Semi- Supervised Human Action Recognition Using Clustering and SVM Classifier

A Project report submitted in partial fulfilment of the requirements for

the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

V PARIMALA	318126510058
V DIVYA	318126510059
K LEELA PRASAD	318126510024
G GOPAL KRISHNA	318126510017

Under esteemed guidance of

G.Gowri Pushpa,M.Tech,(P.hD)

(Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY & SCIENCES

(***Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade***)

SANGIVALASA,VISAKHAPATNAM-531162

2021-2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY & SCIENCES
(UGC AUTONOMOUS)
(*Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)
Sangivalasa, Bheemili mandal, Visakhapatnam dist.(A.P)

BONAFIDE CERTIFICATE

This is to certify that this project report "**Semi-Supervised Human Action Recognition Using Clustering and SVM classifier** " is the bonafide work of **V. Parimala (318126510058), V. Divya (318126510059), K. Leela Prasad (318126510024) and G. Gopal Krishna (318126510017)** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of Anil Neerukonda Institute of Technology & Sciences (ANITS), Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

PROJECT GUIDE

G.GowriPushpa,M.Tech,(Ph.D)
Asst. Professor
Dept of CSE,ANITS

HEAD OF THE DEPARTMENT

Dr.R.SIVARANJANI,M.Tech,Ph.D
Professor
Dept of CSE,ANITS

DECLARATION

We, **V.Parimala, V.Divya, K.Leela Prasad, G.Gopal Krishna**, of final year B.Tech., in the department of Computer Science & Engineering from ANITS,Visakhapatnam, hereby declare that the project report entitled "**Semi-Supervised Human Action Recognition Using Clustering and SVM classifier**" is carried out by us and submitted in partial fulfillment of the requirements for the award of **Bachelor of Technology in Computer Science and Engineering**, under Anil Neerukonda Institute of Technology & Sciences(ANITS) during the academic year 2021-22 and has not been submitted to any other University for the award of any kind of degree.

V PARIMALA	318126510058
V DIVYA	318126510059
K LEELA PRASAD	318126510024
G GOPAL KRISHNA	318126510017

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **G.Gowri Pushpa,** Asst.Professor, Department of Computer Science and Engineering, ANITS, for her guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr.R.SIVARANJANI**, Head of the Department, Computer Science and Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa,** for their encouragement and cooperation to carry out this work.

We express our thanks to Project Coordinator **P.Krishnanjaneyulu,**for his Continuous support and encouragement. We thank all the teaching faculty of the Department of CSE, whose suggestions during reviews helped us in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS:

V PARIMALA	318126510058
V DIVYA	318126510059
K LEELA PRASAD	318126510024
G GOPAL KRISHNA	318126510017

CONTENTS

ABSTRACT

In recent years, Human action recognition gained more importance for its variety of applications like video surveillance, Entertainment, Human-Robot Interaction and so on. Action recognition targets recognising different actions from a sequence of observations and different environmental conditions. This project compares two different handcrafted feature extraction techniques (i.e.. SIFT and optical flow) with SVM classifier on KTH dataset. The SIFT technique detects the interest points for each frame in the video. Then building a codebook using Bag-of-Words approach with the clustering technique as K-means clustering, then the classification is done using the SVM classifier. Similar, process is followed for another feature extraction technique i.e. optical flow. The experimental results of our approach gives an accuracy of 74.34% for SIFT and 84.37% for optical flow.

LIST OF FIGURES:

LIST OF TABLES:

1.INTRODUCTION

Vision-based Human action recognition has received increasing attention in computer vision and has made significant progress in recent years. Adequate training data is essential for detecting specific human actions. Relevant features can be extracted from the videos by applying image processing and machine learning techniques and these features can be used to recognize human actions. This type of recognition not only provides us with knowledge about certain people, but it also assists us in understanding their intentions, feelings, and attitudes.In this project, Human actions are recognised by extracting features from videos using the built-in SIFT and Optical Flow feature extraction algorithms.This project start with extracting features from the 6 action classes of KTH dataset by in built SIFT features extraction method,by applying image processing and machine learning techniques, relevant features are extracted from videos and are subsequently used to model and recognise human actions and then these features are split into test and train feature keypoints, and then these keypoints are grouped into clusters by using k-means algorithm and they are build into bag-of-words vectors and will train the SVM classifier with these vectors and then we will test the SVM classifier with train keypoint vectors and we will construct confusion matrix, With the help of confusion matrix we will find the accuracy.

2.PROBLEM STATEMENT

As a human being, identifying human actions is a quite simple task but for a machine it is a difficult task ,and it is difficult for human beings to identify actions in large datasets but machines can process large amounts of data very easily. So if we train machines to identify the actions that are performed by humans, we can process large amounts of dataset. This project is based on Semi-Supervised Learning which represents a comparison between hand-crafted methods using clustering and SVM classifier based analysis, and discussion on these approaches.

3.RELATED WORK

Human Action Recognition approach mainly focuses on video representations, by extracting the features and describing their formats, there are different feature extraction techniques are available so there is a need to choose the efficient and relevant one. Vision based human action recognition involves feature extraction and action classification by assigning labels to the classes. The assignment of labels is a challenging task as the classifier can misclassify the actions because of the challenges like inter and intra- class variations, Cluttered Background and Camera Motion and Insufficient annotated data etc. Tracking of keypoints generated from SIFT flow trajectory algorithm and descriptor are trained for various trajectories. There are three different types of classifiers: The method of the Bag-Of-Words, Linear and Non-Linear SVM, results of these methods were compared and tabulated. These different techniques were implemented on UCF Sport Action dataset. Two-third video clips of each ten different actions were taken as training dataset and remaining were used for testing purpose. Linear SVM classification showed 87% of accuracy whereas in Non-linear SVM this accuracy was reduced to approximately 80%. Thus, in this proposed method Linear SVM classifier is used for better performance.

4.PROPOSED SYSTEM

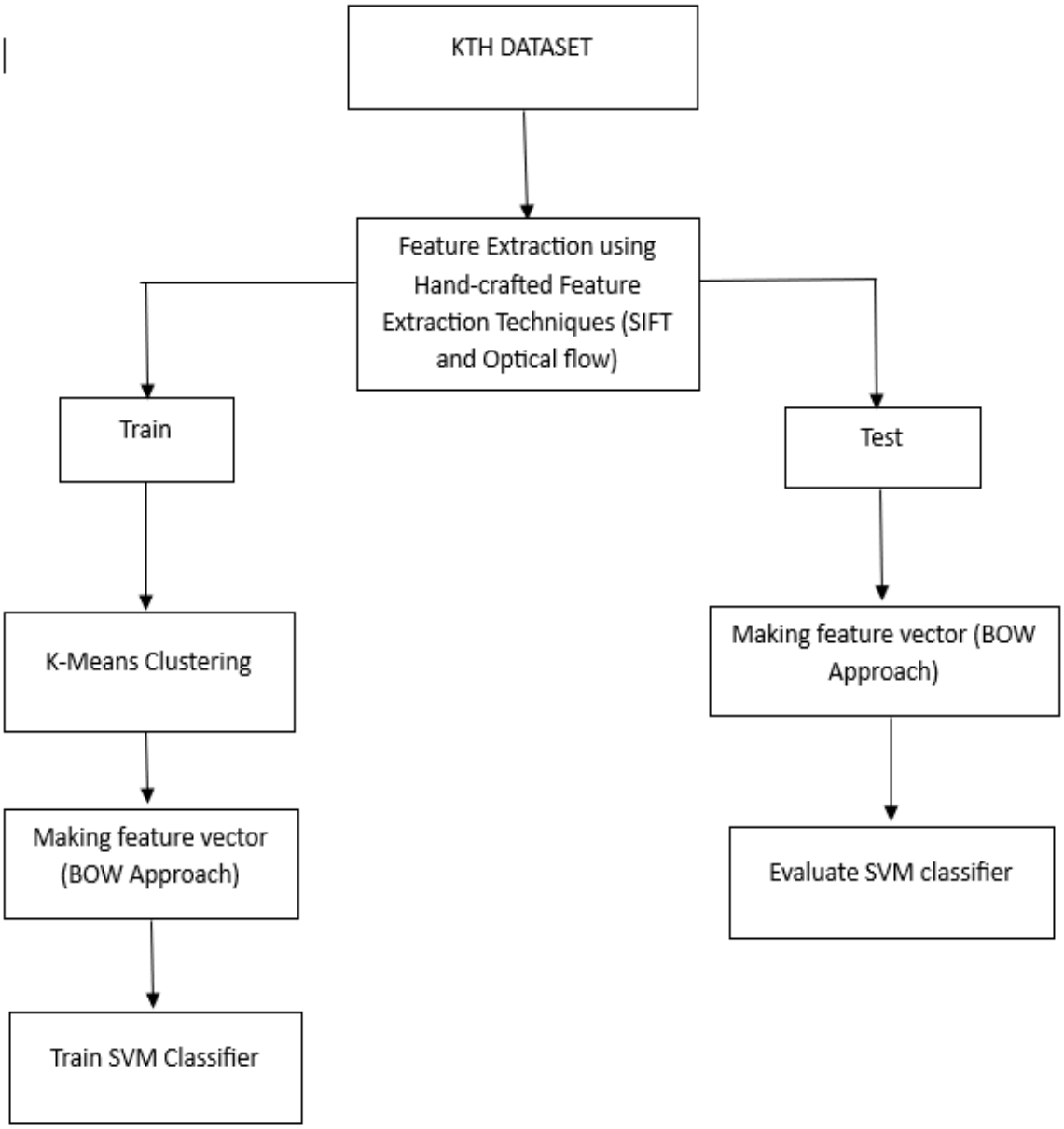


Fig-4.1 : Action Recognition Approach

4.1 About Dataset

In 2004, KTH Royal Institute of Technology began working on to create a non-trivial and publicly available dataset and created KTH dataset.It is the standard dataset and contains 6 action classes: boxing, hand clapping, hand waving, jogging, running, walking and 4 scenarios: outdoor s1,outdoor with scale variation s2, outdoor with different clothes s3, indoor s4) .The below rows in the table (Table3.1) shows the action classes and columns shows the four scenarios in which these actions are performed.

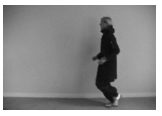
Action	Scenario s1	Scenario s2	Scenario s3	Scenario s4
Boxing				
Hand Clapping				
Hand Waving				
Jogging				
Running				
Walking				

Table-4.1 : Action database : different actions and scenarios

5.METHODOLOGY

1.Features are extracted by using Hand Crafted Feature Extraction techniques such as:

- SIFT(Scale Invariant Feature Transform)
- Optical Flow

2.K-means clustering is used to build codebook.

3.Build Bag-of-words vectors for each video.

4. Train SVM(Support Vector Machine) Classifier.

5.1 Feature Extraction

5.1.1 Scale Invariant Feature Transform (SIFT) :

SIFT is a computer vision feature identification technique that was first presented in 2004 by D.Lowe of the University of British Columbia. This technique is used to locate local features in the frame known as 'key-points'.These keypoints are invariant in scale and rotation.The major advantage of SIFT features, over edge features or hog features, is that they are not affected by the size or orientation of the image. This technique mainly involves 4 basic steps:

5.1.1.1 Scale Space Extrema Detection:

In this step we will blur the image to reduce the noise in the image and keypoints are also identified in this step. Blurring the image is done using "Gaussian Blurring technique". So, for every pixel in an image, the Gaussian Blur calculates a value based on its neighboring pixels. The scale space of an image is a function L(x,y,σ) that is produced from the convolution of a Gaussian kernel(Blurring) at different scales with the input image. Scale-space is separated into octaves and the number of octaves and scale depends on the size of the original image.

5.1.1.1.1 Blurring :

Within an octave, images are progressively blurred using the Gaussian Blur operator. Mathematically, "blurring" is referred to as the convolution of the Gaussian operator and the image. Gaussian blur has a particular expression or "operator" that is applied to each pixel.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad \text{5(1)}$$

Blurred image equation

G is the Gaussian Blur operator and I is an image.While x,y are the location coordinates and σ is the "scale" parameter. Greater the value, greater the blur.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad \text{____5(2)}$$

Gaussian blur operator

5.1.1.1.2 Difference of Gaussian Kernel (DOG) :

This method is used to enhance the features in the image. It involves the subtraction of one blurred version of an original image from another, less blurred version of the original.DoG creates another set of images, for each octave, by subtracting every image from the previous image in the same scale.Here is a visual explanation of how DoG is implemented:

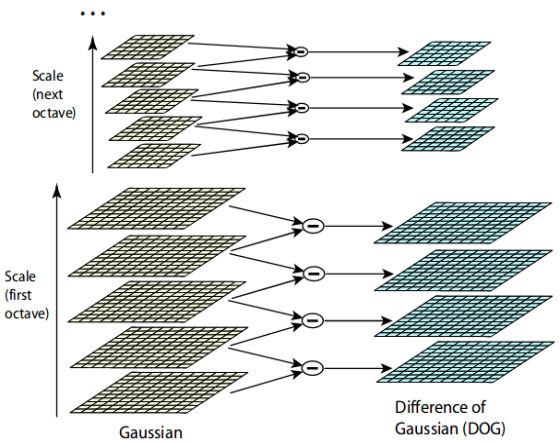


Fig-5.1 : Dog Implementation

5.1.1.1.3 Finding Keypoints :

One pixel in an image is compared with its 8 neighbors as well as 9 pixels in the next scale and 9 pixels in previous scales. This way, a total of 26 checks are made. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale.

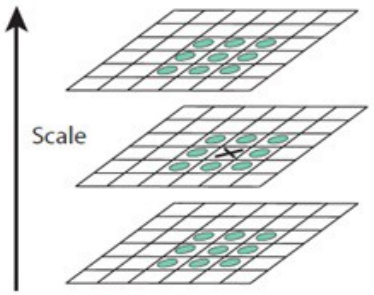


Fig-.5.2 : Keypoint Representation

5.1.1.2 Key Point Localization:

The key points that are identified in the above step are filtered by eliminating sensitive, unwanted key points and by using edge detection process we will eliminate edges and corners are considered.Some of these keypoints may not be robust to noise. This is why there is a need to perform a final check to make sure that we have the most accurate keypoints to represent the image features. Hence, eliminate the keypoints that have low contrast, o r lie very close to the edge. To deal with the low contrast keypoints, a second-order Taylor expansion is computed for each keypoint. If the resulting value is less than 0.03 (in magnitude), we reject the keypoint. A second-order Hessian matrix is used to identify edge keypoints. You can go through the math behind this here.

5.1.1.3 Orientation Assignment:

At this stage, a set of stable keypoints for the images are listed. The next thing is to assign an orientation to each keypoint to make it rotation invariance. Again divide this step into two smaller steps:

a) Calculate the magnitude and orientation
b) Create a histogram for magnitude and orientation

5.1.1.3.1 Calculate the magnitude and orientation :

Calculating the gradients in x and y directions by taking the difference between left & right pixels(Gx) and up & down pixels(Gy) as shown in the sample image :

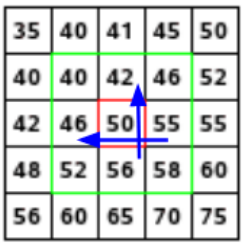


Fig-5.3 : Selecting Gx and Gy

Once we have the gradients, we can find the magnitude and orientation using the following

formulas:

$$\text{Magnitude} = \sqrt{[(G_x)^2+(G_y)^2]} \quad 5(3)$$

$$\Phi = \text{atan}(G_y / G_x) \quad 5(4)$$

The magnitude represents the intensity of the pixel and the orientation gives the direction for the same.

5.1.1.3.2 Creating histogram for the bin values :

On the x-axis, we will have bins for angle values and on the y-axis the magnitudes are mentioned. A sample histogram image is shown below

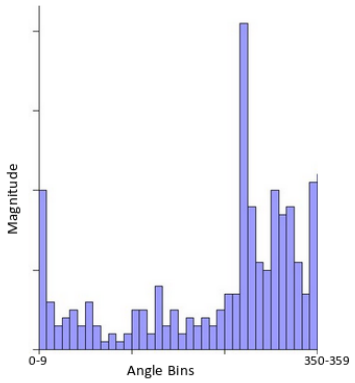


Fig-5.4 : Sample histogram for angle values and magnitudes

The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with the same location and scale, but different directions. It contributes to the stability of matching.

5.1.1.4 Key Point Descriptor:

This is the final step for SIFT. So far, we have stable key points that are scale-invariant and rotation invariant. Next is to compute a descriptor for the local image region about each keypoint that is highly distinctive and invariant as possible to variations such as changes in

viewpoint and illumination. Here there is a use of neighboring pixels, their orientations, and magnitude, to generate a unique fingerprint for this keypoint which called a '**descriptor**'. To do this, a 16x16 window around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So 4 X 4 descriptors over 16 X 16 sample array were used in practice. 4 X 4 X 8 directions give 128 bin values. It is represented as a feature vector to form a keypoint descriptor.

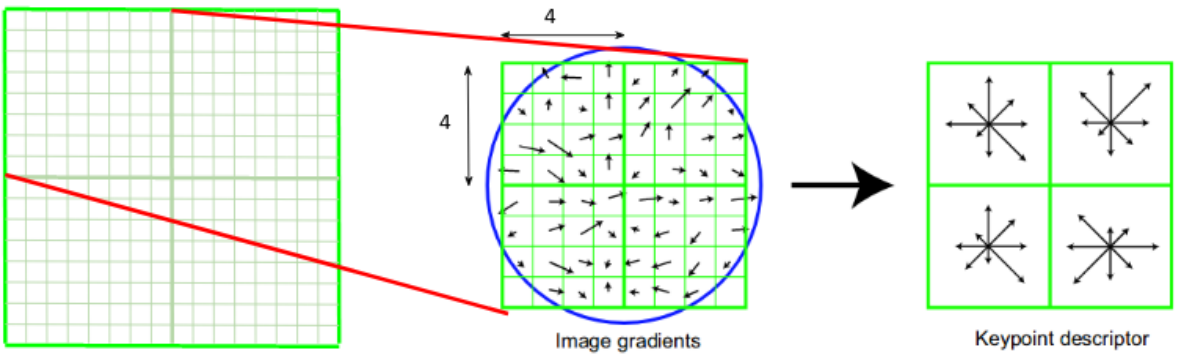


Fig-5.5 : Computing Keypoint Descriptor

This feature vector introduces a few complications. We need to get rid of them before finalizing the fingerprint.

1 Rotation dependence The feature vector uses gradient orientations. Clearly, if you rotate the image, everything changes. All gradient orientations also change. To achieve rotation independence, the keypoint's rotation is subtracted from each orientation. Thus each gradient orientation is relative to the keypoint's orientation.

2 Illumination dependence If we threshold numbers that are big, we can achieve illumination independence. So, any number (of the 128) greater than 0.2 is changed to 0.2. This resultant feature vector is normalized again. And now you have an

illumination independent feature vector.

However in this project we are using inbuilt method for sift feature (i.e. keypoints and descriptors of an image) extraction i.e. SIFT in OpenCV

5.1.1.5 Methods:

sift.detect(): It is used to find the keypoints of an image.

sift.compute():It is used to find the descriptors from the obtained keypoints.

5.1.1.5.1 Code:

```
sift = cv2.SIFT_create();

image=cv2.imread(img.jpg);

g=cv.cvtColor(img,cv.COLOR_BGR2GRAY);

keypoints=sift.detect(g,None);

descriptors=sift.compute(g, keypoints);

img=cv.drawKeypoints(g,keypoints,image);
```

5.1.1.5.2 Output:

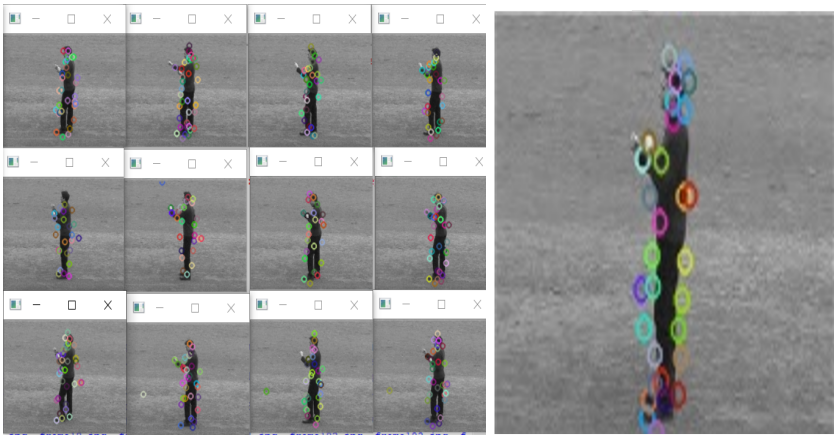


Fig-5.6 : SIFT keypoints

5.1.2. OPTICAL FLOW:

Optical flow is a technique used to describe image motion. It is usually applied to a series of images that have a small time step between them, for example, video frames. An optical flow (OF) algorithm calculates the displacement of brightness patterns from one frame to another.It calculates velocity for points within the images, and provides an estimation of where points could be in the next image sequence.

The most basic assumption made in optical flow calculations is image brightness constancy. Mathematically, this is

$$\mathbf{f(x + \Delta x, y + \Delta y, t + \Delta t) \approx f(x, y, t)} \quad 5(5)$$

where f(x, y, t) is the intensity of the image at position (x,y) and at time t, and Δx, Δy is the change in position and Δt is the change in time.

In this project we use dense optical flow(Attempts to compute the optical flow vector for each frame's pixel. While this method is slower, it produces a more efficient and dense result, which is ideal for applications like learning structure from motion and video segmentation).To analyze dense optical flow, OpenCV provides the function cv2.calcOpticalFlowFarneback.

5.1.2.1 Syntax:

cv2.calcOpticalFlowFarneback(prev, next, pyr_scale, levels, winsize, iterations, poly_n, poly_sigma, flags[, flow])

5.1.2.2 Parameters:

prev: First input image

next: Second input image of same type and same size as prev

pyr_scale: This parameter is used to image scale while building pyramids for each image (scale 0.5).

levels: levels=1 indicates that there are no additional layers (just the first image).

winsize: It is the average window size; the higher the size, the more noise-resistant the technique is and the faster the motion detection is, however the motion fields are blurred (winsize=20).

Iterations: At each pyramid level, the number of iterations to be executed (iterations=1).

poly_n: The size of the pixel neighbourhood is utilized to determine polynomial expansion between pixels, and it is usually 5 or 7 (poly_5).

poly_sigma: As the basis of the polynomial expansion, the standard deviation of the gaussian, which is for derivatives to be smooth. For poly= 5, it can be 1.1, and for poly= 7, it can be 1.5.

5.1.2.3 Code:

cv2.calcOpticalFlowFarneback(prev,next,winsize=20,iterations=1,flags=cv2.OPTFLOW_FARNEBACK_GAUSSIAN,levels=1,pyr_scale=0.5,poly_n=5,poly_sigma=1.1,flow=None)

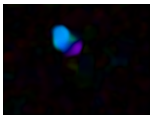
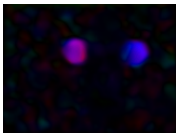
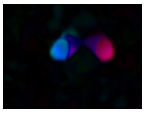
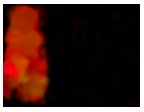
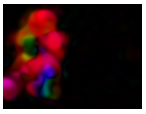
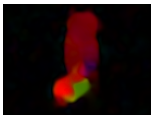
Action	Boxing	hand clapping	hand waving	jogging	running	walking
Before						
After						

Table-5.1:Sample images captured using Optical Flow technique

5.2 Building Bag Of Words Model:

The Bag-of-words model is generally used in Natural Language Processing(NLP) to represent textual information. It ignores the words in the text and only focuses on the number of words in a text. The number of words in the vocabulary form a n- dimensional vector where each number in the vector represents a count of words in the corresponding position in the vocabulary. The specific application is that after using the feature extraction algorithm to extract several feature points of each image, all feature points are treated as visual words which is equivalent to the word in the text. Then the k-means algorithm is used to cluster all

visual words. Next we need to construct the vocabulary of the BOW model. Each image can then be represented by a multidimensional vector to facilitate later classification.

The following techniques are used in building bag of words model:

- Clustering algorithm(K-means)
- Vector Quantization
- TF-IDF

5.2.1 K-means Clustering Algorithm:

K-means clustering is a Unsupervised Learning algorithm that divides the unlabeled dataset into different clusters. K specifies the number of pre-defined clusters that must be produced during the process; for example, if K=4, four clusters will be created, and if K=5, five clusters will be created.

The following steps will explain how the K-Means algorithm works:

Step-1: Choose k value i.e the number of clusters that you want to form.

Step-2: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-3: Calculate the variance and place a new centroid of each cluster.

Step-4: Repeat the third step, which means reassigning each datapoint to the new closest centroid of each cluster.

Step-5: If any reassignment occurs, then go to step-3 else go to finish.

5.2.1.1 Code:

```
kmeans = KMeans(init='k-means++', n_clusters=clusters, n_init=10, n_jobs=2, verbose=1)

kmeans.fit(train_features)
```

5.2.1.2 Parameters:

n_clusters : The number of clusters to be formed as well as the number of centroids to be generated.

k-means++:Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.

n_init:Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

verbose: It is used for producing detailed logging information..

5.2.1.3 Output:

```
========== RESTART: C:/Python36/HAR/HAR_OF/cluste
Loading dataset
Number of feature points to run clustering on: 187069
Running KMeans clustering

Iteration 150, inertia 11689687.867866496
Iteration 151, inertia 11689684.948534662
Iteration 152, inertia 11689682.190207291
Iteration 153, inertia 11689678.616929717
Iteration 154, inertia 11689675.63852898
Iteration 155, inertia 11689673.147707526
Iteration 156, inertia 11689670.724060418
Iteration 157, inertia 11689669.333042668
Iteration 158, inertia 11689667.630482083
Iteration 159, inertia 11689666.368164225
Iteration 160, inertia 11689665.87236607
Iteration 161, inertia 11689665.547234077
Iteration 162, inertia 11689665.191621926
Iteration 163, inertia 11689664.88331369
Converged at iteration 163: center shift 3.97406644424
3.9850693221235894e-05.
>>>
```

5.2.2 Vector Quantization:

The Vector Quantization technique compares each observation vector with the cluster centroids and assigns the observation to the closest cluster. It returns the cluster of each observation and the distortion.There is an in-built method called “vq()” in the python scipy library for vector quantization.

5.2.2.1 Code:

scipy.cluster.vq.vq (obs, code_book)

5.2.2.2 Parameters:

obs : (ndarray) Each row of the 'M' x 'N' array is an observation. The columns are the "features" seen during each observation. The features must be whitened first using the whiten function or something equivalent.

code_book : (ndarray) The code book is usually generated using the k-means algorithm. Each row of the array holds a different code, and the columns are the features of the code

5.2.3 TF-IDF Weighting Scheme :

TF-IDF stands for **Term Frequency-Inverse Document Frequency** and it is a measure, used in the fields of machine learning, that can quantify the importance or relevance of string representations in a document amongst a collection of documents.

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t. \quad \text{5(6)}$$

1 TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:
TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

2 IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:
IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

5.3 CLASSIFICATION (SVM CLASSIFIER):

"Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges.Support Vector Machine is used to classify both linear and non-linear datasets. It classifies the input dataset into different classes. To classify

linear data we will consider the support vectors. Maximum margin is the distance between the two support vectors. The hyperplane in which maximum margin is obtained is known as optical hyperplane

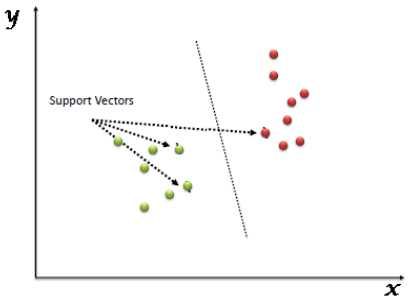


Fig-5.3.1 :Support Vector Machine

Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).One reasonable choice as the best hyperplane is the one that represents the largest separation or margin between the two classes. If such a hyperplane exists it is known as the maximum-margin hyperplane/hard margin.

6. IMPLEMENTATION

6.1 Action Recognition using SIFT feature extraction and SVM classifier

This method treats each frame as a separate instance and classifies each frame rather than the entire video. A video's result is chosen from the majority of all frame recognition .

This method consists of four basic steps:

1. Feature extraction using SIFT
2. Form clusters using K-means clustering
3. Creating Bag-Of-Words vector for each frame
4. Training with SVM

STEP- 1 : Feature extraction using SIFT

(run "extract_sift.py")

CODE :

```
import cv2
import os
import pickle
CATEGORIES = ["boxing","handclapping","handwaving", "jogging","running",
"walking"]
if __name__ == "__main__":
    os.makedirs("data", exist_ok=True)
    sift = cv2.SIFT_create()
    n_processed_files = 0
    for category in CATEGORIES:
        print("Processing category %s" % category)
        folder_path = os.path.join("..", "E:\PROJECT@12A\DataSet\kth_dataset", category)
        filenames = os.listdir(folder_path)
        features = []
       frame_count=0
```

```
    for filename in filenames:
  filepath = os.path.join("..", "E:\PROJECT@12A\DataSet\kth_dataset", category,
      filename)
    vid = cv2.VideoCapture(filepath)
    features_current_file = []
    while vid.isOpened():
      ret, frame = vid.read()
      if not ret:
        break
      frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      frame_count+=1
      kps,desc = sift.detectAndCompute(frame,None)
      if desc is not None:
        features_current_file.append(desc)
    features.append({
      "filename": filename,
      "category": category,
      "features": features_current_file
    })

  print("Number of frames : ",frame_count)
  pickle.dump(features, open("data1/sift_%s.p" % category, "wb"))
```

OUTPUT:

```
>>>
= RESTART: E:\PROJECT@12A\code\KTH-Action-Recognition-master\baseline_sift_bow_svm_individual_frame\extract.py
Processing category boxing
Number of frame for each category....
Category :  boxing 45187
Processing category handclapping
Number of frame for each category....
Category :  handclapping 42667
Processing category handwaving
Number of frame for each category....
Category :  handwaving 53678
Processing category jogging
Number of frame for each category....
Category :  jogging 43884
Processing category running
Number of frame for each category....
Category :  running 38505
Processing category walking
Number of frame for each category....
Category :  walking 65795
>>>
```

STEP - 2 : Creating training and testing sets from the calculated SIFT features. This also creates a file train keypoints.p that contains all SIFT features in the train set in a clustering-friendly format.

(run “make_datset.py”)

Code:

```
import numpy as np
import os
import pickle
CATEGORIES = ["boxing", "handclapping", "handwaving", "jogging", "running","walking"]

TRAIN_PEOPLE_ID = [11, 12, 13, 14, 15, 16, 17, 18,19, 20, 21, 23, 24, 25, 1, 4]
TEST_PEOPLE_ID = [22, 2, 3, 5, 6, 7, 8, 9, 10]
if __name__ == "__main__":
        train = []
        dev = []
        test = []
        train_keypoints = []
        for category in CATEGORIES:
                print("Processing category %s" % category)
                category_features = pickle.load(open("data/optflow_%s.p" % category, "rb"))
                for video in category_features:
                        person_id = int(video["filename"].split("_")[0][6:])
                        if person_id in TRAIN_PEOPLE_ID:
                                train.append(video)
```

```
                    for frame in video["features"]:
                        train_keypoints.append(frame)
                else:
                    test.append(video)
print("Saving train/test set to files")
pickle.dump(train, open("data/train.p", "wb"))
pickle.dump(test, open("data/test.p", "wb"))
print("Saving key points in training set")
pickle.dump(train_keypoints, open("data/train_keypoints.p", "wb"))
```

OUTPUT:

```
= RESTART: E:\PROJECT@12A\code\KTH-Action-Recognition-master\baseline_sift_bow_svm_individual_frame\make_dataset.py
Processing category boxing
Processing category handclapping
Processing category handwaving
Processing category jogging
Processing category running
Processing category walking
Saving dataset to files
Saving keypoints in dataset
Number of SIFT keypoints for each category
boxing: 945162
handclapping: 1141104
handwaving: 1403992
jogging: 202199
running: 91955
walking: 347920
```

STEP - 3: Forming clusters using K-means clustering on “train_keypoints.p”

(run “clustering.py”)

Code:

```
import numpy as np
import os
import pickle
```

```
from sklearn.cluster import KMeans

from numpy import size

if __name__ == "__main__":

        print("Loading dataset")
        train_features=pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-

master/ baseline_sift_bow_svm_individual_frame/data1/train_keypoints1.p", "rb"))

        train=np.vstack(train_features)

        n_features = len(train_features)

        print("Number of feature points to run clustering on: %d" % n_features)

        clusters=100

        print("Running KMeans clustering")

        kmeans = KMeans(init='k-means++', n_clusters=clusters, n_init=10,verbose=1)

        kmeans.fit(train)

        pickle.dump(kmeans, open("data1/cbt_%dclusters.p" % clusters, "wb"))
```

OUTPUT:

```
Number of clusterss :  800
Running KMeans clustering
Initialization complete
Iteration 0, inertia 13345188.0
Iteration 1, inertia 10405822.0
Iteration 2, inertia 10176519.0
Iteration 3, inertia 10085694.0
Iteration 4, inertia 10034166.0
Iteration 5, inertia 10001082.0
Iteration 6, inertia 9976014.0
Iteration 7, inertia 9957783.0
Iteration 8, inertia 9944360.0
Iteration 9, inertia 9934434.0
Iteration 10, inertia 9926662.0
Iteration 11, inertia 9919559.0
Iteration 12, inertia 9913121.0
Iteration 13, inertia 9907492.0
Iteration 14, inertia 9902708.0
Iteration 15, inertia 9898101.0
Iteration 16, inertia 9894391.0
Iteration 17, inertia 9890938.0
Iteration 18, inertia 9887857.0
Iteration 19, inertia 9884977.0
Iteration 20, inertia 9882637.0
Iteration 21, inertia 9880804.0
Iteration 22, inertia 9879218.0
Iteration 23, inertia 9877526.0
Iteration 24, inertia 9875994.0
Iteration 25, inertia 9874630.0
Iteration 26, inertia 9873546.0
Iteration 27, inertia 9872666.0
Iteration 28, inertia 9872042.0
Iteration 29, inertia 9871330.0
Iteration 30, inertia 9870782.0
Iteration 31, inertia 9870225.0
```

STEP - 4 : Making BoW vector for every video frame in the training set, using the computed clusters with TF-IDF weighting scheme.

(run "make_bow_vector.py")

Code:

```
import numpy as np
import os
import pickle
from scipy.cluster.vq import vq


def make_bow(dataset, clusters, tfidf):
    print("Make bow vector for each frame")
    n_frames = 0
    for video in dataset:
        n_frames += len(video["features"])
    bow = np.zeros((n_frames, clusters.shape[0]), dtype=float)
    frame_index = 0
    for video in dataset:
        for frame in video["features"]:
            visual_word_ids = vq(frame, clusters)[0]
            for word_id in visual_word_ids:
                bow[frame_index, word_id] += 1
            frame_index += 1
    if tfidf:
        print("Applying TF-IDF weighting")
```

```
        freq = np.sum((bow > 0) * 1, axis = 0)
        idf = np.log((n_frames + 1) / (freq + 1))
        bow = bow * idf
    frame_index = 0
    for i in range(len(dataset)):
        features = []
        for frame in dataset[i]["features"]:
            features.append(bow[frame_index])
            frame_index += 1
        dataset[i]["features"] = features
        if (i + 1) % 50 == 0:
            print("Processed %d/%d videos" % (i + 1, len(dataset)))
    return dataset
if __name__ == "__main__":
    tfidf=1
    codebook = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
        baseline_sift_bow_svm_individual_frame/data1/cbt_75clusters.p", "rb"))
    clusters = codebook.cluster_centers_
    print("No of cluster centers : ",len(clusters))
    dataset = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
        baseline_sift_bow_svm_individual_frame/data1/train.p", "rb"))
    dataset_bow = make_bow(dataset, clusters, tfidf)
    print("Length of dataset bow : ",len(dataset_bow))
    pickle.dump(dataset_bow, open("data1/train1_bow_100.p", "wb"))
```

OUTPUT:

```
Make bow vector for each frame
Applying TF-IDF weighting
Processed 50/407 videos
Processed 100/407 videos
Processed 150/407 videos
Processed 200/407 videos
Processed 250/407 videos
Processed 300/407 videos
Processed 350/407 videos
Processed 400/407 videos
Length of dataset bow :  407
```

STEP - 5 :Training the SVM classifier with the BoW vectors

(run "train_svm.py")

Code:

```
import numpy as np

import os

import pickle

from sklearn.svm import SVC


if __name__ == "__main__":

    C = 1

    print("Loading training set...")

    dataset= pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
    baseline_sift_bow_svm_individual_frame/data1/train1_bow_75.p", "rb"))

    print(len(dataset))

    X = []

    Y = []

    for video in dataset:

        for frame in video["features"]:
```

```
                X.append(frame)
                Y.append(video["category"])
        print("Length of X : ",len(X))
        print("Length of Y : ",len(Y))
        print("Training Linear SVM.....")
        clf = SVC(C=C, kernel="linear", verbose=True)
        print("Fitting..")
        clf.fit(X, Y)
        print("Saving into file....")
        pickle.dump(clf, open("data1/svm1_75.p", "wb"))
```

OUTPUT:

```
= RESTART: E:\PROJECT@12A\code\KTH-Action-Recognition-master\baseline_sift_bow_svm_individual_frame\train_svm.py
Loading training set...
383
Length of X :  68484
Length of Y :  68484
Training Linear SVM.....
Fitting..
[LibSVM]Saving into file....
>>>
```

STEP-6 : Evaluating SVM classifier against testing set

(run "evaluate.py")

Code:

```
import numpy as np
import os
import pickle
from sklearn.svm import SVC
CATEGORIES = ["boxing", "handclapping", "handwaving", "jogging", "running", "walking"]
if __name__ == "__main__":
        data = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
```

```
baseline_sift_bow_svm_individual_frame/data1/test1_bow_75.p", "rb"))
clf = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
baseline_sift_bow_svm_individual_frame/data1/svm1_75.p", "rb"))
confusion_matrix = np.zeros((6, 6))
correct = 0
for video in data:
        majority = {}
        for category in CATEGORIES:
                majority[category] = 0
for frame in video["features"]:
        predicted = clf.predict([frame])
        majority[predicted[0]] += 1
predicted = None
max_vote = -1
for category in CATEGORIES:
        if(majority[category] > max_vote):
        max_vote = majority[category]
        predicted = category
if(predicted == video["category"]):
        correct += 1
confusion_matrix[CATEGORIES.index(predicted),
        CATEGORIES.index(video["category"])] += 1
print("%d/%d Correct" % (correct, len(data)))
print("Accuracy =", correct / len(data))
```

print("Confusion matrix")

print(confusion_matrix)

OUTPUT:

```
>>>
================ RESTART: C:\Python36\HAR\HAR_SIFT1\evaluate.py =
142/191 Correct
Accuracy = 0.743455497382199
Confusion matrix
[[32.  6.  3. 14. 17.  9.]
 [ 0. 25.  0.  0.  0.  0.]
 [ 0.  0. 29.  0.  0.  0.]
 [ 0.  0.  0. 18.  0.  0.]
 [ 0.  0.  0.  0. 15.  0.]
 [ 0.  0.  0.  0.  0. 23.]]
>>>
```

6.2 Action Recognition using Optical Flow feature extraction and SVM classifier

This method consists of four basic steps:

1. Extract features using optical flow feature extraction technique
2. Form clusters using K-means clustering
3. Creating Bag-Of-Words vector for each frame
4. Training with SVM

STEP - 1 **:** Extract features using optical flow in the x and y direction for each frame in all videos.

(run “extract_optical_flow.py”)

Code :

```
import cv2
import numpy as np
import os
import pickle

CATEGORIES = ["boxing", "handclapping", "handwaving", "jogging", "running",
```

```
"walking"]

if __name__ == "__main__":
    os.makedirs("data", exist_ok=True)
    farneback_params=dict(winsize=20,iterations=1,flags=cv2.OPTFLOW_FARNEBACK
_GAUSSIAN, levels=1,pyr_scale=0.5, poly_n=5, poly_sigma=1.1, flow=None)
    n_processed_files = 0
    for category in CATEGORIES:
        print("Processing category %s" % category)
        folder_path = os.path.join("..", "E:\PROJECT@12A\DataSet\kth_dataset",
    category)
        filenames = os.listdir(folder_path)
        features = []
        frame_count=0
        for filename in filenames:
            filepath = os.path.join("..", "E:\PROJECT@12A\DataSet\ kth_dataset",
    category, filename)
            vid = cv2.VideoCapture(filepath)
            features_current_file = []
            prev_frame = None
            while vid.isOpened():
                ret, frame = vid.read()
                if not ret:
                    break

                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                frame_count+=1
                if prev_frame is not None:
                    flows = cv2.calcOpticalFlowFarneback(prev_frame,
frame,**farneback_params)

                    feature = []
                    for r in range(120):
                        if(r % 10 != 0):
                        continue
```

```
                                        for c in range(160):
                                            if(c % 10 != 0):
                                            continue
                                            feature.append(flows[r,c,0])
                                            feature.append(flows[r,c,1])
                                    feature = np.array(feature)
                                    features_current_file.append(feature)
                                prev_frame = frame
                        features.append({
                                "filename": filename,
                                "category": category,
                                "features": features_current_file})
                print("Number of frames : ",frame_count)
                pickle.dump(features, open("data/optflow_%s.p" % category, "wb"))
```

OUTPUT:

```
>>>
= RESTART: E:\PROJECT@12A\code\KTH-Action-Recognition-master\baseline_optflow_bow_svm\extract_optical_flow.py
Processing category boxing
Done 30 files
Done 60 files
Done 90 files
Processing category handclapping
Done 120 files
Done 150 files
Done 180 files
Processing category handwaving
Done 210 files
Done 240 files
Done 270 files
Processing category jogging
Done 300 files
Done 330 files
Done 360 files
Done 390 files
Processing category running
Done 420 files
Done 450 files
Done 480 files
Processing category walking
Done 510 files
Done 540 files
Done 570 files
```

STEP - 2 : Splitting the computed “Optical Flow” features into training and testing sets.This also generates a file train_keypoints.p of all Optical Flow features in the train set whose format is specifically used for clustering.

(run “make_dataset.py”)

Code :

```
import numpy as np
```

```
import os
import pickle

CATEGORIES = ["boxing", "handclapping", "handwaving", "jogging", "running", "walking"]

TRAIN_PEOPLE_ID = [1,4,11, 12, 13, 14, 15, 16, 17, 18,19,20,21,23,24,25]
TEST_PEOPLE_ID = [22, 2, 3, 5, 6, 7, 8, 9, 10]

if __name__ == "__main__":
    train = []
    test = []

    keypoints=[]
    train_keypoints = []

    for category in CATEGORIES:
        print("Processing category %s" % category)
        category_features = pickle.load(open("data/optflow_%s.p" % category, "rb"))

        for video in category_features:
            person_id = int(video["filename"].split("_")[0][6:])

            for frame in video["features"]:
                keypoints.append(frame)

            if person_id in TRAIN_PEOPLE_ID:
                train.append(video)

                for frame in video["features"]:
                    train_keypoints.append(frame)

            else:
```

```
            test.append(video)

    print("Total number of keypoints : ",len(keypoints))
    print("Number of training keypoints : ",len(train_keypoints))
    print("Saving train/test set to files")
    pickle.dump(train, open("data/train.p", "wb"))
    pickle.dump(test, open("data/test.p", "wb"))

    print("Saving keypoints in training set")
    pickle.dump(train_keypoints, open("data/train_keypoints.p", "wb"))
```

OUTPUT:

```
= RESTART: E:\PROJECT@12A\code\KTH-Action-Recognition-master\baseline_optflow_bow_svm\make_dataset.py
Processing category boxing
Processing category handclapping
Processing category handwaving
Processing category jogging
Processing category running
Processing category walking
Total number of keypoints :  289117
Number of training keypoints :  186805
Saving train/test set to files
Saving keypoints in training set
>>>
```

STEP - 3 : Forming clusters using K-means clustering on “train_keypoints.p”

(run “clustering.py”)

Code :

```
import numpy as np
import os
import pickle
from sklearn.cluster import KMeans
from numpy import size

if __name__ == "__main__":
    print("Loading dataset")
train_features = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
baseline_optflow_bow_svm/data/train_keypoints.p", "rb"))
```

```
train=np.vstack(train_features)
n_features = len(train_features)
print("Number of feature points to run clustering on: %d" % n_features)
clusters=500
print("No of clusterss : ",clusters)
print("Running KMeans clustering")
kmeans = KMeans(init='k-means++', n_clusters=clusters, n_init=10,verbose=1)
kmeans.fit(train)
pickle.dump(kmeans, open("data/cb_%dclusters.p" % clusters, "wb"))
```

OUTPUT:

```
Running KMeans clustering
Initialization complete
Iteration 0, inertia 13076584.0
Iteration 1, inertia 10203468.0
Iteration 2, inertia 9982400.0
Iteration 3, inertia 9894413.0
Iteration 4, inertia 9844023.0
Iteration 5, inertia 9810484.0
Iteration 6, inertia 9787227.0
Iteration 7, inertia 9769511.0
Iteration 8, inertia 9755781.0
Iteration 9, inertia 9744474.0
Iteration 10, inertia 9735468.0
Iteration 11, inertia 9727716.0
Iteration 12, inertia 9720886.0
Iteration 13, inertia 9715418.0
Iteration 14, inertia 9710578.0
Iteration 15, inertia 9706856.0
Iteration 16, inertia 9703824.0
Iteration 17, inertia 9701228.0
Iteration 18, inertia 9699404.0
Iteration 19, inertia 9697814.0
Iteration 20, inertia 9696322.0
Iteration 21, inertia 9694852.0
Iteration 22, inertia 9693425.0
Iteration 23, inertia 9692088.0
Iteration 24, inertia 9691048.0
Iteration 25, inertia 9690198.0
Iteration 26, inertia 9689398.0
Iteration 27, inertia 9688548.0
Iteration 28, inertia 9687635.0
Iteration 29, inertia 9686820.0
Iteration 30, inertia 9686104.0
Iteration 31, inertia 9685425.0
```

STEP - 4 : Making BoW vector for every video frame in the training set, using the computed clusters with TF-IDF weighting scheme.

(run “make_bow_vector.py”)

Code :

```
import numpy as np
import os
import pickle

from scipy.cluster.vq import vq
```

```
def make_bow(dataset, clusters, tfidf):
    print("Make bow vector for each frame")

    n_videos = len(dataset)

    bow = np.zeros((n_videos, clusters.shape[0]), dtype=float)

    video_index = 0
    for video in dataset:
        visual_word_ids = vq(video["features"], clusters)[0]
        for word_id in visual_word_ids:
            bow[video_index, word_id] += 1
        video_index += 1

    if tfidf:
        print("Applying TF-IDF weighting")
        freq = np.sum((bow > 0) * 1, axis = 0)
        idf = np.log((n_videos + 1) / (freq + 1))
        bow = bow * idf

    video_index = 0
    for i in range(len(dataset)):
        dataset[i]["features"] = bow[video_index]
        video_index += 1
        if (i + 1) % 50 == 0:
            print("Processed %d/%d videos" % (i + 1, len(dataset)))
    return dataset

if __name__ == "__main__":
    tfidf=1
    codebook = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
        baseline_optflow_bow_svm/data/cb_600clusters.p", "rb"))
```

```
clusters = codebook.cluster_centers_
print("No of cluster centers : ",len(clusters))
dataset = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
        baseline_optflow_bow_svm/data/train.p", "rb"))
dataset_bow = make_bow(dataset, clusters, tfidf)
print("Length of dataset bow : ",len(dataset_bow))
pickle.dump(dataset_bow, open("data/train_bow_600.p", "wb"))
```

OUTPUT:

```
Make bow vector for each frame
Applying TF-IDF weighting
Processed 50/407 videos
Processed 100/407 videos
Processed 150/407 videos
Processed 200/407 videos
Processed 250/407 videos
Processed 300/407 videos
Processed 350/407 videos
Processed 400/407 videos
Length of dataset bow :  407
```

STEP - 5 : Training the SVM classifier with the BoW vectors

(run "train_svm.py")

Code :

```
import numpy as np
import os
import pickle

from sklearn.svm import SVC

def make_dataset(dataset):
    X = []
    Y = []

    for video in dataset:
```

```
        X.append(video["features"])
        Y.append(video["category"])

    return X, Y

if __name__ == "__main__":

    dataset = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
            baseline_optflow_bow_svm/data/train_bow_600.p", "rb"))
    X, Y = make_dataset(dataset)

    clf = SVC(C=1, kernel="linear", verbose=True)
    clf.fit(X, Y)
    pickle.dump(clf, open("data/svm_C1_c600.p", "wb"))
```

STEP - 6 : Evaluation the SVM classifier against testing set

(run "evaluate.py")

Code:

```
import numpy as np

import os

import pickle

from sklearn.svm import SVC

CATEGORIES = ["boxing", "handclapping", "handwaving", "jogging", "running", "walking"]


if __name__ == "__main__":

        data = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
        baseline_optflow_bow_svm/data/test_bow_c600.p", "rb"))

        clf = pickle.load(open("E:/PROJECT@12A/code/KTH-Action-Recognition-master/
```

```
baseline_optflow_bow_svm/data/svm_C1_c600.p", "rb"))
confusion_matrix = np.zeros((6, 6))
correct = 0
for video in data:
    predicted = clf.predict([video["features"]])
    if predicted == video["category"]:
        correct += 1

    confusion_matrix[CATEGORIES.index(predicted),
        CATEGORIES.index(video["category"])] += 1

print("%d/%d Correct" % (correct, len(data)))
print("Accuracy =", correct / len(data))
print("Confusion matrix")
print(confusion_matrix)
```

OUTPUT:

```
162 Correct
Accuracy = 0.84375
Confusion matrix
[[31.  2.  0.  0.  0.  2.]
 [ 1. 29.  6.  0.  0.  0.]
 [ 0.  0. 25.  0.  0.  0.]
 [ 0.  0.  0. 23.  6.  1.]
 [ 0.  1.  1.  5. 25.  0.]
 [ 0.  0.  0.  4.  1. 29.]]
```

7.RESULT

From the KTH dataset, out of 600 videos, 407 videos are considered for training set and remaining 193 videos for testing. The equation for measuring the accuracy is:

Accuracy = (number of testing videos classified correct)/(total number of testing videos) * 100%____7(1)

For SIFT and optical flow, we have experimented with different numbers of clusters.The major purpose of this section is to determine how many clusters are suited well enough to increase the accuracy using SIFT. The following are the results:

(i) Using SIFT Technique

S.No	'K' Value	Overall Accuracy
1	10	26.17%
2	100	53.92%
3	200	61.25%
4	400	71.20%
5	800	74.34%

Table- 7.1: Overall accuracies obtained using SIFT technique with different number of clusters

(ii) Using Optical Flow Technique

S.No	'K' value	Overall Accuracy
1	10	31.32%
2	100	77.87%
3	200	79.16%
4	400	81.99%
5	800	84.37%

Table-7.2 : Overall accuracies obtained using Optical Flow technique with different number of clusters

Accuracy of each action for 800 clusters :

METHODS	ACTIONS	ACCURACY
SIFT	B0XING	74%
	HAND CLAPPING	96.8%
	HANDWAVING	98.5%
	JOGGING	92.6%
	RUNNING	91%
	WALKING	95.5%
OPTICAL FLOW	B0XING	97.39%
	HAND CLAPPING	94.39%
	HANDWAVING	95.9%
	JOGGING	91.7%
	RUNNING	92.6%
	WALKING	95.8%

Table-7.3 : Comparing accuracies of each action in KTH dataset obtained using SIFT and Optical Flow techniques for 800 clusters

Representing Accuracies for different actions using 800 clusters:

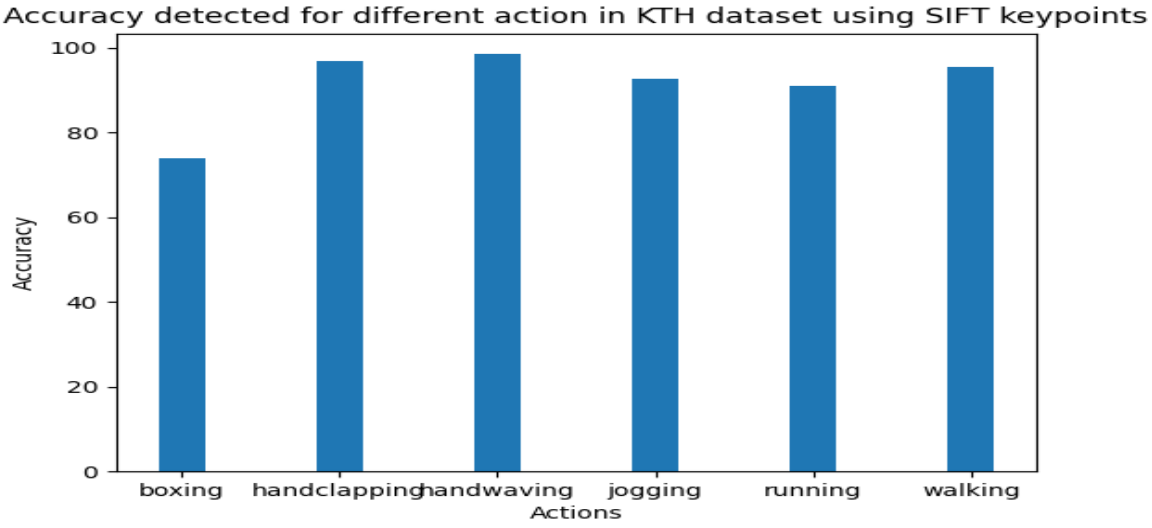


Fig 7.1: Representing accuracies for 6 actions in Kth data set using SIFT technique

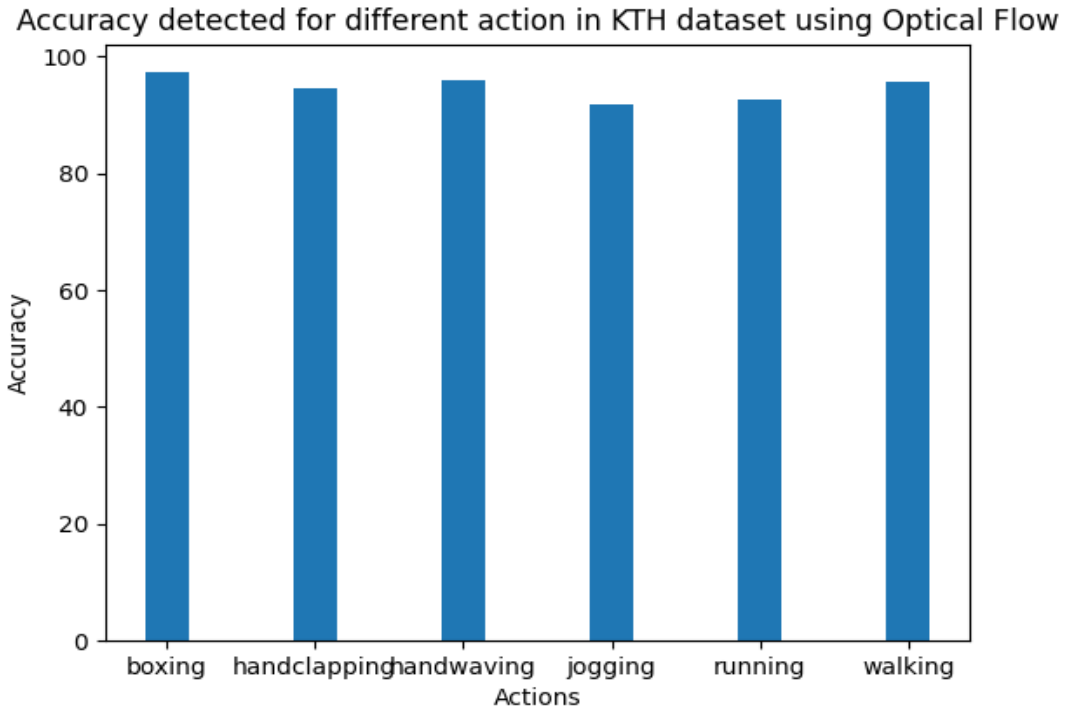


Fig 7.2: Representing accuracies for 6 actions in Kth data set using Optical Flow technique

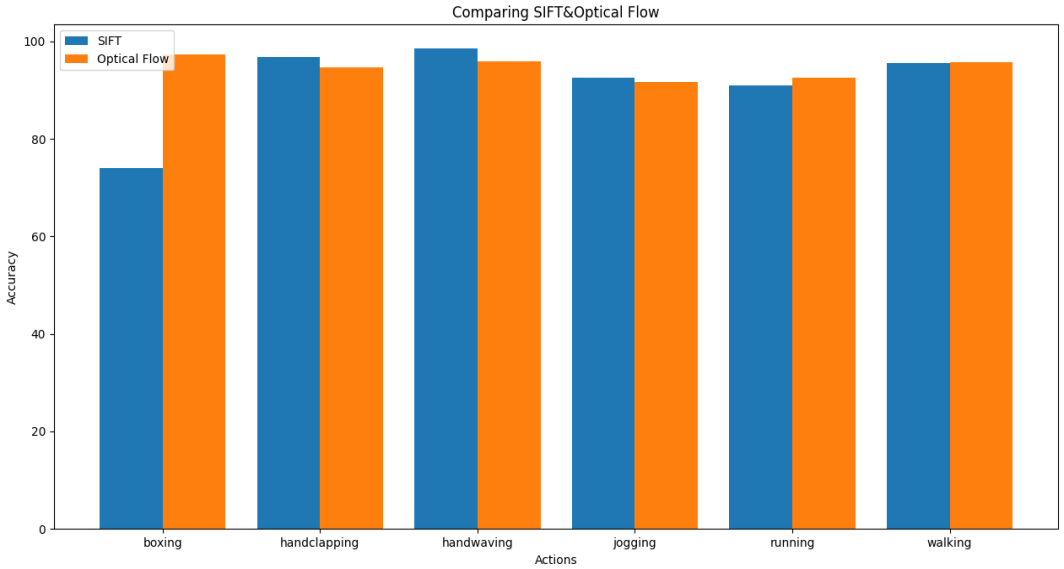


Fig 7.3.Comparing SIFT and Optical Flow Accuracies

8.CONCLUSION AND FUTURE SCOPE

Human Action Recognition proposed in this project provides an effective solution to get an idea of which feature extraction method gives the better accuracy. From the above analysis we can say that using optical flow feature extraction technique gives better results than SIFT feature extraction technique. Based on the above plots, there is much ambiguity in the action “boxing” so using Optical Flow technique the ambiguity is reduced much when compared to SIFT technique.

This scope of study will expand in future to recognize more human actions which can be used for video surveillance, to predict any malicious activities are being done.

9.REFERENCES

[1] Schuldt, C.; Laptev, I.; Caputo, B. (2004). [IEEE Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. - Cambridge, UK (2004.08.26-2004.08.26)] Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. - Recognizing human actions: a local SVM approach. , (), 32–36 Vol.3. doi:10.1109/ICPR.2004.1334462

[2] Yu Kong, Member, IEEE, and Yun Fu, Senior Member,IEEE, " Human Action Recognition and Prediction : A Survey" Journal of Latex class files, Vol-13,No - 9 , September 2018

[3] Qazi, Hassaan Ali; Jahangir, Umar; Yousuf, Bilal M; Noor, Aqib (2017). [IEEE 2017 International Conference on Information and Communication Technologies (ICICT) - Karachi, Pakistan (2017.12.30-2017.12.31)] 2017 International Conference on Information and Communication Technologies (ICICT) - Human action recognition using SIFT and HOG method. , (), 6–10. doi:10.1109/ICICT.2017.8320156

[4] Li, Qilong; Wang, Xiaohong (2018). [IEEE 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS) - Singapore (2018.6.6-2018.6.8)] 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS) - Image Classification Based on SIFT and SVM. , (), 762–765. doi:10.1109/ICIS.2018.8466432

[5] Zhang, Jia-Tao; Tsoi, Ah-Chung; Lo, Sio-Long (2014). [IEEE 2014 International Joint Conference on Neural Networks (IJCNN) - Beijing, China (2014.7.6-2014.7.11)] 2014 International Joint Conference on Neural Networks (IJCNN) - Scale Invariant Feature Transform Flow trajectory approach with applications to human action recognition. , (), 1197–1204. doi:10.1109/ijcnn.2014.6889596

[6] Danafar, S., & Gheissari, N. (n.d.). Action Recognition for Surveillance Applications Using Optic Flow and SVM. Lecture Notes in Computer Science, 457–466. doi:10.1007/978-3-540-76390-1_45

[7] Nagabhushan, T.N.; Aradhya, V. N. Manjunath; Jagadeesh, Prabhudev; Shukla, Seema; M.L., Chayadevi (2018). [Communications in Computer and Information Science] Cognitive Computing and Information Processing Volume 801 || Human Action Detection and Recognition Using SIFT and SVM. , 10.1007/978-981-10-9059-2(Chapter 42), 475–491. doi:10.1007/978-981-10-9059-2_42

[8] Zhang, Jia-Tao; Tsoi, Ah-Chung; Lo, Sio-Long (2014). [IEEE 2014 International Joint Conference on Neural Networks (IJCNN) - Beijing, China (2014.7.6-2014.7.11)] 2014 International Joint Conference on Neural Networks (IJCNN) - Scale Invariant Feature Transform Flow trajectory approach with applications to human action recognition. , (), 1197–1204. doi:10.1109/ijcnn.2014.6889596

[9] Kumar, S. Santhosh; John, Mala (2016). [IEEE 2016 International Carnahan Conference on Security Technology (ICCST) - Orlando, FL, USA (2016.10.24-2016.10.27)] 2016 IEEE International Carnahan Conference on Security Technology (ICCST) - Human activity recognition using optical flow based feature set. , (), 1–5. doi:10.1109/CCST.2016.7815694

[10] Manosha Chathuramali, K. G.; Rodrigo, Ranga (2012). [IEEE 2012 International Conference on Advances in ICT for Emerging Regions (ICTer) - Colombo, Western, Sri Lanka (2012.12.12-2012.12.15)] International Conference on Advances in ICT for Emerging Regions (ICTer2012) - Faster human activity recognition with SVM. , (), 197–203. doi:10.1109/ICTer.2012.6421415

[11] Das Dawn, Debapratim; Shaikh, Soharab Hossain (2015). A comprehensive survey of human action recognition with spatio-temporal interest point (STIP) detector. The Visual Computer, (), –. doi:10.1007/s00371-015-1066-2