Jump Detection Project Report - Second Submission

April 2020

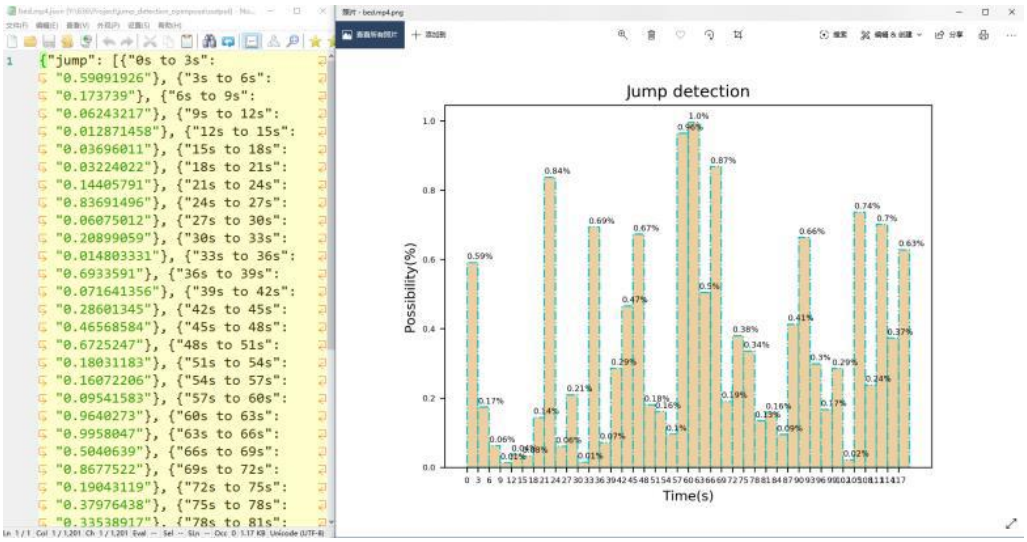
1. Topic

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's landmark files be the 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:



2. Improvement since first submission

a) Dataset:

In this submission, I choose to exclude the HMDB51 from my dataset, for the reason that the video from HMDB51 is from movies, which means that there is more than one person in each video. In smart-home application (the application of this project), usually there is only one person that can be captured by camera. Another reason is that OpenPose performs better when processing video containing single person. For the above two reasons, I choose to delete HMDB51 from my dataset. Therefore, the latest dataset comes from Weizmann, UCF-101, STAIR-actions, and videos that I shoot myself.

Another change on dataset is the sample rate, in my first submission, I mentioned that I cut all videos into many small 3-seconds clips, and from every 3-seconds clip, I averagely take

10 frames from it. But now, I increase the sample rate from 10 frames per clip to 32 frames per clip, for the reason that higher sample rate means more information we get from our dataset.

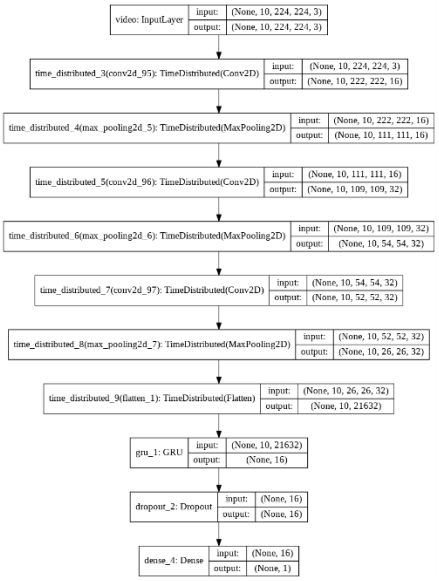
The others regarding the dataset remains unchanged, and the details will be discussed in the following section.

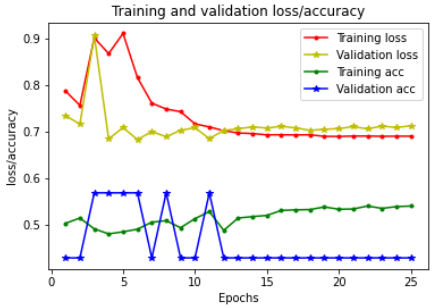
b) Model:

In our first submission, we choose to use ensemble two network in our model, one is a network based on DENSE and RNN which takes body-landmark as input, another is the network based on Conv2D and RNN which takes frames as input.

In this submission, I choose to disregard the information from frames, and only consider the information from body-landmarks. For the following reason:

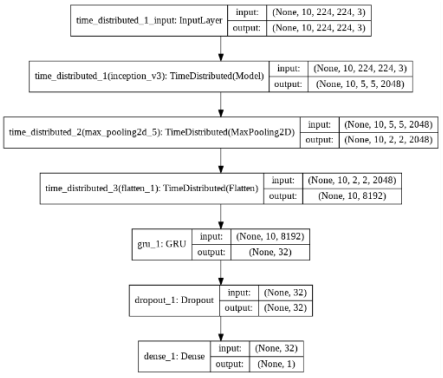
I do several experiments for the network based on Conv2D and RNN, first I designed the following network:

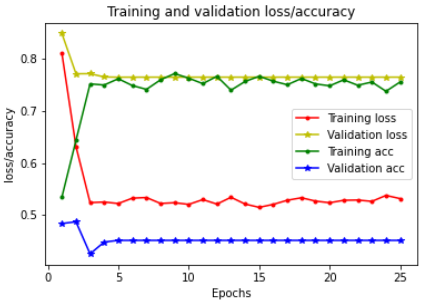




The above network is the network based on Conv2D and RNN, which is a part of model in my first submission. From its loss/acc figure, even the training accuracy remains 50% after several epochs training, indicating that this network does not work for my project.

Then I thought that the reason may be the network is too small to learn features from so many videos, then I designed another network as follows, note that I used transferred learning here to include the pre-trained INCEPTION_V3 network in my mode:

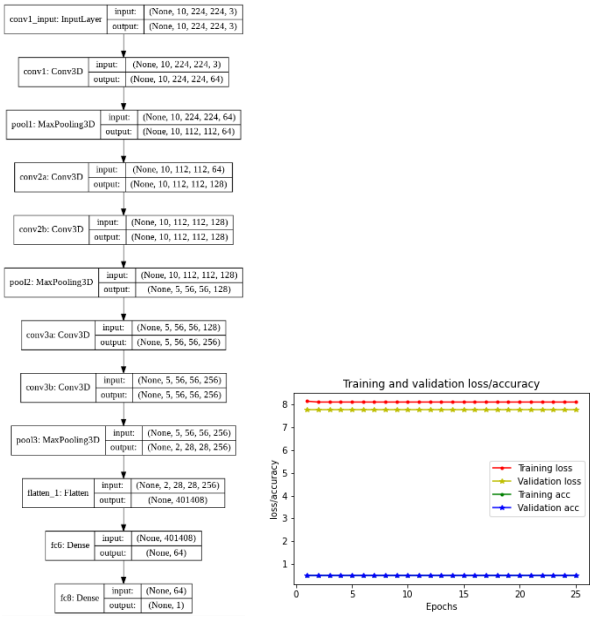




However, even with the pretrained Conv2D network (21,802,784 parameters), the model still works not well on the video input.

Then, I disregard the Conv2D and RNN method, and choose the method from the paper

Learning Spatiotemporal Features with 3D Convolutional Networks. Then I design the following network (I reduce the size of the network in the paper, because the GPU I can access in Colab does not support the big network mentioned in the paper):



It's obviously that this figure also does not work for video input.

After so many tries, I choose to disregard the information from frames temporarily, and focus more on body-landmark part in this submission.

For the new model which takes body-landmark as input, the design of it is based on Modeling Temporal Dynamics and Spatial Configurations of Actions Using Two-Stream Recurrent Neural Networks, more details will be provided in the following section.

3. Dataset

a) Raw dataset: video

I use part videos from 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 body-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:

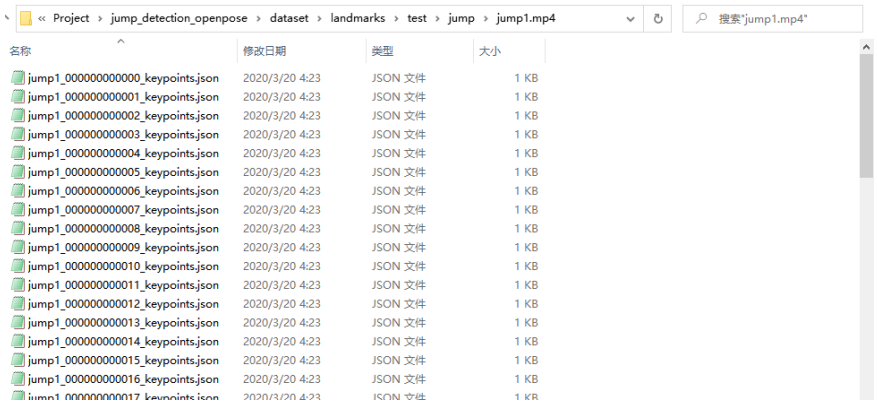
```
Y:\636\Project\jump_detection_openpose\dataset>tree
卷 Work 的文件夹 PATH 列表
卷序列号为 E856-3BE6
Y:.
├─clips
│  ├─test
│  │  ├─jump
│  │  └─others
│  ├─train
│  │  ├─jump
│  │  └─others
│  └─valid
│      ├─jump
│      └─others
└─landmarks
    ├─test
    │  └─jump
    │      ├─jump1.mp4
    │      ├─jump10.mp4
    │      ├─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 body-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 Weizmann, UCF-101, STAIR-actions, these videos are split into around 1800 clips. We choose 1600 of them as train clips, 200 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:



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

d) Load dataset:

There are two ways of loading dataset:

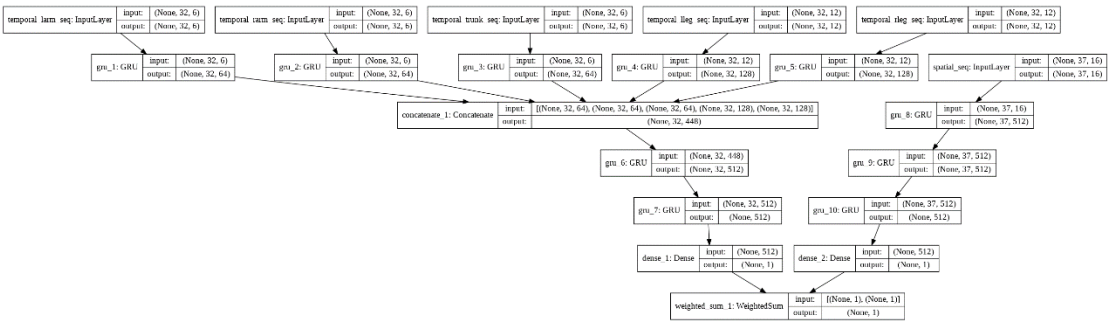
The first way is to use opencv to load clips, and json python library to load landmark files. However, this is very slow, because we have so many files to read and I/O operation is much slower than CPU, so we have the second way:

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:


```
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")
```

4. Network Model

a) Overview:



This model is designed based on [Modeling Temporal Dynamics and Spatial Configurations of Actions Using Two-Stream Recurrent Neural Networks]. It only takes body-landmarks as input. A more straightforward description of this model is as follows:

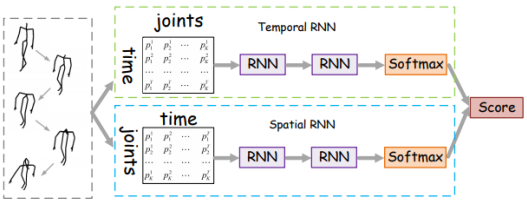


Figure 1. A two-stream RNN architecture for skeleton based action recognition. Here *Softmax* denotes a fully connected layer with a softmax activation function.

This model takes body-landmarks to two-stream, temporal stream and spatial stream, and do parallel computing, and finally collect the outputs from two stream and compute the final result. We will discuss the model from two streams.

b) The temporal stream:

This temporal channel of RNN models the temporal dynamics of skeletons. Here we use Hierarchical RNN, the definition of Hierarchical RNN is:

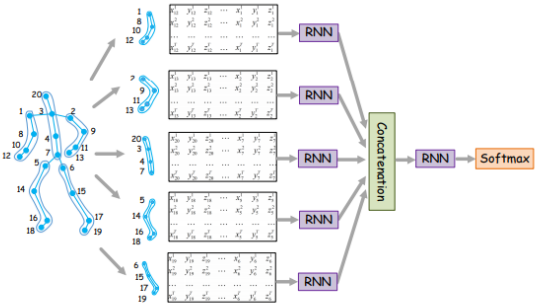


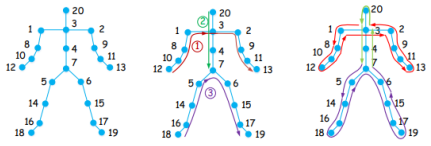
Figure 3. Hierarchical RNN for skeleton based action recognition.

The human skeleton can be divided into five parts, i.e., two arms, two legs and one trunk. We observe that an action is performed by either an independent part or a combination of several parts. For example, kicking depends on legs and running involves both legs and arms. Thus, a hierarchical structure of RNN is used to model the motions of different parts as well as the whole body. Figure 3 shows the proposed structure (From the paper).

The temporal stream is designed based on Hierarchical RNN, which takes body-landmarks of left arm, right arm, trunk, left leg and right leg separately as inputs, and follows three layers RNN, and a final dense layer with sigmoid activation function.

c) The spatial stream:

This spatial channel of RNN models the spatial dynamics of skeletons. The detailed definition is as follows:



(a) Undirected graph of joints. (b) Chain sequence of joints. (c) Traversal sequence of joints.

Figure 4. (a) The physical structure of 20 joints. (b) Convert the joints graph into a sequence. The joints of arms come first, then that of body, finally is that of legs. (c) Use a traversal method to transform the joints graph into a sequence. The order of the sequence is the same as the visiting order of the arrow.

Human body can be considered as an articulated system of rigid segments connected by joints. The physical structure of the 20 joints is represented by an undirected graph in Figure 4(a). Nodes denote the joints and edges denote the physical connections. When an action takes place, this undirected graph displays some varied patterns of spatial structures. For example, clapping is performed with the joints of the two palms striking together, and bending is acted when the joints of the trunk shape into a curve.

To model the spatial dependency of joints, we cast the graph structure into a sequence of joints and exactly develop a relevant RNN architecture. ***The input of the RNN architecture at each step corresponds to the vector of coordinates of a certain joint****. As a joint has only three coordinates, we select a* ***temporal window*** *centered at the time step and concatenate the coordinates inside this window to represent this joint. This RNN architecture models the spatial relationships of joints in a graph structure and is called spatial RNN. (From the paper)*

The important problem here is to convert a skeleton graph to a sequence of joints. And we choose Traversal sequence in our model:

> *Traversal sequence. As illustrated in Figure 4(c), we first select the central spine joint as the starting point, and visit the joints of the left arm. While reaching an end point, it goes back. Then we visit the right arm, the upper trunk, etc. After visiting all joints, it finally returns to the starting point. We arrange the graph into a sequence of joints according to the visiting order. The traversal sequence guarantees the spatial relationships in a graph by accessing most joints twice in both forward and reverse directions. (From the paper)*

According to the experiments done in the paper, when the temporal windows size is one fourth of the total number of frames of one clip, the model performs best. Since in our project, our frame number is 32, the temporal window size we select here is 8.

At the end of the spatial RNN, there is also a dense layer with sigmoid activation function.

d) Weightedsum Layer:

This layer connects the outputs from temporal RNN and spatial RNN, since both RNNs output the possibility of jump, which is between 0 and 1, the Weightedsum layer simply adds them with their weights. According to the paper, the model performs best when we assign weight 0.9 to temporal RNN and 0.1 to spatial RNN.

5. Hyperparameters:

In this submission, I choose batch size as 64, dropout as 0.5, epochs as 100. For 100 epochs of training, I will always save the best model according to val_loss, as well as the final model after 100 epochs of training.

6. Annotated Code:

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

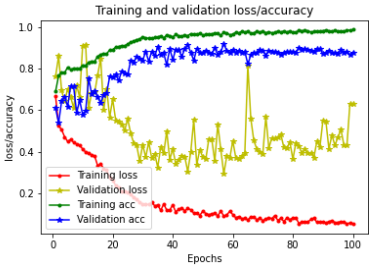
7. Training and Testing Performance

a) Testing:

```
Final test_acc: 0.9482758579583004
Final test_loss: 0.2163128308181105
Best test_acc: 0.9353448316968721
Best test_loss: 0.17720138589883672
```

In first submission, the test acc is 77%, and the current model can have 93% accuracy, which is a big improvement.

b) Training validation



The pic shows that the model works very well on training dataset, and although its performance on validation dataset is unstable, the performance indeed gets better and better after training. The model works best (lowest val_loss) when training around 60 epochs.

8. Performance in real life:

I made some videos by shooting myself jumping, and use the model to detect jump in my video. The follows are some interesting discoveries:

Angle of camera: we can shoot a person from three angles, from the front, from the side, and from the back. From the experiments I have done (you can also repeat the experiments by using the sample video I provided on my YouTube channel), the front and the side angle works better than the back, although the back angle is not too much worse to work. However, if the angle of camera changes when the person is jumping (spin jump), the model cannot detect it. I think the reason is that the model totally relies on body-landmarks generated by OpenPose, and OpenPose cannot generate precise landmarks when the angle of camera is changing.

How to jump: roughly we can separate jump to two classes: first is to jump with full power, another is only use small power. For the full power jump, a person should use all his strength and jump as higher as possible, then this requires him to bend his knees and jump. For the small power jump, a person only needs to let his feet not touch the ground for a while, so it is possible that he does not need to use his leg, he can jump with the strength on his feet (tiptoes). And also for small power jump, the change of height is also smaller than full power jump. The model works pretty well for the full power jump, but not so good on the small power jump. The reason is obvious, the changes happened to body-landmarks are much smaller when small power jump than full power jump, make it harder for the model to detect.

9. Future improvement and challenge:

We have discussed two problems of the model in section 8, the future goal is to solve the problem. From my current thought, I think we should again include CNN based network in our model, which takes original frames as input, to solve the problems.

The cause of the two problems is actually the disadvantage of body-landmarks generated by OpenPose. The OpenPose cannot generate body-landmarks very well when the angle of camera changes. It is also hard for it to show precise difference between body-landmarks, when the action is not so obvious. However, the problems are not hard for CNN based model. Instead of analyzing body-landmarks, the CNN based model can simply extract the feature of the distance between person's feet and the ground, so it can solve the problems easily.

The challenge is huge, for the reason that we need to find an appropriate CNN based network that works for our project, considering the fact I have tried lots of CNN based network and they all failed. One way I can think of is to read more papers and try their models one by one.

10. 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 find it on the appendix of this report.

c) How to run my code for test

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

If you want to get the sample videos and their landmark files, please find them on the appendix of this report.

Appendix

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
side_small_power.mp4	https://youtu.be/Xkee64oEeBM
side_full_power.mp4	https://youtu.be/SoTE9ZN9QKI
front_small_power.mp4	https://youtu.be/j1_AP10X798
front_full_power.mp4	https://youtu.be/hh1LJGicr8U
back_small_power.mp4	https://youtu.be/2XeIm2Hywd4
back_full_power.mp4	https://youtu.be/QZUxwz2MYOY
random.mp4	https://youtu.be/J6nTsPEfElQ
landmark files	https://drive.google.com/file/d/1TO9qZnFNA0U0Kj7CNqGJTSWKzFSaqOC5/view?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/file/d/1shPnXQeDR2yWOFankFqCrR7JyakFenSl/view?usp=sharing
.NPY files (for fast loading dataset)	https://drive.google.com/drive/folders/1b11D5WAf7ELt4FV2HNGCvS3ZCKkNhYJS?usp=sharing