Hikmet Orhun Guley S00442

Assignment 3 Report

Introduction

In I tried to classify the action in a given video. The dataset is from the following website (http://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html). The same dataset is used in for the well known paper “Actions as Space-Time Shapes” [1]. On this dataset, I applied HOOF [1] (Histogram of Oriented Optical Flow), PCA (Principal Component Analysis) and k-Nearest Neighbors (k-NN) for classification.

Dataset

The dataset is consist of 93 low-resolution (180 x 144, deinterlaced 50 fps) video sequences showing nine different people, each performing 10 natural actions such as “run,” “walk,” “skip,” “jumping-jack” (or shortly “jack”), “jump-forward-on-two-legs” (or “jump”), “jump-in-place-on-two-legs” (or “pjump”), “gallopsideways” (or “side”), “wave-two-hands” (or “wave2”), “waveone- hand” (or “wave1”), or “bend.”

Implementation

Implementation of this project basically follows the steps in the project description document, which are;

• create temporal mean HOOF for each training sample,
• apply **PCA** to **HOOF** and reserve 90% of the total variance,
• create temporal **mean HOOF** and apply **PCA** transform on training samples,
• **classify** via k-NN. Your goal is to achieve the best result by choosing **k** and **number bins** for HOOF.

1) Dense Optical Flow

Since it is easier to apply and close performance wise I used Dense Optical flow method instead of Lucas-Kanade method. It computes the optical flow for all the points in the frame. It is based on Gunner Farneback's algorithm which is explained in "Two-Frame Motion Estimation Based on Polynomial Expansion" by Gunner Farneback in 2003. I used OpenCV built-in functions for this calculation [2].

2) Histogram of Oriented Optical Flow (HOOF)

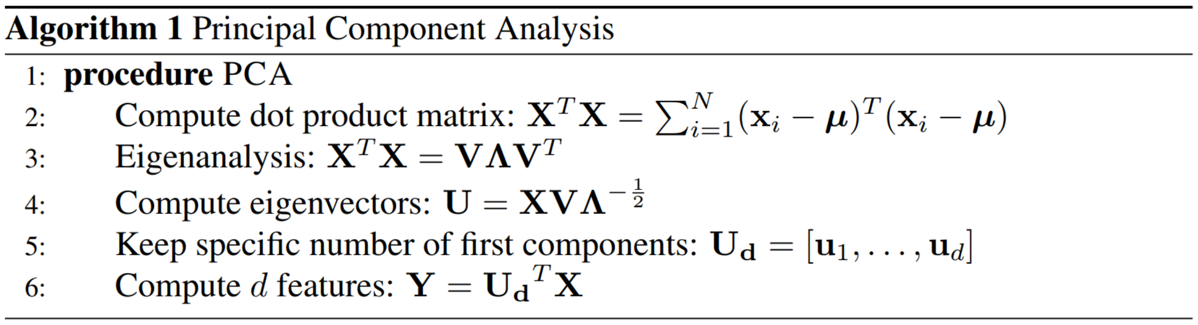
Histogram of Oriented Optical Flow (HOOF) is calculated as follows

```
Algorithm HOOF
1: Optical flow is computed at every frame of the video
2: Each flow vector is binned according to its primary angle from the horizontal axis and weighted according to its magnitude. Thus, all optical flow vectors, v = $[x, y]^T$ with direction, θ = $tan^{-1}(y/x)$ in the range,
   $-\frac{\pi}{2} + \pi\frac{b-1}{B} \leq \theta < -\frac{\pi}{2} + \pi\frac{b}{B}$
   will contribute by $\sqrt{x^2 + y^2}$ to the sum in bin $b$, $1 \leq b \leq B$ out of a total of $B$ bins.
3: Normalize HOOF to sum up to 1.
4: Since the videos of different number of frames, calculate the temporal mean of the HOOF's calculated as follows,
   $\bar{h} = \frac{1}{T}\sum_{t}^{T} h_t$
   $\bar{h}$ is the temporal mean HOOF for a video in dataset.
5: Create a matrix of calculated by temporal means,
   $\mathbf{h}_t = [\mathbf{h}_{t;1}, \mathbf{h}_{t;2}, \ldots, \mathbf{h}_{t;B}]^\top$
```

3) Principal Component Analysis (PCA)

Principal Component analysis is a powerful and widely used technique the reduce the dimensions of high dimensional data.

The algorithm works as follows,



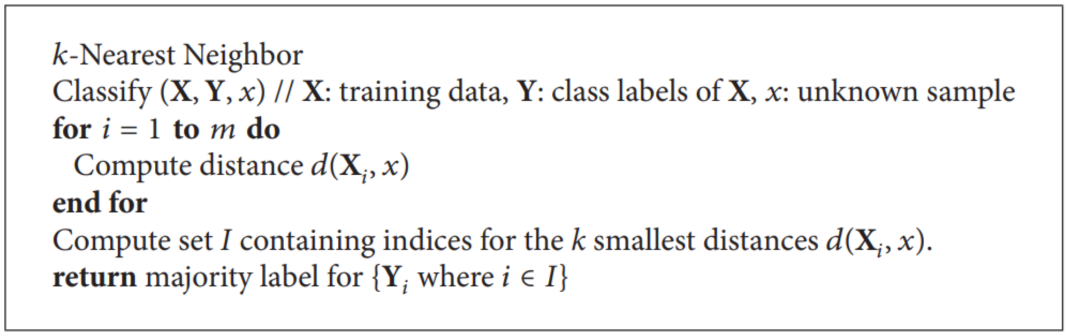
```
Algorithm 1 Principal Component Analysis
1: procedure PCA
2:     Compute dot product matrix: $\mathbf{X}^T\mathbf{X} = \sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})^T(\mathbf{x}_i - \boldsymbol{\mu})$
3:     Eigenanalysis: $\mathbf{X}^T\mathbf{X} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$
4:     Compute eigenvectors: $\mathbf{U} = \mathbf{X}\mathbf{V}\boldsymbol{\Lambda}^{-\frac{1}{2}}$
5:     Keep specific number of first components: $\mathbf{U}_\mathbf{d} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$
6:     Compute $d$ features: $\mathbf{Y} = \mathbf{U}_\mathbf{d}{}^T\mathbf{X}$
```

Screenshot taken from (https://ibug.doc.ic.ac.uk/media/uploads/documents/notes_implementation_component_analysis.pdf)

By using PCA algorithm I apply PCA to temporal mean HOOF's and reserve the 90% of the variance. As an example, for the case with 30 bins, the PCA algorithm is able to reduce to dimensionality to 4 while reserving the 90% of the variance.

4) k-Nearest Neighbour (kNN)

K-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. The pseudocode of the algorithm is as follows,



```
k-Nearest Neighbor
Classify ($\mathbf{X}, \mathbf{Y}, x$) // $\mathbf{X}$: training data, $\mathbf{Y}$: class labels of $\mathbf{X}$, $x$: unknown sample
for $i = 1$ to $m$ do
  Compute distance $d(\mathbf{X}_i, x)$
end for
Compute set $I$ containing indices for the $k$ smallest distances $d(\mathbf{X}_i, x)$.
return majority label for $\{\mathbf{Y}_i$ where $i \in I\}$
```

Screenshot taken from the paper "A Machine Learning Approach for Specification of Spinal Cord Injuries Using Fractional Anisotropy Values Obtained from Diffusion Tensor Images"

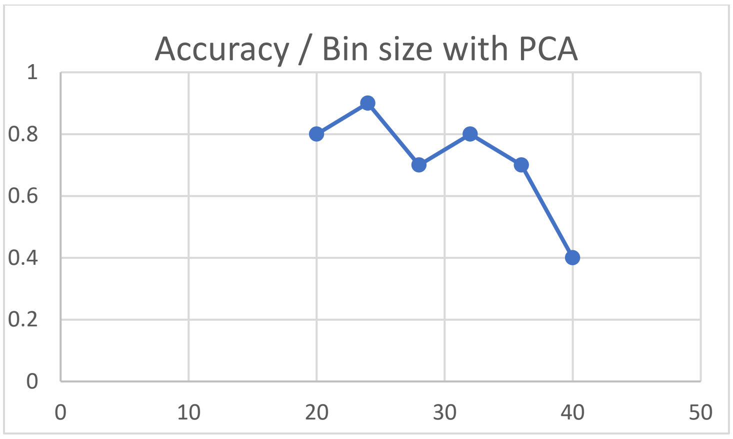
In the assignment, I used the kNN algorithm to classify the videos in the dataset. The classes are as follows,

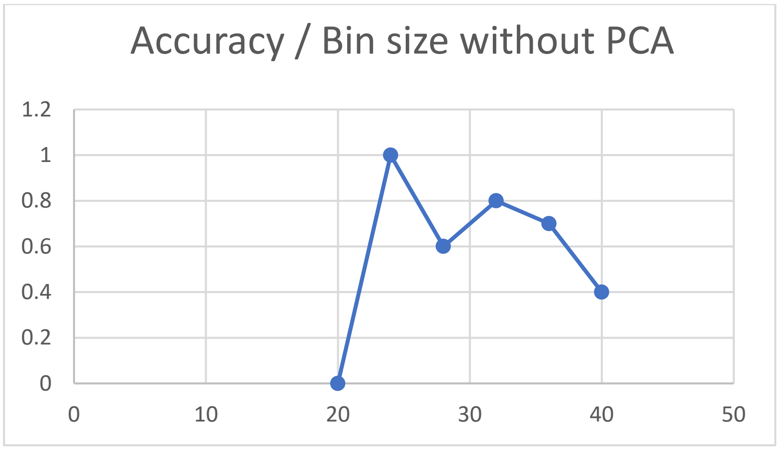
- bend
- jack
- jump
- pjump
- run
- side
- skip
- walk
- wave1
- wave2

I used the built-in kNN function to classify videos in the dataset. In the function, I used brute force k finder to find optimal k.

Computational Results

Below you can see the graps comparing the bin size with the accuricies. The first graph implies the relation between PCA applied data and bin size. The second graph implies the relation between the data without PCA and bin size.





Since I don't have enough time to construct this graphs in matplotlib, I looked at the data manually and construct the graph manually in MS Excel.

Conclusion

As conclusion, this project was very intuitive for me in terms of understanding the algorithms and techniques such as optical flow, HOOF, PCA and kNN. Gaining knowledge about histograms and applying machine learning techniques to classify videos was a good experience for me

References

[1] Chaudhry, Rizwan & Ravichandran, Avinash & Hager, Gregory & Vidal, René. (2009). Histograms of oriented optical flow and Binet-Cauchy kernels on nonlinear dynamical systems for the recognition of human actions. 2012 IEEE Conference on Computer Vision and Pattern Recognition. 1932-1939.10.1109/CVPRW.2009.5206821.

[2] https://docs.opencv.org/3.4/d7/d8b/tutorial_py_lucas_kanade.html

[3]https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html

[4] https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/