the inputs to the outputs. This is essentially the average generalisation error when the function $\hat{f}$ is applied to independent test samples from the joint probability distribution of X and Y [25].

2.1.3 Loss Functions

Entropy

The entropy H of a discrete probability distribution defined of discrete random variable X is given by:

$$H(X) = -\sum_i P(X_i) \ln(P(X_i)) \tag{2.7}$$

Entropy is a measure of uncertainty in predictions, and is often used in classification contexts. It is the most commonly-used loss function for neural networks when performing classification tasks [13; 19]. It is a natural measure for such tasks since it is a measure of how confident the model is in its predicted class versus the other possible classes.

2.1.4 Mathematics

Vector

A vector is an ordered array of numbers. We can identify any particular number in the array by using its corresponding index in the array. Further, we say that if each element of the vector is an element of the set $\mathcal{S}$, and the vector has n elements, then the vector is an element of the Cartesian product of the set $\mathcal{S}$ n times. More concretely, given a vector $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \\ \vdots \\ x_n \end{bmatrix} \tag{2.8}$$

and $x_i \in \mathcal{S}, \forall i = 1, \ldots, n$, then we have that $\mathbf{x} \in \mathcal{S}^n$. If $x_i \in \mathbb{R}, \forall i = 1, \ldots, n$ (i.e. $\mathbf{x} \in \mathbb{R}^n$), then the vector $\mathbf{x}$ represents a point in an n-dimensional real vector space.

Matrix

A matrix is a two-dimensional array of numbers. Similarly to vectors, elements can be identified by their corresponding indices. However, there are two indices - i and j. If a matrix $\boldsymbol{A}$ has a height of m and a width of n, and each element of $\boldsymbol{A}$ is in the set $\mathcal{S}$, then we have that $\boldsymbol{A} \in \mathcal{S}^{m \times n}$. If the matrix is a real matrix, we have that $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. The element in the i^{th} row and j^{th} column is then denoted $\boldsymbol{A}_{i,j}$. There is a special kind of matrix known as the *identity matrix*, where each element along the main diagonal is equal to one, and all other elements are zero. This matrix therefore has the following property:

$$\forall \mathbf{x} \in \mathbb{R}^n, \boldsymbol{I}_n \mathbf{x} = \mathbf{x} \tag{2.9}$$

where $\mathbf{x}$ is an n-dimensional vector, and $\boldsymbol{I}_n \in \mathbb{R}^{n \times n}$ is the identity matrix of dimension $n \times n$.

Transpose

The transpose is an operator defined for matrices that is essentially the mirror image of a matrix across a diagonal line [26]. The transpose of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^T$, and can formally be defined as:

$$(\boldsymbol{A}^T)_{i,j} = \boldsymbol{A}_{j,i} \tag{2.10}$$

where i and j are indices running through the rows and columns of the matrices, respectively.

Trace

The trace is an operator defined for matrices. It yields the sum of all the diagonal elements of a matrix $\boldsymbol{A}$. Formally, the trace Tr can be defined as follows:

$$\text{Tr}(\boldsymbol{A}) = \sum_i \boldsymbol{A}_{i,i} \tag{2.11}$$

It can therefore be formalised as the following mapping: Tr : $\mathbb{R}^{m\times n} \mapsto \mathbb{R}$. The trace operator is invariant to the transpose operator, as well as to cyclic permutations of matrix products.

Determinant

The determinant is an operator defined only for square matrices: $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. It is equal to the product of all the eigenvalues of the matrix [26]. The absolute value of the determinant can intuitively be thought of as a measure of how much multiplication by the matrix expands or contracts the space it is operating upon.

Norm

A norm is a function that allows for the computation of the size of a vector. The L^p-norm of a vector $\mathbf{x}$ is formally given by [26]:

$$||\mathbf{x}||_p = (\sum_i |x_i|^p)^{\frac{1}{p}} \tag{2.12}$$

where $p \in \mathbb{R}$ and $p \geq 1$. Norms are essentially functions that map vectors to non-negative values. Formally, a norm (including the L^p norm) is any function f that satisfies the following properties:

1. $f(\mathbf{x}) = 0 \implies \mathbf{x} = \mathbf{0}$;

2. $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$;

3. $\forall \alpha \in \mathbb{R}, f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x})$;

where $|\cdot|$ is the absolute value function; and $\mathbf{0}$ is the zero vector.

Hessian

Consider a multivariate function $f(\mathbf{x})$, where $\mathbf{x} = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^n$. The *Hessian matrix* $\boldsymbol{H}$ is defined as the matrix of second derivatives of f:

$$\boldsymbol{H}(f)(\mathbf{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x}) \tag{2.13}$$

This is equivalent to the Jacobian of the gradient [26].

Homography

A homography, in a computer vision context, is a perspective transformation of a plane. That is, it is a re-projection of a plane from one camera view to another, subject to translation (i.e. position) and rotation (i.e. orientation) transformations of the camera.

Histogram

A histogram is a discrete approximation to a distribution function. The values being quantised are divided into so-called *bins*. The occurrences of the values or range of values are calculated for each bin. This is known as the *bin count* for that particular bin. A vector is then formed from these bin counts to create the histogram.

2.1.5 Image Processing and Computer Vision

Image

A digital image, or more commonly referred to simply as an image, described in a two-dimensional discrete space is derived from an analog image in a two-dimensional continuous space through a sampling process known as digitization [27]. The two-dimensional continuous image is divided into m rows and n columns. The intersection of a row and a column is termed a *pixel*. A pixel has an associated value s. This value can be any $s \in \mathbb{R}$, but typically $s \in \mathcal{G}$, where $\mathcal{G} = \{0, 1, \ldots, 255\}$ for 8-bit images. $\mathcal{G}$ is known as the gray level or grayscale. We term $|\mathcal{G}|$ the number of gray levels of the image. Often-times we work with colour images. In colour images, the image has a certain number L of *channels*. For example, grayscale images have $L = 1$, while RGB images have $L = 3$.

Grayscale and colour images can, alternatively, be defined in terms of matrices. A grayscale image is a 2D matrix, whereas a colour image (e.g. RGB image) is a 3D matrix.

Video

A video is essentially a three-dimensional volume, where frames are sampled at a certain frequency (frames per second) to form a sequence of multiple frames. The frames are images, and have the same properties as regular two-dimensional colour or grayscale images.

Convolution

Convolution may be defined as a function that maps two (possibly multi-dimensional) input signals to a (possibly multi-dimensional) output signal. Since we work with images, all definitions are (without loss of generality) limited to the 2D space. In the 2D discrete space, convolution is defined as [26]:

$$c(m, n) = a(m, n) * b(m, n) = \sum_j \sum_k a(j, k)b(m - j, n - k) \tag{2.14}$$

where $*$ is the convolution operation. The convolution operator has the following properties:

1. Commutativity: $c = a * b = b * a$;
2. Associativity: $c = a * (b * d) = (a * b) * d = a * b * d$;

3. Distributivity: $c = a * (b + d) = (a * b) + (a * d)$;

where a, b, c, and d are all 2D signals/images, either continuous or discrete [27]. Convolution is associated with a similar operation known as cross-correlation, which is the same as convolution, but without 'flipping' the *kernel* [26]. The *kernel* is an image/matrix with which the image is convolved.

However, since in recent human action recognition research 3D convolutional neural networks are being employed, and are achieving state-of-the-art performance, we provide a definition. 3D discrete convolution is a simple extension of the 2D case, and it is clear from the above definition for 2D convolution that the 3D case is defined as:

$$c(m, n, p) = a(m, n, p) * b(m, n, p) = \sum_{j} \sum_{k} \sum_{l} a(j, k, l) b(m - j, n - k, p - l) \tag{2.15}$$

where a, b, c, and d are all 3D signals/images.

RANSAC

Random Sample Consensus (RANSAC) [28] is an algorithm for estimating the parameters of a model, while attempting to give no influence to outliers in the data. It is technique employed in the extraction of the state-of-the-art hand-crafted features for action recognition, known as improved dense trajectories. Assume we have a model whose parameters can be estimated using a set of at least n samples. Further, consider a set of samples P such that $|P| > n$, and a subset S_1 of samples are randomly sampled from P such that $|S_1| = n$. The model M_1 is then instantiated. Then, a subset S_1^* of points in P that are within some error threshold of M_1 is determined. S_1^* is termed the *consensus set* of S_1.

If $|S_1^*| > t$ (where t is a threshold), which is a function of the estimate of the number of gross errors in P, S_1^* is used to compute a new model M_1^*. This new model is often computed using the least-squares method. However, if $|S_1^*| < t$, a new subset S_2 is randomly sampled, and the above process is repeated. Then, if after a set number of trials, no consensus set with t or more members is found, the model is solved with the largest consensus set found until that point, or terminate in failure [28].

2.2 Interest Point Detection

Interest point detection is typically the first step of many approaches to action recognition [17; 29; 18; 11]. The general idea is to localise points in an image or video from which feature descriptions can be computed. Below, the most common detectors are described.

2.2.1 SIFT

Scale Invariant Feature Transform (SIFT) features are perhaps the most well-known image features. They are formed by computinmg the image gradient at each pixel in a 16×16 window around the detected keypoint, using the appropriate levels of a Gaussian pyramid at which the keypoint was detected [30]. In order to reduce the effects of gradient far from the center/keypoint, gradient magnitudes are weighted according to a Gaussian kernel. The 16×16 window is then broken down into four 4×4 quadrants. The gradient magnitudes in each quadrant are then quantised into a histograms, binned according to one of eight orientations. Finally, trilinear interpolation is then performed to minimise the effects location and/or dominant orientation misestimation [30]. This results in a 128-dimensional histogram with non-negative elements. The last two steps are to normalise the vector to unit length (to reduce contrast effect), clip the values to the histogram to 0.2, and then to normalise in the same way once again. This finally yields the SIFT descriptor. A visual of the SIFT descriptor can be seen in Figure 2.1. SURF (Speeded Up Robust Features) is a popular variant of the SIFT descriptor, where box filters are used to approximate the derivatives and integrals used in the computation of the SIFT descriptor [30]. It is sometimes used instead of SIFT, since SURF interest points and descriptors are comparatively much faster to compute, while being similarly distinctive and robust.

2.2.2 Hessian

The Hessian affine region detector is based on the Hessian matrix, and has been used as the interest point detection mechanism in action recognition research. As with many interest point detection algorithms, the points are detected at a particular location and scale in the image. For the Hessian affine detector, the determinant of the Hessian matrix provides a measure for selecting these two metrics of the interest point. Formally, given an image I and point $\mathbf{x} = (x, y)$, the Hessian matrix $\boldsymbol{H}(\mathbf{x}, \sigma)$ at a spatial scale σ

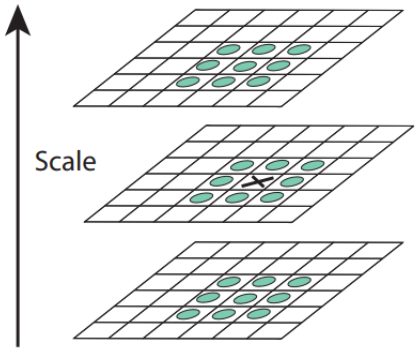


Figure 2.1: For the SIFT descriptor, a keypoint must be maximal in both space and scale. In this case, the point is compared with its 26 neighbours in space and scale to see if it is maximum. These three images are difference-of-Gaussians (DoG) images [5].

is defined as follows [31]:

$$\boldsymbol{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \tag{2.16}$$

where $L_{xx}(\mathbf{x}, \sigma)$ is the convolution of I with the second-order derivative of a Gaussian kernel with respect to x, where $L_{xx}(\mathbf{x}, \sigma) = I * G_{xx}(\sigma)$ (and similarly for the other partial derivatives).

2.2.3 Space-Time Interest Points

Space-Time Interest Points [6] is a sparse interest point detection mechanism, which has seen much use in early action recognition research. It is an extension of the 2D Harris interest point detector into 3D. Formally, given an input image I, its linear scale-space representation L is computed by convolving I with an anisotropic, spatio-temporal Gaussian kernel g given by [6]:

$$g(x, y, t, \sigma_l^2, \tau_l^2) = \frac{1}{\sqrt{(2\pi)^3 \sigma_l^4 \tau_l^2}} \exp(-\frac{x^2 + y^2}{2\sigma_l^2} - \frac{t^2}{2\tau_l^2}) \tag{2.17}$$

where x, y, and t are the two spatial co-ordinates, and temporal co-ordinate, respectively; σ_l^2 is the spatial variance; and τ_l^2 is the temporal variance. That is, $L = g * I$, where $*$ is the convolution operation. The second-moment matrix $\boldsymbol{\mu}$ is then computed by [6]:

$$\boldsymbol{\mu} = g(\cdot, \sigma_i^2, \tau_i^2) * \begin{pmatrix} L_x^2 & L_x L_y & L_x L_t \\ L_x L_y & L_y^2 & L_y L_t \\ L_x L_t & L_y L_t & L_t^2 \end{pmatrix} \tag{2.18}$$

Figure 2.2: Example of detected STIP interest points [6]. These frames relate to the action *walking* from the KTH dataset. For the action of walking, intuitively, interest points would be things like joints such as ankles, knees, and similar areas. This is because they relate directly to the action of walking and walking can be defined by these areas and their associated motion. We can see that the STIP detector detects such points, but not necessarily in every frame. It is important to note that this is a case with a simple background with no other unrelated motion that could possibly confuse the detector into detecting false positives.

where $\sigma_i^2 = s\sigma_l^2$ and $\tau_i^2 = s\tau_l^2$ for some $s \in \mathbb{R}$ are the integration scales; and $L_z = \frac{\partial L}{\partial z} = \frac{\partial}{\partial z}(g * I)$ for $z \in \{x, y, t\}$. This second-moment matrix is the basis of the so-called *Harris response function*, which gives a measure of 'cornerness' for a given input.

The Space-Time Interest Points (STIPs) are then defined as the maxima of the Harris response function H:

$$H = \det(\boldsymbol{\mu}) - k\text{trace}^3(\boldsymbol{\mu}) \tag{2.19}$$

where $k \in \mathbb{R}$; det is the determinant operator; and trace is the trace operator. Non-maximal suppression (i.e. finding local maxima in the interest function, typically with a window filter of size 3×3) may or may not be performed before this step. Examples of STIPs are shown in Figure 2.2. In the figure, we can see examples of some of the interest points identified by the algorithm. It intuitively makes sense that they would be localising points such as feet and joints, as these types of points are key to the action, and would be identified when incorporating temporal information into the interest point detection as STIP does.

2.2.4 Dense Sampling

Dense sampling is an approach for sampling interest points not based on any detection criterion such as in STIPs, but rather these points are sampled at regular grid intervals on an image. Sometimes, all pixels are used as interest points when performing dense sampling. This results in far more interest points and computation than a sparse detector such as STIPs, Hessian, or SIFT, but in the action recognition domain, it often yields better performance.

2.3 Optical Flow

Optical flow [32], in a computer vision context, is a method for motion analysis that aims to compute displacement of intensity patterns. Optical flow is often used in action recognition as a stage of the temporal feature extraction process, as it can be used to create temporal feature vector representation of videos [18; 11; 19]. The key assumption to many optical flow techniques is the *brightness constancy constraint*. This assumption is formalised as [30]:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.20}$$

This constraint states that the intensity of moving pixels remains constant during motion [33]. We then take the Taylor series expansion around zero (i.e. the MacLaurin series expansion) of this equation:

$$I(x+dx, y+dy, t+dt) = I(0,0,0)+\frac{\partial I}{\partial x}(x+dx)+\frac{\partial I}{\partial y}(y+dy)+\frac{\partial I}{\partial t}(t+dt)+O(\text{higher orders}) \tag{2.21}$$

Let $I_x := \frac{\partial I}{\partial x}$, and similarly for y and t. Then, setting this expansion to 0, we obtain $I_x dx+I_y dy+I_t dt = 0$ (since $I(0,0,0) = 0$). Dividing through by d_t yields:

$$I_x u I_y v + I_t = 0 \tag{2.22}$$

where $u = dx/dt$ and $v = dy/dt$. This is known as the optical flow (constraint) equation. Since we want to solve for u and v, the system is under-constrained.

2.3.1 Lucas-Kanade

Lucas-Kanade [34] is a very popular algorithm used to compute optical flow, usually to track sparse interest points such as Shi-Thomasi's *good features to track* [35], or space-time interest points (STIPs) [6]. The algorithm overcomes the fact that the above system is under-constrained by considering local optical flow - considering a $2k+1 \times 2k+1$ (where $k \in \mathbb{Z}^+$) window. This yields the following system of equations [34]:

$$\begin{bmatrix} I_{x,1} & I_{y,1} \\ I_{x,2} & I_{y,2} \\ \dots & \dots \\ I_{x,2k\times1} & I_{y,2k\times1} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_{t,1} \\ \vdots \\ -I_{t,2k\times1} \end{bmatrix} \tag{2.23}$$

We can shorten this equation in matrix-vector form as $\boldsymbol{A}\mathbf{u} = \mathbf{I_t}$. The issue with solving this system is that A is non-invertible due to it not necessarily being square. We therefore compute the pseudo-inverse to find a solution:

$$\begin{aligned} \boldsymbol{A}^T\boldsymbol{A}\mathbf{u} &= \boldsymbol{A}^T\mathbf{I_t} \\ \mathbf{u} &= (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\mathbf{I_t} \end{aligned}$$

2.3.2 Farneback

Sparse interest point tracking was the de-facto standard in early action recognition research, however, recently dense optical flow has been preferred. In dense optical flow, dense interest points are sampled at regular grid locations on the image (this may even be every pixel). A dense optical flow algorithm, such as Farneback's method [7] is applied to track the dense interest points over time. Although this is much more computationally-intensive than tracking sparse interest points, it often yields better, more accurate results. An example visualisation of Farneback's method can be seen in Figure 2.3.

Farneback's method assumes that the neighbourhood of each pixel can be approximated by a quadratic polynomial. These coefficients of the quadratic polynomial are estimated using a least-squares fit to the signal values of the neighbourhood. Consider a displacement $\mathbf{d}$ from the exact version of the quadratic

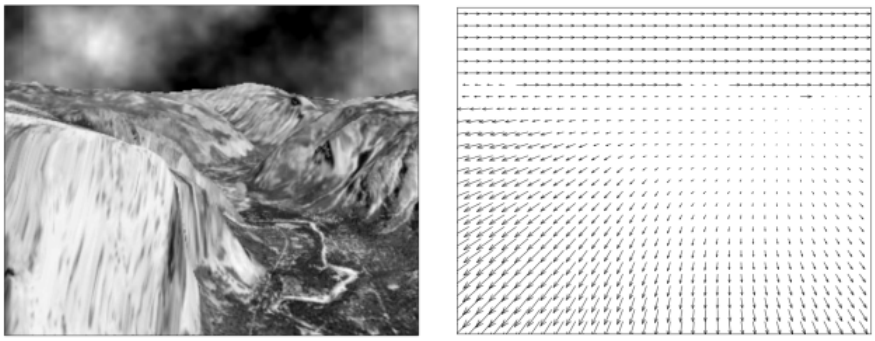


Figure 2.3: Illustration of Farneback's dense optical flow algorithm [7]. (Left) Frame from the Yosemite sequence. (Right) Corresponding optical flow vector field, computed using Farneback's method. Each pixel has a corresponding flow vector.

polynomial $I_1(\mathbf{x}) = \mathbf{x}^T \boldsymbol{A}_1 \mathbf{x} + \mathbf{b}_1^T \mathbf{x} + c_1$, given by [7]:

$$\begin{aligned} I_2(\mathbf{x}) = I_1(\mathbf{x} - \mathbf{d}) &= (\mathbf{x} - \mathbf{d})^T \boldsymbol{A}_1 (\mathbf{x} - \mathbf{d}) + \mathbf{b}_1^T (\mathbf{x} - \mathbf{d}) + c_1 \\ &= \mathbf{x}^T \boldsymbol{A}_1 \mathbf{x} + (\mathbf{b}_1 - 2\boldsymbol{A}_1 \mathbf{d})^T \mathbf{x} + \mathbf{d}^T \boldsymbol{A}_1 \mathbf{d} - \mathbf{b}_1^T \mathbf{d} + c_1 \\ &= \mathbf{x}^T \boldsymbol{A}_2 \mathbf{x} + \mathbf{b}_2^T + c_2 \end{aligned}$$

This is a new quadratic polynomial signal. Equating coefficients yields: $\boldsymbol{A}_2 = \boldsymbol{A}_1$; $\mathbf{b}_2 = \mathbf{b}_1 - 2\boldsymbol{A}_1 \mathbf{d}$; and $c_2 = \mathbf{d}^T \boldsymbol{A}_1 \mathbf{d} - \mathbf{b}_1^T \mathbf{d} + c_1$ [7]. This means that, if $\boldsymbol{A}_1$ is non-singular, we can solve for the global displacement variable $\mathbf{d}$ [7]:

$$\begin{aligned} 2\boldsymbol{A}_1 \mathbf{d} &= -(\mathbf{b}_2 - \mathbf{b}_1) \\ \mathbf{d} &= -\frac{1}{2} \boldsymbol{A}_1^{-1} (\mathbf{b}_2 - \mathbf{b}_1) \end{aligned}$$

2.4 Feature Description

There are many image feature descriptors used in human action recognition, some of which are described below. These descriptors can be grouped, at a high level, into gradient-based, and optical-flow-based, descriptors.

2.4.1 N-Jet

An N-jet, formally, is the set of derivatives or partial derivatives of a function up to, and including, order N. In a human action recognition context, N-jets were used in early research [6; 17] to encode local interest points. In such computer vision domains, an N-jet is computed by computing the partial derivatives in the scale-space of the image (both the spatial and temporal scale space when working with videos). Formally, the N-jet l for a scale-space defined in space (x and y) and time (t) is defined as follows:

$$l = (\partial_{x^m y^n t^k} L) \ni m + n + k \leq N \tag{2.24}$$

where $\partial_{x^m y^n t^k} L$ is the partial derivative of the scale space representation with respect to x with order m; y with order n and t with order k. For example, the N-jet for such a scale-space representation L over the spatial domain, with $N = 2$, is defined as:

$$l = (L_x, L_y, L_{xx}, L_{xy}, L_{yx}, L_{yy}) \tag{2.25}$$

2.4.2 Histograms of Oriented Gradients

Histograms of Oriented Gradients (HOG) is an image feature introduced by [21] for the purpose of human detection, however, the feature is essentially a general descriptor that can be used to describe the shape and/or appearance of any object in an image. They have seen widespread use in human action recognition, as well as in many areas of computer vision. The essential intuition behind HOG features is that, even without knowing anything about edge or gradient positions a-priori, the distribution of local edge directions and intensity gradients can often be used to characterise local object appearance and shape [21]. A description of how to compute these features is given below.

First, the input image is gamma-corrected and normalised. Next, the image is divided into small spatial regions known as cells. This can be seen in Figure 2.4. The gradients are then computed. A 1-D histogram is then computed for each cell, containing gradient directions and edge orientations. Contrast normalisation on the responses is then performed. Collecting all these 1-D HOG vectors provides the final feature descriptor. Extensions and variations of the above base HOG descriptor have been proposed [36; 16]. A visualisation of the HOG descriptors for an image can be seen in Figure 2.5.

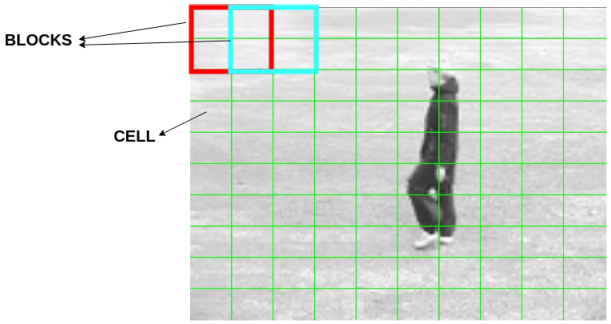
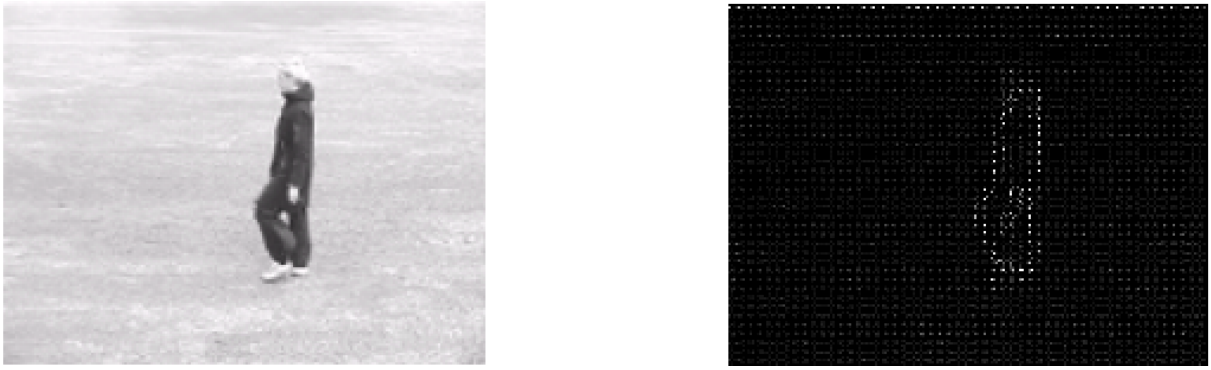


Figure 2.4: Illustration of how the HOG algorithm breaks an image down into cells and blocks. A histogram of gradient directions or edge orientations is accumulated for each cell.



(a) (b)

Figure 2.5: Illustration of the HOG algorithm. (a) Input frame from a video from the KTH dataset. (b) Visualisation of the HOG descriptors computed for the image. It is clear from this visual that HOG attempts to encode local appearance and shape information, as we can see a silhouette of the person from the input frame.

2.4.3 Histogram of Optical Flow

Histograms of Optical Flow (HOF) [8; 37] is a feature for describing motion information in an image sequence or video. Part of the computation leverages optical flow / point tracking algorithms to find optical flow vectors. By encoding the distribution of local optical flow, local motion information is captured by the HOF descriptor. An example of the high-level stages of computation can be seen in Figure 2.6. Each optical flow vector is quantised into the histogram by binning according to its orientation, and weighting according to its magnitude [8]. Thus, all flow vectors $v = [x, y]$ with direction $\theta = \tan^{-1}(y/x)$ will contribute [8]:

$$\sqrt{x^2 + y^2} \tag{2.26}$$

to the sum of its corresponding bin. An example of the binning procedure for the case of 4 bins can be seen in Figure 2.7. The histogram is then normalised to sum to 1. This makes the HOF representation

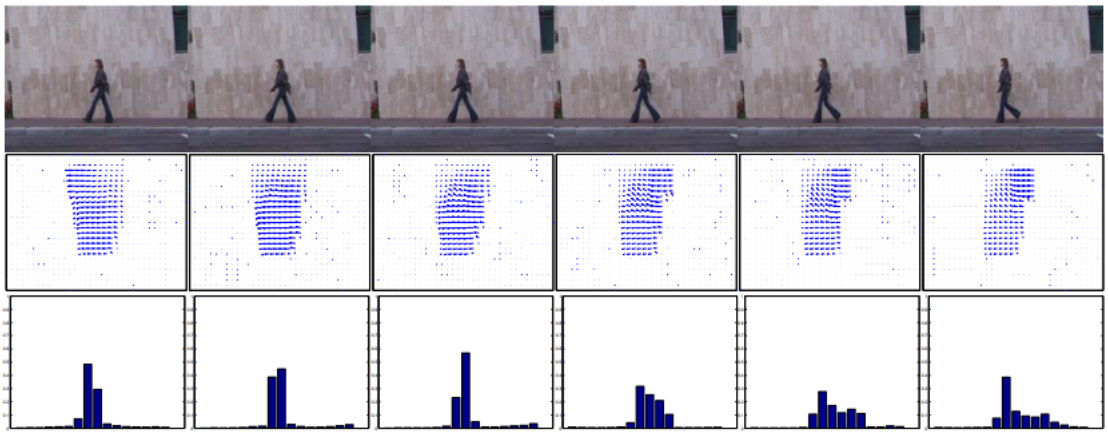


Figure 2.6: HOF computation [8] for an example sequence. The top row shows a sequence of frames of a video. The second row shows the optical flow fields for the input sequences. These flow fields encode the relative local motion between frames. The bottom row depicts the histogram of optical flow. The flow vectors from the second row are quantised into histograms by binning them according to their direction and weighting them according to their magnitude.

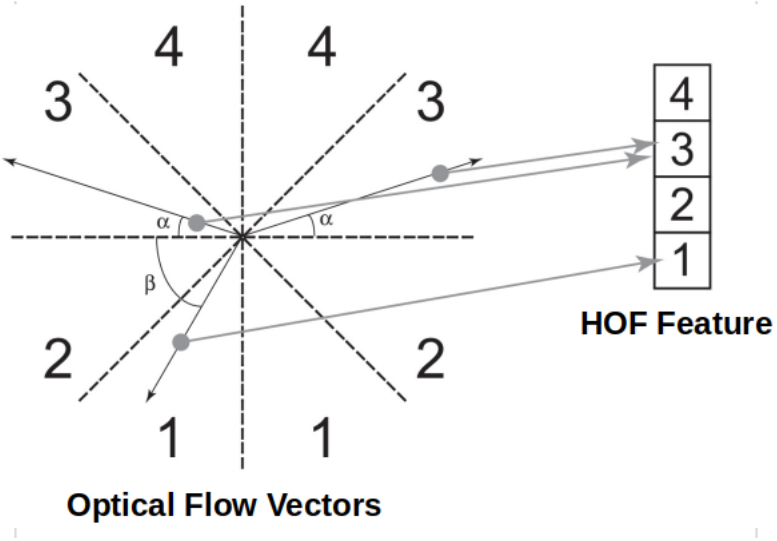


Figure 2.7: Illustration of the binning procedure of the HOF algorithm for the case of 4 bins [8]. Three example flow vectors are shown. Two have a orientation of α, which falls into the range defined for histogram bin 3. One flow vector has orientation β, which falls into the range defined for histogram bin 1.

invariant to direction of motion as well as scale. Further, small noise optical flow measurements will not affect the representation.

2.4.4 Dense Trajectories

Dense trajectories were introduced by [11] for the purpose of human action recognition. The approach is inspired by the achievements in the image classification domain from applying dense sampling. Interest

points are sampled, using dense sampling, with a grid size of $W \times W$ pixels, at multiple spatial scales. Each of these points $P_t = (x_t, y_t)$ at frame t is tracked in a dense optical flow field $\omega = (u_t, v_t)$ using the median filtering algorithm [11]:

$$P_{t+1} = (x_{t+1}, y_{t+1}) = (x_t, y_t) + (M * \omega)|_{(\bar{x}_t, \bar{y}_t)} \tag{2.27}$$

where $(\bar{x}_t, \bar{y}_t)$ is the rounded position of (x_t, y_t); and M is the median filtering kernel [11]. The concatenation of these tracked points form the trajectory feature [11]:

$$(P_t, P_{t+1}, P_{t+2}, \dots) \tag{2.28}$$

Further, the trajectories are limited to a length of L, due to the well-known drifting problem in tracking. Points satifying the Shi-Thomasi criterion [35] are sampled and added to the tracking process if no tracked points are found in a $W \times W$ neighbourhood (in an effort to ensure dense coverage). This criterion is defined by the smallest eigenvalue of the point's autocorrelation matrix being below a certain threshold. Static trajectories are pruned, as it is posited that these do not contain any relevant information for action recognition (i.e. dynamic information). If large, sudden displacements occur, it is likely that this is an error in tracking. Therefore, such trajectories are also removed. The shape of trajectory $\mathbf{S}$ is defined as [11]:

$$\mathbf{S} = (\Delta P_t, \dots, \Delta P_{t+L-1}) \tag{2.29}$$

where $\Delta P_t = P_{t+1} - P_t = (x_{t+1} - x_t, y_{t+1} - y_t)$ [11]. This $\mathbf{S}$ vector is normalised, resulting in a vector, $\mathbf{S}'$, termed the *trajectory descriptor*.

2.4.5 Improved Dense Trajectories

Improved dense trajectores [18] (commonly known as iDT) are an extension of the dense trajectory descriptor [11]. The essential idea behind iDTs are to improve on dense trajectories by estimating camera motion, and correcting for it accordingly. This is due to the fact that camera motion negatively affects motion-based descriptors, and by extension the recognition performance of the system. This is because camera motion, in an action recognition context, is erroneous information. Camera motion, in a sense, 'confuses' the motion descriptors into thinking the motion pertains to the action under consideration, when it in fact does not.

To estimate the camera motion, the key assumption made is that any two consecutive frames are related by a homography [18]. Although this may seem to be a rather stringent assumption, it is often valid since there is not usually much global motion between consecutive frames, especially for a high-frame-rate video sequence. Candidate matches of sufficient quantity and quality need to be found in order to estimate the homography reliably. This is done by 1) finding SURF feature points in both frames, and matching them based on the nearest-neighbour rule; and 2) sampling motion vectors from a dense optical flow using Farneback's method [7], where sampling is based on the above-mentioned Shi-Thomasi [35] criterion. These two methods for extracting suitable features/matches are complementary since SIFT focuses on detecting blob-like structures, whereas Shi-Thomasi's *good features to track* focuses on points that are corner-like or edge-like. The homography can then be estimated using these matched points by using the RANSAC algorithm [28]. This allows for the correction of (i.e. removal of) camera motion.

2.4.6 Motion Boundary Histograms

Motion boundary histograms (MBHs) are a descriptor introduced by [38] to encode relative motion information. The application in the initial research was human detection. However, these have been widely used in human action recognition research [11; 18]. In order to compute the MBH descriptor, a dense optical flow field needs to be computed. The horizontal and vertical components are then separated from this optical flow field: $I_\omega = (I_x, I_y)$ [38]. The spatial derivatives are then computed for both components. Finally, the orientation information of these derivatives is then quantised into histograms with a pre-defined number of bins (8 bins in this case) [11]. This quantisation into histograms is akin to the quantisation computation of the HOG descriptor. These histograms are then separately normalised using the L_2-norm.

2.5 Clustering

Since the above feature description techniques typically result in a variable number of local features per video or image, clustering techniques are often employed as the first step of encoding these many local features into global, fixed-length vectors. The two most common forms of clustering in action recognition are discussed below. The first - K-Means clustering - is a form of *hard clustering*. The other - Gaussian mixture models - is a form of *soft clustering* (a probabilistic approach to clustering).

2.5.1 K-Means Clustering

K-Means clustering is an algorithm used to identify clusters or groups of data points in a multi-dimensional space [23]. Suppose we have a set of vectors $\mathcal{S} = \{\mathbf{x}_i\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^D$. The goal is to partition $\mathcal{S}$ into K clusters/groups, where each group is similar in some way, where similarity is defined by some distance metric (in this case, the Euclidean distance). To do so, we introduce a set of K D-dimensional vectors $\boldsymbol{\mu}_k \in \mathbb{R}^D$, $k = 1, \ldots, K$. These are known as the cluster centres, and are the protoypes associated with cluster k [23]. We then find the assignments of the data points to their closest associated cluster centroid. Once found, we update the location of the cluster centres to the mean of the points assigned to it. We repeat this process until we minimise the so-called distortion function J, which is defined as [24]:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2 \tag{2.30}$$

where $r_{nk} \in \{0, 1\}$ is an indicator function denoting which of the K clusters data point $\mathbf{x}_n$ is assigned to [24]. It should be noticed that this function is non-convex, and thus the K-Means algorithm can result in local minima.

2.5.2 Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is akin to K-Means clustering, but whereas K-Means clustering is a form of hard clustering, GMMs are instead a form of soft clustering. A Gaussian mixture model can be expressed as the following linear combination [24]:

$$P(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k) \tag{2.31}$$

where K is the number of Gaussians in the mixture; π_k are the mixing coefficients; μ_k are the means; and Σ_k are the covariance matrices. Define $\mathbf{z}$ to be a K-dimensional binary random variable, such that $\mathbf{z}$ is a one-hot encoding. We can then define the marginal distribution over $\mathbf{z}$ as [24]:

$$P(z_k = 1) = \pi_k \tag{2.32}$$

where $\pi_k \in [0, 1]$; and $\sum_k \pi_k = 1$. Since $\mathbf{z}$ is a one-hot encoding, the marginal distribution $P(\mathbf{z})$ can instead be expressed as [24]:

$$P(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k} \tag{2.33}$$

Further, the conditional distribution can be expressed as [24]:

$$P(\mathbf{x}|\mathbf{z}) = \prod_{k} \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)^{z_k} \tag{2.34}$$

We can then compute the marginal distirbution $P(\mathbf{x})$ [24]:

$$P(\mathbf{x}) = \sum_{\mathbf{z}} P(\mathbf{z})P(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k) \tag{2.35}$$

Since the goal of a GMM is to maximise the likelihood function with respect to the parameters (means, covariances, and mixing coefficients), we can employ the Expectation-Maximisation (EM) algorithm to estimate these parameters. EM [39] is an algorithm for finding maximum likelihood solutions latent variable models [24]. In the context of GMMs, the first part of the EM algorithm is to initialise the parameters mixture model: the mixing coefficients π_k, the means $\boldsymbol{\mu}_k$, and covariance matrices $\boldsymbol{\Sigma}_k$. This can be done using K-Means. Then, we iteratively perform the following three steps [24]:

1. **E Step**: Compute the expected values for latent variables / parameters [24]:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{2.36}$$

2. **M Step**: Estimate new values for the latent variables / parameters using the expected values from the previous step [24]:

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{2.37}$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T \tag{2.38}$$

$$\pi_k^{\text{new}} = \frac{N_k}{N} \tag{2.39}$$

where

$$N_k = \boldsymbol{\Sigma}_{n=1}^{N} \gamma(z_{nk}) \tag{2.40}$$

3. Check for convergence by computing the log likelihood. Repeat the process from the **E step** if the algorithm has not converged:

$$\ln p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\} \tag{2.41}$$

2.6 Feature Encoding and Quantisation

Many approaches to human action recognition extract local features, whether it be for temporal or spatial feature extraction. This means that every video necessarily has a different number of local features. In order to obtain a fixed-length descriptor per video from these local features, we need to employ some feature encoding technique. The most common way to do this is to utilise a Bag-of-Words (BoW) or Fisher Vectors. These encoding / quantisation techniques leverage some form of clustering as part of their process.

2.6.1 Bag-of-Words

Affine-invariant local features can be quantised into fixed-length histograms using a Bag-of-Words approach [40]. Suppose we have a set of m videos $\{V_i\}_{i=1}^{m}$. For each video, we have $n_i \in \mathbb{N}$ local features. The n_is are necessarily distinct. Suppose, for a particular video V_i, these n_i local features are given by the matrix $\boldsymbol{M}_i \in \mathbb{R}^{n_i \times k}$:

$$\begin{bmatrix} \cdots & \boldsymbol{\gamma}_{i,1} & \cdots \\ \cdots & \boldsymbol{\gamma}_{i,2} & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \boldsymbol{\gamma}_{i,n_i} & \cdots \end{bmatrix} \tag{2.42}$$

where k is the dimensionality of the feature; and $\boldsymbol{\gamma}_{i,j} \in \mathbb{R}^k$ is the j^{th} local feature for video i, $\forall i = 1, \ldots, n_i$. We then sample $L \in \mathbb{Z}^+$ features from the set of all features from all videos. This means we sample L features from a total choice of $\sum_{i=1}^{m} n_i$ features. We then have a set $\mathcal{F}$ of features where $|\mathcal{F}| = L$. We perform K-Means clustering on this set of features, with a pre-defined *codebook* size of K, which is the number of 'visual word' clusters to learn. Then, for each local feature of a particular

video, we find which visual word it belongs to defined by some distance metric. This process results in a histogram of visual word occurrences $H \in \mathbb{R}^K$ - the number of occurrences of particular patterns in the video [40]. The normalised version of this histogram is then the final fixed-length representation of the video.

These histograms of visual words are computationally efficient; invariant to affine transformations of the object class; and invariant to occlusion, lighting, and intra-class variations [40].

2.6.2 Fisher Vectors

The Fisher Vector [41] (with visualisation depicted in Figure 2.8) is a more recent approach to local feature encoding and state-of-the-art patch encoding, and has been widely applied in recent human action recognition research. It is an alternative to the above-described BoW approach, is computationally efficient, provides good results, and can be compressed with a minimal loss of accuracy. The Fisher Vector relies on the underlying principle of the Fisher Kernel.

Suppose we have a set of M local descriptors $X = \{\mathbf{v}_i\}_{i=1}^{M}$. Assume the data-generating PDF of X is given by u_λ with parameters λ. Then, X can be described by the following gradient vector [41]:

$$G_\lambda^X = \frac{1}{M}\nabla_\lambda \log u_\lambda(X) \tag{2.43}$$

Intuitively, G_λ^X describes the relative contribution of the parameters to the generative process [41]. Further, it is clear that the dimensionality of G_λ^X is invariant to M, and depends only on the number of parameters in λ. A kernel for the above gradient vectors can be defined as [41]:

$$K(X,Y) = G_\lambda^{X^T} \boldsymbol{F}_\lambda^{-1} G_\lambda^Y \tag{2.44}$$

where $\boldsymbol{F}_\lambda^{-1}$ is known as the Fisher information matrix (FIM) of u_λ, which is both symmetric and positive definite. We can therefore decompose this FIM into its Cholesky decomposition, thereby allowing us to rewrite K in terms of dot products between normalised vectors. These normalised vectors are defined as [41]:

$$\mathcal{G}_\lambda^X = L_\lambda G_\lambda^X \tag{2.45}$$

where $F_\lambda = L_\lambda^T L_\lambda$ is the Cholesky decomposition of the FIM. We term $\mathcal{G}_\lambda^X$ the *Fisher Vector*. Often,

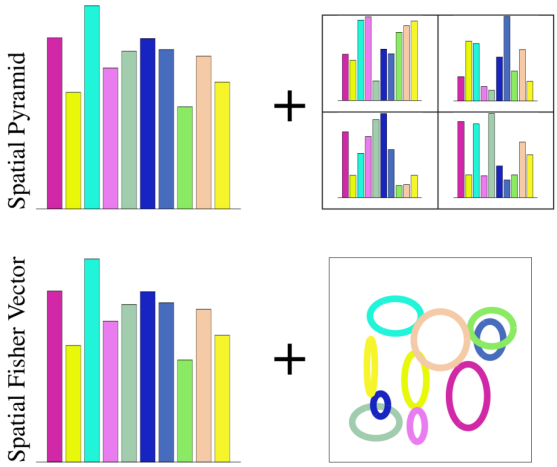


Figure 2.8: Comparison of Fisher vector representation against normal spatial pyramid image representations [9]. The spatial pyramid representation (above), based on the common bag-of-words framework, simply concatenates the histograms of visual word occurrences of the image. The Fisher vector representation (below), however, models an image using means and variances. The ellipses represent Gaussians of a GMM. Thus, the Fisher vector can be thought of as a 'soft' (i.e. probabilistic) version of the typical bag-of-words / histogram of visual words approach.

u_λ is chosen to be a GMM, with $\lambda = \{\pi_i, \mu_i, \Sigma_i\}_{i=1}^{K}$ being the set of mixture weights, means, and covariance matrices of the K-mode GMM.

2.6.3 VLAD

The VLAD (Vector of Locally Aggregated Descriptors) is an image representation based on a locality criterion in the feature space [42]. It can be seen as a non-probabilistic simplification of Fisher vectors. It is computed by first computing a codebook of k visual words using K-Means. Each descriptor $\mathbf{x} \in \mathbb{R}^d$ in the feature space is assigned to a visual word based on the nearest neighbour rule: $\mathbf{c}_i = \text{NN}(\mathbf{x})$.

Then, for each visual word $\mathbf{c}_i$, we accumulate the residuals $\mathbf{x} - \mathbf{c}_i$ of descriptors $\mathbf{x}$ assigned to visual word $\mathbf{c}_i$. This essentially encodes the distribution of the descriptors with respect to the centre. The VLAD is then an L_2-normalised version of the vector containing these accumulations of residuals for each of the k visual words. Thus, the dimension of the VLAD is kd.

2.7 Dimensionality Reduction

Often, the feature encoding processes described above (as well as many other steps in different modelling processes) result in very high-dimensional features. These can be, at times, on the order of millions of

dimensions. As such, dimensionality reduction is sometimes performed [12; 43]. Typically, principal components analysis is the algorithm of choice.

2.7.1 Principal Components Analysis

Principal Components Analysis (PCA) - otherwise known as the Karhunen-Loeve transform - is a technique most often used to perform dimensionality reduction. However, other applications such as data visualisation; feature extraction; and lossy data compression [44]. It is a projection of data onto a lower dimensional space, known as the *principal subspace*. The data is projected such that the variance is maximised [24]. PCA essentially learns the continuous latent variables that correspond to orthogonal directions of maximum variance, where orthogonal directions are termed the principal components of the data. The n principal components correspond to the n eigenvectors with largest associated eigenvalues.

Consider a set of data points $\{\mathbf{x}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^D$. The goal is to then project the data onto a subspace with dimensionality $M < D$, such that the variance of the projected data is maximal. Without loss of generality, we project the data onto a one-dimensional subspace (i.e. $M = 1$ for simplicity of the explanation). The direction of this space can be defined using a vector $\mathbf{u}_1 \in \mathbb{R}^D$, such that $\mathbf{u}_1^T\mathbf{u}_1 = 1$. Each data point $\mathbf{x}_i$ is projected onto the one-dimensional subspace by computing $\mathbf{u}_1^T\mathbf{x}_i$. The mean of projected data is given by $\mathbf{u}_1^T\bar{\mathbf{x}}$, where $\bar{\mathbf{x}}$ is given by:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i \tag{2.46}$$

and similarly, the variance of the projected data is given by:

$$\frac{1}{N}\sum_{i=1}^{N}[\mathbf{u}_1^T\mathbf{x}_i - \mathbf{u}_1^T\bar{\mathbf{x}}]^2 = \mathbf{u}_1^T\boldsymbol{S}\mathbf{u}_1 \tag{2.47}$$

where $\boldsymbol{S} = (1/N)\sum_i(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$. Since we want to maximise variance, we formulate the Lagrangian and find its maxima. We require $\mathbf{u}_1^T\mathbf{u}_1 = 1$, since unconstraining the maximisation problem could yield $||\mathbf{u}_1|| \to \infty$. Thus, we obtain the following Lagrangian:

$$\mathcal{L}(\mathbf{u}_1) = \mathbf{u}_1^T\boldsymbol{S}\mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^T\mathbf{u}_1) \tag{2.48}$$

where $\lambda_1 \in \mathbb{R}$ is a Lagrange multiplier. Computing the partial derivative of $\mathcal{L}$ with respect to $\mathbf{u}_1$ and

setting to zero yields:

$$\boldsymbol{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \tag{2.49}$$

which means that $\mathbf{u}_1$ must be an eigenvector of the covariance matrix $\boldsymbol{S}$. If we left-multiply by $\mathbf{u}_1$, we see that variance is given by $\mathbf{u}_1^T \boldsymbol{S}\mathbf{u}_1 = \lambda_1$. This essentially means that variance will be at a maximum when we set $\mathbf{u}_1$ to be the eigenvector with largest associated eigenvalue. This eigenvector is known as the first principal component of the data. If we make M larger, this will yield more principal components with $\mathbf{u}_i^T \mathbf{u_j} = 0$ for all $i \neq j$ and $\mathbf{u}_i^T \mathbf{u_j} = 1$ for all $i = j$.

2.8 Modelling

The last stage of action recognition approaches is to apply some machine learning model to your data. This either engineered features or raw inputs such as RGB videos or optical flow videos. The two most common algorithms by far are the support vector machine (typically used in tandem with engineered features), and neural networks (typically used to automatically learn features).

2.8.1 Support Vector Machines

The vast majority of early approaches to human action recognition opt for a hand-crafted feature extraction process, as opposed to automatically-learned feature extraction. Further, the extracted feature vectors are often very high-dimensional - hundreds or thousands of dimensions. Therefore, linear classifiers are often used, and the most popular by far is the Support Vector Machine (SVM). This classifier (later extended to regression problems) was first introduced by [45]. The basic, conceptual idea of the SVM is to map the input feature vectors to a very high-dimensional (even infinite) feature space, and then find a maximal-margin separating hyperplane in this high-dimensional space. This linear decision surface separates the classes perfectly in a so-called *hard-margin* SVM when the data is linearly separable, whereas a *soft-margin* SVM allows for misclassification in linearly nonseparable data contexts.

Formally, consider a compact metric space (X, d) and (without loss of generality) a binary classification context: $Y = \{-1, 1\}$. Additionally, define p to be the probability distribution on $Z := X \times Y$. We attempt to find the mapping $f : X \mapsto Y$.

A q-norm soft-margin SVM is estimated using samples from a reproducing kernel Hilbert space (RKHS)

$\mathcal{H}_K$ with an associated *Mercer kernel* K [46]. It is through the use of this kernel that the feature vectors are mapped to the high-dimensional feature/vector space. The q-norm soft margin SVM classifier is then defined as $\text{sign}(f_{\mathcal{Z}})$, where $f_{\mathcal{Z}}$ is the minimizer of the following [46]:

$$\begin{aligned} \underset{f \in \bar{\mathcal{H}}_K}{\arg\min} \quad & \frac{1}{2}||f^*||_K^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i^q \\ \text{s.t.} \quad & y_i f(\mathbf{x}_i) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \end{aligned}$$

where $\bar{\mathcal{H}}_K := \mathcal{H}_K + \mathbb{R}$; $C \in \mathbb{R}$ such that $C > 0$; and $\mathcal{Z} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \in Z^m$ are random samples drawn i.i.d from p.

2.8.2 Neural Networks

Deep Neural Networks

Deep neural networks (DNNs) are essentially regular artificial neural networks (ANNs), but with multiple hidden layers. The goal of DNNs, and indeed any neural network architecture, is to approximate some function f^*. Estimation of the function is performed by minimising some cost function using samples, and in turn learning the set of weights θ that best approximate the function.

The majority of networks are trained using maximum likelihood estimation (MLE). Thus, the cost function minimised during training the cross-entropy between the training data and the model distribution. Mathematically, this function is given by:

$$J(\theta) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(\mathbf{y}|\mathbf{x})] \tag{2.50}$$

where p_{model} is the data-generating distribution. This cost function is the general form for (most) neural network architecture. For example, if we have that $\log p_{\text{model}}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; f(\mathbf{x};\theta), \boldsymbol{I})$ ($\boldsymbol{I}$ is the identity matrix), then the cost function simply simplifies to mean squared error.

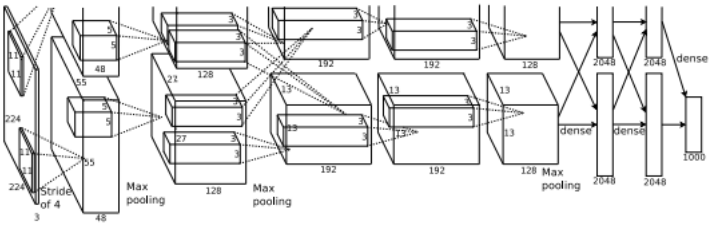


Figure 2.9: Architecture used by [10] known as AlexNet. This architecture pioneered a wave of research into 2D CNNs for computer vision tasks. The basic idea of the architecture is to have successive convolutional and max-pooling layers (with activation functions applied), and finally a fully-connected layer and classification layer at the end.

Convolutional Neural Networks

With the comparatively-recent onset of deep learning and deep neural architectures being employed in many computer vision and machine learning problems, such as image classification [10]; natural language processing and speech recognition [47]; and video classification [13], many recent approaches to human action recognition have utilised, in some form, deep learning techniques. One such technique is the convolutional neural network (CNN). CNNs have an architecture specifically designed to process data that has a known grid-like topology [26]. An example of such data are images, which are essentially just a grid of pixel intensities. To process the data, CNNs employ convolution in place of regular matrix multiplication (as in conventional neural networks) in at least one of the layers.

Advantages of CNNs are 1) they are able to account for the natural spatial relationships inherent in images; 2) employ parameter sharing, which reduces the number of trainable parameters that need to be optimised; and 3) can learn deep, rich features/descriptions of the input image data. In this way, CNNs pre-trained on large, varietal datasets, such as ImageNet [48], often serve as useful, generic feature extractors. Feature vectors can be sampled by simply performing inference on the network using a test image, and thus computing a particular layer's output. Usually we discard the final softmax (classification) layer, and instead use the penultimate layer's output. A common 2D CNN architecture can be seen in Figure 2.9.

For the domain of action recognition in which samples are in the form of video data, those using CNNs often extend the network to a three-dimensional architecture [13]. This allows the network to naturally incorporate temporal relationships in the video, as opposed to training a 2D CNN frame-by-frame. This can in turn provide for more descriptive features for both spatial and temporal descriptions of the video and/or action. Convolutional networks often employ multiple successive convolutional-pooling layer

pairs. The pooling (such as max pooling or average pooling) provides the model with approximate translation invariance, and to a lesser degree the mitigation of over-fitting [26]. Moreover, pooling usually occurs after a non-linearity has been applied to the output of the artificial neuron, and provides a summary statistic of the nearby outputs.

Recurrent Neural Networks

In the action recognition domain, the data is almost always exclusively video data. Video is inherently sequential data, as a frame V_t depends on, and correlates with, previous frames $V_{t-l}, V_{t-l+1}, \ldots, V_{t-1}$, for $l \in \mathbb{N}$. These inter-frame temporal dependencies and sequential nature of videos make the application of recurrent neural networks (RNNs) to such problems [49] intuitive and logical. RNNs [50] are a family of neural network architectures particularly adept at processing sequential data. Using particular recurrent architectures such as long short-term memory (LSTM) [51] or gated recurrent units (GRU) [52], the network can learn long-term temporal dependencies, without suffering from the exploding or vanishing gradient problems [26].

The word *recurrent* in RNNs implies that there are feedback loops from neurons back to themselves. This means that a particular neuron's output depends not only on the output from the previous layer (as in vanilla neural networks), but also the output from the current layer as well. Formally, for each time step $t = 1, \ldots, \tau$, we have the following update equations:

$$\begin{aligned}
\mathbf{a}^{(t)} &= \mathbf{b} + \boldsymbol{W}\mathbf{h}^{(t-1)} + \boldsymbol{U}\mathbf{x}^{(t)} \\
\mathbf{h}^{(t)} &= \phi(\mathbf{a}^{(t)}) \\
\mathbf{o}^{(t)} &= \mathbf{c} + \boldsymbol{V}\mathbf{h}^{(t)} \\
\hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)})
\end{aligned}$$

where $\mathbf{a}^{(t)}$ is the activation at time step t; $\mathbf{h}^{(t)}$ is the hidden layer representation at time t; $\mathbf{o}^{(t)}$ is the output at time t; $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$ are weight matrices; ϕ is an arbitrary activation function; $\mathbf{b}$ and $\mathbf{c}$ are bias vectors; $\mathbf{x}^{(t)}$ is the input at time t; and $\hat{\mathbf{y}}^{(t)}$ is the output probability distribution over a set of discrete random variables (this discrete assumption holds without loss of generality).

LSTM networks are the state-of-the-art recurrent architectures in many sequential modeling domains. They achieve this by overcoming the decaying error back flow in backpropagation through time (BPTT),

and instead enforce a constant error flow within special gated, recurrent units. These gates, all with their own trainable weights, learn to open and close access to the aforementioned constant error flow [51]. This is the property of LSTM (and similarly, GRU) units, that allows them to learn long-term dependencies in data.

Activation Functions

The *universal approximation theorem* states that every continuous function defined on compact subsets of $\mathbb{R}^n$ can be approximated arbitrarily well with a neural network with one hidden layer, under certain mild assumptions of a given *activation function* ϕ [53]. For instance, the function must be continuous and differentiable everywhere. It is the activation function ϕ that introduces non-linearity to the network, and allows it to learn and approximate highly-nonlinear decision surfaces. Some of the more common activation functions are discussed below.

Linear The linear activation function is essentially not transforming the output of a neuron non-linearly, and instead making the output to be the regular matrix multiplication from the previous layer: $\hat{\mathbf{y}} = \boldsymbol{W}^T\mathbf{h} + \mathbf{b}$, where $\hat{\mathbf{y}}$ is the output; $\boldsymbol{W}$ is the weight matrix; $\mathbf{h}$ is the previous layer's ouput; and $\mathbf{b}$ is the bias vector. Linear units are typically used to compute the mean of a conditional Gaussian distribution [26]. It should be noted that the assumptions for the activation function listed above all hold for a linear activation, and thus the universal approximation theorem still holds for these types of activations. It is simply the multi-layer, feed-forward architecture itself that gives rise to the universal approximator property.

Sigmoid The sigmoid activation function is defined as:

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{h} + b) \tag{2.51}$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the logistic sigmoid function; $\mathbf{w}$ is a weight vector; $\mathbf{h}$ is the output feature vector (i.e. logit) from the previous layer; and b is the bias term. The sigmoid activation was the de-facto standard in early neural network research, and today is often used as the final layer of a network in binary classification problems, since we can define the logistic output as: $P(y = 1|z) := \hat{y} = \sigma(z)$ [26], where $z = \mathbf{w}^T\mathbf{h} + b$ is the logit. The major disadvantage of sigmoid units are the fact that as

the number of layers in the network increases, the sigmoid activations saturate and thus, the gradient vanishes and is not communicated to the early layers effectively.

Softmax The softmax activation function is almost always used when using neural networks for classification problems. The softmax activation allows one to represent a probability distribution over dicrete variable with t possible values [26]. Formally, the softmax activation is defined as:

$$\text{softmax}_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{2.52}$$

for all $i = 1, \ldots, t$, where z_i is the logit of the i^{th} neuron.

ReLU ReLU units are perhaps the most common activation function used in current neural network research. They are defined as $g(z) = \max\{0, z\}$, for some logit z. These types of units are easy to optimise as they are similar to linear units, except for the fact that they are zero across half their domain. This means that they are much less likely to saturate, and thus will allow the error derivatives to pass through, somewhat mitigating the issues that plague sigmoid units. There have been many extensions to vanilla ReLU activations, such as Leaky ReLU and parametric ReLU (PReLU).

Chapter 3

Literature Review

The vast majority of early approaches to the problem of human action recognition employ a hand-crafted feature extraction process. In early work in action recognition, this process involved detecting interest points using some form of interest point detector [6; 43; 17; 29]. One of the most popular of these detectors was an extension to the well-known Harris interest point detector into the video domain. It is known as space-time interest points (STIPs) [6]. The approach leverages a three-dimensional Gaussian kernel extended into the temporal domain. Another common method to obtain interest points is dense sampling. Some common local descriptors computed around interest points include HOG3D (a three-dimensional extension of the HOG descriptor), and jets. Since then, there have been many different approaches that attempt to solve the problem, many of the recent approaches fully or partially leveraging deep learning techniques.

3.1 Engineered Features

Many early approaches to action recognition adopt hand-crafted features. Such approaches attempt to incorporate domain knowledge into the extracted features. These can be loosely grouped into three groups: *spatial*, *motion*, and *other*. *Spatial* are approaches that only explicitly incorporate spatial or appearance information into the features. *Motion* are approaches approaches that explicitly incorporate both spatial / appearance and motion / flow information into the features. Lastly, *other* are all other approaches. We structure this section in accordance with this loose grouping. A summary of previous literature using engineered features is given in Table 3.1.

Table 3.1: Summary of previous approaches adopting the engineered features approach. The first second column describes the features used in the approach. *Traj* means a trajectory feature, dense means dense sampling, and *CMC* means camera motion correction.

Approach	Features	Classifier	Dataset	Result
[17]	STIPs + 4-Jets + BoW	SVM	KTH	71.7
[54]	STIPs + HOG/HOF + BoW	Chi-squared SVM	KTH	91.8
[55]	Object Bank	SVM	KTH	98.2
[55]	Object Bank	SVM	UCF Sports	95.0
[55]	Object Bank	SVM	UCF50	57.9
[55]	Object Bank	SVM	HMDB51	26.9
[56]	Action MACH	Action MACH	KTH	88.66
[57]	Hough Forest	Hough Forest	KTH	92.0
[57]	Hough Forest	Hough Forest	Weizmann	95.6
[16]	STIPs + 3D HOG + BoW	Chi-squared SVM	KTH	91.4
[16]	STIPs + 3D HOG + BoW	Chi-squared SVM	Weizmann	84.3
[16]	STIPs + 3D HOG + BoW	Chi-squared SVM	Hollywood	24.7
[58]	STIPs + HOG/HOF + Hierarchical BoW	MKL + SVM	KTH	95.53
[58]	Dense + 3D HOG + Hierarchical BoW	MKL + SVM	UCF Sports	87.27
[11]	Dense + Traj + HOG + HOF + MBH + BoW	Chi-squared SVM	KTH	94.53
[11]	Dense + Traj + HOG + HOF + MBH + BoW	Chi-squared SVM	YouTube	84.2
[11]	Dense + Traj + HOG + HOF + MBH + BoW	Chi-squared SVM	Hollywood2	58.3
[11]	Dense + Traj + HOG + HOF + MBH + BoW	Chi-squared SVM	UCF Sports	88.2
[18]	Dense + Traj + HOG + HOF + MBH + Fisher Vector + CMC	Chi-squared SVM	Hollywood2	64.3
[18]	Dense + Traj + HOG + HOF + MBH + Fisher Vector + CMC	Chi-squared SVM	HMDB51	57.2
[18]	Dense + Traj + HOG + HOF + MBH + Fisher Vector + CMC	Chi-squared SVM	Olympic Sports	91.1
[18]	Dense + Traj + HOG + HOF + MBH + Fisher Vector + CMC	Chi-squared SVM	UCF50	91.2
[59]	MPEG flow + iDT	Chi-squared SVM	HMDB51	46.7
[59]	MPEG flow + iDT	Chi-squared SVM	Hollywood2	60.3
[12]	iDT + Stacked Fisher Vectors	Linear SVM	HMDB51	66.79

3.1.1 Spatial

To obtain an action representation for a video sequence $I(x, y, t)$, [17] first detects STIPs. Fourth-order local jets are then computed around these STIPs at the relevant spatial and temporal scales σ and τ [17]:

$$\mathbf{l} = (L_x, L_y, L_t, L_{xx}, \ldots, L_{tttt}) \tag{3.1}$$

where L is the scale-space representation of the input sequence; and $L_{x^m y^n t^k} = \sigma^{m+n}\tau^k(\partial_{x^m y^n t^k} g) * I$ [17] are the normalised derivatives of L. The jets are tasked with encoding pertinent appearance and motion information in the neighbourhood of the points.

These local jets are then quantised into a global feature vector using a bag-of-words approach with K-Means clustering to form a histogram of visual word occurrences. The K-Means clustering results in a vocabulary of visual words h_i. The histogram of occurrences for each visual word is then used as the final representation. Thus, the method evaluates two representations: the local jet features, and the quantised histogram feature. An SVM is then trained on these two representations. The kernel for the

SVM trained on histogram features is defined as $K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma\chi^2(\mathbf{x}, \mathbf{y}))$ [17]. The kernel used in the SVM for the local jet features is given by $K_L(\boldsymbol{L}_h, \boldsymbol{L}_k) = (1/2)[\hat{K}_L(\boldsymbol{L}_h, \boldsymbol{L}_k) + \hat{K}_L(\boldsymbol{L}_h, \boldsymbol{L}_k)]$, with [17]:

$$\hat{K}_L(\boldsymbol{L}_h, \boldsymbol{L}_k) = \frac{1}{n_h} \sum_{j_h=1}^{n_h} \max_{j_k=1,\ldots,n_k} \{K_l(\mathbf{l}_{jh}, \mathbf{l}_{jk})\} \tag{3.2}$$

where $\boldsymbol{L}_i = \{\mathbf{l}\}_{j=1}^{n_i}$; $\mathbf{l}_j$ is a jet descriptor of STIP j in video sequence i; and [17]

$$K_l(\mathbf{x}, \mathbf{y}) = \exp(-\rho(1 - \frac{< \mathbf{x} - \boldsymbol{\mu}_\mathbf{x} | \mathbf{y} - \boldsymbol{\mu}_\mathbf{y} >}{||\mathbf{x} - \boldsymbol{\mu}_\mathbf{x}|| \cdot ||\mathbf{y} - \boldsymbol{\mu}_\mathbf{y}||})) \tag{3.3}$$

where $\boldsymbol{\mu}_\mathbf{x}$ is the mean of $\mathbf{x}$. This approach resulted in a best average accuracy of 71.7% on the KTH dataset. This is relatively low performance on the KTH dataset, and the performance can most likely be attributed to the local features not being able to capture as much of the pertinent motion information as needed, and neither representation is encoding much spatial information of the actions either. However, the method is relatively fast to compute when compared to some of the other methods described below.

[16] created a novel, gradient-based 3D descriptor that is an extension of HOG3D [36]. In order to make gradient computation more efficient, mean gradient vectors are computed using an *integral video* scheme. In this way, 3D gradient vectors can be computed efficiently at any arbitrary scale in a video. The integral video is an extension of the *integral image* popularised by [60] in order compute efficient Haar features. Formally, the integral video iv_{∂_x} for v_{∂_x} is given by [16]:

$$iv_{\partial_x} = \sum_{x' \leq x, y' \leq y, t' \leq t} v_{\partial_x}(x', y', t') \tag{3.4}$$

where v_{∂_x} is the partial derivative of the video with respect to x. The computation is similar for iv_{∂_y} and iv_{∂_t}. Thus, the mean gradient $\bar{\mathbf{g}}_\mathbf{b} = (\bar{g}_{\mathbf{b}\partial_x}, \bar{g}_{\mathbf{b}\partial_y}, \bar{g}_{\mathbf{b}\partial_t})$ can be efficiently computed for any 3D volume $\mathbf{b}$ [16]:

$$\begin{aligned}\bar{g}_{\mathbf{b}\partial_x} = [&iv_{\partial_x}(x+w, y+h, t+l) - iv_{\partial_x}(x, y+h, t+l) - iv_{\partial_x}(x+w, y, t+l) + iv_{\partial_x}(x, y, t+l)] \\ -[&iv_{\partial_x}(x+w, y+h, t) - iv_{\partial_x}(x, y+h, t) - iv_{\partial_x}(x+w, y, t) + iv_{\partial_x}(x, y, t)]\end{aligned} \tag{3.5}$$

where (x, y, t) is the volume's position, and w, h, and l are the width, height, and length of the volume, respectively. The computation is similar for $\bar{g}_{\mathbf{b}\partial_y}$ and $\bar{g}_{\mathbf{b}\partial_t}$. The next phase is the gradient orientation

quantisation phase. In 2D, n-bin histograms of gradient orientations can be seen as an approximation of a circle with a regular n-sided polygon, where a bin of the histogram for the gradient orientation quantisation corresponds to a side of this 2D polygon. In 3D, the chosen polyhedrons in this work are the 12-sided dodecahedron and the 20-sided icosahedron (since in 2D the $8-sided$ octagon is commonly chosen). It should be noted that these are both *platonic solids* [16].

Assume the centre of gravity of a regular n-sided polygon is at the origin of a 3D Euclidean space. To quantise gradient vector $\bar{\mathbf{b}}_\mathbf{b}$ with respect to its orientation, it first needs to be projected onto the axes of the origin of the Euclidean space, at the centre position of all the faces of the polygon. This is achieved through multiplication with a matrix $\boldsymbol{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n)$, where $\mathbf{p}_i = (x_i, y_i, t_i)$ are the centre positions of all n faces. The projection $\hat{\mathbf{q}}_\mathbf{b}$ of $\bar{\mathbf{g}}_\mathbf{b}$ is then computed as follows [16]:

$$\hat{\mathbf{q}}_\mathbf{b} = (\hat{q}_{\mathbf{b}1}, \hat{q}_{\mathbf{b}2}, \ldots, \hat{q}_{\mathbf{b}n}) = \frac{\boldsymbol{P} \cdot \bar{\mathbf{g}}_\mathbf{b}}{||\bar{\mathbf{g}}_\mathbf{b}||_2} \tag{3.6}$$

This normalised projected vector $\hat{\mathbf{q}}_\mathbf{b}$ is then thresholded. The threshold is given by $t = \mathbf{p}_i^T \mathbf{p}_j$ for neighbouring axes $\mathbf{p}_i$ and $\mathbf{p}_j$. Then, all negative elements are set to zero after subtracting t from $\hat{\mathbf{q}}_\mathbf{b}$. This results in a thresholded histogram $\hat{\mathbf{q}'}_\mathbf{b}$. $\hat{\mathbf{q}'}_\mathbf{b}$ then defines the distribution of the gradient magnitudes [16]:

$$\mathbf{q}_\mathbf{b} = \frac{||\bar{\mathbf{b}_\mathbf{b}}||_2 \cdot \hat{\mathbf{q}'}_\mathbf{b}}{||\hat{\mathbf{q}'}_\mathbf{b}||_2} \tag{3.7}$$

Next, consider a cuboid $\mathbf{c}$ in a video. Compute the histogram over the set of $S \times S \times S$ sub-volumes of $\mathbf{c}$. Then, for each sub-volume, the integral video representation is used to compute the mean gradient vectors, followed by quantisation into $\mathbf{q}_{\mathbf{b}_i}$. This histogram $\mathbf{h}_\mathbf{c}$ for cuboid $\mathbf{c}$ is then defined as [16]:

$$\mathbf{h}_\mathbf{c} = \sum_{i=1}^{S^3} \mathbf{q}_{\mathbf{b}_i} \tag{3.8}$$

This histogram can be computed for any arbitrary scale since the number of supporting mean gradient vectors is fixed, and by using the integral video representation. To compute the final descriptor, the above histogram is computed for each of the pre-defined $M \times M \times N$ cells in the video. All these histograms are normalised, concatenated, clipped, and re-normalised. This re-normalised descriptor is then the final feature representation for the video. Similar to above approaches, a video sequence is represented as a bag-of-words using STIPs. A random sampling of the training descriptors are sampled and clustered into a visual vocabulary using K-Means. Then, histograms of visual word occurrences

are fed into a SVM with the χ^2 kernel for classification. Accuracies of 91.4%, 84.3%, and 24.7% were obtained on the KTH, Weizmann, and Hollywood datasets, respectively. Although performance on the first two datasets is reasonable, performance on the more difficult dataset, Hollywood, is relatively poor in comparison. This suggests that these 3D gradient features are not able to encode enough spatial and temporal information to perform well in more difficult, realistic video settings.

3.1.2 Motion

[54] detected STIPs at multiple spatial and temporal scales (σ_i^2, τ_j^2), where $\sigma_i = 2^{(i+1)/2}$, $i = 1, \ldots, 6$; and $\tau_j = 2^{j/2}$, $j = 1, 2$. The multiple scale approach is chosen over the scale selection method for computational complexity reasons, and independence from scale selection artefacts. To encode the motion and appearance information from these detected local features, histograms are computed in the space-time volumes around the points / features. The size of these space-time volumes $(\Delta_x, \Delta_y, \Delta_t)$ depends on the spatial and temporal detection scales: $\Delta_x, \Delta_y = 2k\sigma$; and $\Delta_t = 2k\tau$, for some $k \in \mathbb{R}$. Then, each of these volumes is subdivided further into a $n_x \times n_y \times n_t$ grid of cuboids. For each cuboid, HOG and HOF features are computed. These histograms are normalised and concatenated. A bag-of-words is then built by computing a visual vocabulary using the K-Means clustering algorithm on a subset of the above-described feature vectors. Multiple different grid shapes for the spatio-temporal volumes were tested. For classification a multi-channel χ^2 SVM was used. This approach resulted in an average accuracy of 91.8% on the KTH dataset. This method was state-of-the-art at the time, due to the features being able to capture more of the pertinent motion and appearance information than that of [17], for example.

As with most early action recognition work, [61] detect STIPs as a base to compute video descriptors. Multiple descriptors were evaluated. Firstly, histograms of spatio-temporal gradients and optical flow are computed over either the entire neighbourhood of the interest point, as well as over several $M \times M \times M$ smaller neighbourhoods around the interest point (resulting in so-called *position-dependent histograms*). Another descriptor is the D-dimensional PCA projection of either the optical flow or spatio-temporal gradient vectors computed over local scale and velocity-normalised spatio-temporal neighbourhoods around the interest points. In order to classify actions, multiple similarity / distance metrics are evaluated in a nearest neighbour classification scheme: 1. normalised scalar product; 2. Euclidean distance; and 3. the χ^2 distance. The best performance achieved was by position-dependent histograms of optical flow.

[29] evaluated various interest point and descriptor combinations on a host of datasets. The interest point

detection algorithms evaluated were: 1. **Harris3D** Computed at multiple spatial and temporal scales instead of performing scale selection; 2. **Cuboids** Based on temporal Gabor filters; 3. **Hessian** An extension Hessian saliency measure into the temporal domain; and 4. **Dense Sampling** Points extracted at regular positions and scales in both space and time. The descriptor types that were evaluated were: 1. **HOG/HOF** Computed in the same way as [54]; 2. **HOG3D** An extension of the SIFT descriptor into the temporal domain. 3. **ESURF** An extension of the regular SURF descriptor to videos. Similar to the approaches above, video sequences are represented as a bag-of-words. The descriptors are quantised into visual words using K-Means clustering with $V = 4000$. These visual words are then used to quantise the local features into global histograms of visual word occurrences, which are the final video representation. An SVM is used with the χ^2 kernel (defined above) is used for classification. One key finding is that in realistic video settings, dense sampling outperforms other interest point detection methods, while increasing the number of features by a factor of 15. Another key finding is that the HOG/HOF descriptors seem to, in general, perform well. This research suggests that focusing on dense sampling as a detection mechanism might prove useful.

The approach taken by [58] is to develop an improved technique for bag-of-words-based recognition, by creating a vocabulary that is richer and more representative of the actions. For interest points, two methods are investigated: dense sampling and STIPs. For dense sampling, HOG3D descriptors are extracted, whereas for STIPS, HOG and HOF descriptors are used. A *level-0* vocabulary is then formed by clustering each feature type separately using K-Means with k centres. These k centres are the level-0 visual words. Then, each video clip V can be described by [58]:

$$V = \{< x_1, y_1, t_1, w_1^{(0)} >, \ldots, < x_{n_v}, y_{n_v}, t_{n_v}, w_{n_v}^{(0)} >\} \tag{3.9}$$

where each $< x_i, y_i, t_i, w_i^{(0)} >$ tuple consists of the spatial location, frame number, and word index, respectively (per interest point). The total number of interest points for the video is given by n_v. The 0 in $w^{(0)}$ indicates that the vocabulary is a level-0 vocabulary. Lastly, $w_i^{(0)} \in \{1, 2, \ldots, k\}$.

To form the level-1 vocabulary, for each interest point, the N closest interest points are collected. Let the set of N closest points to p be defined by [58]:

$$\mathcal{N}(p) := \{p, q_1, q_2, \ldots, q_{N-1}\} \tag{3.10}$$

Closeness is defined by a normalised Euclidean distance on the point's 3D position. These points can then be quantised, depending on its location in space and time with respect to the central point, into one of eight orientation bins. A matrix of dimension $N \times 8k$ is computed to facilitate this quantisation. The r^{th} row of this matrix defines a cumulative histogram of that point's r ranked neighbours [58]. The row entries themselves define the number of neighbours in each bin. This matrix is then flattened into a vector of dimension $8kN$. This vector serves as the representation for a single level-1 descriptor. PCA is then applied to these descriptors to reduce their dimensionality. It should be noted that, since the point's neighbours are put into the descriptor according to their rank (rather than distance), the orientation bins do not have a pre-determined scale. With these neighbourhood descriptors, the above process is repeated using the neighbourhoods themselves as the central points. K-Means is run on a sample of reduced-dimension descriptors, thus quantising the neighbourhoods into a visual vocabulary. This whole process is repeated until $L + 1$ vocabularies are formed. Each level of this hierarchy encodes the space-time layout of the component features [58]. We can then map an the raw features on a video to $L + 1$ BoW histograms. A MKL (multiple kernel learning) technique is used as part of the modelling process. These learned kernels are then used with an SVM to classify the clips into the action classes. 95.53% and 87.27% accuracies on KTH and UCF Sports were obtained. This was, at the time, state-of-the-art performance of both datasets. The hierarchical nature of the approach was able to encode more pertinent spatial and temporal information about the actions.

A seminal approach for the current state-of-the-art and recent promising results was introduced by [11]. The research was inspired by the great results of dense sampling for the task of image classification. Firstly, dense trajectory features are computed. We can define the shape S of a trajectory as a sequence of displacement vectors $S = (\Delta P_t, \ldots, \Delta P_{t+L-1})$, where $\Delta P_t = (P_{t+1} - P_t) = (x_{t+1} - x_t, y_{t+1} - y_t)$. This shape of a trajectory encodes local motion patterns. This vector is then normalised by the sum of the displacement vectors [11]:

$$S' = \frac{(\Delta P_t, \ldots, \Delta P_{t+L-1})}{\sum_{j=t}^{t+L-1} ||\Delta P_j||} \tag{3.11}$$

S' is then referred to as the *trajectory descriptor*. To leverage motion information from these trajectory descriptors, descriptors are computed within a space-time volume around the trajectory. First, HOG and HOF descriptors are computed, with both being L_2-normalised. Then, MBH descriptors are computed and L_2-normalised. A visual of the full flow to compute the final representation is shown in Figure 3.1. Again, a bag-of-words approach is taken by estimating a visual vocabulary using K-Means and using the

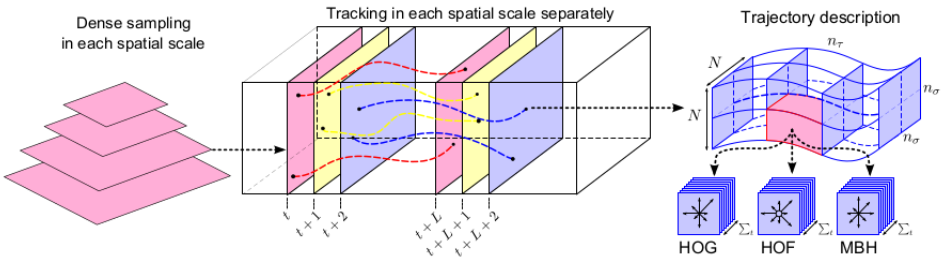


Figure 3.1: Illustration of the dense trajectory description [11].

Figure 3.2: Comparison of dense trajectory features vs regular KLT-based motion features [11]. Red dots are point positions in the current frame. It is clear that dense trajectories are more robust to noise and irregular motions, particularly at shot boundaries. Further, they capture complex motion patterns more accurately.

resulting histogram of visual word occurrences to train a SVM with a χ^2 kernel. By way of comparison, the features computed using Farneback's dense optical flow were compared with using a sparse interest point tracker such as KLT (Lucas-Kanade). The features from the dense optical flow consistently outperformed the KLT features on all datasets, often significantly. This is not surprising, as can be seen by Figure 3.2, which visualises these different trajectories.

Results include average accuracies of 94.53%, 84.2%, 58.3%, and 88.2% on the KTH, YouTube, Hollywood2, and UCF Sports datasets, respectively. This shows the effectiveness of the method, as it is able to perform well even on the more difficult, realistic datasets such as YouTube and Hollywood2. This is likely due to the fact that both zero-order (HOF) and first-order (MBH) motion information is taken into

account, in addition to appearance (HOG) and trajectory information. All of this combined provides a rich representation of the video, and by extension, the action.

[18] improved this approach by accounting for camera motion, which can greatly negatively affect the effetiveness and robustness of motion features in realistic videos. To estimate camera motion (i.e. global background motion), it is assumed that two consecutive frames are related by a homography. This assumption holds since, in most cases, global motion between consecutive frames is often small. It is essential that this homography estimation is robust in order to ensure inconsistent matches can be removed. First, frame-to-frame correspondences must be found. Two approaches are performed in tandem to do this. These two approaches allow for the finding of sufficient and complementary matches. SURF features are found and matched based on the nearest-neighbour rule. Additional to these SURF features, motion vectors are sampled from the optical flow performed using Farneback's method. Motion vectors are only sampled if the corresponding feature point satisfies the Shi-Thomasi criterion. These two approaches are complementary since SURF focuses on blob-like structures, while the Shi-Thomasi criterion ensures motion points focus on corner-like and edge-like structures [18]. The homography is then estimated using the RANSAC [28] algorithm. This allows for the image to be rectified by removing the camera motion. The removal of camera motion is important since motion-based descriptors like HOF and MBH became far less reliable / powerful in the presence of camera motion. Another advantage is the trajectories generated by camera motion can also be removed applying a threshold to the displacement vectors in the camera-motion-corrected optical flow field.

One issue with the camera motion estimation is that it is very common for a human to dominate the frame, and camera motion is often inconsistent with human motion. Thus, a technique is proposed to remove these inconsistent matches due to human motion using a state-of-the-art human detector. Matches inside the bounding box around the detected human are removed so that the homography can be estimated more accurately. Without this human detection phase, the estimation of the homography can become inaccurate since many features from the human motion become inlier matches [18]. Similar to the approach in [11] above, points are densely sampled at several spatial scales; points in homogeneous regions are pruned; tracking of points through the median filtering technique; trajectories are limited to a length L; static feature trajectories are pruned; and trajectories with large sudden displacements are pruned. Then, various descriptors are computed for each trajectory: Trajectory (30-dimensional); HOG (96-dimensional); HOF (108-dimensional); and MBH (192-dimensional). Normalised displacement vectors

are concatenated to form the trajectory descriptor. The HOG, HOF, and MBH descriptors are normalised using the *rootSIFT* approach, which is the square root of each dimension after L_1 normalisation. Lastly, a bag-of-words approach is taken by estimating a vocabulary of visual words using K-Means clustering on these descriptors. Then histograms of visual word occurrences are used to train an SVM with the χ^2 kernel for classification. Results show, firstly, that the iDT feature consistently outperforms the original dense trajectory feature [11] on all datasets evaluated. Secondly, another feature encoding technique - Fisher vectors - was also evaluated to compare with the bag-of-words approach. This showed that Fisher vectors consistently outperformed bag-of-words on all datasets. Also, results were compared when the removal of inconsistent matches due to humans using the human detector was, and was not, performed. Results show that using the human detector helped in all cases (i.e. accuracy was higher on all datasets). Results obtained using this approach were 64.3%, 57.2%, 91.1%, and 91.2% on the Hollywood2, HMDB51, Olympic Sports, and UCF-50 datasets, respectively. These are the state-of-the-art hand-crafted features for human action recognition. Since many different types of information are combined, they perform well on different datasets of varying difficulties. Trajectories encode local motion patterns; HOG encodes appearance and spatial information; HOF encodes zero-order motion information; and MBH encodes first-order motion information. Lastly, the removal of camera motion makes the motion-based features even more effective. This means that the feature benefits from many sources to provide a rich, distinctive representation for a video sequence.

[59] attempted to improve the real-time capabilities of iDTs by making them more computationally efficient. To achieve this, firstly, motion information is extracted directly from the video compression used by the video codec when compressing a video (termed *MPEG flow*). The second way of achieving this was by constructing descriptors from very sparse motion measurements. Although the *MPEG flow* is slightly more noisy than its Lucas-Kanade and Farneback counterparts, it provides for a reliable motion estimate. Using Fisher vectors to encode the extract trajectory, HOG, HOF, and MBH features yields accuracies of 60.3% and 46.7% on the Hollywood2 and HMDB51 datasets, respectively.

The approach introduced by [12] was partly inspired by deep neural networks, and how they learn representations of data. The lower layers of the networks learn low-level features of the input, and as the network gets deep, the features become more semantic and richer representations of the input. The representation in this work is known as *stacked Fisher vectors* (SFVs). Fisher vectors essentially aggregate local patterns into one long vector by encoding local features of actions in a video in a high-dimensional

space. Therefore, global structure cannot be effectively encoded using this traditional Fisher vector representation, since it represents a video from the perspective of a local feature space. SFVs are obtained by stacking two Fisher vector encoding layers. In this way, they are a 'deep' structure. The motivation of the SFV is to describe a video at a higher-level that captures semantic, global information about the video.

The first-layer Fisher vector is computed as follows. Consider a video V of size $W \times H \times L$ (width, height, and number of frames, respectively). iDTs (characterised by concatenated trajectory, HOG, HOF, and MBH descriptors) are extracted. Let $\boldsymbol{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times N}$ be the trajectory features for V. The features are whitened and decorrelated using PCA before being encoded as Fisher vectors. This is done in order to meet the diagonal covariance assumption of the GMM used in the Fisher vector encoding scheme. The features are then encoded as Fisher vectors using a GMM with K_1 modes. Call this set of sparse, high-dimensional Fisher vectors $\boldsymbol{X'} = [\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_n]$ *tiny Fisher vectors*. These tiny Fisher vectors are then aggregated for multiple spatial and temporal scales (two scales for width and height, and three for time) for densely sampled cuboids, with spatial and temporal strides of δ_s and δ_t. Aggregation is only performed for sub-volumes that have more than a threshold T trajectories, to avoid incorporating sub-volumes with no motion (which could have detrimental effects on the representation). These locally-aggregated Fisher vectors $\boldsymbol{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n] \in \mathbb{R}^{2K_1 d \times M}$ are termed *local Fisher vectors*, where M is the number of sub-volumes with more than T trajectories. These local Fisher vectors are then power and L_2-normalised for each dimension, and then jointly L_2-normalised.

Since these local Fisher vectors are very high-dimensional, max-margin dimensionality reduction is applied such that the compressed local Fisher vectors have a dimensionality comparable to the local features in the first layer. These compressed Fisher vectors are then de-correlated using PCA and whitened. These are then fed into the second layer Fisher vector. After a GMM with K_2 modes is estimated, another layer of Fisher vectors is added with the reduced-dimensionality local Fisher vectors from the first layer, and are aggregated over the entire video (similarly to the first layer). The same normalisation scheme as the first layer is then applied. This normalised representation serves as the final *stacked Fisher vector* representation for the video. A linear SVM is then trained on the stacked Fisher vectors for classification. Results are an accuracy of 66.79% on the HMDB51 dataset. This performance illustrates the representational power of the Fisher vector, and when combined in a multi-layer fashion, can increase performance even further. Performance on the HMDB51 with a normal Fisher vector encoding (i.e. not

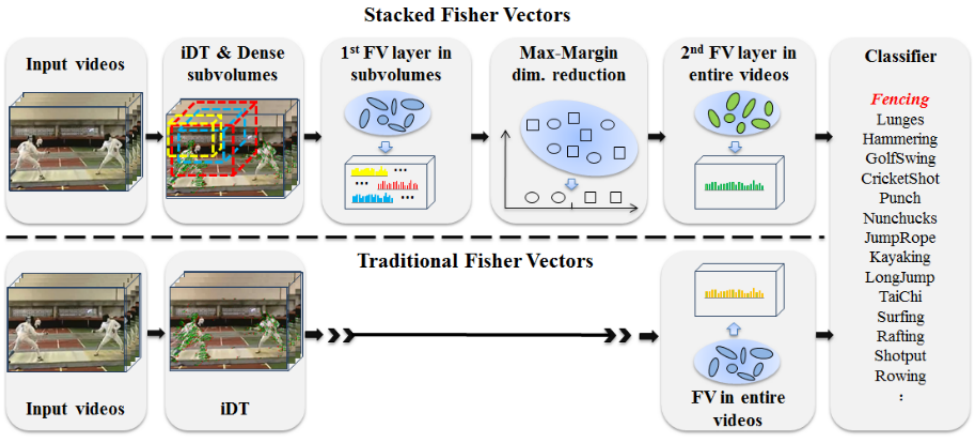


Figure 3.3: Comparison between the stacked, and traditional, Fisher vector representations [12].

stacked) was 57.2%, which shows how the proposed stacking approach aids performance. A comparison of how traditional Fisher vectors differ from stacked Fisher vectors can be seen in Figure 3.3.

3.1.3 Other

Some other interesting approaches include [55], in which the *Object Bank* [62] approach to image representation was extended into video to form a representation termed *Action Bank*. This method is the current state-of-the-art on the KTH dataset. This approach uses a combined output of many action detectors. Each of these detectors outputs a correlation volume to serve as the representation of a video sequence. More formally, for a given action bank with N_a detectors, each detector is run for N_s spatio-temporal scales to yield $N_a \times N_s$ correlation volumes. These volumes go through a max-pooling scheme adapted for the volumetric case. Each action-scale pair has an associated 73-dimensional vector. This results in a final action bank feature vector that is $N_a \times N_s \times 73$-dimensional. The action detector used in the work is the *action spotting* detector [63] since this detector is invariant to appearance variation, is able to localise an action from a single template, is efficient, and can be naturally interpreted as a decomposition of a video into space-time energies such as *leftward motion* and *flicker* [55]. The method is based on the assumptions that an action is comprised of these low-level motion energies in different spatio-temporal orientations. A regular SVM is used for classification. Results obtained were 98.2% on KTH, 95.0% on UCF Sports, 57.9% on UCF50, and 26.9% on HMDB51. The two main downsides of this approach are its massive computational inefficiency (e.g. a video from UCF50 took anywhere from 0.4 - 34 hours), which is likely the reason the method has not been researched further, as well as the fact it performs comparatively poorly on realistic datasets.

[56] generalise the traditional MACH filter to encompass human actions. This filter aggregates all training images into a single template. This template composition is achieved by optimising four metrics: 1. Average Correlation Height; 2. Average Correlation Energy; 3. Average Similarity Measure; and 4. Output Noise Variance [56]. The resultant composite template effectively expresses the rough shape and appearance of an object [56]. The Fast Fourier Transform is then applied to these templates to map them into the frequency domain. It is in the frequency domain that mapped templates are correlated with test images. This results in a surface in which the highest peak corresponds to the most likely location of the object of interest in the image [56]. This MACH filter is generalised to encompass actions by synthesising a template estimated from the spatio-temporal volumes of action sequences [56]. This resulted in an average accuracy of 88.66% on the KTH dataset.

[57] implement a Hough-transform-based approach for human action recognition. They assume that for each training sample (i.e. video), there is an associated: 1. 2D bounding box for each frame around the person; 2. action label; and 3. temporal boundaries of the action cycle in the video [57]. It should be noted that these are fairly strong assumptions. These assumptions essentially mean that for each training sample, we have the track for the action, and that the action only occurs once during that video's duration. In order to learn the mapping from the action tracks to a Hough space, a Hough forest structure is used (as it is more efficient than conventional codebook approaches). Each tree in the forest is constructed from a set of patches $\{\mathcal{P}_i = (\mathcal{I}_i, c_i, \mathbf{d}_i)\}$, where $\mathcal{P}_i$ is a sampled cuboid from the action track; $\mathcal{I}_i$ are features extracted at a patch; c_i is the target class; and $\mathbf{d}_i \in \mathbb{R}^3$ is a displacement vector from the centre of the patch to the centre of the action track [57]. The extracted features are the grayscale intensity, absolute value of x and y, time derivatives, and the absolute value of optical flow in x and y [57]. This approach resulted in a 95.6% and 92.0% accuracy on the Weizmann and KTH datasets, respectively.

3.2 Learned Features

Many recent approaches to action recognition adopt deep learning. Such approaches attempt to automatically learn good representations of the input videos in order to classify the actions. These approaches can be grouped into two groups: *Without Flow* and *With Flow*. *Without Flow* are approaches that do not include optical flow in the modelling process, whereas *With Flow* are approaches approaches that do include optical flow. We structure this section in accordance with this grouping. A summary of previous literature using learned features is given in Table 3.2.

Table 3.2: Summary of previous approaches adopting the learned features approach.

Approach	Model	Dataset	Result
[43]	CNN + ISA + PCA	KTH	93.9
[43]	CNN + ISA + PCA	Hollywood2	53.3
[43]	CNN + ISA + PCA	UCF Sports	86.5
[43]	CNN + ISA + PCA	YouTube	75.8
[49]	3D CNN + RNN	KTH	92.17
[14]	2D RGB CNN + 3D Flow CNN + SVM	UCF-101	88.0
[14]	2D RGB CNN + 3D Flow CNN + SVM	HMDB51	59.4
[64]	3D RGB CNN + 3D Flow CNN + ImageNet	UCF-101	92.5
[64]	3D RGB CNN + 3D Flow CNN + ImageNet	HMDB51	65.4
[64]	3D RGB CNN + 3D Flow CNN + ImageNet + iDT	UCF-101	93.5
[64]	3D RGB CNN + 3D Flow CNN + ImageNet + iDT	HMDB51	69.2
[13]	3D CNN + Linear SVM	UCF-101	85.2
[13]	3D CNN + iDT + Linear SVM	UCF-101	90.4
[65]	Large temporal resolution + 3D CNN + Brox Flow	UCF-101	91.7
[65]	Large temporal resolution + 3D CNN + Brox Flow	HMDB51	64.8
[65]	Large temporal resolution + 3D CNN + Brox Flow + iDT	UCF-101	92.7
[65]	Large temporal resolution + 3D CNN + Brox Flow + iDT	HMDB51	67.2
[19]	3D RGB CNN + 3D Flow CNN + ImageNet + Kinetics	UCF-101	98.0
[19]	3D RGB CNN + 3D Flow CNN + ImageNet + Kinetics	HMDB51	80.9
[66]	2D Pose CNN	UCF-101	65.2
[66]	2D Pose CNN	HMDB51	43.7
[66]	I3D + 2D Pose CNN	UCF-101	98.2
[66]	I3D + 2D Pose CNN	HMDB51	80.9

3.2.1 Without Flow

The Independent Subspace Analysis (ISA) algorithm was extended to video by [43] in order to perform human action recognition. ISA is an unsupervised algorithm that learns from unlabelled image patches. The conventional ISA algorithm for static images can be seen as a two-layer network with square and square-root activation functions in the first and second layers, respectively. The weights in the first layer W are learned, while the second-layer weights V are fixed. The freezing of weights V is done in order to represent the subspace structure of the neurons in the first layer [43]. This is achieved by, in the second layer, pooling over a small neighbourhood of units in the first layer. Formally, given input x^t, the activation for the i^{th} second-layer unit is $p_i(x^t; W, V) = \sqrt{\sum_{k=1}^{m} V_{ik} (\sum_{j=1}^{n} W_{kj} x_j^t)^2}$. ISA then learns

weight matrix W by solving [43]:

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{m} p_i(x^t; W, V) \quad \text{subject to } WW^T = \boldsymbol{I} \tag{3.12}$$

where the x^ts are the whitened input samples. $W \in \mathbb{R}^{k \times n}$ are the first-layer weights, and $V \in \mathbb{R}^{m \times k}$ are the second-layer weights. The orthonormal constraints ensure that the features are diverse. This ISA algorithm becomes prohibitive when the input patches become too large, such as in the case of video. This is because an orthogonalisation step must be performed at every step of the projected gradient descent algorithm, and the cost of this step grows as a cubic function of the input dimension. To solve this, a CNN architecture is designed such that PCA and ISA are used as sub-units for unsupervised learning. Essentially, the ISA algorithm is first trained on small input patches. This learned network is then convolved with a larger region of the input image. The output of this convolutional step is preprocessed by PCA and fed into the next layer which is again ISA. This process ensures that this next layer only works with low-dimensional inputs. Videos are fed into this network by flattening a sequence of image patches into a vector, and feeding this as input into the network. In order to discard non-informative features, a thresholding technique is employed where a feature is discarded if the L_1-norm of its activation is below a certain threshold δ. This approach resulted in 93.9%, 53.3%, 86.5%, and 75.8% average accuracies on the KTH, Hollywood2, UCF Sports, and YouTube datasets, respectively.

A seemingly-natural deep learning approach to classify human actions was taken by [49]. Firstly, a 3D CNN is created by simply using 3D kernels instead of 2D kernels as in conventional 2D CNNs. The architecture is 10 layers in total: the input layer, two convolutional-rectification-pooling layers, and two dense layers. This 3D CNN operates on input volumes of dimension $34 \times 54 \times 9$, and is tasked with automatically learning salient spatio-temporal features directly from these input RGB volumes. Then, an RNN was used to learn a labelling of the input sequence based on the accumulation of multiple individual decisions, where each decision corresponds to a small temporal neighbourhood from the 3D CNN's learning process [49]. This is achieved by using one hidden layer of LSTM cells in the RNN. The input into this RNN is 120 output values from the final convolutional layer in the 3D CNN. The only forms of pre-processing are a spatial down-sampling of the videos by a factor of two (to reduce memory requirements), and 3D contrast normalisation in a $7 \times 7 \times 7$ neighbourhood. Additionally, data augmentation is performed, to increase the number of training examples, by generating vertically-

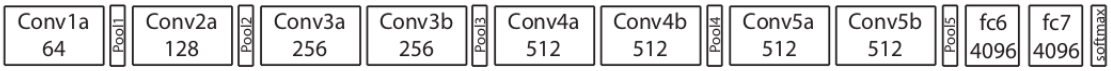


Figure 3.4: Illustration of the C3D network architecture [13].

flipped and mirrored versions of each training sample. A 92.17% accuracy on KTH dataset was achieved. This approach shows the advantage of automatically-learned features through the use of deep learning. With minimal preprocessing and feature engineering, a high accuracy was achieved on the KTH dataset. One issue with this method is the unknown with regards to performance on more complicated, realistic datasets.

[13] employed a 3D CNN that was tasked with learning the spatio-temporal features necessary to perform accurate action recognition. A 3D architecture was preferred over a 2D architecture since the temporal information inherent in the input volumes is lost with a 2D architecture. 3D convolutions preserve this temporal information, as does 3D pooling compared to 2D pooling. All videos are resized to have a spatial dimension of 128×171, and training examples are randomly-cropped $16 \times 112 \times 112$ clips from these resized videos. Also, the dimensions of the kernels for convolution are all set to $3 \times 3 \times 3$, as empirical evidence shows that this was the best performing size. All 3D convolution kernels are of dimension $3 \times 3 \times 3$ with a stride of $1 \times 1 \times 1$. All 3D pooling layers are of dimension $2 \times 2 \times 2$ with a stride of $2 \times 2 \times 2$. This network architecture is termed *C3D*. The network architecture can be seen in Figure 3.4. Layers with *conv* are convolutional layers, *pool* are pooling layers, and layers with *fc* are fully-connected / dense layers.

This network is trained from scratch. The learning rate is halved after a certain number of iterations, and trained for 13 epochs in total. In order to perform classification, two methods are used. The first is an average of the predictions of 10 clips randomly extracted from the video, where a clip's prediction is given by the network's prediction of the centre crop. The other is feature vectors obtained by sampling the result of the final dense layer of the network (i.e. the layer before the classification layer). These vectors (known as C3D features) are then used to train a linear SVM. This results in an accuracy of 85.2% on the UCF101 dataset. However, when combined with iDT features, the accuracy increases to 90.4% on UCF101. These results further suggest that a combination of hand-crafted and deep learning features is a good approach to take for action recognition.

3.2.2 With Flow

A method that spawned much further research in action recognition was introduced by [14]. This method is based on the intuition that video can naturally be broken down into spatial and temporal components. The spatial component, defined by the individual frames of the videos, carries information about scenes and objects in the video. The temporal component, defined by motion between frames, carries information about the movement in the video (e.g. that of camera motion and the objects). Both of these components, or streams, are implemented as CNNs. Multiple fusion methods of the output of the two networks are evaluated.

The spatial stream CNN operates on individual RGB frames of the video. Thus, this network makes action predictions for each frame. The static appearance of a frame is hypothesised to be a useful cue for action recognition since actions are often strongly associated with particular objects. The temporal stream CNN is fed some form of pre-computed optical flow input. This is done so that the temporal network does not have to estimate the motion itself. Three different types of inputs are considered for the temporal CNN. The first is optical flow stacking. This is where optical flow vector fields are stacked. An optical flow vector field $\mathbf{d}_t(u, v)$ from frame t to $t + 1$ can be seen as a two-channel image, where the channels correspond to the horizontal and vertical components of the vector field. This stacking of vector fields process results in a volume of dimension $w \times h \times 2L$, where $w \times h$ are the dimensions of the frames; and L is the number of stacked vector fields. An alternative optical flow representation to feed into the temporal CNN is termed *trajectory stacking*. This method replaces flow estimates sampled at the same locations across multiple frames with the flow estimates along trajectories [14]. To fuse the output of both the spatial and temporal networks, two methods are considered: averaging and training a linear SVM on stacked L_2-normalised softmax vectors as features for the SVM. This approach resulted in 88.0% and 59.4% accuracies on the UCF-101 and HMDB51 datasets, respectively. Due to the impressive performance of this approach, especially for deep learning-based approaches to action recognition, it has seen much interest in recent action recognition research. This novel two-stream approach, as well as the optical flow volume inputs, have been widely studied and adapted to try improve the recognition performance.

Different fusing techniques for the two-stream architecture introduced by [14] were evaluated by [64]. The architecture introduced in [14] has two main flaws. Firstly, due to late-fusion being employed in the network, it is unable to learn pixel-wise dependencies between the spatial and temporal features.

Secondly, since the spatial CNN operates on single frames, it is limited in the temporal scale, and as such cannot learn frame-to-frame spatial relationships. In order to define the different fusing techniques, a fusion function must first be defined. A fusion function $f : \mathbf{x}_t^a, \mathbf{x}_t^b \mapsto \mathbf{y}_t$ fuses two input feature maps $\mathbf{x}_t^a \in \mathbb{R}^{H \times W \times D}$ and $\mathbf{x}_t^b \in \mathbb{R}^{H' \times W' \times D'}$ (which are the outputs of a particular convolutional layer in the CNN), at time t, to produce an output feature map $\mathbf{y}_t \in \mathbb{R}^{H'' \times W'' \times D''}$, where W, H, and D are the width, height, and number of channels of the corresponding feature maps, respectively. For simplicity, only the case of $W = W' = W''$, $H = H' = H''$, and $D = D' = D''$ was considered. The following spatial fusion techniques are introduced [64]:

1. **Sum fusion**: $\mathbf{y}^{\text{sum}} = f^{\text{sum}}(\mathbf{x}^a, \mathbf{x}^b)$. This is the element-wise sum of the two input feature maps:

$$y_{i,j,d}^{\text{sum}} = x_{i,j,d}^a + x_{i,j,d}^b \tag{3.13}$$

   where $1 \leq i \leq H$, $1 \leq j \leq W$, $1 \leq d \leq D$. Sum fusion simply defines an arbitrary correspondence between the network outputs being fused.

2. **Max fusion**: $\mathbf{y}^{\text{max}} = f^{\text{max}}(\mathbf{x}^a, \mathbf{x}^b)$, which is simply the element-wise max of the two input feature maps:

$$y_{i,j,d}^{\text{mas}} = \max\{x_{i,j,d}^a, x_{i,j,d}^b\} \tag{3.14}$$

   This also simply defines an arbitrary correspondence between the network outputs being fused.

3. **Concatenation fusion**: $\mathbf{y}^{\text{cat}} = f^{\text{cat}}(\mathbf{x}^a, \mathbf{x}^b)$. This method stacks the two feature maps in corresponding pixel locations over the channels:

$$\begin{aligned} y_{i,j,2d}^{\text{cat}} &= x_{i,j,d}^a \\ y_{i,j,2d-1}^{\text{cat}} &= x_{i,j,d}^b \end{aligned} \tag{3.15}$$

   where $\mathbf{y} \in \mathbb{R}^{H \times W \times 2D}$. Concatenation does not define any correspondence between the input feature maps, and instead tasks layers in the network to learn a suitable correspondence.

4. **Conv fusion**: $\mathbf{y}^{\text{conv}} = f^{\text{conv}}(\mathbf{x}^a, \mathbf{x}^b)$. This method performs concatenation fusion, followed by convolution with a bank of filters $\mathbf{f} \in 1 \times 1 \times 2D \times D$ and biases $b \in \mathbb{R}^D$. $\mathbf{f}$ halves the dimensionality of the input feature maps. Additionally, it is able to model weighted pixel-wise combinations of the two feature maps. If $\mathbf{f}$ is left as a trainable filter in the CNN, it learns the correspondences

between the feature maps.

5. **Bilinear fusion**: $\mathbf{y}^{\text{bil}} = f^{\text{bil}}(\mathbf{x}^a, \mathbf{x}^b)$. This fusion computes a pixel-wise matrix outer product over the feature maps, followed by a summation over the locations:

$$\mathbf{y}^{\text{bil}} = \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{x}_{i,j}^{a^T} \otimes \mathbf{x}_{i,j}^{b^T} \tag{3.16}$$

   This fusion method is able to capture the multiplicative interactions between the input feature maps at corresponding spatial locations. However, the feature is high-dimensional (i.e. D^2-dimensional).

The two temporal fusion methods considered were:

1. **3D Pooling**: This is a simple extension of regular 2D max-pooling into the temporal domain.

2. **3D Conv + Pooling**: This is convolution with a bank of filters followed by 3D max-pooling.

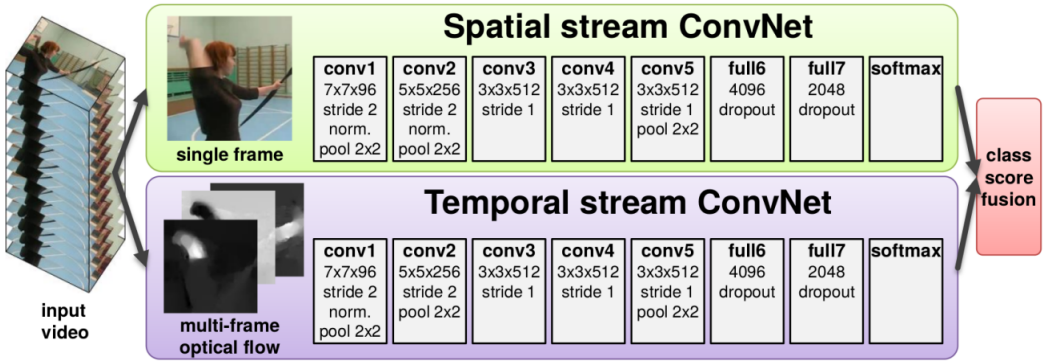


Figure 3.5: Illustration of the two-stream network architecture for action classification [14].

The architecture implemented fuses the networks at the last convolutional layer into the spatial stream CNN. This in turn creates a spatio-temporal stream by using 3D Conv fusion followed by 3D pooling [14]. ImageNet pre-training is applied to the networks separately. Predictions are made by simply averaging the predictions of the two streams. All hidden layer activations are ReLU, and max pooling is performed over a 3×3 window with a stride of 2. For the spatial CNN, 224×224 sub-images are randomly cropped from the input frames, and undergo random horizontal flipping and RGB jittering. All videos are pre-processed by resizing them such that the smaller side is of length 256. During training, the learning rate is decreased after a certain number of iterations. This two-stream approach results in an accuracy of 92.5% and 65.4% on UCF101 and HMDB51, respectively. This is impressive performance,

and shows that the two-stream approach combined with appropriate fusion is promising for action recognition. However, the authors performed additional experiments to see how performance changes when combining state-of-the-art hand-crafted iDT features with the two-stream representation using late fusion. Predictions are made by averaging SVM scores of the Fisher vector encoded iDT descriptors with the predictions of the two-stream CNN. This results in even better performance on the two datasets, with accuracies of 93.5% and 69.2%, respectively. This suggests that there is still some merit in utilising both engineering and automatically-learned features.

[65] attempted to design a network architecture that can model the full temporal extent of actions in video, as previous works could not achieve this since temporal resolutions were typically too low (on the order of a couple of frames per video). The proposed network architecture consists of five 3D convolutional layers, followed by three fully-connected layers (two with 2048 units each, and one with the number of classes). These convolutional layers contain 64, 128, 256, and 256 filters each. All convolutional layers use filters of size $3 \times 3 \times 3$, and each convolutional layer is followed by a ReLU activation and a space-time max-pooling layer. Inputs with different temporal extents (i.e. volumes with different numbers of frames) are evaluated to see the impact on performance: 16 frames (16f) and 60 frames (60f). For the 16f network, patches of dimension $112 \times 112 \times 16$ are randomly cropped from videos with a spatial resolution of 171×128 pixels. This is the baseline architecture. The spatial resolution is decreased for 60f networks in order to preserve network complexity: patches of dimension $58 \times 58 \times 60$ are randomly cropped from input videos with a spatial resolution of 89×67 pixels. It is hypothesised that, since the 60f network will have a longer temporal extent in the higher convolutional layers, more complex temporal patterns will be learned, which will affect performance positively. Data augmentation is also performed through random clipping and multi-scale cropping. Dropout is also applied with probability 90%.

Different types of inputs are also evaluated. First, raw RGB values from video frames are used. Flow fields in x and y are used for motion inputs, and are computed for the original videos, and then rescaled by a scale of the spatial subsampling to maintain correct flow values. The dependency of action recognition of the quality of the motion estimation is investigated by computing three different types of flow inputs: 1. MPEG flow; 2. Farneback; and 3. Brox. Since Brox is the most sophisticated of these motion estimation methods, it performs the best of the three. Its motion estimates are the least noisy and the most accurate. A test video is then divided into clips each of length T frames, with a temporal stride of 4 frames. 10 additional test clips are used: the four corners and centre crop, and their horizontal flips. The video score

is computed by averaging over clip and crop scores [65]. With both RGB and Flow inputs, accuracies of 91.7% and 64.8% are obtained on the UCF101 and HMDB51 datasets, respectively. However, when combined with iDT with Fisher vector encoding, performance increases to 92.7% and 67.2% on the same respective datasets. These results are very helpful for a number of reasons. They highlight how important temporal extent is when modeling videos using CNNs. They also again reinforce how useful deep models are when performing action recognition. Lastly, they show how useful hand-engineered features (iDTs encoded as Fisher vectors) can be when using them as complementary features to deep features in action recognition. The research also suggests that spatial resolution is less important than temporal resolution / extent when performing action recognition using deep networks.

A new CNN architecture for action recognition was proposed by [19] known as Two-Stream Inflated 3D CNNs, or I3D. This I3D architecture has seen much interest in recent action recognition research due to its current state-of-the-art results on some benchmark datasets. The first contribution of the work was to show how a 3D CNN can benefit from 2D ImageNet CNN designs, and possibly their learned parameters. In order to inflate a 2D CNN into a 3D CNN, the filters can simply be inflated from 2D to 3D by adding a temporal dimension to them by making them cubic (i.e. of dimension $N \times N \times N$, for some $N \in \mathbb{N}$). In order to make use of pre-trained weights from ImageNet in the 3D architecture, the learned 2D filters from the 2D CNN can be repeated N times to make a simple video sequence. Due to the linearity property, this 3D filter can then be divided by N to ensure the response from this 3D filter is the same as the 2D filter. Temporal pooling is not performed in the first two max-pooling layers, but symmetric pooling is performed in all other layers. This was shown to help performance. Lastly, a two-stream architecture, similar to [14], was proposed, in which one I3D network takes RGB volumes as inputs, and another I3D network takes optical flow volumes as inputs. The two networks are trained separately and their predictions are combined at test time. 64-frame snippets are used to train the networks. The I3D networks themselves were pre-trained on the Kinetics datasets, which is a large-scale video dataset consisting of 400 classes with over 400 videos per class. This is in addition to the ImageNet pre-training of the 2D versions of the networks, before they were 'inflated'. Training was done using stochastic gradient descent with a momentum of 0.9. 3D CNNs were trained on Kinetics using 64 GPUs, while trained on HMDB51 and UCF101 using 16 GPUs. Data augmentation was also performed by random cropping of clips with spatial dimension 224×224 from videos resized such that the shorter side has length 256 (similar to previous approaches [14]). This results in a 98.0% accuracy on UCF-101, and 80.9% on the HMDB51 dataset. This is state-of-the-art performance on these datasets, and significantly outperforms

previous results. The main reasons for this massive performance jump from previous research are: 1. the networks are larger than previous 3D CNN architectures used for action recognition; 2. good use of transfer learning was made with pre-training on such massive datasets; 3. the computing power behind the research, which for some networks was as much as 64 GPUs simultaneously. This allows bigger inputs to be fed into the network, which allows it to learn from a longer temporal sequence than previous architecture (64 frames for this research vs. the 16 frames in C3D [13]). These results suggest that extracting features using deep learning for action recognition is a promising method. However, it also suggests that to get state-of-the-art results using deep learning, massive computing power and resources are needed.

An approach that attempted to build on the above I3D framework by incorporating additional cues into the representation was introduced by [66]. The work introduces a cue in addition to appearance and motion - pose. The PoTion (pose motion) representation is created as follows. 2D pose estimation methods most commonly output human joint heatmaps, which indicates the probability of each joint at each pixel. The pose detection method used in the PoTion representation is Part Affinity Fields. This method is robust to occlusion and truncation, and has the ability to handle the presence of multiple people in the image. Part Affinity Fields extracts joints heatmaps and affinities between pairs of joints corresponding to bones. Only the heatmaps (not the pair-wise affinities) are considered for the representation.

Each frame of a video is run through the Part Affinity Fields algorithm pre-trained on the MS COCO dataset. The result for each frame is 19 heatmaps; one for each of the 18 joints considered, and one for the background. Formally, the heatmap at frame t for joint j is defined as $\mathcal{H}_j^t$. Thus, $\mathcal{H}_j^t[x, y]$ defines the probability of joint j being located at pixel (x, y) at frame t. This dimension of the heatmap is necessarily less than that of the input due to the stride of the network used in the algorithm. All heatmaps are rescaled such that the smaller dimension is 64 pixels, where H is the height of the heatmap after rescaling, and W is the width. The heatmap values are also clipped to be in the range $[0, 1]$, since the values might be slightly outside this range, and thus would not truly represent probabilities.

Next, a time-dependent heatmap colourisation technique is introduced. The heatmaps are colourised such that the colour of the current frame depends on its relative time in the video clip. That is, each heatmap $\mathcal{H}_j^t \in [0, 1]^{H \times W}$ is transformed into a C-channel image $\mathcal{C}_j^t \in \mathbb{R}^{H \times W \times C}$. These C channels can be interpreted as colour channels, where colour is defined as $\mathbf{o} \in \mathbb{R}^C$. The same colour $\mathbf{o}(t)$ is applied to all joint heatmaps at a particular frame t. This means that the colour only depends on t. The colourisation

scheme is defined for the case of $C = 2$ [66]:

$$\mathbf{o}(t) = (\frac{t-1}{T-1}, 1 - \frac{t-1}{T-1}) \tag{3.17}$$

where T is the number of frames. This means the colourised heatmap of joint j, pixel location (x, y), channel c, at time t is given by [66]:

$$\mathcal{C}_j^t[x, y, c] = \mathcal{H}_j^t[x, y]\mathbf{o}_c(t) \tag{3.18}$$

These colourised heatmaps then need to be aggregated over time, so that a fixed-size PoTion representation can be obtained. That is, the PoTion representation must be invariant to the duration of the video clip. To do this, the sum of the colourised heatmaps over time for each joint is computed [66]:

$$\mathcal{S}_j = \sum_{t=1}^{T} \mathcal{C}_j^t \tag{3.19}$$

Since $\mathcal{S}$ depends on the number of frames T, the channel c is normalised by its maximum value in order to obtain an invariant representation. This results in a C-channel image $\mathcal{U}_j[x, y, c]$ [66]:

$$\mathcal{U}_j[x, y, c] = \frac{\mathcal{S}_j[x, y, c]}{\max_{x',y'} \mathcal{S}_j[x', y', c]} \tag{3.20}$$

The colour in this image encodes the temporal evolution of keypoint positions in the video. This means that if a keypoint was stationary for some time, a stronger intensity will be accumulated at that position. This is posited to have potentially detrimental effects on performance, thus an intensity normalisation scheme is proposed. This involves first computing a single-channel intensity image that contains the sum of all channels for every pixel [66]:

$$\mathcal{I}_j[x, y] = \sum_{c=1}^{C} \mathcal{U}_j[x, y, c] \tag{3.21}$$

Then, the final normalised C-channel PoTion representation $\mathcal{N}_j[x, y, c]$ can be computed [66]:

$$\mathcal{N}_j[x, y, c] = \frac{\mathcal{U}_j[x, y, c]}{\mathcal{I}_j[x, y] + \epsilon} \tag{3.22}$$

where $\epsilon = 1$ for numerical stability. It is clear that the dimension of this PoTion representation is

invariant to the duration of the input video.

Since the PoTion representation is much less textured than normal natural images, the network needed to model the representation does not require any pre-training, nor does it have to be deep. The network architecture consists of 6 convolutional layers followed by a single fully-connected layer. The input into the network are images that consist of $19 \times (2C + 1)$ channels. This is a result of stacking $\mathcal{U}_j$, $\mathcal{I}_j$, and $\mathcal{N}_j$ for all joints. This approach results in accuracies of 43.7% and 65.2% on the HMDB51 and UCF101 datasets, respectively. However, when the PoTion representation is combined with the two I3D networks, accuracy for the two datasets goes up to 80.9% and 98.2%. This is state-of-the-art for UCF101, however the same for HMDB51. This shows that the PoTion representation provides *some* complementary information for action recognition in addition to appearance and motion.

3.3 Conclusion

This chapter presented a detailed literature review of previous research and approaches in the domain. It is clear that the main focus of most research is the feature extraction process. The aim is to extract the most reliable and robust features that can accurately discriminate between action classes on a variety of datasets. However, it is clear that it is difficult to find such a set of features. The direction of current research is to make use of deep learning to learn rich representations of the data from simple input cues such as appearance (RGB values), motion (flow values), or pose (human pose estimates). This is because such techniques have proved useful, and have resulted in a state-of-the-art performance on all common benchmark datasets. Further, the jump in performance from previous state-of-the-art techniques was so large with current deep learning approaches, and as such, the shift in direction to such approaches makes sense.

However, these deep learning approaches have a somewhat limited applicability. They require huge amounts of computational power to perform well. The inputs into the networks need to be large enough (in terms of both spatial and temporal resolution) in order to learn salient features and rich representations of the actions. Some of the state-of-the-art techniques [19] train on more than 60 large GPUs. As such, it is powerful to find ways to compute representations of actions that do not require such a large computational overhead. This could be in the form of classical computer vision techniques, or simpler inputs that leverage neural networks that do not require training on large inputs for long, such as the

PoTion representation [66].

Chapter 4

Research Methodology

4.1 Introduction

The previous chapters thoroughly discussed background concepts pertaining to the problem domain, as well as presented a literature review of the core research in the field of action recognition. This chapter serves to discuss the research aim, present a set of hypotheses to be empirically verified or rejected through the research, and present the proposed methodology implemented in the research.

4.2 Research Questions

The primary goal of the research is to find a robust, extensible approach to human action recognition that performs well based on local deep CNN features and linear SVMs.

Research Question 1: Does feature extraction using a fine-tuned, pre-trained three-dimensional convolutional neural network on a large, varietal, unrelated dataset, provide discriminative, strong local features for human action recognition?

Research Question 2: What feature aggregation technique will prove most effective for human action recognition? A simple consensus voting scheme, or linear SVMs?

4.3 Research Hypothesis

There is strong evidence to suggest that approaches to human action recognition that leverage deep neural architectures such as CNNs and RNNs result in good recognition performance. Further, many approaches adopt a separate strategy for extracting and combining spatial and temporal features from the video data. This naturally leads to the following research hypotheses:

Research Hypothesis 1: Local features from a pre-trained 3D CNN will contain rich enough information to result in good action recognition performance.

Research Hypothesis 2: Feeding the local deep features into a linear SVM will result in better performance than a consensus voting scheme.

4.4 Proposed Method

The intuition behind the proposed approach is that actions in video can naturally be decomposed into temporal and spatial components, and we can model actions by separately modelling these components of the video. This intuition has been applied in action recognition research by employing separate CNNs to operate on RGB inputs and optical flow inputs [19; 66]. We base our work on this framework. We also posit that using such networks as feature vector extractors, and then applying a model on those vectors is better than obtaining predictions directly from the networks. The full method is described below and can be seen in Figure 4.5. Our approach is also partially inspired by the findings from the work in [65], in which it was found that although both high spatial and temporal resolution are important for action recognition, temporal resolution is somewhat more important. We take cognisance of this finding, and use temporal resolutions that are as high as possible (within computational resource constraints) to model the data with the I3D networks. Further, we leave all the parameters of the I3D networks trainable during the fine-tuning phase (i.e. we do not freeze any of the weights).

4.4.1 Method

Assume we have a video $V \in \mathcal{G}^{t_v \times r_v \times c_v \times d_v}$. We first compute the optical flow video at this original resolution. An optical flow video is computed by applying a dense optical flow algorithm to every pair of frames in the video. This flow algorithm can be defined by the function $h(I_1, I_2)$, where I_1 and I_2 are

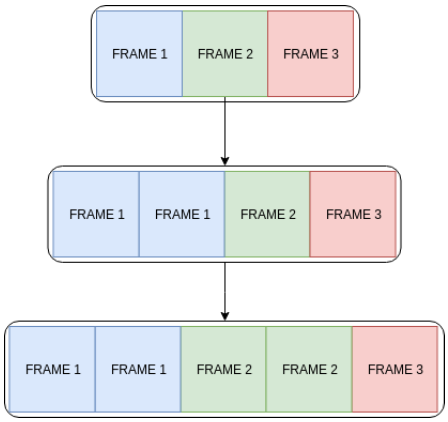


Figure 4.1: Visualisation of how the case of $t_v < T$ works for a video of length 3 frames, needing to increase its temporal resolution to 5 frames. This is to ensure that the logical flow of the action still holds when we need to increase the temporal resolution of a video. This approach is in contrast to the method of simply duplicating the last frame $T - t_v$ times to fill the deficit. We posit that this approach is better for such cases.

grayscale input frames. h then maps these two input frames to a new, two-channel 'image', where the two channels represent the optical flow vector fields for the horizontal, x, and vertical, y, components of the flow, respectively. This process yields a flow video $F \in \mathbb{R}^{t_f \times r_f \times c_f \times d_f}$.

We normalise the RGB video V by dividing by 255. Then, we limit the temporal resolution to a length of T frames. This results in three cases. If $t_v < T$, then we duplicate frame i into position $i + 1$ repeatedly until we have the required number of frames. A visualisation of the algorithm for this case can be seen in Figure 4.1. If $t_v > T$, then we sample T equally-spaced frames from V. Lastly, we sample the whole volume if $T = t_v$. We apply the same temporal sampling procedure for F. This results in volumes $V_s \in [0, 1]^{T \times r_v \times c_v \times d_v}$ and $F_s \in \mathbb{R}^{T \times r_f \times c_f \times d_f}$. Next, we resize both V and F such that the smaller side is equal to S_1, preserving aspect ratio in the process. This yields volumes V_r and F_r.

An additional rescaling process is performed on the flow video F_r, similar to the process performed in [65]. The intuition behind this rescaling is that, if the flow for a particular pixel was l at a resolution of $N \times N$, then the flow at a resized resolution of $N/2 \times N/2$ should be $l/2$ (i.e. no longer l). This rescaling proved useful for discriminating between temporally-similar actions such as *walk* and *run*. This rescaling is defined by two scaling factors s_x and s_y, representing the scaling factors for the x flow and y flow components, respectively. Formally, given original spatial resolution $r_o \times c_o$ and new resized

(a)

(b)

Figure 4.2: Consecutive input frames from the HMDB51 dataset for the action *fencing*. These are used in the illustrated flow calculation.

resolution $r_n \times c_n$, we have that:

$$s_x = \begin{cases} 1 - \frac{c_o - c_n}{c_o} & \text{if } c_n < c_o \\ 1 + \frac{c_o - c_n}{c_n} & \text{if } c_n > c_o \\ 1 & \text{if } c_n = c_o \end{cases} \tag{4.1}$$

and

$$s_y = \begin{cases} 1 - \frac{r_o - r_n}{r_o} & \text{if } r_n < r_o \\ 1 + \frac{r_o - r_n}{r_n} & \text{if } r_n > r_o \\ 1 & \text{if } r_n = r_o \end{cases} \tag{4.2}$$

The flow is then rescaled by multiplying all elements of the x vector fields by s_x, and all elements of the y vector fields by s_y.

A visualisation of this scaling process for the horizontal and vertical flow components can be seen in Figures 4.2, 4.3, and 4.4. The visualisations in these figures are such that the brighter the image, the higher the flow value. The original resolution of the video is 240×352, and the resolution after resizing is 128×188. However, we can see from both Figure 4.3b and 4.4b that the flow at the smaller, resized video has the same magnitude as the flow at the original, higher resolution. This, we posit, can negatively affect the flow representation of our method. Thus, we rescale the flow values as can be seen in Figures 4.3c and 4.4c. The values have a smaller magnitude (as seen by the relative brightness compared to the original flow images), which makes sense since at a smaller resolution, the flow should not be as large as at a higher resolution for the same input.

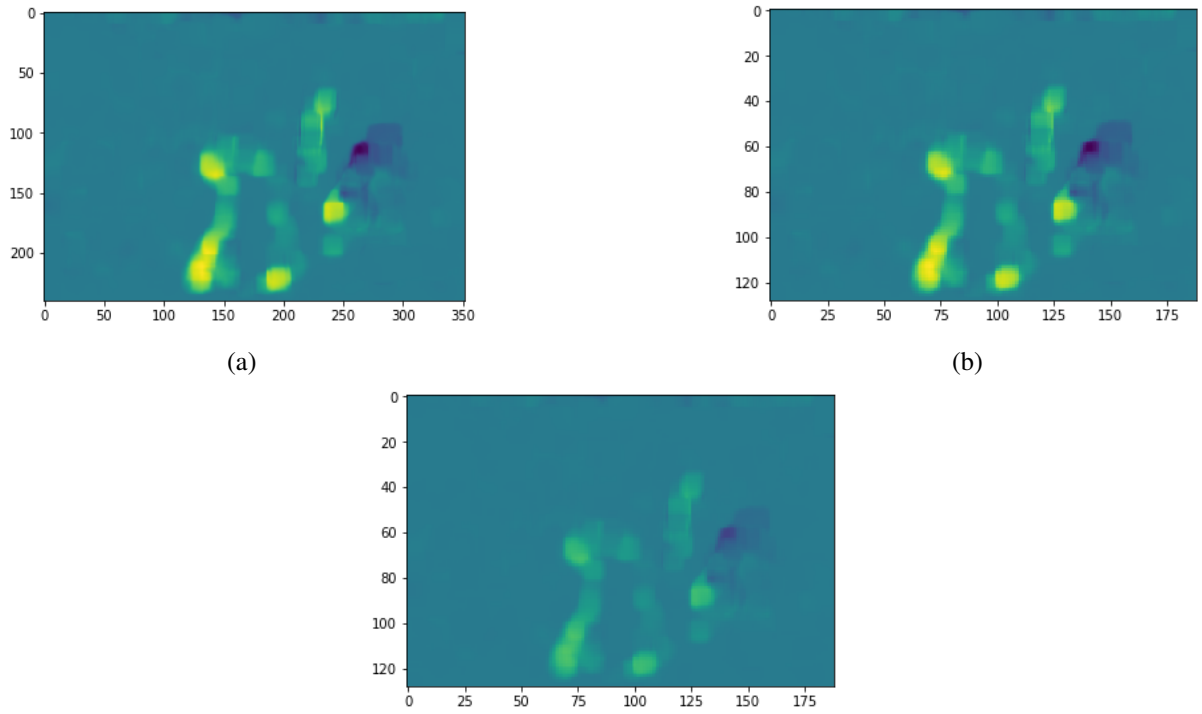


Figure 4.3: Horizontal flow estimates. (a) Flow computed at the original resolution of the video. (b) Flow at the resized resolution of the input video. (c) Flow after rescaling. The relative lower brightness (signifying flow of lower magnitude) of the rescaled flow makes intuitive sense, since the resolution is lower.

Finally, we crop 5 volumes each from both V and F, where each crop V_c from V_r and crop F_c from F_r is such that:

$$V_c \in \mathbb{R}^{T \times \frac{7}{8}S_1 \times \frac{7}{8}S_1 \times d_v} \tag{4.3}$$

and

$$F_c \in \mathbb{R}^{T \times \frac{7}{8}S_1 \times \frac{7}{8}S_1 \times d_f} \tag{4.4}$$

The 5 crops are the four corners, and the centre crop. The coefficient $7/8$ ensures that the 5 crops have good coverage of the volumes.

Next, we make use of the state-of-the-art three-dimensional convolutional neural network architecture I3D. I3D consists of an RGB network and a flow network. We use the versions of these networks pre-trained on the ImageNet and Kinetics datasets. We then fine-tune the RGB network on the V_c crops, and fine-tune the flow network on the F_c crops. Define the RGB network as the mapping:

$$f_{\text{rgb}} : \mathbb{R}^{T \times \frac{7}{8}S_1 \times \frac{7}{8}S_1 \times d_v} \mapsto [0,1]^{|\mathcal{C}|} \tag{4.5}$$

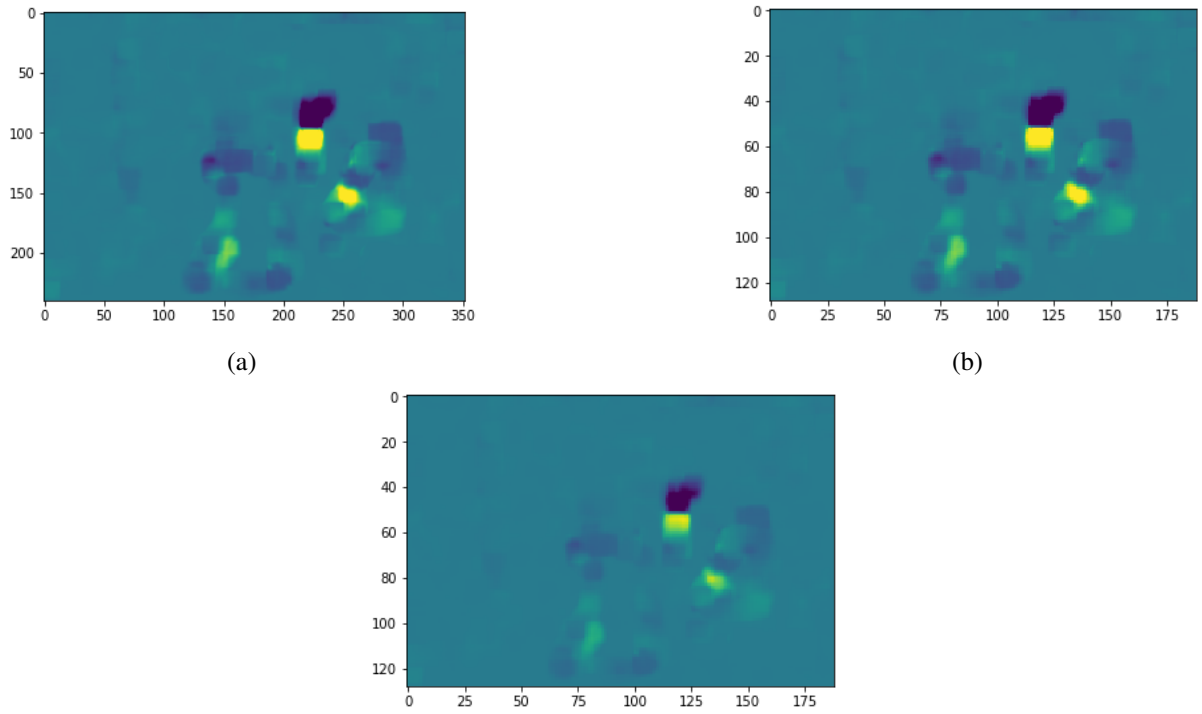


Figure 4.4: Vertical flow estimates. (a) Flow computed at the original resolution of the video. (b) Flow at the resized resolution of the input video. (c) Flow after rescaling.

and the flow network as the mapping:

$$f_{\text{flow}} : \mathbb{R}^{T \times \frac{7}{8} S_1 \times \frac{7}{8} S_1 \times d_f} \mapsto [0,1]^{|\mathcal{C}|} \tag{4.6}$$

Given these networks, we can then classify videos according to the following rule, which we use as our baseline approach. Given a test video V_{test} decomposed into its 5 RGB crops $[V_{\text{test}}^{(\text{rgb}_1)}, \ldots, V_{\text{test}}^{(\text{rgb}_5)}]$ and 5 flow crops $[V_{\text{test}}^{(\text{flow}_1)}, \ldots, V_{\text{test}}^{(\text{flow}_5)}]$, we can obtain action classes from f_{rgb} and f_{flow} by simply taking $c_{\text{rgb}_i} := \arg\max_j f_{\text{rgb}}(V_{\text{test}}^{(\text{rgb}_i)})_j$ for $j = 1, \ldots, |\mathcal{C}|$ (and similarly for the flow network), which is just the most likely class defined by the maximum probability in the final, softmax layer of the networks. We can then use the following simple rule for classifying the video into an action class c:

$$c = \text{mode}([c_{\text{rgb}_1}, \ldots, c_{\text{rgb}_5}, c_{\text{flow}_1}, \ldots, c_{\text{flow}_5}]) \tag{4.7}$$

This forms our baseline approach. However, since this rule is very naive, we also attempt the following procedure. The penultimate layer of both of these I3D networks can be used as generic 'action' features.

To this end, we sample 5 vectors for both RGB and flow corresponding to the 5 crops for each volume. Assume these vectors are D-dimensional. We then concatenate these 10 vectors (5 for RGB and 5 for flow) to form one vector $\mathbf{v} \in \mathbb{R}^{10D}$. We then power-normalise each vector $\mathbf{v}$, which is defined by:

$$\mathbf{v}_{\text{norm}} := \text{sign}(\mathbf{v})|\mathbf{v}|^{\alpha} \tag{4.8}$$

We set $\alpha = 0.5$ in our normalisation. We use power normalisation in place of regular L_2-normalisation since it has shown to give better performance [18]. Since D may be in the order of tens of thousands of dimensions, we first reduce the dimensionality of these vectors using PCA (PCA is applied by default unless stated otherwise). This reduced-dimension vector is used as the final 'action' representation. A linear SVM is then trained on these vectors, and learns to classify actions from this representation. We posit that this can achieve better performance than making predictions directly from the network for three reasons. Firstly, the majority voting scheme from above is simple. Secondly, the layers of the network up until before the final layer can be seen to learn a rich representation of the input that the final layer can classify using a softmax classifier. We posit that an SVM is more effective for classifying from these representations than the softmax classifier. Lastly, the SVM will have the information about all RGB and flow crops from the $\mathbf{v}$ representation, whereas the network makes predictions on a crop-by-crop basis, and thus does not have the information from other crops to aid its prediction.

Since we have a fixed number of crops per video - 5 - there is no need to encode these extracted feature vectors as Fisher vectors, or any other feature quantisation technique (e.g. BoW or VLAD), since the dimensionality remains constant at $10D$ for any video. Such quantisation techniques are typically applied in this domain to create a fixed-length, global representation of an image or video from a variable number of local features per image / video.

For a test video, the following process is followed. We perform optical flow for the video. We then perform the requisite pre-processing (resizing, normalisation, and scaling). We then extract the 5 crops from both the RGB video and the flow video. These crops are sent through their relevant networks, and baseline predictions are made by taking the majority vote of the softmax classifications. SVM predictions are made by extracting the penultimate layers' features, concatenating these 10 resultant vectors, globally power-normalising this vector, and obtaining an action prediction from the linear SVM.

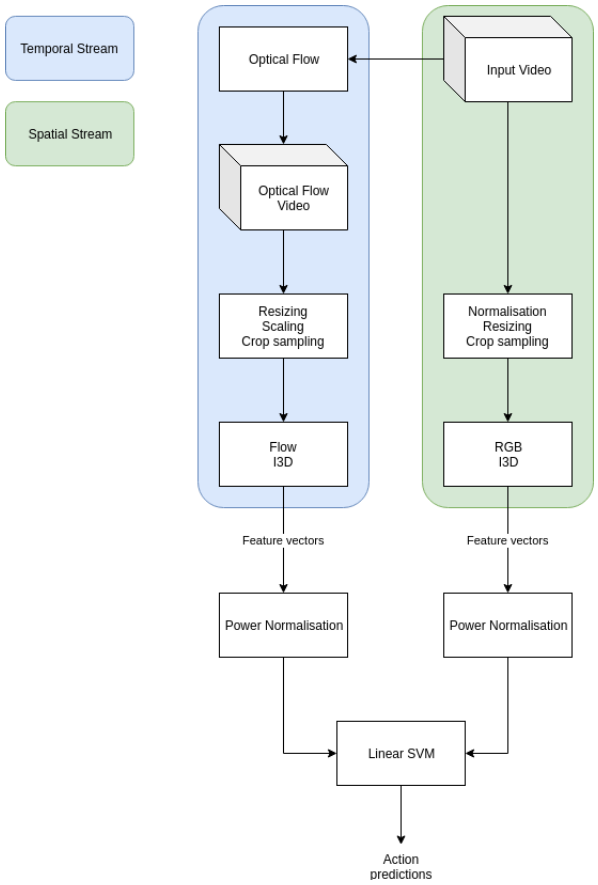


Figure 4.5: Process flow of the proposed method.

4.4.2 Hyperparameter Optimisation

For the linear SVMs, the only parameter we optimise for is the penalty parameter C. We use a grid search approach, and choose the C that maximises five-fold cross-validation accuracy. The parameters for the I3D networks, such as number of layers and number of filters / neurons per layer, are left as default as per the recommendation of the authors in [19]. The weights for the network are set to that setting which minimises the cross-entropy loss on a validation dataset.

4.5 Action Recognition Datasets

In this research, we study and apply our proposed technique to the KTH and HMDB51 datasets. This provides a good framework to test our method on a simple dataset, KTH (as a sort of regression test), and a more difficult, complex dataset, HMDB51.

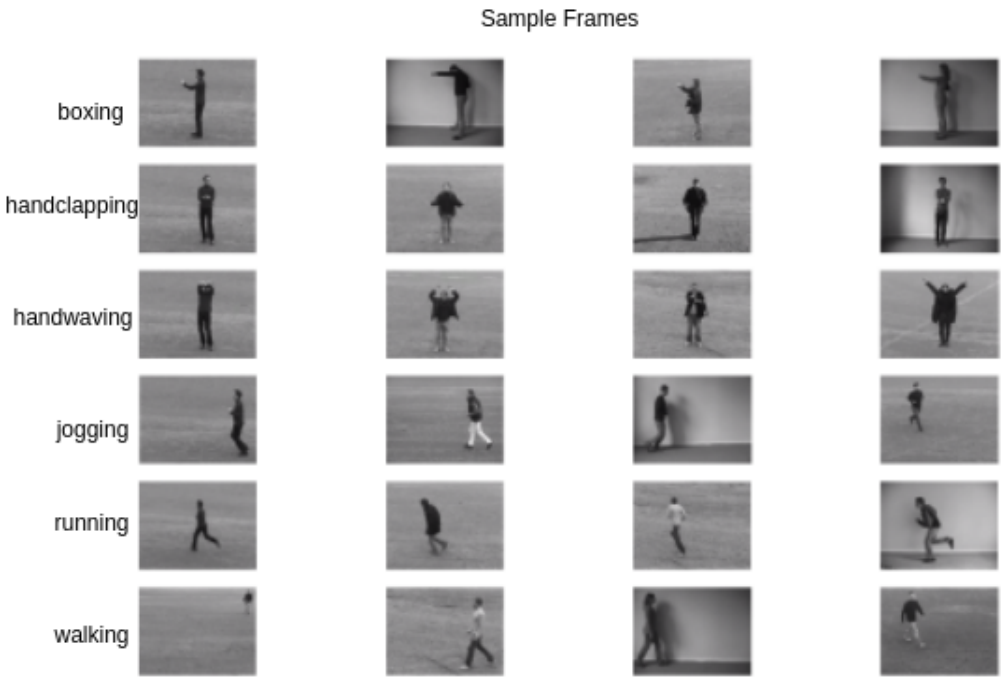


Figure 4.6: Sample frames and actions from the KTH dataset.

4.5.1 KTH

The KTH dataset consists of fairly-static backgrounds, with actions performed 25 different actors. It consists of 599 videos of 25 people repeatedly performing one of the six actions. Altogether, of which there are 2391 subsequences of actions being performed. There are four different scenarios spread across the videos, namely *s1*: outdoors; *s2*: outdoors with scale variation; *s3*: outdoors with different clothes; and *s4*: indoors. Further, the dataset consists of the following six action classes: Walking; Jogging; Running; Boxing; Handclapping; and Handwaving. Example frames from the dataset are shown in Figure 4.6. Presented below are some basic statistics regarding the number of frames in the 2391 snippets of the KTH dataset. We take these statistics into consideration during the modelling process:

1. Mean: 94.1 frames
2. Max: 361 frames
3. Min: 23 frames

4.5.2 HMDB51

HMDB51 [3] is a large human action recognition dataset, often used to thoroughly test a particular approach to action recognition, as it is widely considered one of the more difficult benchmark datasets in this domain. It consists of videos from a variety of sources such as films, public databases, YouTube, and Google videos. There are 6676 clips divided into 51 different categories/classes. Each class contains a minimum of 101 videos. Example frames from the dataset are shown in Figure 4.7. Rather than enumerate all 51 action classes, below is a description of the 5 general classes that the 51 categories are grouped into [3]:

1. General facial actions
2. Facial actions with body manipulations
3. General body movements
4. Body movements with object interactions
5. Body movements for human interaction

Presented below are some basic statistics regarding the number of frames in the 6766 videos of the HMDB51 dataset. We take these statistics into consideration during the modelling process:

1. Mean: 93.4 frames
2. Max: 1062 frames
3. Min: 18 frames

4.5.3 Performance Evaluation

For both datasets, we use the average accuracy over the classes as the main metric for evaluating performance. This is the standard evaluation metric from previous literature. It is formally defined as [67]:

$$\text{avg\_acc} = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \frac{\text{TP}_c + \text{TN}_c}{\text{TP}_c + \text{TN}_c + \text{FP}_c + \text{FN}_c} \tag{4.9}$$

where TP_c is the number of true positives for action class c; TN_c is the number of true negatives for action class c; FP_c is the number of false negatives for action class c; and FN_c is the number of false

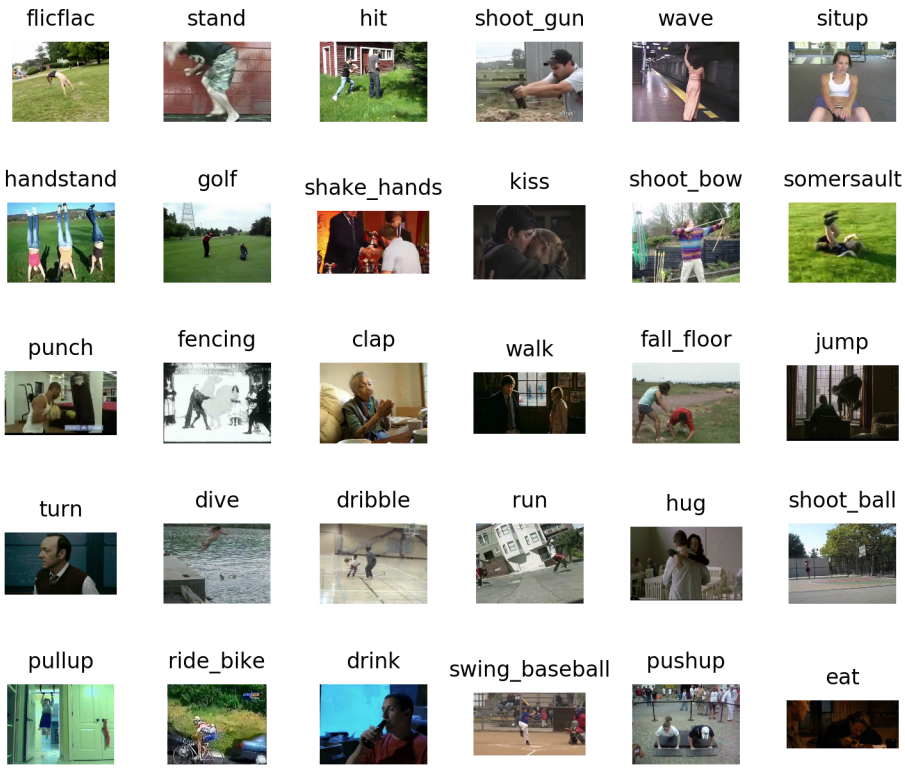


Figure 4.7: Sample frames and actions from the HMDB51 dataset [3].

negatives for action class c. We also provide measures of performance in terms of precision, recall, and F1-score, which are formally defined as [68]:

$$\begin{aligned} \text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{F1-score} &= \frac{2 * \text{recall} * \text{precision}}{\text{recall} + \text{precision}} \end{aligned} \tag{4.10}$$

For the KTH dataset, the data is split according to the standard split of 8 people for training, 8 for validation, and 9 for testing. We then average the results over a pre-defined number of trials of this train/test split framework. Performance is then measured by average accuracy over these splits. For the HMDB51 dataset, there are three pre-defined train/test splits that come with the dataset, as defined by the introducing authors. Performance is then measured by average accuracy over these three splits. We also provide performance measures in the form of classification performance tables (showing class-by-class precision, recall, and F1-score), and confusion matrices.

4.6 Conclusion

In this section, the proposed method was introduced. The novelty in the method comes in treating the two-stream network architecture as a local feature extraction paradigm for another class of model to then learn to map to action classes. The intuition is that the networks can learn rich spatial and flow 'action' representations that another class of model can leverage more effectively than the de-facto softmax classifier at the end of the I3D networks. Further, the two simple, intuitive pre-processing steps - crop filling and optical flow scaling - should prove to be useful in making the inputs into the networks easier to learn rich representations from, thereby resulting in better recognition performance.

In this chapter, the datasets that the proposed method is evaluated and tested on are also introduced. We also introduce how we measure the accuracy of our system, by following the standard evaluation frameworks for both datasets. It should be noted that researchers in this domain often test the same datasets (e.g. KTH) in different ways, and depending on this experimental setup, results can skew significantly. Due to the vast computational resources that the state-of-the-art methods use for HMDB51, we are unable to test the method to the degree we would want, since we are limited in the spatial and temporal resolution we can employ for the input videos. The networks our approach are based on, I3D, were trained with up to 64 large GPUs by the authors in the original work [19]. Further, limited time was also an issue in being able to train larger, more competitive networks.

Chapter 5

Experimental Results

In the previous section, the proposed method was discussed in detail, along with the datasets that the method will be benchmarked on. We evaluate the datasets in the same way as the majority of other researchers (in terms of train/test splits and performance metrics), as described in the previous chapter. Furthermore, we make comparisons between each method and previous results, and extensively test the proposed method. We also analyse the performance of the spatial and temporal streams of the system for the KTH dataset, and analyse the benefits of performing the optical flow scaling and crop filling procedures on both datasets. Lastly, we attempt to quantify the performance of the proposed methods, and draw any possible conclusions from there.

5.1 Experimental Setup

Technologies used to run the experiments are Python with various machine learning, computer vision, and deep learning libraries. Experiments were run on a 32-core machine with 128GB of RAM and a RTX 2080Ti GPU, as well as a separate 8-core machine, with 32GB of RAM, and a GTX 1080 GPU.

5.2 Results on KTH Dataset

To test on the KTH dataset we set parameter $S_1 = 128$ and parameter $L = 40$. Such a setting, we posit, should be able to adequately model the comparatively-simple 2391 snippets of the dataset. We split the dataset in the recommended manner of 8 people for training, 8 for validation, and 9 for testing.

We fine-tune the I3D networks for a certain number of epochs, and use the model that minimises the cross-entropy loss on the validation set.

Due to the KTH dataset's relative simplicity, we would expect performance to be fairly high. We can see a comparison of our proposed approaches against previous methods in Table 5.1. It is clear that our method is competitive with state-of-the-art approaches, achieving a highest accuracy of 96.6%, which is second only to the state-of-the-art approach by [55] with 98.2%. Even with the baseline majority voting approach, performance is still competitive. We can most likely attribute this performance to the KTH's simplicity. The backgrounds are mostly static, and homogeneous. There is also not much clutter and occlusion to contend with. The I3D networks are, therefore, able to model the actions from such videos well. The ActionBank method proposed by [55] performs better most likely due to the amount of domain knowledge incorporated into the hand-crafted features, however, it should be noted that this method does not generalise well to complex settings / datasets such as HMDB51. Our method is, instead, simply tasked with learning the actions directly from RGB pixel values and optical flow vector fields, and generalises better than ActionBank to such settings. The baseline approach is less powerful simply because it makes predictions on a crop-by-crop basis and performs a majority vote. There are no steps taken to use information from the other crops (e.g. in the form of pooling or the SVM approach) to make the predictions. However, it is still relatively high, since much of the context about an action get be determined from a single video crop (as a result of the simple background, minimal noise, and limited clutter / occlusion).

Our approach performs better than the other methods because our method is structured in such a way that it makes no stringent assumptions about the actions. Instead, we feed raw RGB and optical flow inputs and task the networks with learning everything. The majority of the previous approaches hand-craft features that may work well on some simpler datasets, but struggle to generalise to more complex settings. Our method has no such limitation, and as such has a higher accuracy. Some of the previous works do compete with our method, such as [11] and [58]. The dense trajectory representation [11] is strong since it extracts both zero-order (HOF) and first-order (MBH) motion information, as well as trajectory and appearance (HOG) information. It then encodes and quantises these descriptors using the state-of-the-art Fisher vector representation. This means the representation is powerful enough to perform well on the KTH dataset since it takes many different cues into account to represent an action (and similarly for the hierarchical representation employed in [58], where the features are able to encode

enough pertinent information about the actions to perform well).

Table 5.1: Results on the KTH dataset.

Method	**Accuracy**
N-Jets + BoW [17]	71.7
STIP + HOG/HOF [54]	91.8
Dense Sampling + HOG/HOF [29]	92.1
Action Bank [55]	98.2
Action MACH [56]	88.66
Hough Forest [57]	92.0
3D HOG [16]	91.4
Space-Time Hierarchy [58]	95.53
ISA [43]	93.9
Dense Trajectories [11]	94.53
Ours - baseline	**94.5**
Ours - SVM	**96.6**

5.2.1 Baseline Approach

From the following tables, we can see that the baseline approach performs best when both the RGB and flow components are combined. This is an expected result, since there is more information about the actions available when performing the consensus vote. We can also see that the flow component by itself results in a higher accuracy on the dataset than the RGB component by itself. We can attribute this to the fact that in a simple dataset like KTH, where the background is simple and consistent across samples, motion information (as represented by our flow component) provides more useful cues for determining the actions than appearance information (as represented by our RGB component). We can see from the tables, particularly Figures 5.2 and 5.3, that the flow component is a marginally more useful cue in distinguishing between temporally-similar actions such as *boxing* and *clapping* or *jogging* and *running*. Spatial information alone is not rich enough to discriminate between such actions as they are too spatially similar and thus RGB information does not help as much as motion information. Please note that all reported results from here onwards are for the same *particular* train/test split of the dataset, whereas Table 5.1 reports values averaged over a number of train/test splits (as per standard performance evaluation for the dataset).

It is also clear from the baseline results that, for this dataset, the flow component by itself is noticeably better at recognising the three most similar actions of the dataset - *walking*, *jogging*, and *running* - than the RGB component by itself. The optical flow vector fields provide more useful information than RGB

Table 5.2: Classification performance for the KTH dataset using the baseline method with the flow and RGB components combined.

Action Class	Precision	Recall	F1-Score
boxing	0.9862	1.0000	0.9931
clapping	0.9660	0.9861	0.9759
jogging	0.8581	0.9236	0.8896
running	0.9457	0.8472	0.8938
walking	0.9595	0.9861	0.9726
waving	0.9928	0.9583	0.9753
Avg/Total	**0.9513**	**0.9502**	**0.9500**

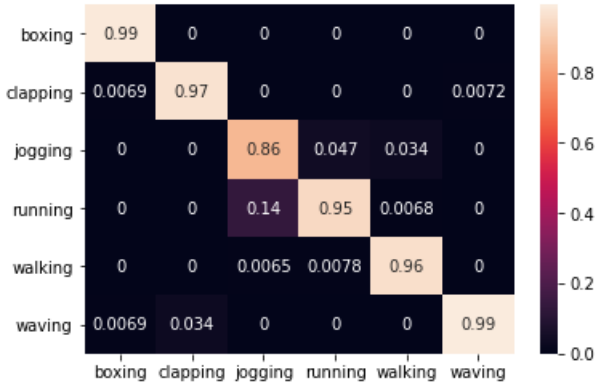


Figure 5.1: Confusion matrix for the baseline method with both the RGB and flow components combined on the KTH dataset.

Table 5.3: Classification performance for the KTH dataset using the baseline method with only the RGB component.

Action Class	Precision	Recall	F1-Score
boxing	0.9728	1.0000	0.9862
clapping	1.0000	0.9722	0.9859
jogging	0.8446	0.8681	0.8562
running	0.8601	0.8542	0.8571
walking	0.9858	0.9653	0.9754
waving	0.9792	0.9792	0.9792
Avg/Total	**0.9404**	**0.9397**	**0.9400**

does at discriminating between such similar actions. However, it should be noted that for this simple dataset, the difference in recognition accuracy between the RGB and flow components is quite small, since the accuracies are already so high.

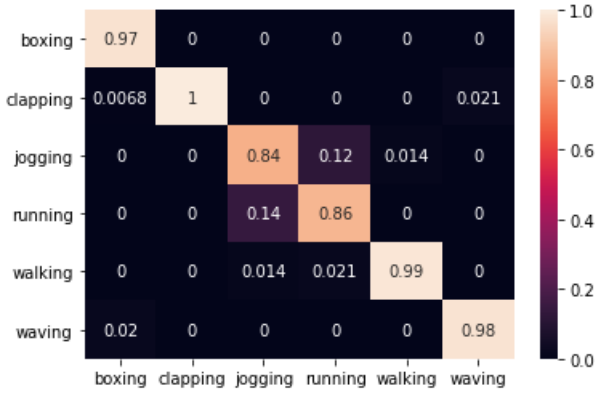


Figure 5.2: Confusion matrix using the baseline method using only the RGB component on the KTH dataset.

Table 5.4: Classification performance for the KTH dataset using the baseline method with only the flow component.

Action Class	Precision	Recall	F1-Score
boxing	0.9862	1.0000	0.9931
clapping	0.9660	0.9861	0.9759
jogging	0.8794	0.8611	0.8702
running	0.9333	0.8750	0.9032
walking	0.9226	0.9931	0.9565
waving	0.9929	0.9653	0.9789
Avg/Total	**0.9467**	**0.9467**	**0.9462**

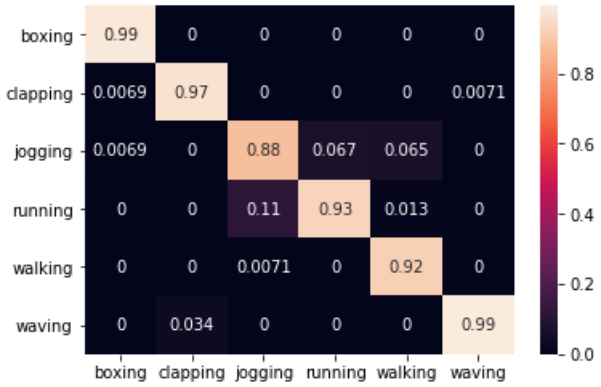


Figure 5.3: Confusion matrix using the baseline method with only the flow component on the KTH dataset.

5.2.2 SVM Approach

We see a similar expected trend with the SVM approach to the baseline approach in that combining the RGB and flow components of the system results in higher performance than either one separately. This reinforces the suggestion that both flow and RGB components combined are useful cues in the

Table 5.5: Classification performance on the KTH dataset using the SVM method with both the RGB and flow components combined.

Action Class	Precision	Recall	F1-Score
boxing	1.0000	1.0000	1.0000
clapping	0.9728	0.9931	0.9828
jogging	0.8343	0.9792	0.9010
running	0.9752	0.8194	0.8906
walking	1.0000	0.9861	0.9930
waving	0.9929	0.9722	0.9825
Avg/Total	**0.9625**	**0.9583**	**0.9583**

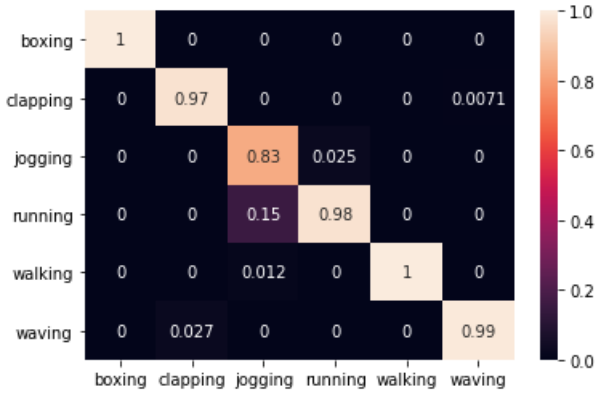


Figure 5.4: Confusion matrix using the SVM method with both RGB and flow components combined on the KTH dataset.

recognition of human actions, and that actions can indeed be naturally broken down into spatial and temporal components. The recognition accuracy of the SVM method, 95.8%, is higher than the baseline of 95.0%. This suggests that there is a benefit to using the SVM method, and by extension, incorporating the information from all video crops into a model to make action predictions, though the benefit is more clearly evident in the more complex HMDB51 dataset.

We also notice that for all three configurations - 1. RGB + flow; 2. flow; and 3. RGB - that the SVM approach performed better than the baseline equivalent. This suggests that treating the networks as feature extractors, and feeding these local, crop-based features into another model for classification is a beneficial approach. Similarly to the baseline approach, flow alone performed better than RGB alone, which again suggests that temporal information is a more useful cue than spatial information for action recognition on the KTH dataset.

We ran an additional experiment to investigate the effect of PCA on performance. Surprisingly, reducing the dimension from the original 20400 dimensions to approximately 1500 dimensions resulted in a marginally higher accuracy (i.e. the system's performance is slightly worse *without* PCA). This is likely

Table 5.6: Classification performance on the KTH dataset using the SVM method with only the RGB component.

Action Class	Precision	Recall	F1-Score
boxing	0.9862	1.0000	0.9931
clapping	1.0000	0.9722	0.9859
jogging	0.7941	0.9375	0.8599
running	0.9286	0.8125	0.8667
walking	1.0000	0.9444	0.9714
waving	0.9795	0.9931	0.9862
Avg/Total	**0.9480**	**0.9432**	**0.9438**

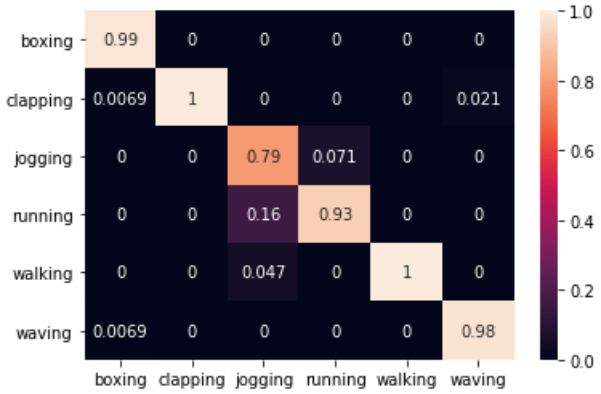


Figure 5.5: Confusion matrix using the SVM method with only the RGB component on the KTH dataset.

Table 5.7: Classification performance on the KTH dataset using the SVM method with only the flow component.

Action Class	Precision	Recall	F1-Score
boxing	0.9931	1.0000	0.9965
clapping	0.9724	0.9792	0.9758
jogging	0.8447	0.9444	0.8918
running	0.9597	0.8264	0.8881
walking	0.9728	0.9931	0.9828
waving	0.9859	0.9722	0.9790
Avg/Total	**0.9547**	**0.9525**	**0.9523**

due to the curse of dimensionality, although the difference is so negligable, that we are unable to draw any real conclusions. These very large dimensions also discount the use of non-linear kernels in the SVM, as they typically do not perform well in such contexts. Moreover, linear SVMs have been shown to perform well in a high-dimensional action recognition context [11; 18].

As part of our method, we perform what we term as ‘crop filling’, which is appending frames to a crop of a video if the number of frames is less than the desired number for the input into the networks. This is typically done by repeatedly appending the last frame the required number of times until the deficit is

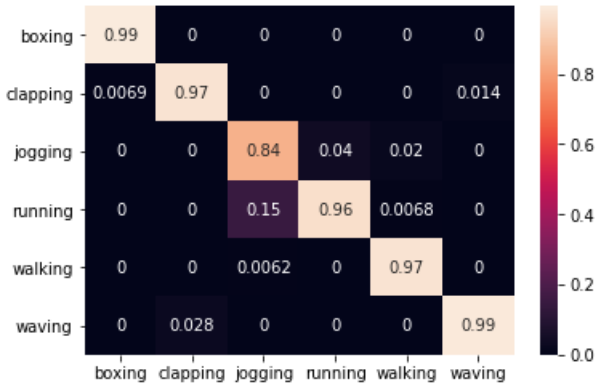


Figure 5.6: Confusion matrix using the SVM method with only the flow component on the KTH dataset.

Table 5.8: Results of the proposed method with and without the flow scaling and crop filling procedures on the KTH dataset.

	Baseline	**SVM**
No Flow Scaling + Crop Filling	91.5	95.7
Flow Scaling + Crop Filling	**95.0**	**95.8**

filled. We instead propose a scheme as described in the previous chapter, and can be seen in Figure 4.1. Additionally, we also perform optical flow scaling as opposed to leaving the flow values untouched as often done in previous research. We investigate leaving out these two steps - that is, appending the last frame and performing no flow scaling. This resulted in a baseline accuracy of 91.5%, which is noticeably lower than the baseline accuracy compared with when the two proposed methods are employed - 95.0%. This suggests that performing these two steps aids performance. Interestingly, the performance of the SVM method drops only very slightly to 95.7%. Since it is difficult to tell how much these two techniques help on the KTH dataset, we see that they are indeed beneficial in more complex settings, such as on the HMDB51 dataset. We can see a tabulation of these aforementioned results in Table 5.8.

5.3 Results on HMDB51 dataset

To test on the HMDB51 dataset, we set parameter $S_1 = 256$ and $L = 40$. This setting is partially inspired by the findings by [65], in which it is shown that temporal resolution is somewhat more important in action recognition than spatial resolution. Thus, we attempt to make the temporal resolution as long as possible, within the constraints of computational resource (and time) limitations (40 being the limit in the circumstances). A large temporal resolution is needed for complex, varietal datasets such as HMDB51 since the videos are fairly long, and, more importantly, the majority of the full temporal evolution of the

action needs to be modelled to ensure enough of the variation in the data is being captured. The latter is because the dataset has some actions that are *very* temporally (and spatially) similar - *talk* vs *laugh*, *draw sword* vs *sword exercise*, and *kick* vs *kick ball*. This is in tandem with the other existing difficulties related to the dataset, such as very complicated, non-static backgrounds, clutter, and occlusion. Performance for this dataset is represented as accuracy. We fine-tune the I3D networks for a certain number of epochs, and use the model that minimises the cross-entropy loss on the validation set, where the validation set is given by 20% of the training data.

Results on the HMDB51 dataset can be seen in Table 5.9. It is clear that the method is not competitive with state-of-the-art approaches [19] and [66]. We attribute this to computational resource limitations, as we were not able to train for temporal (and spatial) resolutions as large as those approaches used (i.e. $>= 64$ frames). This makes a large difference since this is a 60% increase in temporal resolution that the networks can learn from, which is vital for complex datasets such as HMDB51 (and less so for a dataset such as KTH). The authors from these approaches had more than 60 GPUs to train their networks, whereas we had only 1 available. Time available for the research was also a prohibitive factor. Further, these state-of-the-art approaches employ the more accurate TVL-1 optical flow algorithm to compute the optical flow videos, whereas we instead opt for Farneback's optical flow algorithm. We do this since the TVL-1 algorithm is very computationally intensive and prohibits such systems' real-time capabilities (even though the resulting flow estimates are more accurate). However, it should be noted that the goal of this work is not to necessarily to achieve state-of-the-art performance on these datasets, but rather to thoroughly investigate using RGB (spatial) and flow (temporal) information to model actions in video, and introduce potential improvements to the baseline, such as optical flow scaling and using another class of model on the extracted features of the networks (in our case, a linear SVM). These improvements may prove useful in competing with state-of-the-art when these networks can be trained on larger batches of crops with higher temporal and spatial resolutions. This, however, is left as future work.

Given the circumstance of computational resource limitations, the method still performs well given the limited temporal resolution we can use for such a complicated dataset. The method outperforms [55] (i.e. the state-of-the-art method on KTH), suggesting that our method can generalise to more complicated datasets better than [55], while still performing very well on these simpler datasets. We attribute this to the fact that our method is more general since it keeps hard-coding of features to a minimum, and instead tasks the I3D networks with learning salient representations of actions directly from raw pixel and flow

Table 5.9: Results on the HMDB51 dataset.

Method	**Accuracy**
Action Bank [55]	26.9
iDT + FV [18]	57.2
MPEG Flow [59]	46.7
iDT + Stack Fisher vectors [12]	66.7
Two-Stream CNN [14]	59.4
Two-Stream CNN + Fusion [64]	65.4
Long-Term Temporal Convolution [65]	64.8
I3D [19]	80.9
PoTion + I3D [66]	80.9
Ours - baseline	**50.7**
Ours - SVM	**62.8**

values. This is useful for complicated, varietal datasets where it is difficult to hand-craft reliable, general features (whereas the method in [55] outperforms our method on the simple KTH dataset).

Our method outperforms the solely hand-crafted approaches such as [18] and [59], since these video representations are not rich enough to generalise as well as our method for similar reasons to [55]. They are inherently limited in the amount of salient information they can capture and represent in complex settings. However, the stacked Fisher vector representation [12] performs slightly better than ours. We posit that this is due to the fact that the method employs a hierarchical representation, similar to that of a deep neural network, and uses an already-strong base to do so: iDTs [18] encoded as Fisher vectors. It is interesting to note that our method performs comparatively with both [64] and [65]. This is noteworthy since the work in [65] in particular use large temporal resolutions of the videos (up to 100 frames), but with a very low spatial resolution. Our method performs similarly due to the transfer learning employed in our method by pre-training on both the ImageNet and Kinetics datasets before fine-tuning on HMDB51. This pre-training makes up for the lack of temporal resolution in this case. However, the state-of-the-art methods employ a large temporal (64 frames and above) and spatial (224×224 and larger) resolution, as well as the pre-training on the same datasets. They also train the networks on larger batches, for longer. This is the main reason for the disparity between methods such as ours, [65], and [64], compared with state-of-the-art methods. Lastly, state-of-the-art methods also employ data augmentation by randomly flipping, cropping, and applying jitter to the input videos. This allows the networks to learn from more data, and also increase generalisation performance. We were not able to employ this data augmentation step due to computational resource and time limitations. Please note that all reported results from here onwards are (with loss of generality with respect to the analyses and subsequent conclusions) for split

1 of the dataset, whereas Table 5.9 reports the average accuracy over the three train/test splits for the dataset.

5.3.1 Baseline Approach

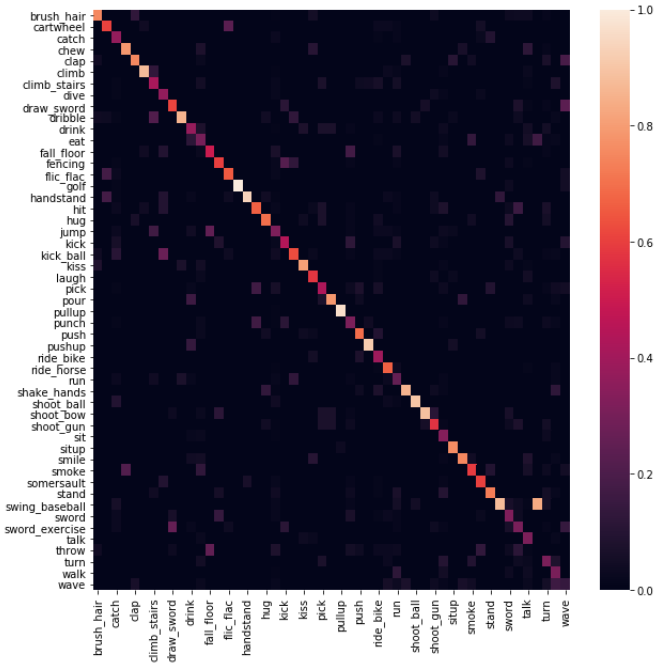


Figure 5.7: Confusion matrix using the baseline approach with both the RGB and flow components on the HMDB51 dataset (split 1).

5.3.2 SVM Approach

In the KTH dataset, the benefits of the SVM approach over the baseline approach were marginal due to the simplistic nature of the KTH dataset, and the already-high accuracies. The benefits are much more evident for the HMDB51 dataset. The average accuracy jumps from a baseline of 50.6% to 64.1%. This is a 26.6% relative increase in performance. We posit that a combination of the fact that, 1. information about all video crops being available to the SVM to learn from; and 2. the power normalisation, are the main contributing factors to this performance increase. The baseline for KTH was already so strong that the benefit of the SVM was not clear. The jump arises from the fact that the SVM has the full context of the action from all 10 crops available to it to learn from (due to the concatenation of all 10 crops' feature vectors), whereas the network has to make predictions from a single crop at a time. The representation of the actions has many feature dimensions, and linear SVMs have been shown to work

Table 5.10: Classification performance on the HMDB51 dataset (split 1) with the baseline approach with both the RGB and flow components.

Action Class	Precision	Recall	F1-Score
brush_hair	0.7500	0.6000	0.6667
cartwheel	0.6071	0.5667	0.5862
catch	0.3692	0.7742	0.5000
chew	0.7857	0.3667	0.5000
clap	0.7500	0.4286	0.5455
climb	0.8750	0.7000	0.7778
climb_stairs	0.4167	0.3333	0.3704
dive	0.3636	0.4000	0.3810
draw_sword	0.6129	0.6333	0.6230
dribble	0.8571	0.4000	0.5455
drink	0.3721	0.4706	0.4156
eat	0.3000	0.3871	0.3380
fall_floor	0.5000	0.0667	0.1176
fencing	0.6000	0.6774	0.6364
flic_flac	0.6538	0.5667	0.6071
golf	1.0000	0.9000	0.9474
handstand	0.9375	0.5000	0.6522
hit	0.6667	0.1379	0.2286
hug	0.6957	0.5333	0.6038
jump	0.3168	0.6531	0.4267
kick	0.4444	0.1333	0.2051
kick_ball	0.6250	0.1667	0.2632
kiss	0.8077	0.7000	0.7500
laugh	0.5789	0.7333	0.6471
pick	0.4286	0.2000	0.2727
pour	0.7857	0.4231	0.5500
pullup	0.9667	0.9667	0.9667
punch	0.3103	0.6000	0.4091
push	0.6923	0.6000	0.6429
pushup	0.9167	0.7333	0.8148
ride_bike	0.3939	0.8667	0.5417
ride_horse	0.6667	0.8000	0.7273
run	0.2542	0.5556	0.3488
shake_hands	0.8571	0.4000	0.5455
shoot_ball	0.9000	0.6000	0.7200
shoot_bow	0.8889	0.5333	0.6667
shoot_gun	0.5714	0.5333	0.5517
sit	0.3333	0.8077	0.4719
situp	0.7632	0.9667	0.8529
smile	0.7500	0.4000	0.5217
smoke	0.5909	0.4333	0.5000
somersault	0.6053	0.7667	0.6765
stand	0.7273	0.2500	0.3721
swing_baseball	0.8750	0.2333	0.3684
sword	0.3226	0.3333	0.3279
sword_exercise	0.3077	0.2667	0.2857
talk	0.3133	0.8667	0.4602
throw	0.0000	0.0000	0.0000
turn	0.3103	0.4865	0.3789
walk	0.3061	0.5000	0.3797
wave	0.1429	0.1034	0.1200
Avg/Total	**0.5840**	**0.5137**	**0.5063**

well in such circumstances [18; 11; 58; 12]. The jump in performance is so large for the HMDB51 dataset compared to the KTH dataset since contexts from the different crops is much more important in the more complicated videos in HMDB51. A large amount of context for an action can be determined

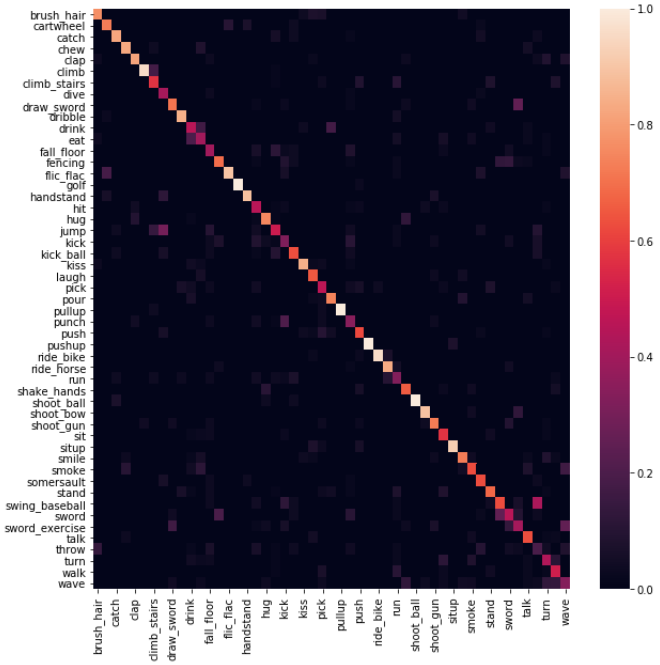


Figure 5.8: Confusion matrix using the SVM approach with both the RGB and flow components on the HMDB51 dataset (split 1).

from a single crop in the KTH dataset, whereas in the HMDB51 dataset, a single crop does not provide enough valuable information about the action. This is due to the similarities between actions and the complex nature of the dataset.

It is clear from Tables 5.10 and 5.11 that our method does well with certain classes such as *golf*, *pushup*, and *climb*. However, certain actions, particularly ones related to motion with the hands and arms, prove difficult for the algorithm to classify. Examples of such actions are *eat*, *punch*, *throw*, and *wave*. These actions, particularly the latter two, are very similar both in appearance and motion. This confusion by the algorithm is thus, to a certain degree, understandable. Overall, the network was not able to perform as we would like due to the time and computational limitations discussed previously. The significantly smaller temporal resolution means that the network did not see enough of the variation in the data, and enough of the temporal evolution of the actions, to be competitive with state-of-the-art approaches.

We ran an additional experiment to investigate the effect of not applying PCA to the features before normalising and feeding them into the SVM. The results without performing PCA are negligible. The dimension is reduced from 20400 to 3500. This suggests that there is a low-dimensional basis governing these local CNN features, as we can reduce the number of dimensions by over 80% and see no difference in performance. Also, we can conclude that, for the HMDB51 dataset, applying PCA is a better idea than

Table 5.11: Classification performance on the HMDB51 dataset (split 1) with the SVM approach with both the RGB and flow components.

Action Class	Precision	Recall	F1-Score
brush_hair	0.7667	0.7667	0.7667
cartwheel	0.7273	0.8000	0.7619
catch	0.8214	0.7419	0.7797
chew	0.8276	0.8000	0.8136
clap	0.8182	0.6429	0.7200
climb	0.9600	0.8000	0.8727
climb_stairs	0.5714	0.5333	0.5517
dive	0.4118	0.7000	0.5185
draw_sword	0.7083	0.5667	0.6296
dribble	0.8438	0.9000	0.8710
drink	0.4565	0.6176	0.5250
eat	0.4000	0.4516	0.4242
fall_floor	0.4000	0.4000	0.4000
fencing	0.6923	0.5806	0.6316
flic_flac	0.8947	0.5667	0.6939
golf	1.0000	0.9667	0.9831
handstand	0.8889	0.8000	0.8421
hit	0.4565	0.7241	0.5600
hug	0.7500	0.7000	0.7241
jump	0.5000	0.5306	0.5149
kick	0.3235	0.3667	0.3438
kick_ball	0.6296	0.5667	0.5965
kiss	0.8462	0.7333	0.7857
laugh	0.6500	0.8667	0.7429
pick	0.4667	0.4667	0.4667
pour	0.7391	0.6538	0.6939
pullup	1.0000	0.9333	0.9655
punch	0.3404	0.5333	0.4156
push	0.6176	0.7000	0.6562
pushup	1.0000	0.9333	0.9655
ride_bike	0.9630	0.8667	0.9123
ride_horse	0.8333	0.8333	0.8333
run	0.3243	0.4444	0.3750
shake_hands	0.6562	0.7000	0.6774
shoot_ball	1.0000	0.8667	0.9286
shoot_bow	0.8929	0.8333	0.8621
shoot_gun	0.7241	0.7000	0.7119
sit	0.5714	0.7692	0.6557
situp	0.9231	0.8000	0.8571
smile	0.7273	0.5333	0.6154
smoke	0.6250	0.5000	0.5556
somersault	0.6250	0.8333	0.7143
stand	0.6800	0.5312	0.5965
swing_baseball	0.6250	0.1667	0.2632
sword	0.4545	0.3333	0.3846
sword_exercise	0.4062	0.4333	0.4194
talk	0.6316	0.8000	0.7059
throw	0.1935	0.2000	0.1967
turn	0.4348	0.5405	0.4819
walk	0.5135	0.6333	0.5672
wave	0.3333	0.1379	0.1951
Avg/Total	**0.6602**	**0.6405**	**0.6412**

Table 5.12: Results of the proposed method with and without the flow scaling and crop filling procedures on the HMDB51 dataset (split 1).

	Baseline	**SVM**
No Flow Scaling + Crop Filling	48.4	59.9
Flow Scaling + Crop Filling	**50.6**	**64.1**

not applying it, as we are left with fewer dimensions to deal with, and there will be a resultant reduction in training and testing time (with no effect on performance) for the linear SVM.

Similarly to the KTH dataset, we perform an investigation of not performing the 'crop filling' and flow scaling procedures. This resulted in a baseline accuracy of 48.4%, which is lower than the baseline accuracy compared with when the two proposed methods are employed - 50.6%. The SVM-based approach also drops from 64.1% to 59.9%. This reinforces the suggestion that performing these two steps aids performance, and the benefits of doing so are more clear on the HMDB51 dataset than on the KTH dataset. We posit that the scaling procedure results in flow values that are more representative of what the flow values should be at the resolution of the input. The 'filling' procedure, we posit, represents a more natural progression of the video, and by extension, the action, than simply repeatedly appending the last frame would. We can see a tabulation of these aforementioned results in Table 5.12.

5.4 Conclusion

The method performs admirably for a few reasons. Firstly, given the limited temporal resolution we use for the crops to train the I3D networks, we are able to achieve relatively good performance. We compete with state-of-the-art on the simpler KTH dataset where temporal resolution is not such a vital factor for performance, and we perform less well on more complicated, varietal datasets in which temporal resolution is of utmost importance in order to capture the full temporal extent of the actions and what distinguishes them from very similar actions. For such complicated datasets (i.e. HMDB51), we perform appropriately given the constraints, and better than many hand-crafted feature approaches, and better than some deep-learning-based approaches. Further, we show that a performance gain is possible by treating the RGB and flow networks as generic local action feature vector extractors, and training another model (in this case an SVM) on these feature vectors to task them instead with learning the mapping to action classes. This was true for both datasets evaluated.

We also show that performing two simple steps of crop filling and optical flow scaling results in per-

formance gains in all cases. The intuitions behind performing both of these steps seems to hold. Crop filling results a more natural progression of the action / video that simply stacking the final frame does. Further, scaling to optical flow according to the spatial down-sampling or up-sampling results in more reasonable optical flow values for the new resolution. Lastly, it is interesting to note that our findings for the KTH dataset of flow information being a more important and useful cue for action recognition than spatial information is consistent with findings previous research [19].

Chapter 6

Conclusion and Future Work

Human action recognition is a very active area of research, and many new methods and techniques are being released and invented on an almost-weekly basis. This is because video recognition has not seen the same sort of performance gains and massive success that its two-dimensional counterpart, image recognition, has seen in recent years, even with the popularisation of deep learning, and by extension, convolutional neural networks. This lack of success is most likely due to the action recognition problem extending into the temporal domain, causing an extra dimension to consider when modelling the problem.

However, recently, large strides in the problem domain have been made with the recent availability of very large labeled video datasets such as Kinetics and YouTube-8M. These sort of datasets were not available until recently; we were instead limited to relatively small labeled video datasets, such as UCF-101, and HMDB51. Thus, video action recognition systems could not leverage transfer learning at the scale of image recognition systems (as very large labelled image datasets have been readily available for years), which is a large part of the success of convolutional neural networks on many problems. Another issue was that simply extending successful 2D CNNs into the temporal domain by making them 3D was not effective enough, since the networks were too small due to computational limitations, and therefore could not learn effectively. A combination of more intelligent 3D architectures [19] and being able to leverage transfer learning with datasets such as Kinetics has led to big advances in video action recognition.

This is the main reason that we use I3D pre-trained on these datasets as the backbone for our method. We extend it by scaling the optical flow inputs, and treating the networks as local action feature extractors. We then feed the concatenated, crop-level features from the networks into a linear SVM for action

classification. This method, given the constraints at hand (time and computational resources), proves useful and achieves results worthy of further investigation. The method competes with state-of-the-art on the smaller, simpler dataset KTH, achieving a highest accuracy of 96.6%. However, the method does not compete with state-of-the-art on the larger, more complex dataset HMDB51, achieving an average accuracy of 62.8% over the three train/test splits. This suggests that computational resources, and thus by extension, temporal and spatial resolution of the input samples for the networks, had a large impact on performance of the system for HMDB51. Not performing data augmentation (again, due to computational constraints) is also a factor that likely negatively affected performance of our approach, as this is common in deep learning approaches since it greatly aids performance. However, considering these constraints, we still perform admirably for this dataset, competing with other deep learning approaches, and outperforming many hand-crafted feature approaches. Lastly, we show that performing the proposed crop filling and flow scaling procedures is beneficial, as both the baseline and SVM approach improved on both studied datasets when employing these two techniques.

Based on our results, we can deem both research hypotheses correct. Local features from a pre-trained 3D CNN contain rich enough information to result in good action recognition performance. Further, treating the networks as feature extractors, and feeding these local features into a linear SVM results in significantly better performance than using a simple majority voting scheme from the networks' predictions, for both studied datasets. We leave it as future work to determine if the method can compete with state-of-the-art approaches when trained on larger batches of video crops with larger temporal and spatial resolutions.

6.1 Future Work

The most important extension is to increase the temporal resolution (and spatial resolution) to see how this affects performance, and at one point does performance plateau - when the increase in resolution no longer aids performance. Furthermore, different, more accurate, optical flow algorithms could be used in place of Farneback's methods (such as TVL-1 [69] or Brox [70]). However, these methods are generally much slower to compute, and thus to make them feasible for a real-time human action recognition system one would likely need to use their GPU implementations. Another possible extension could be to add additional cues to the recognition, such as pose (in addition to the appearance and motion cues in this work), or to train the networks on more crops of the videos (by sampling more crops, or not limiting the

temporal resolution and then sampling from the full video).

Bibliography

[1] UCF, "Ucf sports action dataset," 2017.

[2] I. Laptev, "Recognition of human actions," 2017.

[3] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: a large video database for human motion recognition," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

[4] I. Laptev, "Human actions and scene dataset," 2009.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[6] I. Laptev, "On space-time interest points," *Int. J. Comput. Vision*, vol. 64, pp. 107–123, Sept. 2005.

[7] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, (Berlin, Heidelberg), pp. 363–370, Springer-Verlag, 2003.

[8] R. Chaudry, A. Ravichandran, G. Hager, and R. Vidal, "Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions," *Center for Imaging Science, Johns Hopkins University*, 2005.

[9] J. Krapac, J. Verbeek, and F. Jurie, "Modeling spatial layout with fisher vectors for image categorization," in *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, (Washington, DC, USA), pp. 1487–1494, IEEE Computer Society, 2011.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[11] H. Wang, A. Klaser, C. Schmid, and C.-L. Liu, "Action recognition by dense trajectories," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, (Washington, DC, USA), pp. 3169–3176, IEEE Computer Society, 2011.

[12] X. Peng, C. Zou, Y. Qiao, and Q. Peng, "Action recognition with stacked fisher vectors," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 581–595, Springer International Publishing, 2014.

[13] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, (Washington, DC, USA), pp. 4489–4497, IEEE Computer Society, 2015.

[14] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, (Cambridge, MA, USA), pp. 568–576, MIT Press, 2014.

[15] S. Ke, H. Thue, Y. Lee, J. Hwang, J. Yoo, and K. Choi, "A review on video-based human activity recognition," *Computers*, 2013.

[16] A. Klaser, M. Marszalek, and C. Schmid, "A Spatio-Temporal Descriptor Based on 3D-Gradients," in *BMVC 2008 - 19th British Machine Vision Conference* (M. Everingham, C. Needham, and R. Fraile, eds.), (Leeds, United Kingdom), pp. 275:1–10, British Machine Vision Association, Sept. 2008.

[17] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local svm approach," in *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03*, ICPR '04, (Washington, DC, USA), pp. 32–36, IEEE Computer Society, 2004.

[18] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, (Washington, DC, USA), pp. 3551–3558, IEEE Computer Society, 2013.

[19] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the kinetics dataset," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 4724–4733, 2017.

[20] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Val Gool, "Temporal segment networks: Towards good practices for deep action recognition," in *ECCV*, 2016.

[21] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, (Washington, DC, USA), pp. 886–893, IEEE Computer Society, 2005.

[22] R. Girdhar, D. Ramanan, A. Gupta, J. Sivic, and B. Russell, "ActionVLAD: Learning spatio-temporal aggregation for action classification," in *CVPR*, 2017.

[23] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[24] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[25] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[27] C. Solomon and T. Breckon, *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. Wiley Publishing, 1st ed., 2011.

[28] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[29] H. Wang, M. Muneeb Ullah, A. Klser, I. Laptev, and C. Schmid, "Evaluation of local spatio-temporal features for action recognition.," in *Proceedings of the British Machine Vision Conference*, 09 2009.

[30] R. Szeliski, *Computer Vision: Algorithms and Applications*. New York, NY, USA: Springer-Verlag New York, Inc., 1st ed., 2010.

[31] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A comparison of affine region detectors," *Int. J. Comput. Vision*, vol. 65, pp. 43–72, Nov. 2005.

[32] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artif. Intell.*, vol. 17, pp. 185–203, Aug. 1981.

[33] D. Fortun, P. Bouthemy, and C. Kervrann, "Optical flow modeling and computation: a survey," *Computer Vision and Image Understanding*, vol. 134, p. 21, May 2015.

[34] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[35] J. Shi and C. Tomasi, "Good features to track," tech. rep., Ithaca, NY, USA, 1993.

[36] N. Buch, J. Orwell, and S. Velastin, "3d extended histogram of oriented gradients (3dhog) for classification of road users in urban scenes," 2009.

[37] R. V. H. M. Colque, C. A. C. Júnior, and W. R. Schwartz, "Histograms of optical flow orientation and magnitude to detect anomalous events in videos," in *Proceedings of the 2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, SIBGRAPI '15, (Washington, DC, USA), pp. 126–133, IEEE Computer Society, 2015.

[38] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *Proceedings of the 9th European Conference on Computer Vision - Volume Part II*, ECCV'06, (Berlin, Heidelberg), pp. 428–441, Springer-Verlag, 2006.

[39] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[40] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *In Workshop on Statistical Learning in Computer Vision, ECCV*, pp. 1–22, 2004.

[41] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *Int. J. Comput. Vision*, vol. 105, pp. 222–245, Dec. 2013.

[42] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *CVPR 2010 - 23rd IEEE Conference on Computer Vision & Pattern Recognition*, (San Francisco, United States), pp. 3304–3311, IEEE Computer Society, June 2010.

[43] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng, "Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, (Washington, DC, USA), pp. 3361–3368, IEEE Computer Society, 2011.

[44] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[45] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, Sept. 1995.

[46] D.-R. Chen, Q. Wu, Y. Ying, and D.-X. Zhou, "Support vector machine soft margin classifiers: Error analysis," *J. Mach. Learn. Res.*, vol. 5, pp. 1143–1175, Dec. 2004.

[47] A. Graves, A. rahman Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, 2013.

[48] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[49] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," in *Proceedings of the Second International Conference on Human Behavior Unterstanding*, HBU'11, (Berlin, Heidelberg), pp. 29–39, Springer-Verlag, 2011.

[50] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, (USA), pp. 1033–1040, Omnipress, 2011.

[51] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[52] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax,*

Semantics and Structure in Statistical Translation, (Doha, Qatar), pp. 103–111, Association for Computational Linguistics, Oct. 2014.

[53] B. C. Csaji, "Approximation with artificial neural networks," *Faculty of Sciences, Eotvos University, Hungary*.

[54] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," *CVPR '08*, 2008.

[55] S. Sadan and J. J. Corso, "Action bank: A high-level representation of activity in video," in *In: CVPR. (2012*.

[56] M. D. Rodriguez, J. Ahmed, and M. Shah, "Action mach: a spatio-temporal maximum average correlation height filter for action recognition," in *In Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition*, 2008.

[57] A. Yao, J. Gall, and L. Gool, "A hough transform-based voting framework for action recognition," 2010.

[58] A. Kovashka and K. Grauman, "Learning a hierarchy of discriminative space-time neighborhood features for human action recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[59] V. Kantorov and I. Laptev, "Efficient feature extraction, encoding, and classification for action recognition," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2593–2600, June 2014.

[60] P. Viola and M. Jones, "Robust real-time face detection," *International Journal of Computer Vision 57*, 2004.

[61] I. Laptev and T. Lindeberg, *Local Descriptors for Spatio-temporal Recognition*. 2006.

[62] L.-J. Li, H. Su, Y. Lim, and L. Fei-Fei, "Object bank: An object-level image representation for high-level visual recognition," *Int. J. Comput. Vision*, vol. 107, pp. 20–39, Mar. 2014.

[63] K. G. Derpanis, M. Sizintsev, K. J. Cannons, and R. P. Wildes, "Efficient action spotting based on a spacetime oriented structure representation," *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1990–1997, 2010.

[64] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[65] G. Varol, I. Laptev, and C. Schmid, "Long-term temporal convolutions for action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1510–1517, 2018.

[66] V. Choutas, P. Weinzaepfel, J. Revaud, and C. Schmid, "Potion: Pose motion representation for action recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 2018.

[67] W. Zhu, N. Zeng, and N. Wang, "Sensitivity, specificity, accuracy, associated confidence interval and roc analysis with practical sas implementations," *NorthEast SAS users group, health care and life sciences*, 01 2010.

[68] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, (New York, NY, USA), pp. 233–240, ACM, 2006.

[69] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime tv-l1 optical flow," in *Proceedings of the 29th DAGM Conference on Pattern Recognition*, (Berlin, Heidelberg), pp. 214–223, Springer-Verlag, 2007.

[70] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," in *European Conference on Computer Vision (ECCV)*, vol. 3024 of *Lecture Notes in Computer Science*, pp. 25–36, Springer, May 2004.