Zohreh Shaghaghian / Project05/ Action detection: Pushing

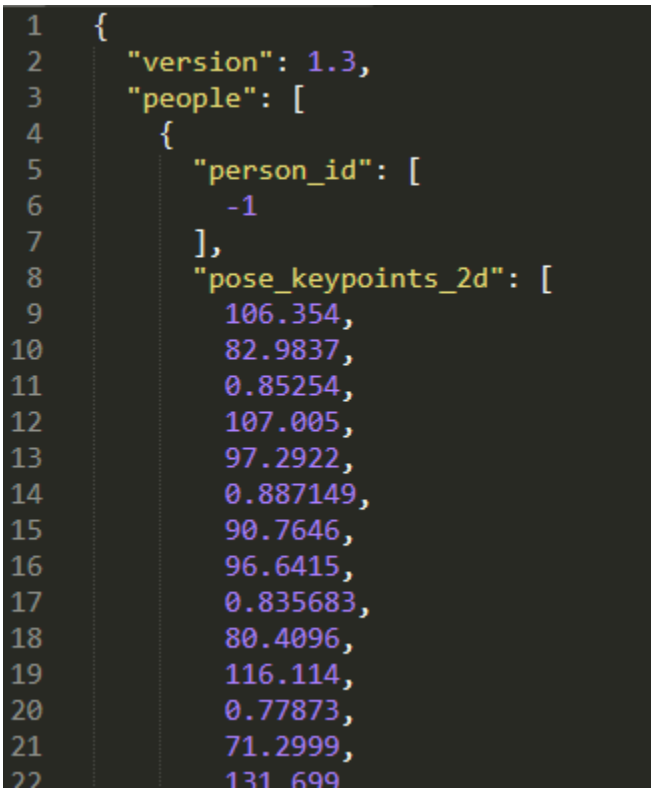
The original dataset is 50 videos of pushing and 50 videos of other activity.The reason to separate videos is to facilitate data preparation and avoid manual labeling.

In previous submission I used image frames of videos as input data and I used two models to tarin my data (CNN and CNN+LSTM) separately. However the model's accuracy was not very good and the model's behavior was very fluctuating. In fact the CNN architecture was not strong enough to get the features.

In this submission I used openpose json files as input data and the result improved a lot.

The openpose is run in colab and each video returns #n number of json files where n represents the number of frames for each video.

The json files consist an object called 'people' , that has another object named as 'pose_kepoints_2d'. Some of the json files consist other objects relating to facial expression or 3d movement. But The common object was 'pose_kepoints_2d' which I used. The landmarks represent (x ,y, acc) of each detected point by openpose.



Then the json files are loaded, read and the landmarks are saved as csv file. The following codes show how I saved the landmarks in a csv file. A dictionary is used to save the labels and their corresponding landmarks. Some of the json files' people object were empty (mostly the last

frames) and I assume the reason is that the person is disappeared in that video. So I had to handle this in my code.

So a dictionary is defined where keys are the label of each video (push_0, push_1, ..., other_0, other_1, ..) and values are vector of size (n,75) where 'n' is the number of frames and 75 is fixed for all json files which represent the vector size of the land mark: 'pose_keypoints_2d'

```
def get_video_name(file_name):
    parts = file_name.split('_')
    return (parts[0] + '_' + parts[1])


# the if statement is added since some json files did not have 'pose_keypoints_2d'

# returning a dict where keys are video labels(push_0,push_1, ..)
# and values are landmark vectors corresponding to frames of the video
def read_data_from_landmarks(json_dir):
    video_to_landmarks = {}
    for file in os.listdir(json_dir):
        file_path = json_dir + '\\' + file
        temp = json.load(open(file_path))
        if len(temp['people']) == 0:
            pass
        else:
            key = get_video_name(file)
            value = temp['people'][0]['pose_keypoints_2d']
            video_to_landmarks.setdefault(key, []).append(value)

    return video_to_landmarks
```

Then the data is shuffled based on the keys (labels)

```
# dictonary is a hashtable  and shuffle not work.To shuffle :
# get the list of keys, shuffle the keys and get the values of the corresponding keys
# define a seed for random to be iterable --> random.Random(seed)
def shuffle_data (video_to_landmarks):
    shuffled_keys = list(video_to_landmarks.keys())
    random.Random(4).shuffle(shuffled_keys)
    labels = shuffled_keys
    train_data = []
    for key in labels:
        train_data.append(video_to_landmarks[key])

    return train_data, labels
```

Since the length of the videos were different (min~3 sec, and max~10 sec), openpose created different number of frames. So I used padding to get a constant number of frames. This step is crucial later in the model and is presented as n_timestep as a parameter of LSTM layer. Maxlen is used as a Stabilizer. Without maxlen the data will be padded to the maximum length of the existing frame and this results in too many zero padding for short videos and may have disruptive impact in training. The maxlen=125 is chosen based on the average length of my videos, 4 sec.

```
# padding frames of different videos, the bellow function will automatically pad to max length
input_data = pad_sequences(train_data, dtype='float32', maxlen=125, padding='post')
```

Reshaping the data so it could be saved as dataframe in a csv file:

```
# get dataframe so the data could be saved as CSV file as input: (dataframe input should be 2d)
# reshaping the data so it could convertto dataframe
r = input_data.shape[0]
m = input_data.shape[1]
n = input_data.shape[2]

reshaped_inputdata = input_data.reshape(r, m*n)
input_df = pd.DataFrame(data=reshaped_inputdata, index=labels)
csv_filepath = "D:\\tamu\\courses\\DeepLearning\\ProjectPart5\\input_data_2.csv"
input_df.to_csv(csv_filepath)


print('input_data.shape: ',input_data.shape)
print('reshaped_inputdata.shape: ', reshaped_inputdata.shape)
```

From this step I only needed to read the csv file, reshape it again to get back to the original shape and use it as input data. Training model is done in colab:

Reading data from csv file:

```
base_dir = "/content/drive/My Drive/Google_Colab"
df = pd.read_csv(base_dir + '/' + 'input_data_2.csv', index_col = 0)

# print (df.head())
```

Get corresponding labels:

Note: array of zero or one are added to match the shape of the x_traing

```
labels = df.index.tolist()
# get binary labels: if label=push, then is 1, and if label='other', then is 0
def binary_label(labels):
    binary_labels = []
    for l in labels:
        if l.split('_')[0]=='push':
            binary_labels.append(np.ones(m))
        else:
            binary_labels.append(np.zeros(m))
    return binary_labels

# get list of labels as 0 and 1
train_label_list = binary_label(labels)
```

Normalizing x_train (Neural network does not work well with big numbers)

```
# get x_train and Normalize it so all numbers get scaled between 0-1
input_data = df.values
min_max_scaler = preprocessing.MinMaxScaler()
x_train_scaled = min_max_scaler.fit_transform(input_data)
```

Reshaping the input data to its original shape:

```
x_train= x_train_scaled.reshape(r, m, n)
y_train = np.asarray(train_label_list)     #convert label list to numpy array

print(x_train.shape)
print (y_train.shape)
```

Note: y_train shape still need a change. The reason is to match the last layer (Dense classifier) of the model.

```
[ ] y_train = y_train.reshape(y_train.shape[0], y_train.shape[1], 1)
    print(y_train.shape)
```

```
(100, 125, 1)
```

```
[ ] print(x_train.shape)
    print (y_train.shape)
```

```
(100, 125, 75)
(100, 125, 1)
```

Model Configuration:

For this submission I used one LSTM layer with 5 cells and one Dense layer as classifier. There are some keypoints here so the model work correctly.

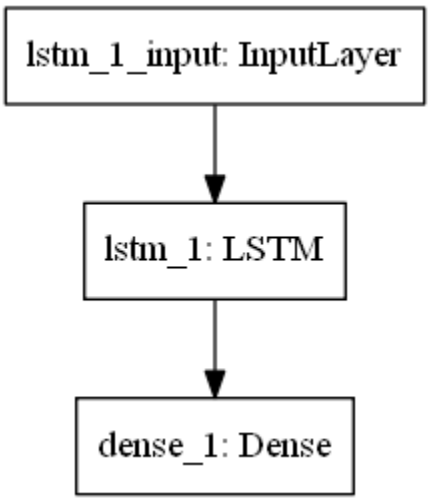
1. Input_shape (n_timesteps,n_features) → this should match the shape of my x_train (100, 125,75)
   Note that 100 is number of my videos and it will be later fed during fed through batch_size
2. return_sequences = True → This Argument plays crucial role in model training and keeping the time_steps in the last layer (classifier). If This argument is not mentioned, the last layer shape will be (None,1) instead of (None,125,1) and this means that all 125 frames will be predicted as one probability instead of 125 probabilities. So this is the reason why y_train shape is defined as ((100, 125, 1)
3. The last layer is a Dense layer representing 1 neuron (binary: push=1, other=0)

```
## Deep learning model:
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, LSTM, Dropout
from keras import optimizers

n_timesteps = 125   # same as maxlen padded
n_features = 75    # same as vector size of landmark
cells = 5

model = Sequential()
model.add(LSTM(cells, input_shape=(n_timesteps,n_features), dropout=0.1, recurrent_dropout=0.5, return_sequences = True))
model.add(Dense(1))
model.summary()

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])                # optmizer='rmsprop'/ 'sgd'
```

Model layers and parameters:

```
Model: "sequential_44"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_40 (LSTM)               (None, 125, 5)            1620
_________________________________________________________________
dense_47 (Dense)             (None, 125, 1)            6
=================================================================
Total params: 1,626
Trainable params: 1,626
Non-trainable params: 0
```

Model Architecture:



As you can see the architecture of the model is quite simple and much simpler than my previous submission. However, this model works much better compare to my previous models and this shows the power of openpose landmarks!

The model is trained for 8 epochs, with batch size of 5 and validation split of 15%.

```
[ ] epochs = 8    #10
    history = model.fit(x=x_train, y=y_train, validation_split = 0.15, batch_size=5, epochs=epochs, shuffle=True)
```

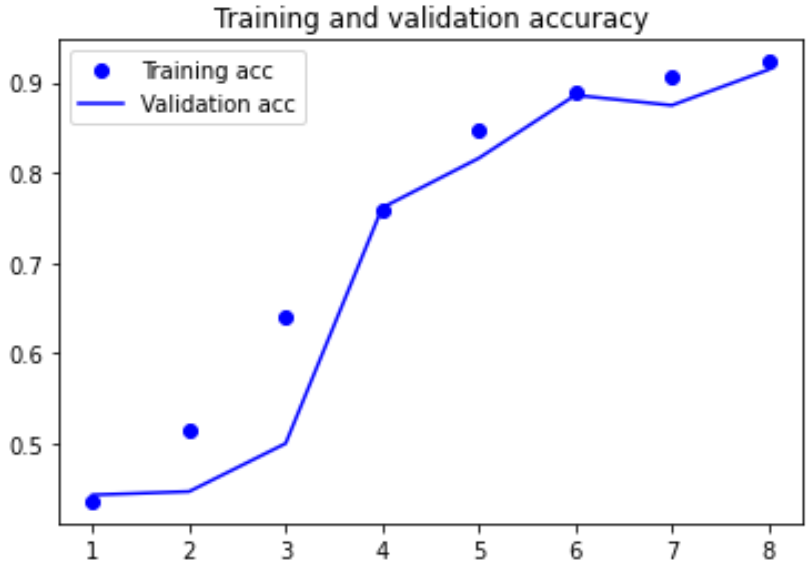
```
Epoch 1/8
17/17 [==============================] - 1s 70ms/step - loss: 4.9410 - accuracy: 0.4357 - val_loss: 1.1827 - val_accuracy: 0.4427
Epoch 2/8
17/17 [==============================] - 1s 57ms/step - loss: 0.9198 - accuracy: 0.5145 - val_loss: 0.8950 - val_accuracy: 0.4464
Epoch 3/8
17/17 [==============================] - 1s 57ms/step - loss: 0.6694 - accuracy: 0.6413 - val_loss: 0.7595 - val_accuracy: 0.4997
Epoch 4/8
17/17 [==============================] - 1s 60ms/step - loss: 0.5425 - accuracy: 0.7575 - val_loss: 0.5808 - val_accuracy: 0.7611
Epoch 5/8
17/17 [==============================] - 1s 63ms/step - loss: 0.4296 - accuracy: 0.8475 - val_loss: 0.4269 - val_accuracy: 0.8160
Epoch 6/8
17/17 [==============================] - 1s 60ms/step - loss: 0.4279 - accuracy: 0.8893 - val_loss: 0.3311 - val_accuracy: 0.8859
Epoch 7/8
17/17 [==============================] - 1s 58ms/step - loss: 0.3213 - accuracy: 0.9072 - val_loss: 0.3205 - val_accuracy: 0.8747
Epoch 8/8
17/17 [==============================] - 1s 66ms/step - loss: 0.2406 - accuracy: 0.9230 - val_loss: 0.2480 - val_accuracy: 0.9141
```

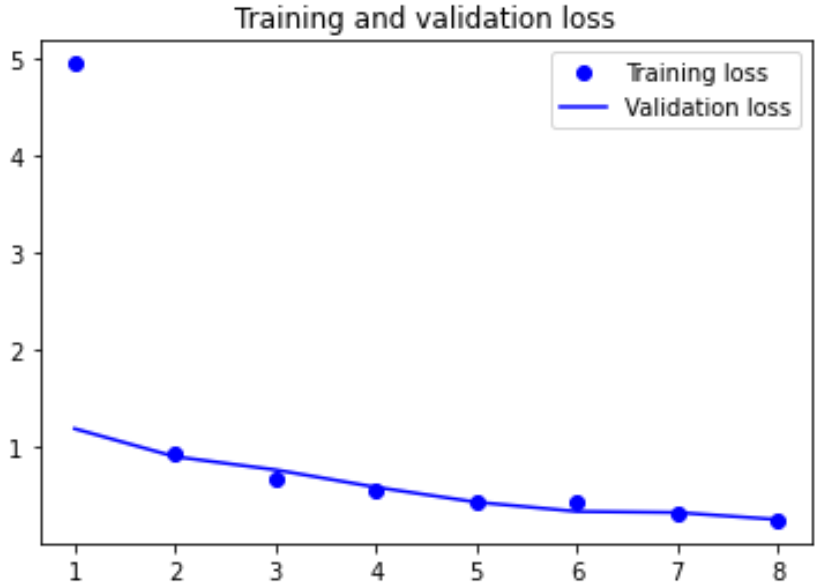
Saving the model:

```
import json

model_dir = '/content/drive/My Drive/Google_Colab/saved_models'

model_json = model.to_json()
with open(model_dir + "/P05_LSTM_model6.json", "w") as json_file:
    json_file.write(model_json)
    # serialize weights to HDF5
model.save_weights(model_dir + "/P05_LSTM_model6.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

Plotting Results:





Evaluating the model on sample tests:

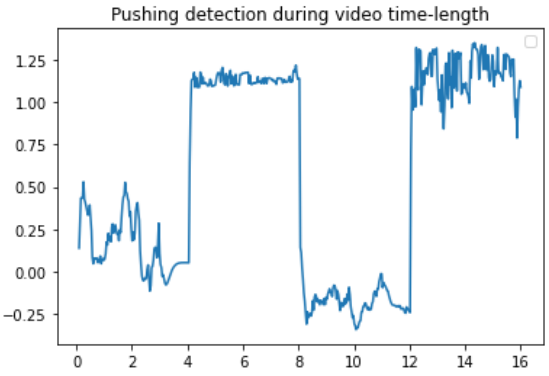
In total I had 20 short videos for test (10 pushing and 10 other). Each sample test consists 4 videos made up of 2 pushing and 2 other that are coupled as one video sample and given as x_test to evaluate the model on unseen samples. For example sample 1video consists: other+pushing+other+pushing

jsonfiles are saved with openpose and saved as csv file. Later I manually have handled them to create 5 sheets in excel each represents landmarks for video sample test. The excel file is attached in my submission. All video length are 16 seconds based on my padding.

Results sample 1:

```
scores = loaded_model.evaluate(x_test, y_test,batch_size=1)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1]*100))
```

```
4/4 [==============================] - 0s 9ms/step - loss: 0.0525 - accuracy: 0.9960
accuracy: 99.60%
```



Results sample 2:

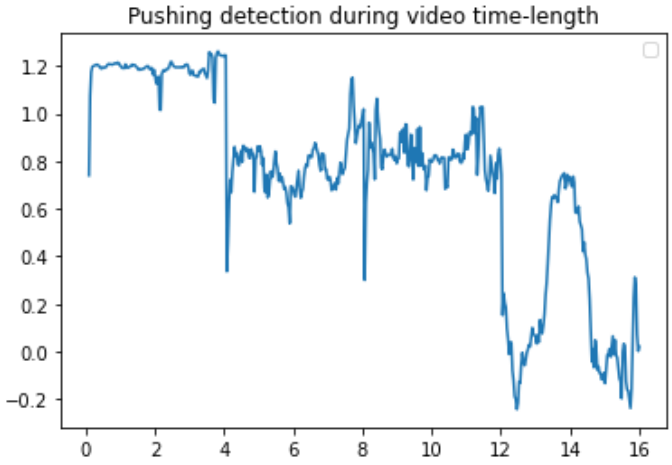
```
[119] scores = loaded_model.evaluate(x_test, y_test,batch_size=1)
      print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1]*100))
```

```
4/4 [==============================] - 0s 9ms/step - loss: 0.0396 - accuracy: 1.0000
accuracy: 100.00%
```

Results sample 3:

```
[124] scores = loaded_model.evaluate(x_test, y_test,batch_size=1)
      print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1]*100))
```

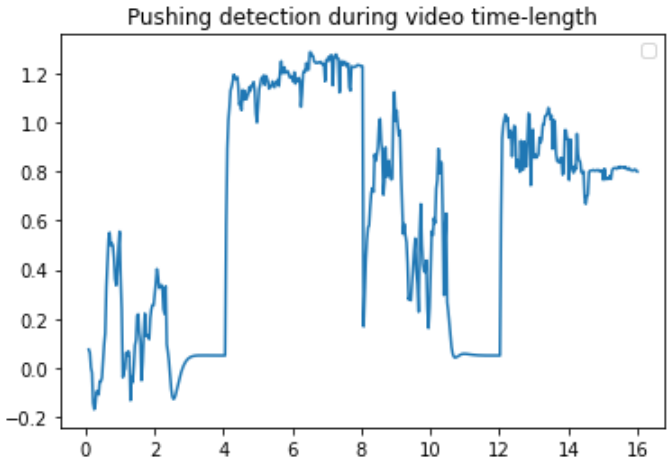
```
4/4 [==============================] - 0s 8ms/step - loss: 0.6662 - accuracy: 0.6920
accuracy: 69.20%
```



Results sample 4:

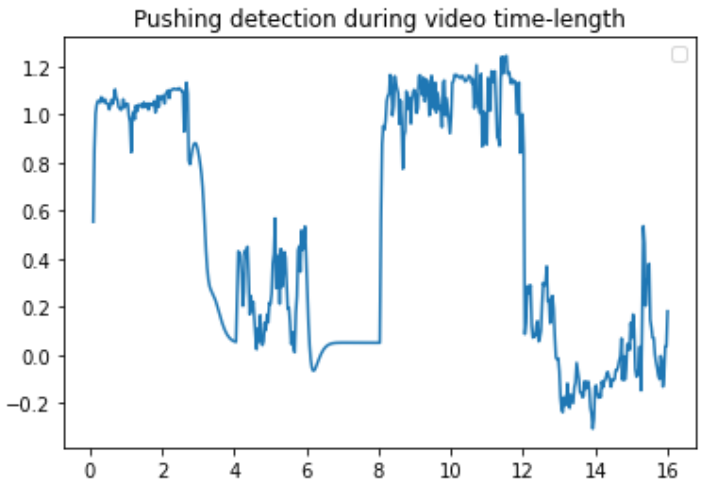
```
[129] scores = loaded_model.evaluate(x_test, y_test,batch_size=1)
      print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1]*100))
```

```
4/4 [==============================] - 0s 8ms/step - loss: 0.4065 - accuracy: 0.8840
accuracy: 88.40%
```



Results sample 5:

```
[134] scores = loaded_model.evaluate(x_test, y_test,batch_size=1)
      print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1]*100))

      4/4 [==============================] - 0s 9ms/step - loss: 0.1871 - accuracy: 0.9380
      accuracy: 93.80%
```



Conclusion and future work:

The overall results show the model can predict the unseen videos fairly well! Except sample 3, other samples show satisfying results. However, I need to see the problems and flaws of the model or the specification of samples like sample 3 that the model could not predict very well.

For the next submission I will try to make my model more complex and try to detect the flaws of my model where it fails to predict accurately.