CSCE636 Project

Final submission

Yipeng LU

April 2020

1. Topic

In this project I built a neural network and train it on Google colab to recognize stretching body poses against everyday home actions (drinking, walking and playing with tablet) using video recorded by myself. At the end, I wrote a function that takes in path of the video, and show a figure, where the x-axis is time, and the y-axis is the corresponding label.

2. Dataset

I record videos of myself and cut the videos to 2s, 10fps video slices to be the dataset. Stretching videos include actions of sky reaching, waist twisting and leg stretching. Non-stretching videos include actions of playing with tablet, drinking and walking. Each action is recorded under both standing and sitting conditions, and through varies of angles. There are 346 positive training examples, 343 negative training examples, 173 positive validation examples, 173 negative validation examples, 172 positive testing examples, and 172 negative testing examples. Each action from each angle under each sitting condition has similar amount of examples in the dataset, and training dataset, validation dataset and testing dataset are made from different videos.

3. Model

3.1 Architecture

My model is a combination of time distributed dense layer and stacking RNN layers.

Time distributed dense layer with 4 50-neuron dense layers having RELU activation is applied to key points of each frame for feature extraction. Batch normalization is applied to the output of each of the 30-neuron dense layer for faster convergence.

The output of the time distributed dense layer, applied on each of the 20 frames, is then passed to a stacking RNN layer with one 32-unit GRU layer and one 64-unit GRU layer. The output of the stacking RNN layer is passed to a dense layer having sigmoid activation to make the final prediction.

The architecture of the model is shown below in figure 1.

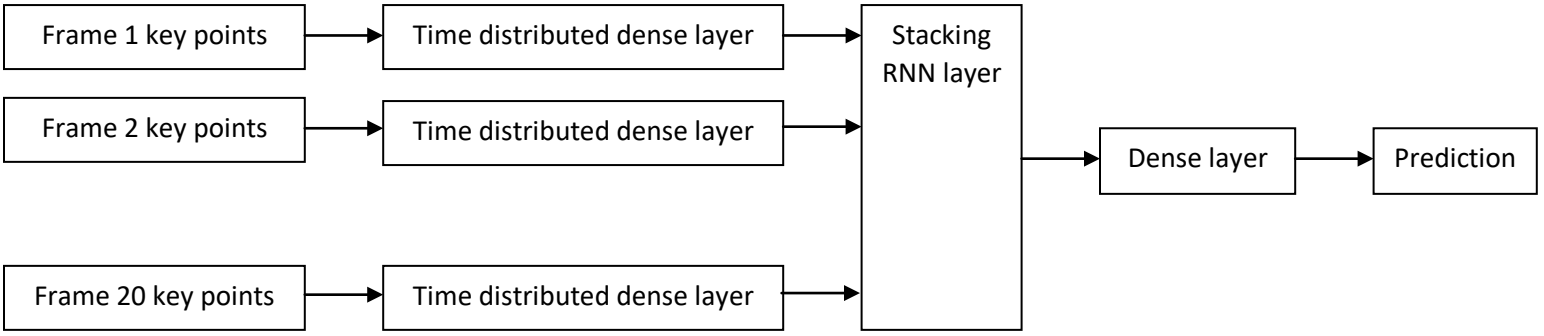

Figure 1

3.2 Input: Shape of tensor

The input tensor has a shape of 20*50 which denotes the 20 sequential frames with 25 key points having X and Y coordinates each.

3.3 Output: Shape of tensor

The shape of output is 1*1.

3.4 Shape of output tensor for each layer

Shape of output of the time distributed dense layer is 20*50.

Shape of output of the stacking RNN layer is 64*1.

Shape of output of the final dense layer is 1*1.

4. Hyper parameters

4.1 Lists of hyper parameters

In this project, there are three hyper parameters I choose to tune: number of dense layer, dropout rate, and number of neurons in each dense layer.

4.2 Range of Value of Hyper parameters Tried

number of the dense layer	2, 3, 4,6,8
dropout rate of the dense layer	0, 0.1,0.2
number of neurons	10,30,50,80,100

4.3 Optimal Hyper parameters Found

number of the dense layer	4
dropout rate of the dense layer	0
number of neurons	50

For the number of the dense layer and number of neurons for each dense layer, the performance do improves as the capacity of the network increases. But after some point, no matter how I increase the capacity of the network, the network neither over fit nor becomes better, and that is the time when I stop increasing the capacity of my network.

For the dropout rate of the dense layer, I find out that the performance of my network is the best when there are no dropout

layers. Since my model does not over fit anyway, I do not place dropout layers inside my model.

5. Data preprocessing, performance and assembly learning

5.1 Data preprocessing

First of all, I cut the long videos that I recorded into 2s, 10fps slices using moviepy library, then I use openpose to extract the key points of each frame and store the key point data of each frame in a json file containing 25 triples in the form of (X coordinate, Y coordinate, Possibility).

Because the possibility argument is not useful for activity recognition, I create a new list containing 25 doubles in the form of (X coordinate, Y coordinate). Then, I divide each X coordinate by the width of the video and divide each Y coordinate by the height of the video so that each number in the list is between 0 and 1. Next, because zooming out the video frame should not influence its label, I zoom out the original video frame by a random number j between 1 and 2, by dividing all the X and Y coordinate by j. Lastly, because shifting the video frame should not influence its label, I shift the X coordinate by a random number k between -0.3 and 0.3, and shift the Y coordinate by a random number l between -0.3 and 0.3. The value of random variable j, k and l is the same for all the 20 frames of a video slices, and is initialized every video slices.

Because storing the key points of all frame in a list takes too much RAM, I decide to use a data generate to feed the data to the model. As the default data generator in keras does not satisfy my need, I write my own data generator. My own data generator can output the key points of each of the 20 frames of the video slices as well as the label of the

video slices in the form of ((batch_size,20,50),(batch_size,1)). Each time all the video slices has been output once, all the video slices are shuffled so that same step in different epochs does not output the same key points data.

5.2 Performance

The model is trained for 300 epochs. The loss and accuracy of training data and validation data is shown in figure 2 and figure 3. The result of the trained model on the test set is loss 0.34 and accuracy 0.943.

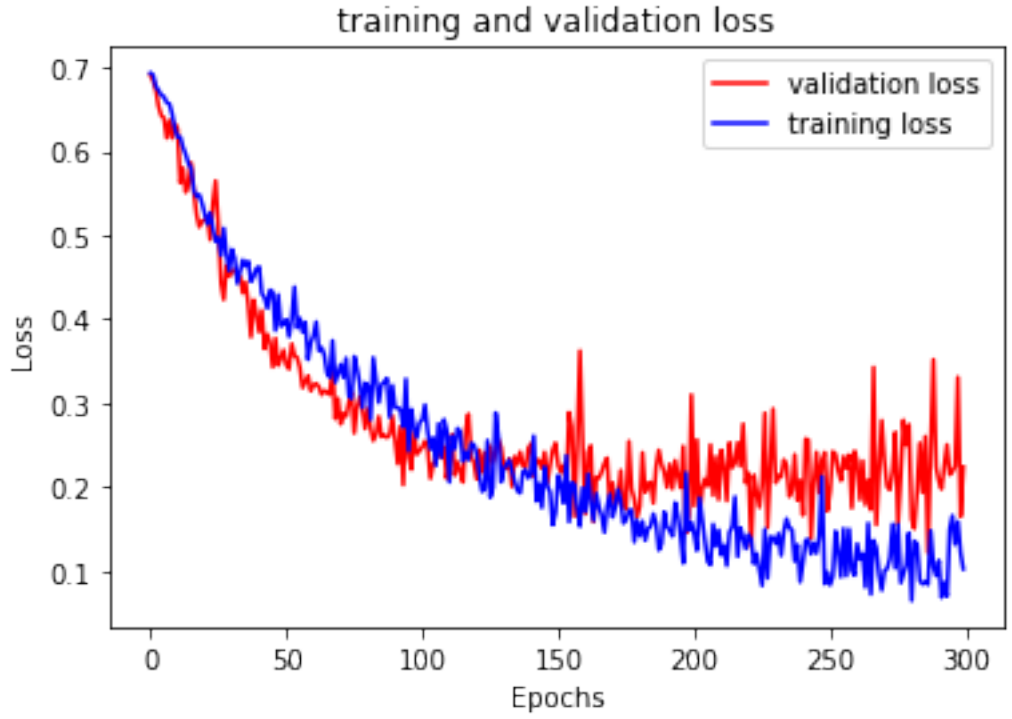

Figure 2

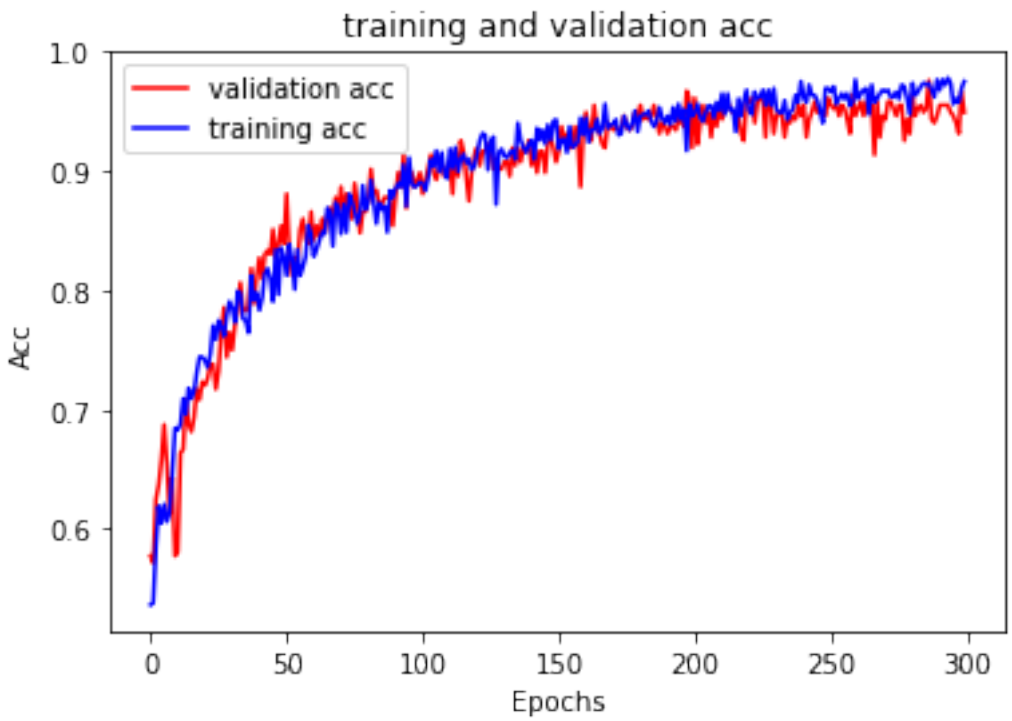

Figure 3

5.3 Assembly learning

The majority of positive examples in validation set and testing set that are wrongly labeled are related to actions of stretching legs from the back, which is shown in figure 4.

Figure 4

The majority of negative examples in validation set and testing set that are wrongly labeled are related to actions of drinking from the sides, which is shown in figure 5.

Figure 5

Because the false negative rate is higher than the false positive rate, I adjust the threshold to balance the positive and negative examples. The example is considered false negative if its original label is 1 and its prediction label is less than 0.5. The example is considered false positive if its original label is 0 and its prediction label is more than 0.4. Then, I get 22 false negative examples and 20 false positive examples.

I construct a new model m2 with same architecture as my original model m1, and trained on these 42 wrongly-labeled examples for 300 epochs. Then I construct a new model m3 which makes decision based on the prediction of m1 and m2 using the keras functional API. M1 and m2 both take in the key points of 20 frames as input, and is set to be non-trainable. The final dense layer takes in the prediction of m1 and m2, and output the final decision of m3. The architecture of m3 is shown in figure 6.

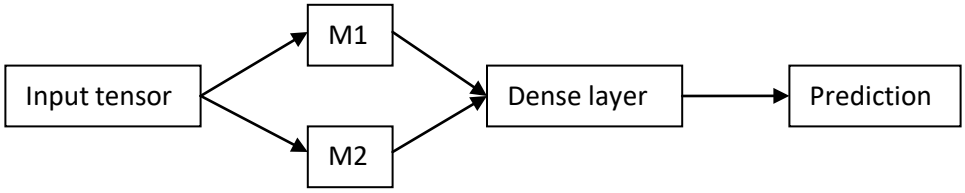

Figure 6

The model is trained for 300 epochs. The loss and accuracy of training data and validation data is shown in figure 7 and figure 8. The result of the trained model on the test set is loss 0.35 and accuracy 0.932. Since its performance is not better than the original model, I did not consider this assembly learning model as my final model.

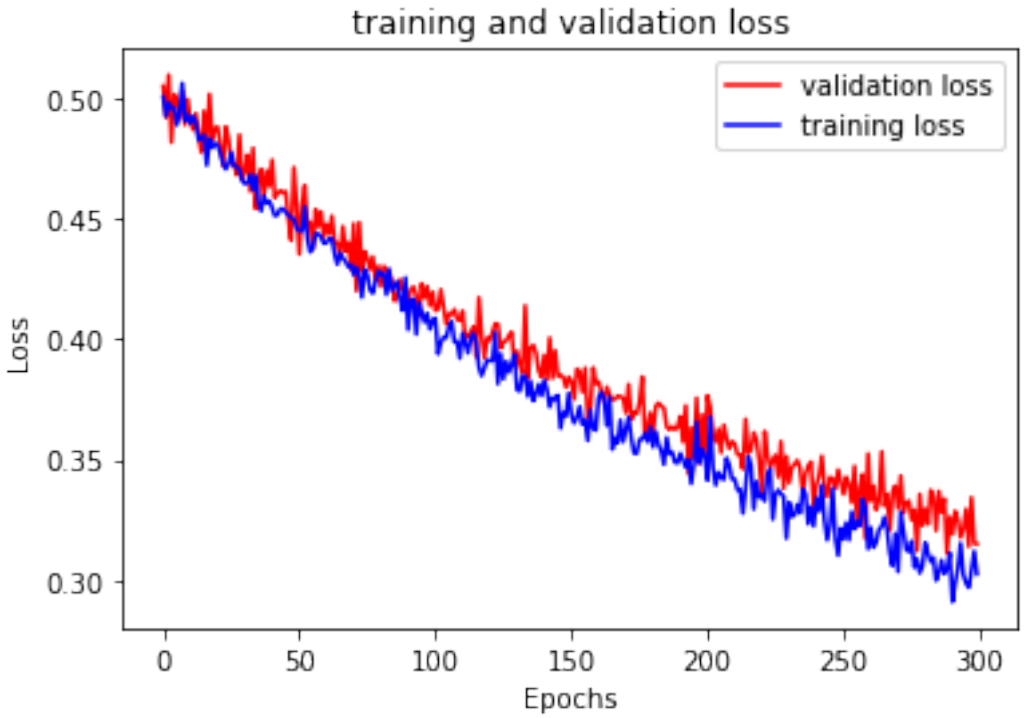

Figure 7

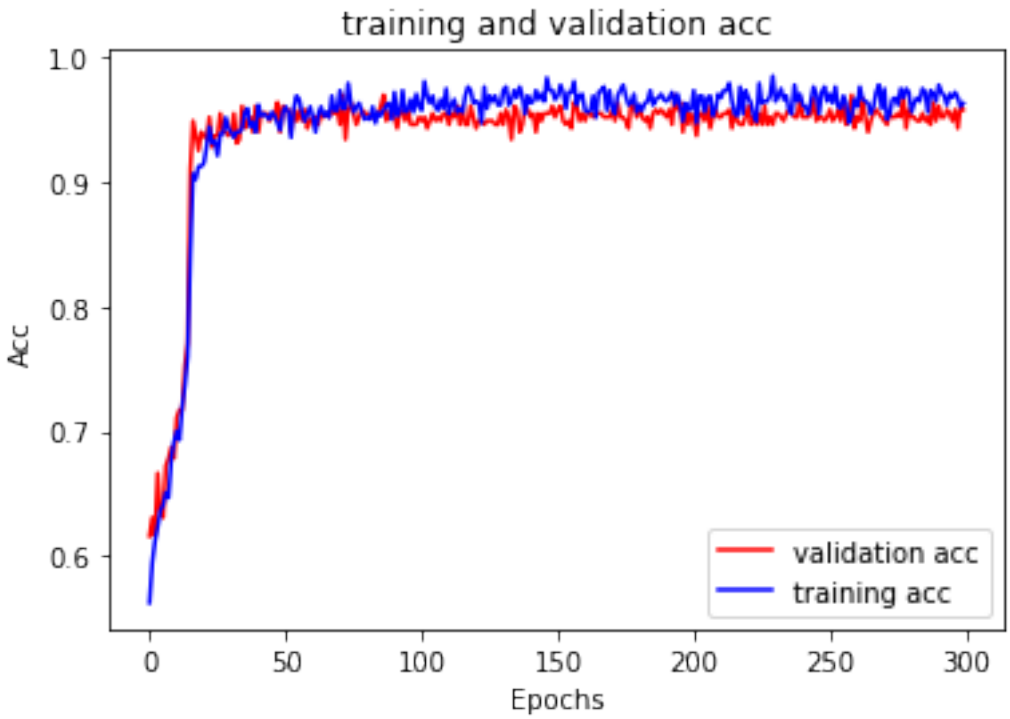

Figure 8

6. Code, demo video and improvements

The code of my prediction function and the model is on https://github.com/Yipeng-LU/CSCE636_Last_Submission

The demo video is on https://youtu.be/9tY4zlN_Cys The demo video is 2 min 56 seconds long, and consists of various stretching actions and non-stretching actions. The plot that my prediction function generate is shown in figure 9. Because the video is relatively long, the x axis is very crowded.

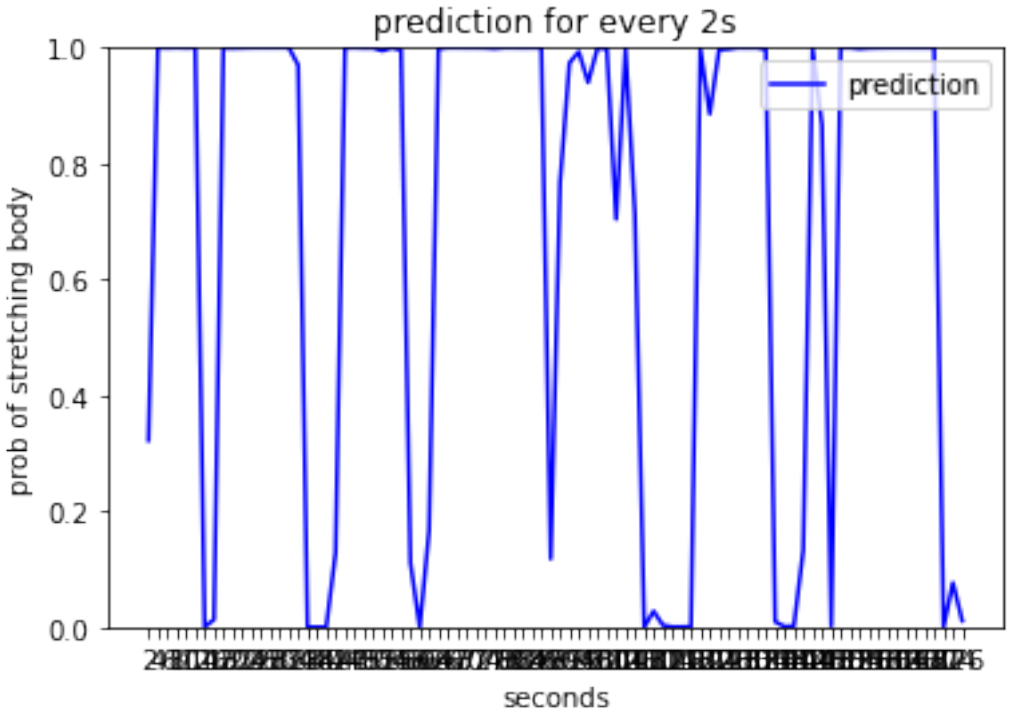

Figure 9

The improvement of this submission compared to submission 3 is:

1. In the data augmentation step before the data generator generate the data, I add a new operation which randomly shifts the key points in both X and Y directions besides just zooming out the key points.
2. Because the possibilities of the key points are not useful for training, I abandoned them and only left X and Y coordinate of the key points.
3. I tried to train a new model on false examples, and trained a dense layer that takes in predictions of both models and makes final prediction.
4. More hyper parameters are tried in order to reach the best performance of the model.