Shaohua Yuan shyuan@tamu.edu 629009213

Jump Detection Project Report- First Submission

March 2020

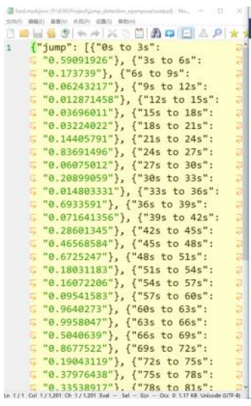
1. Topic

a) Introduction:

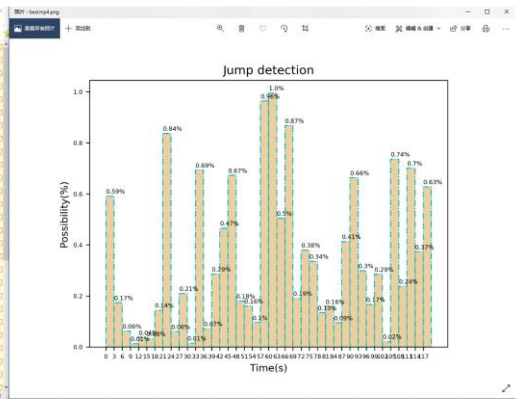
In this project, I will build a neural network project that can detect whether and when the human action of jump presents in videos. This project can be used in smart home applications for detecting human action of jump.

b) Solution:

- For a video, cut it into small clips whose time-length are fixed to 3 seconds.
- Use the OpenPose project to generate landmark json files for each clip.
- Let one clip and its landmark files be one input tensor of the neural network, the output will be the possibility that jump is detected in this clip.
- For a video, input all its clips to the neural network in time order, and get the outputs in time order.
- Collect the outputs and generate a figure and json file, statistically showing jump detection in the video. As following:



```
{"jump": [{"0s to 3s":
 "0.59091926"}, {"3s to 6s":
 "0.173739"}, {"6s to 9s":
 "0.06243217"}, {"9s to 12s":
 "0.012871458"}, {"12s to 15s":
 "0.03696011"}, {"15s to 18s":
 "0.03224022"}, {"18s to 21s":
 "0.14405791"}, {"21s to 24s":
 "0.83691496"}, {"24s to 27s":
 "0.06075012"}, {"27s to 30s":
 "0.20899059"}, {"30s to 33s":
 "0.014803331"}, {"33s to 36s":
 "0.6933591"}, {"36s to 39s":
 "0.071641356"}, {"39s to 42s":
 "0.28601345"}, {"42s to 45s":
 "0.46568584"}, {"45s to 48s":
 "0.6725247"}, {"48s to 51s":
 "0.18031183"}, {"51s to 54s":
 "0.16072206"}, {"54s to 57s":
 "0.09541583"}, {"57s to 60s":
 "0.9640273"}, {"60s to 63s":
 "0.9958047"}, {"63s to 66s":
 "0.5040639"}, {"66s to 69s":
 "0.8677522"}, {"69s to 72s":
 "0.19043119"}, {"72s to 75s":
 "0.37976438"}, {"75s to 78s":
 "0.33538917"}, {"78s to 81s":
```



2. Dataset

a) Raw dataset: video

I use part videos from HMDB51, Weizmann, UCF-101, STAIR-actions. These are all datasets including different human actions (jump, run, walk, sit down, exercise, etc.). I collect around 900 videos from these datasets. Half of their topics are jump, another half videos' topics are others (run, walk, sit down, exercise, stand up).

Besides, I also shoot myself jumping or not to 6 videos, each of them has 2 mins time-length.

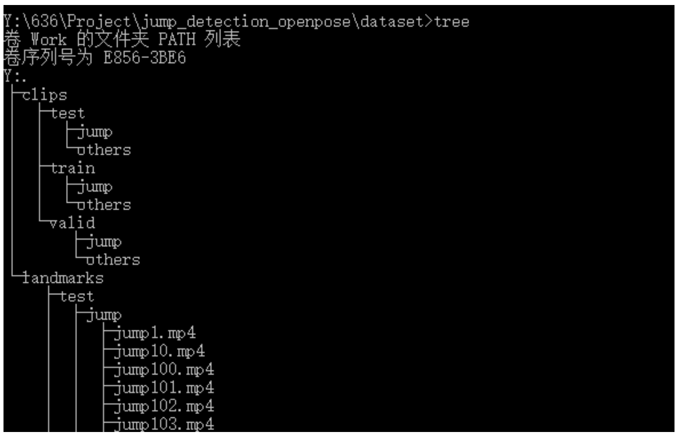
b) Process data: 3-seconds clips and landmarks

- Clips: Use ffmpeg and Python script to split all videos into 3-seconds clips.
- Landmarks: Use OpenPose alongside with Google Colab & Google Drive to generate landmark json files for each clip. Use the following command in Colab:

```
!cd openpose && ./build/examples/openpose/openpose.bin --video 'root_of_video' --write_json 'directory_of_output' --display 0 --render_pose 0 -keypoint_scale 3
```

```
!cd openpose && ./build/examples/openpose/openpose.bin --video sample/bed.mp4 --write_json sample/bed/ --display 0 --render_pose 0 -keypoint_scale 3
```

c) Dataset Directory Structure:



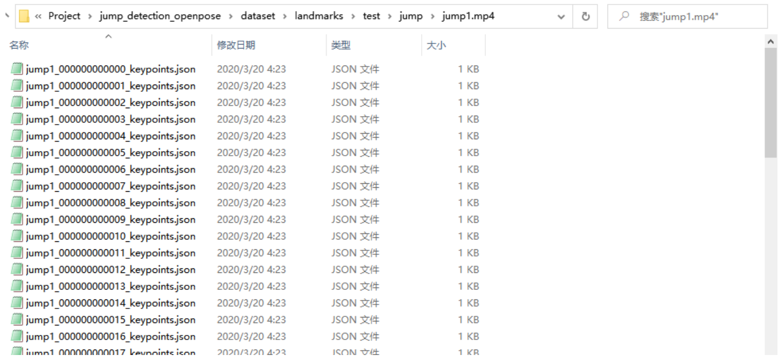
```
Y:\636\Project\jump_detection_openpose\dataset>tree
卷 Work 的文件夹 PATH 列表
卷序列号为 E856-3BE6
Y:.
├─clips
│  ├─test
│  │  ├─jump
│  │  └─others
│  ├─train
│  │  ├─jump
│  │  └─others
│  └─valid
│      ├─jump
│      └─others
└─landmarks
    ├─test
    │  ├─jump
    │  │  ├─jump1.mp4
    │  │  ├─jump10.mp4
    │  │  ├─jump100.mp4
    │  │  ├─jump101.mp4
    │  │  ├─jump102.mp4
    │  │  ├─jump103.mp4
```

Since there are lots of sub-directories under the "dataset/landmarks/" directory, we only post a part of command "tree" output.

The whole dataset (clips and landmark files) is stored in "dataset/". Clips are stored in "dataset/clips/", while landmark files are stored in "dataset/landmarks/".

We get 900 videos from HMDB51, Weizmann, UCF-101, STAIR-actions, these videos are split into around 2300 clips. We choose 2000 of them as train clips, 300 of them as valid clips. I also get 6 videos by shooting myself using smartphone. These 6 videos are split to 200 clips, and we use the 200 clips as test clips.

For each video, there exist a directory under "dataset/landmarks/" that has the same name as the video name. For example, for video "dataset/clips/test/jump/jump1.mp4", there is a directory "dataset/landmarks/test/jump/jump1.mp4/", and this directory stores the landmark json files of video "dataset/clips/test/jump/jump1.mp4", as following:


« Project › jump_detection_openpose › dataset › landmarks › test › jump › jump1.mp4
搜索"jump1.mp4"

名称	修改日期	类型	大小
jump1_000000000000_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000001_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000002_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000003_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000004_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000005_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000006_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000007_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000008_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000009_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000010_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000011_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000012_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000013_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000014_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000015_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000016_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB
jump1_000000000017_keypoints.json	2020/3/20 4:23	JSON 文件	1 KB

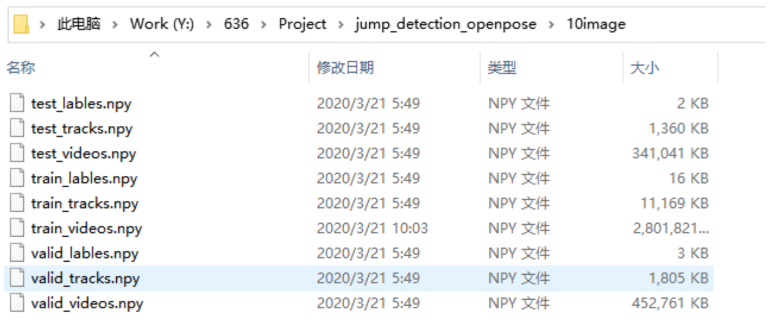
The number of json files equals to the number of frames in that video.

d) Load dataset:

There are two ways of loading dataset:

The first way is use opencv to load clips, and json python library to load landmark files. However, this is very slow, because we have too many files to read and I/O operation is much slower than CPU, so we have second way:

After we use the first way to load all dataset, we use numpy to store this data as npy files. Therefore, when we want to train our model again, we only need to load these npy files, which is much faster. The files and code are as follows:



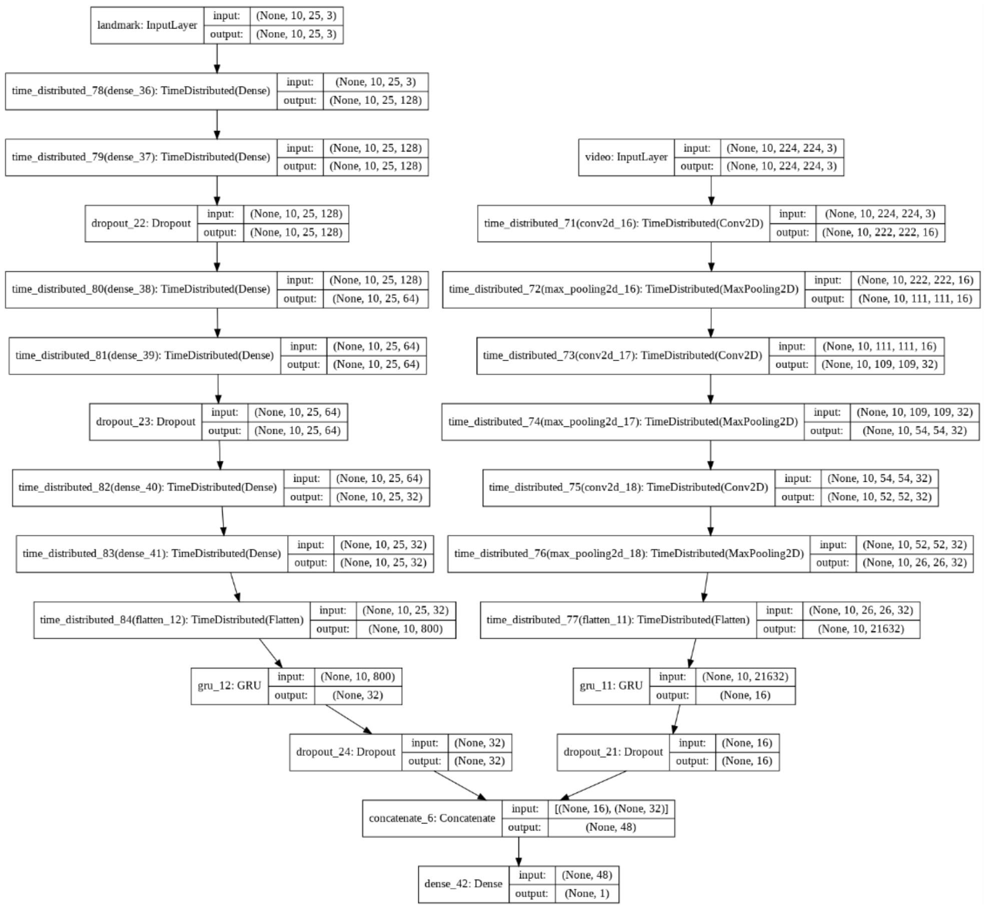
› 此电脑 › Work (Y:) › 636 › Project › jump_detection_openpose › 10image

名称	修改日期	类型	大小
test_lables.npy	2020/3/21 5:49	NPY 文件	2 KB
test_tracks.npy	2020/3/21 5:49	NPY 文件	1,360 KB
test_videos.npy	2020/3/21 5:49	NPY 文件	341,041 KB
train_lables.npy	2020/3/21 5:49	NPY 文件	16 KB
train_tracks.npy	2020/3/21 5:49	NPY 文件	11,169 KB
train_videos.npy	2020/3/21 10:03	NPY 文件	2,801,821...
valid_lables.npy	2020/3/21 5:49	NPY 文件	3 KB
valid_tracks.npy	2020/3/21 5:49	NPY 文件	1,805 KB
valid_videos.npy	2020/3/21 5:49	NPY 文件	452,761 KB

```
import numpy as np
train_videos = np.load("10image/train_videos.npy")
train_tracks = np.load("10image/train_tracks.npy")
train_lables = np.load("10image/train_lables.npy")
valid_videos = np.load("10image/valid_videos.npy")
valid_tracks = np.load("10image/valid_tracks.npy")
valid_lables = np.load("10image/valid_lables.npy")
test_videos = np.load("10image/test_videos.npy")
test_tracks = np.load("10image/test_tracks.npy")
test_lables = np.load("10image/test_lables.npy")
```

3. Network Model

a) Overview


landmark: InputLayer | input: (None, 10, 25, 3) | output: (None, 10, 25, 3)
time_distributed_78(dense_36): TimeDistributed(Dense) | input: (None, 10, 25, 3) | output: (None, 10, 25, 128)
time_distributed_79(dense_37): TimeDistributed(Dense) | input: (None, 10, 25, 128) | output: (None, 10, 25, 128)
dropout_22: Dropout | input: (None, 10, 25, 128) | output: (None, 10, 25, 128)
time_distributed_80(dense_38): TimeDistributed(Dense) | input: (None, 10, 25, 128) | output: (None, 10, 25, 64)
time_distributed_81(dense_39): TimeDistributed(Dense) | input: (None, 10, 25, 64) | output: (None, 10, 25, 64)
dropout_23: Dropout | input: (None, 10, 25, 64) | output: (None, 10, 25, 64)
time_distributed_82(dense_40): TimeDistributed(Dense) | input: (None, 10, 25, 64) | output: (None, 10, 25, 32)
time_distributed_83(dense_41): TimeDistributed(Dense) | input: (None, 10, 25, 32) | output: (None, 10, 25, 32)
time_distributed_84(flatten_12): TimeDistributed(Flatten) | input: (None, 10, 25, 32) | output: (None, 10, 800)
gru_12: GRU | input: (None, 10, 800) | output: (None, 32)
dropout_24: Dropout | input: (None, 32) | output: (None, 32)
video: InputLayer | input: (None, 10, 224, 224, 3) | output: (None, 10, 224, 224, 3)
time_distributed_71(conv2d_16): TimeDistributed(Conv2D) | input: (None, 10, 224, 224, 3) | output: (None, 10, 222, 222, 16)
time_distributed_72(max_pooling2d_16): TimeDistributed(MaxPooling2D) | input: (None, 10, 222, 222, 16) | output: (None, 10, 111, 111, 16)
time_distributed_73(conv2d_17): TimeDistributed(Conv2D) | input: (None, 10, 111, 111, 16) | output: (None, 10, 109, 109, 32)
time_distributed_74(max_pooling2d_17): TimeDistributed(MaxPooling2D) | input: (None, 10, 109, 109, 32) | output: (None, 10, 54, 54, 32)
time_distributed_75(conv2d_18): TimeDistributed(Conv2D) | input: (None, 10, 54, 54, 32) | output: (None, 10, 52, 52, 32)
time_distributed_76(max_pooling2d_18): TimeDistributed(MaxPooling2D) | input: (None, 10, 52, 52, 32) | output: (None, 10, 26, 26, 32)
time_distributed_77(flatten_11): TimeDistributed(Flatten) | input: (None, 10, 26, 26, 32) | output: (None, 10, 21632)
gru_11: GRU | input: (None, 10, 21632) | output: (None, 16)
dropout_21: Dropout | input: (None, 16) | output: (None, 16)
concatenate_6: Concatenate | input: [(None, 16), (None, 32)] | output: (None, 48)
dense_42: Dense | input: (None, 48) | output: (None, 1)

The model is likely a combination of two sequential model connected by a concatenate layer. The details are as follows

b) The right-side sequential sub-network:

The right-side sequential sub-network is built on CNN and RNN. Since we know that RNN is order sensitive, that is the order of input will influence the output of RNN; and because jump action is also order sensitive, that is we first bend our knees, and jump to the air, finally touch the ground, we can use RNN to detect jump action.

The input of the right sub-network is 3-seconds clips. We know a clip is a collection of frames. Therefore, we do data preprocessing: for each clip, we averagely take 10 frames from it, for each frame, we resize it into (224, 224). Then the one clip will be converted to one (10, 224, 224, 3) input tensor of the sub-network.

Because the clips are converted to images, we can use CNN to extra features from these images and provide the features to RNN. Since the input has time sequence, we can use Keras TimeDistributed layer wrapper to let CNN has time sequence. Note that we use dropout to reduce overfit.

Finally, we connect CNN to RNN. The sub-network code are as follows:

```
video_input = Input(shape=(num_image, 224, 224, 3), name='video')
x = TimeDistributed(Conv2D(16, (3, 3), activation='relu'))(video_input)
x = TimeDistributed(MaxPooling2D(2,2))(x)
x = TimeDistributed(Conv2D(32, (3, 3), activation='relu'))(x)
x = TimeDistributed(MaxPooling2D(2,2))(x)
x = TimeDistributed(Conv2D(32, (3, 3), activation='relu'))(x)
x = TimeDistributed(MaxPooling2D(2,2))(x)
x = TimeDistributed(Flatten())(x)
x = GRU(16, return_sequences=False)(x)
x = Dropout(0.5)(x)
```

c) The left-side sequential sub-network:

The left-side sequential sub-network is built on Dense and RNN. The reason for RNN is same as the right side.

The input of the left sub-network is landmark files of 3-seconds clips. We know a clip is a collection of frames. And the number of landmark files equals to the number of frames. Since the right sub-network averagely take 10 frames as its input, here we averagely take 10 landmark files as the input of the left sub-network. One landmark file can be converted to (25, 3) matrix. Therefore, the input tensor will be (10, 25, 3)

We choose dense layer to analyze landmarks data and provide the result to RNN. Since the input has time sequence, we can use Keras TimeDistributed layer wrapper to let dense layers have time sequence. Note that we use dropout to reduce overfit.

Finally, we connect dense layers to RNN. The sub-network code are as follows:

```
landmark_input = Input(shape=(num_image, 25, 3), name='landmark')
y = TimeDistributed(Dense(128,kernel_regularizer=regularizers.l2(0.001), activation='relu'))(landmark_input)
y = TimeDistributed(Dense(128,kernel_regularizer=regularizers.l2(0.001), activation='relu'))(y)
y = Dropout(0.5)(y)
y = TimeDistributed(Dense(64,kernel_regularizer=regularizers.l2(0.001), activation='relu'))(y)
y = TimeDistributed(Dense(64,kernel_regularizer=regularizers.l2(0.001), activation='relu'))(y)
```

```
y = Dropout(0.5)(y)
y = TimeDistributed(Dense(32,kernel_regularizer=regularizers.l2(0.001), activation='relu'))(y)
y = TimeDistributed(Dense(32,kernel_regularizer=regularizers.l2(0.001), activation='relu'))(y)
y = TimeDistributed(Flatten())(y)
y = GRU(32, return_sequences=False)(y)
y = Dropout(0.5)(y)
```

d) The output of the model

We use a concatenate layer to combine the outputs of left-side and right-side, and then use a dense layer with "sigmoid" activation to make the output value between 0 and 1, where 1 means that jump detected in the 3-seconds clip, 0 means no jump detected. Therefore, the output will be (1,). The code is as follows:

```
concatenated = concatenate([x, y])
output = Dense(1, activation='sigmoid')(concatenated)
model = Model([video_input, landmark_input], output)
model.summary()
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
```

4. Hyperparameters

In this project, we need to choose batch size, epochs and dropout. I choose batch size as 16, epochs as 50, dropout as 0.5. For 50 epochs training, we will always save the best model according to val_acc, and the final model after 50 training. The code is:

```
callbacks_list = [
    keras.callbacks.ModelCheckpoint(
        filepath=path+model_name+'_best.h5',
        monitor='val_acc',
        save_best_only=True,
    )
]
history = model.fit([train_videos, train_tracks],    train_lables,    epochs=50,    batch_size=16,
        callbacks=callbacks_list,    verbose=2,    validation_data=([valid_videos, valid_tracks], valid_lables))
model.save(path+model_name+'_final.h5')
```

5. Annotated Code

Please find my code at my Github(private)
(https://github.com/shyuan7-software/jump_detection_openpose)

6. Training and Testing Performance

a) Test:

Test code is:

```
test_loss, test_acc = model.evaluate([test_videos, test_tracks], test_lables, verbose=2, batch_size=16)
print('Final test_acc:', test_acc)
model.save(path+model_name+'_final.h5')
best_model = load_model(path+model_name+'_best.h5')
```

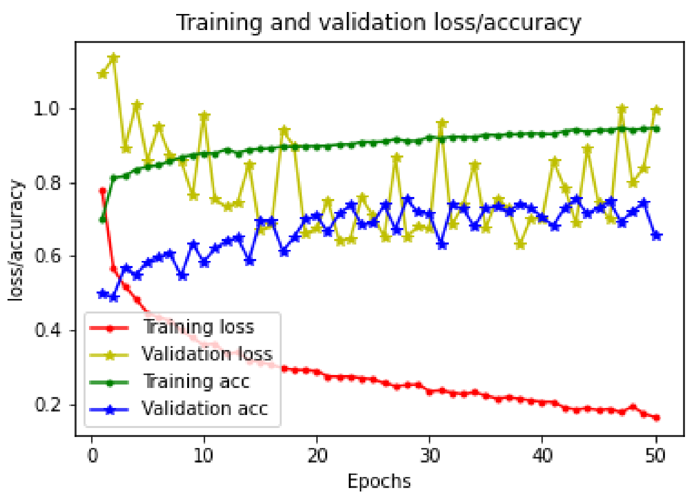
```
test_loss, test_acc =
    best_model.evaluate([test_videos, test_tracks], test_lables, verbose=2, batch_size=16)
print('Best test_acc:', test_acc)
```

Test result is:

```
Final test_acc: 0.5689655172413793
Best test_acc: 0.771551724137931
```

The best model has 77% accuracy on test dataset, while the final model (trained for 50 epochs) have only 56% accuracy, obviously there exists overfit.

b) Train and Validation:



From the pic, the model works very well on training dataset, but it is unstable on validating dataset, and there is overfit after around 40 epochs.

7. Instruction on how to run your code for training and testing your neural network

a) Install Dependencies

- Python3
- Keras
- TensorFlow
- Numpy
- Matplotlib
- OpenCV

b) How to run my code for training

For training a network using my code, just use the following command:

```
python train_model.py
```

Note that the code on GitHub does not include dataset (because it is too large to upload), so if you clone from my GitHub and run this command, it would fail. If you want to get the dataset, please send me an email shyuan@tamu.edu.

c) How to run my code for test

Please find this video (private) (https://youtu.be/RItYYhAImLs)

For your test convenience, I provide some sample videos and their landmark files as following:

sofa.mp4:	https://youtu.be/Il0c4LaDsQI
raw_video.mp4:	https://youtu.be/sOZx88CmMOE
no_jump.mp4:	https://youtu.be/wdpJLK31rIY
keep_jump.mp4:	https://youtu.be/xO2b17cI9Wo
intermedia.mp4:	https://youtu.be/1KJj8GDu0d4
bed.mp4:	https://youtu.be/JiuroZvrt8M
landmark files:	https://drive.google.com/drive/folders/1St1RiO6kB9MlPiOF6ItiIuPoCYV1uahQ?usp=sharing

If you want to access my dataset (train, valid, and test), please access it through google drive:

Clips and Landmark files	https://drive.google.com/drive/folders/1vUYK2-X1HWBWLH3C1e4IYcaMAN_CzRjg?usp=sharing
.NPY files (for fast loading dataset)	https://drive.google.com/drive/folders/1V6PB5sE8K8jLW1UnnoBPDZVhQO-tLXAh?usp=sharing