Real-time Action Recognition
Based on Human Skeleton in Video

Feiyu Chen

2019 Mar 20th

Final Project of EECS-433 Pattern Recognition

Teacher: Prof. Ying Wu

Contents

1. Introduction

1.1 Overview of the project

In this project, I implemented an action recognition system which can recognize 9 types of human actions in video. A demo is shown in **Fig. 1.1**, where the action "Wave" is correctly recognized. More video demonstrations and software are on my Github[1].

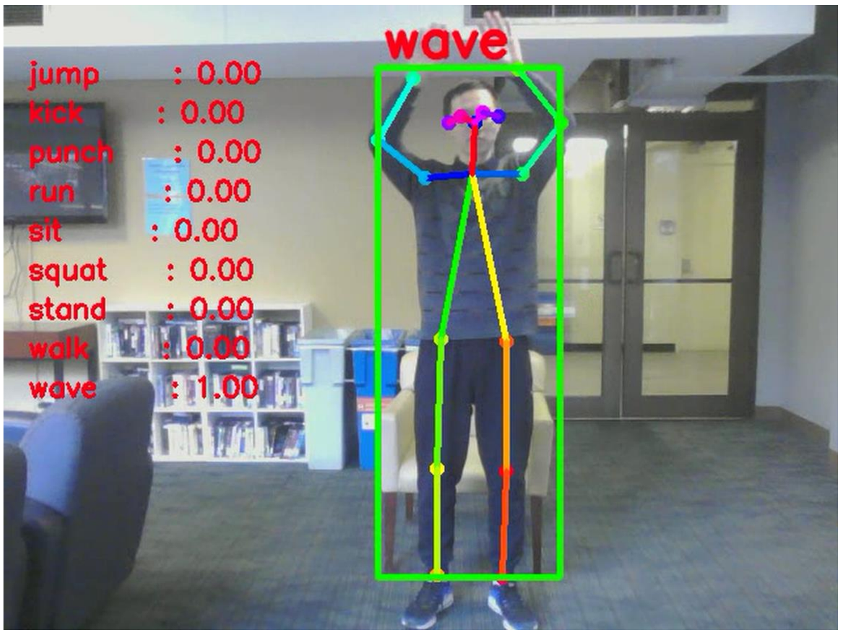


Fig. 1.1. Recognizing human actions from video.

1.2 Algorithm for action recognition

In this project, the recognition of human action is based on the human skeleton data. I used an open-source algorithm, OpenPose[2], to detect human skeleton (joint positions) from each video frame, and then utilized the skeleton as raw data to extract features and make classification by using machine learning algorithms. Details of the algorithm are described in Chapter 3 Algorithm.

There are other methods for action recognition, such as using 3D Convolution Neural Network[3] to directly recognize actions from video. However, it is time consuming and difficult to train the large neural network, and also lacks the interpretability. On the contrary, the features of human skeleton are concise, intuitive, and easy for differentiating different human actions. Thus, I chose the human skeleton as the base features to complete my action recognition project.

[1] https://github.com/felixchenfy/Realtime-Action-Recognition

[2] 2017 CVPR: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields

[3] 2017 CVPR: Quo vadis, action recognition? a new model and the kinetics dataset

1.3 Applications of action recognition

The technology of human action recognition based on video data has diverse applications. For example, it can be used for interactive games where a person's movement is recognized by the computer and used as the input for playing games. Another important application of action recognition is monitoring health for the elderly and kid who stay alone at home. The action recognition can detect dangerous situations (such as sudden fainting, falling down, or other injuries of the elderly/kid) and report the emergency to the family or hospital in time. Besides, there are also other applications such as security surveillance and video classification.

2. Training Data

I collected videos of 9 types of actions, including (1) Wave, (2) Stand, (3) Punch, (4) Kick, (5) Squat, (6) Sit, (7) Walk, (8) Run, and (9) Jump, as shown in **Fig. 2.1**.

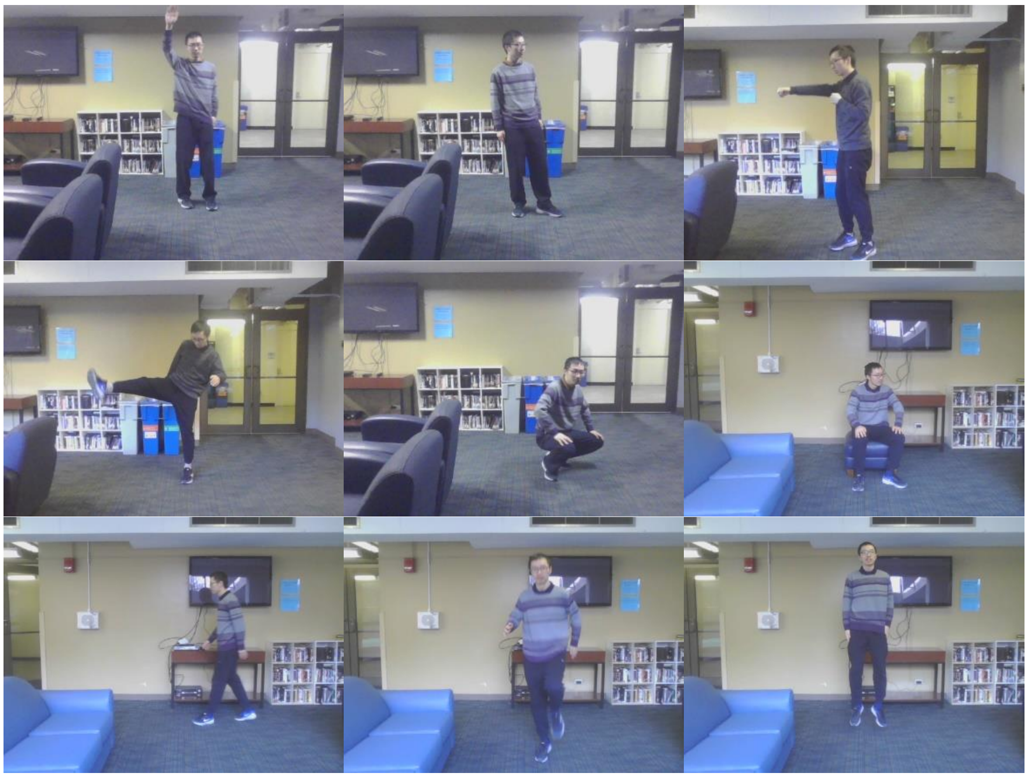


Fig. 2.1 Snapshots of 9 types of actions in the training data, including: Wave, Stand, Punch, Kick, Squat, Sit, Walk, Run, and Jump.

The length of each video ranges from 0.8 seconds to 2 minutes, and each video only contains one type of action. For example, in one video, I did a kick in 0.8 seconds; In another video, I kept waving my arms for 2 minutes.

These videos were recorded with a size of 640x480 and a framerate of 10 frames/second, so that it's fast enough for capturing the whole movement of an action. The total number of frames of each action is shown in table 2.1.

Table 2.1. Number of frames of the 9 actions as training data.

Actions	1	2	3	4	5	6	7	8	9	Total
	wave	stand	punch	kick	squat	sit	walk	run	jump	
Number of frames	1239	1703	799	1162	964	1908	1220	1033	1174	11202

3. Algorithm

3.1 Overview

The overall workflow of the action recognition algorithm is shown in **Fig. 3.1**.

The input to the system is a video stream, either coming from a camera or a video file. Then the OpenPose algorithm is adopted to detect the human skeleton (joint positions) from each frame. Next, a sliding window of size N aggregates the skeleton data of the first N frames. These skeleton data are preprocessed and used for feature extraction, which are then fed into a classifier to obtain the final recognition result (of this window). Similarly, to achieve a real-time recognition framework, the window is slided frame by frame along the time dimension of the video, and outputs a label for each video frame. Here the window size N is set as 5 after experiment, which equals to a length of 0.5 seconds.

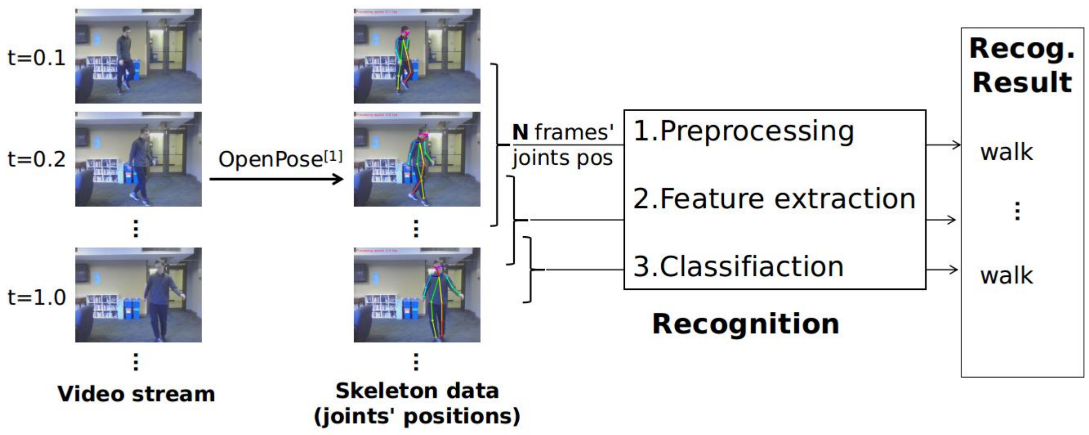
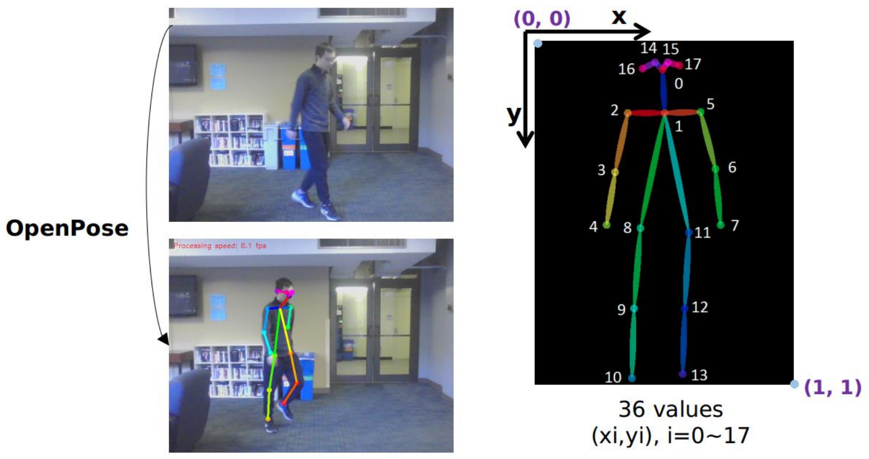


Fig. 3.1. Workflow of the human action recognition system based on skeleton data.

3.2 Detecting human skeleton from image

The OpenPose[4] algorithm is adopted to detect human skeleton from the image. The key idea of OpenPose is using Convolutional Neural Network to produce two heapmaps, one for predicting joint positions, and the other for associating the joints into human skeletons. In short, the input to OpenPose is an image, and the output are the skeletons of all the humans this algorithm detects. Each skeleton has 18 joints, including head, neck, arms and legs, as shown in **Fig. 3.2**. Each joint position is represented in the image coordinate with coordinate values of x and y, so there is a total of 36 values of each skeleton.



(a) (b)

Fig. 3.2. (a) Detecting human skeleton by OpenPose. (b) Each skeleton has 18 joints.

3.3 Preprocessing features

The raw skeleton data are preprocessed before extracting features. The preprocessing includes 4 steps as summarized in **Fig. 3.3**, and are illustrated as follows:

(1) Scale the coordinate

The original joint's position outputted by OpenPose has a different unit for x coordinate and y coordinate. I scaled them to be the same unit to deal with the images with different height/width ratios.

(2) Remove all joints on head

OpenPose outputs 5 joints on head, including 1 Head, 2 Eyes, and 2 Ears. However, for the actions of my training set, the head's position helps little for the classification. What matters is the configuration of the body and limbs. Thus, I manually removed the five joints on head to make the features more meaningful.

[4] The Github of OpenPose is: https://github.com/CMU-Perceptual-Computing-Lab/openpose

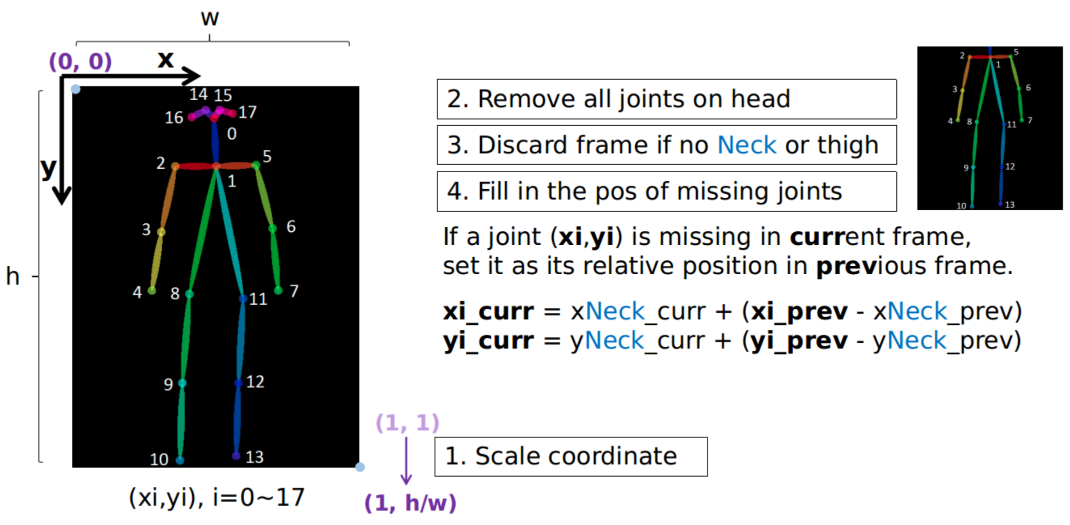

w
(0, 0)
x
y
h
14 15
16
17
0
2
5
1
3
6
4
8
11
7
9
12
10
13
(xi,yi), i=0~17
(1, 1)
(1, h/w)
1. Scale coordinate
2. Remove all joints on head
3. Discard frame if no Neck or thigh
4. Fill in the pos of missing joints
If a joint (xi,yi) is missing in current frame,
set it as its relative position in previous frame.
xi_curr = xNeck_curr + (xi_prev - xNeck_prev)
yi_curr = yNeck_curr + (yi_prev - yNeck_prev)

Fig. 3.3. Preprocessing joint positions in 4 steps.

(3) Discard frames who have no Neck or Thigh

If in a frame there is no human skeleton detected by OpenPose, or the detected skeleton has no Neck or no Thigh, then this frame is considered as invalid and is discarded. Besides, the sliding window should re-initialize on the next frame.

(4) Fill in the missing joints

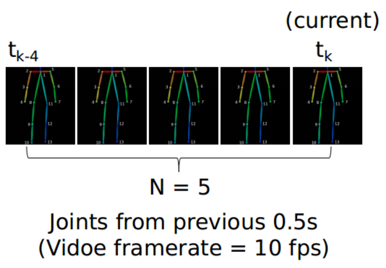
In some cases, OpenPose might fail to detect a complete human skeleton from the image, causing some blanks in the joint positions. These joints must be filled with some values in order to maintain a fixed-size feature vector for the following feature classification procedure.

Two bad solutions are: (1) Discard this frame. However, in such way, the algorithm would never be able to detect the action when the person is standing sideways and not facing the camera. (2) Fill in the positions with some value outside a reasonable range. Theoretically, when the classifier is strong enough, this method could work. However, my experiment results gave really bad recognition accuracy. Thus, I didn't adopt this method.

The solution I used is to fill in a joint's position based on its relative position in the previous frame with respect to the neck. Suppose in last frame, the hand is 10 pixels on the right side of the neck. Then in this frame, if the hand is missing, I will set it as 10 pixels on the right side of the neck of the current frame. The experimental result shows that this method works well.

3.4 Feature Extraction

After the previous preprocessing step, the joints' positions are complete and good to use. In this section, I will use the joint positions from N=5 frames as raw features, and then manually design and extract more salient features that might be useful for distinguishing the action types. A summary of the computed features is shown in **Fig. 3.4**.



Possible Selections of Features

Symbol	Meaning
Xs	Concatinatation of joints' pos of N frames
H	in Xs: Average skeleton height
V_body	in Xs: (Velocity of neck) / H
X	Normalized pos: X = (Xs - mean(Xs)) / H
V_joints	in X: Velocity of all joints, $\{X[t_k]-X[t_{k-1}]\}$
JointAngles	in X: convert pos to joint angles
LimbLens	in X: convert pos to length of limbs

Fig. 3.4. A list of manually designed features extracted from the raw joint positions.

I computed the following features:

(1) Xs: A direct concatenation of joints positions of the N frames.
Dimension = 13 joints * 2 pos/joint * N = 130, where N = 5.

(2) H: Average height of the skeleton of the previous N frames. This height equals the length from Neck to Thigh. It's used for normalizing all features described below.
Dimension = 1

(3) V_body: Velocity of the neck divided by H.
Dimension = N -1 = 4

(4) X: Normalized joint positions, which equals [Xs – mean(Xs)]/H
Dimension = 130, same as Xs

(5) V_joints: Velocities of the joints, $\{X[t_k]-X[t_{k-1}]\}$, computed in the normalized coordinate.
Dimension = 13 joints * 2 vels/joint * (N-1) = 104

(6) Joint angles computed from joint positions.

(7) Length of each limb computed from joint positions.

After experiment, I chose the normalized joint positions (**X**), moving velocity of the body (**V_bod**y, duplicated for 10 times to increase weight), and joint velocities (**V_joints**) as the features. These features are concatenated to form a feature vector of dimension 314. Then, I adopted the PCA algorithm to reduce the feature vector to 50 dimensions. The sum of the eigen values of the new coordinates accounts for 97% of the total sum. The diagram of the resultant features is shown in **Fig. 3.5** below.

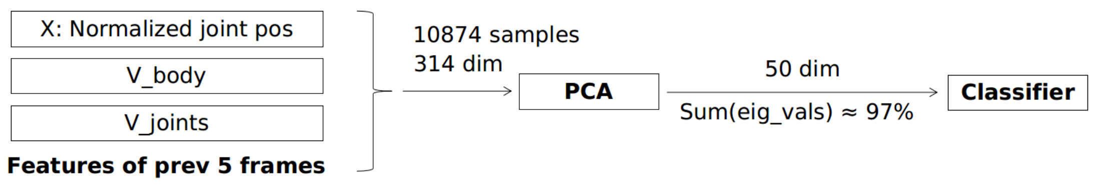


Fig. 3.5. Overview of the feature extraction step: Three features are selected for classification, including normalized joint positions, body velocity, and joints velocities. Besides, PCA is applied for feature reduction.

To sum up, the resultant feature vector has a dimension of 50. The number of samples are 10874, which is smaller than the original 11202 frames due to the setting of sliding window and missing data. These features are ready for classification.

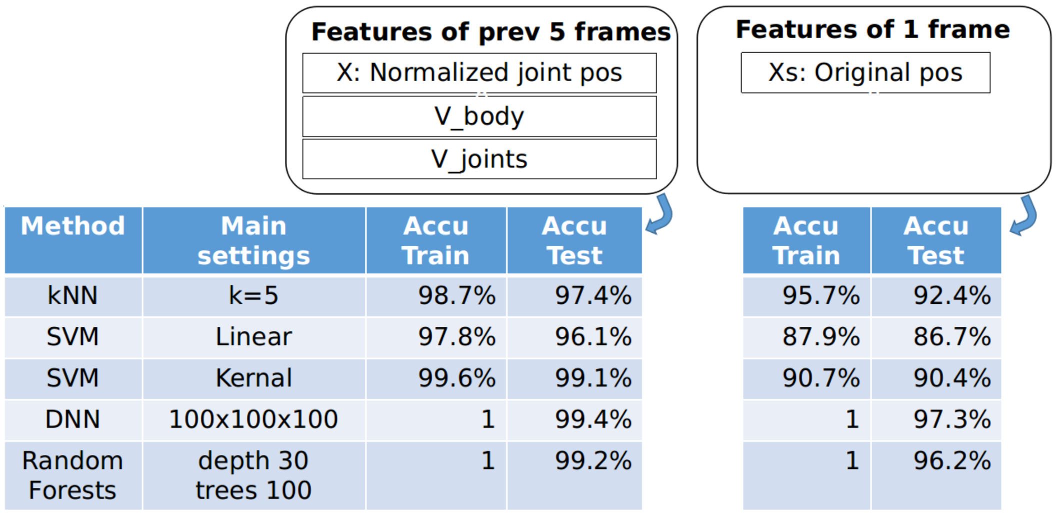
3.5 Classification

The total training data are split into two sets: 70% for training, and 30% for testing. Five different classifiers are experimented, including kNN, SVM, SVM with kernel method, Deep Neural Network, and Random Forests. The implementation of these methods is from the Python library "sklearn". I tuned the major parameters of each method in order to obtain a better result, which is discussed in next chapter.

4. Result

4.1 Recognition accuracy

The final recognition (classification) accuracy is shown in **Fig. 4.1**(a), where the features are extracted as described in last section. The result shows that the accuracy of all 5 models are higher than 96%. Furthermore, three of the models achieved an accuracy of more than 99%, including SVM (kernel method), DNN (3 layers, 100x100x100), and Random Forests (100 trees with depth 30).



Features of prev 5 frames
X: Normalized joint pos
V_body
V_joints
Features of 1 frame
Xs: Original pos

Method	Main settings	Accu Train	Accu Test
kNN	k=5	98.7%	97.4%
SVM	Linear	97.8%	96.1%
SVM	Kernal	99.6%	99.1%
DNN	100x100x100	1	99.4%
Random Forests	depth 30 trees 100	1	99.2%

Accu Train	Accu Test
95.7%	92.4%
87.9%	86.7%
90.7%	90.4%
1	97.3%
1	96.2%

(a) (b)

Fig. 4.1. Recognition accuracy of five models, when using (a) selected features from five frames, or (b) raw features from single frame.

To evaluate whether the selected features improves the accuracy or not, a comparison experiment was conducted. This time, the window size is set as 1 frame, and only the unnormalized joint positions are used as features. The result in **Fig. 4.1**(b) shows that accuracies of all the methods drop for some several percentages. Besides, there is some overfitting for DNN and Random Forests even after finetuning the parameters. This result indicates that the selection of features does help to improve the recognition accuracy.

4.2 Speed of algorithms

The speed of the algorithms was tested on a laptop with an Intel Core i7 CPU and a GTX-1070 GPU. The OpenPose runs at about 7 fps when the image is resized to 432x368. The time cost for feature extraction and classification is less than 0.01s per frame for all classifiers, since the models are all relative shallow.

With some future optimization of the program and using better hardware, this action recognition system should be able to run in real time with a framerate of 10 fps.

4.3 Performance on video of mine

In this section, I tested the performance of the proposed action recognition system on a real video. The actions in this video are similar to the training set, because both actions were performed by me, and the action types are the same as the training set.

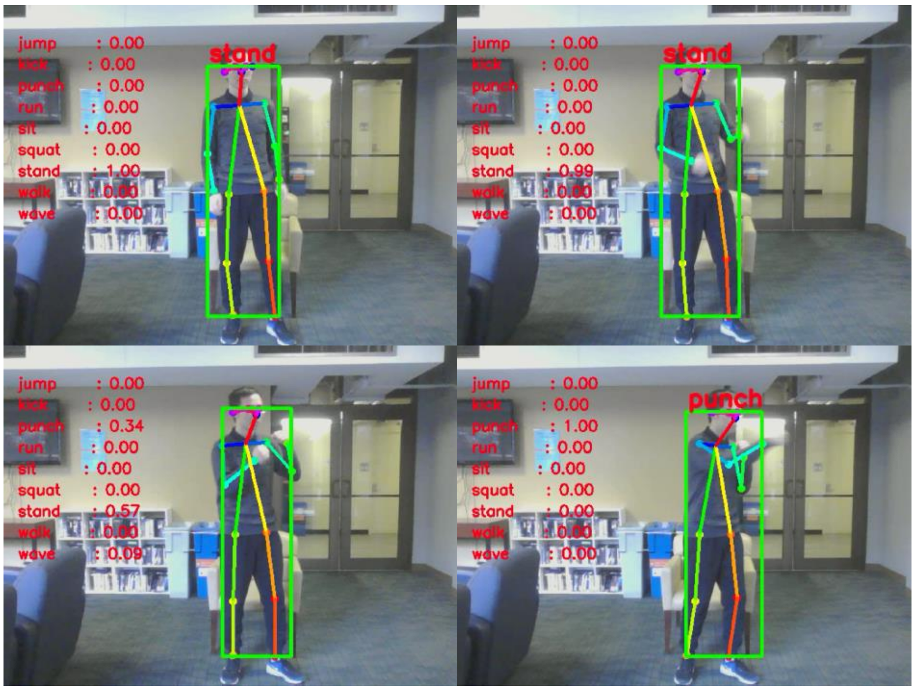


Fig. 4.2. Performance of action recognition on a video where I performed the actions with the same types as the training set. The algorithm successfully detects the action of Punch.

The classifier I chose is DNN. I then added a filtering procedure by averaging the prediction scores of the adjacent two frames, as shown in the red numbers in the left column of the images in Fig. 4.2. Only when the score is higher than 0.85 will the predicted label be shown above the person. Besides, the currently algorithm only tracks one person and recognizes his/her actions.

The action recognition performance on the video performed by myself is pretty accurate and stable. A sequence of snapshots is shown in **Fig. 4.2**. As I changed my action from Stand to Punch, the prediction scores also varied accordingly.

4.4 Performance on videos from NWU/UCLA dataset

I then did a test on two videos from Northwestern-UCLA Multiview Action 3D Dataset[5].

For one video, the action of walk was correctly recognized, as shown in **Fig. 4.3**(a). However, for the other, as shown in **Fig. 4.3**(b), the action of Sit was wrongly recognized as Kick. One possible reason is that the action of Kick and Sit have a similar configuration of skeleton, because for both actions the leg is bending in front of the body.

The wrong result reveals some limitations of the current action recognition system. In future work, more training data should be collected. Besides, a better model is required to extract more import features for differentiating different actions.

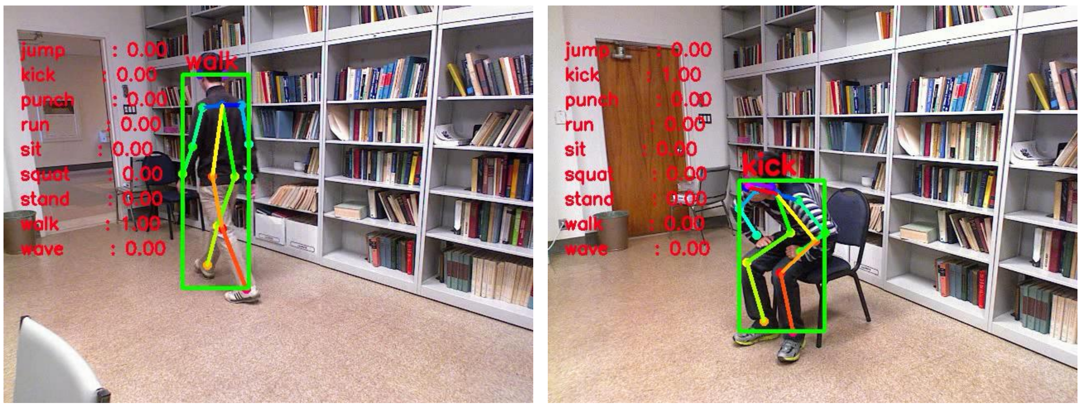


Fig. 4.3. Action recognition result on two public videos.

4.5 Performance on video of dancing

I also tested the action recognition result on a video of dancing. Four snapshots and recognition results are shown in **Fig. 4.4**. The recognized actions are Kick, Punch, Wave, and Jump respectively. They are all kind of reasonable and fits the common sense.

[5] http://users.eecs.northwestern.edu/~jwa368/my_data.html

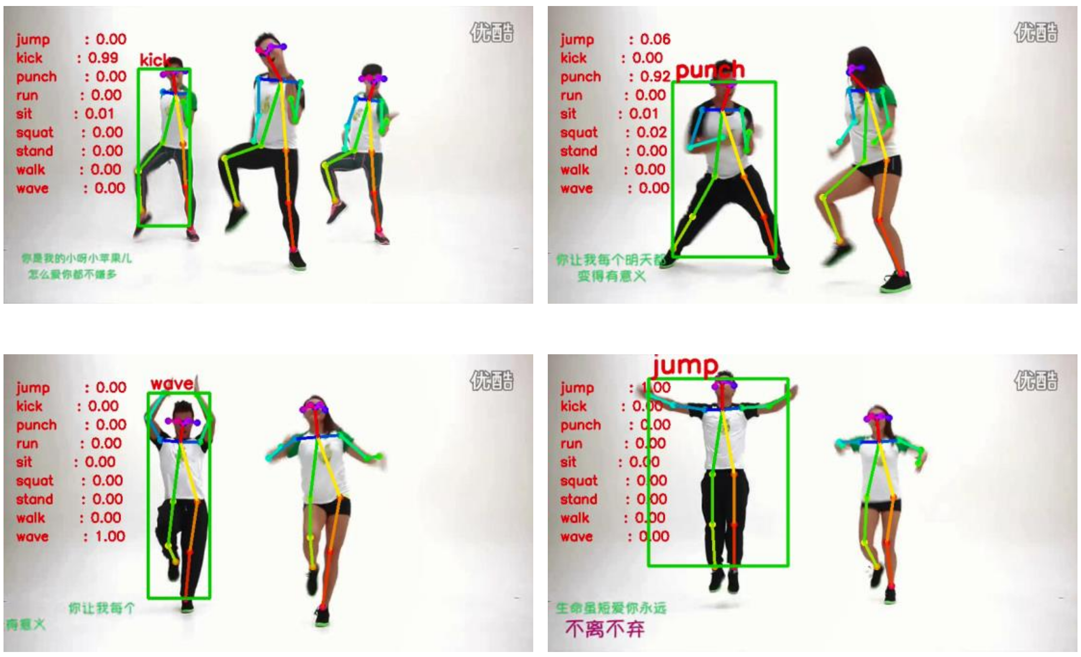


Fig. 4.4. Action recogniton result on a video of dancing.

5. Conclusion

In this project, I implemented a human action recognition system that can recognize 9 types of actions. The algorithm is based on a real-time framework by aggregating the skeleton data of a 0.5s window for feature extraction and classification. The recognition accuracy was up to 99% on the training set composed of more than 10000 samples. This action recognition system was then tested on real world videos: It achieved stable and accurate recognition performance on a video similar to the training set performed by me, and achieved relatively good result on other videos.