## Final Report:

## Zohreh Shaghaghian / ProjectPart08 /action detection: **Pushing**

Date: 05/01/2020

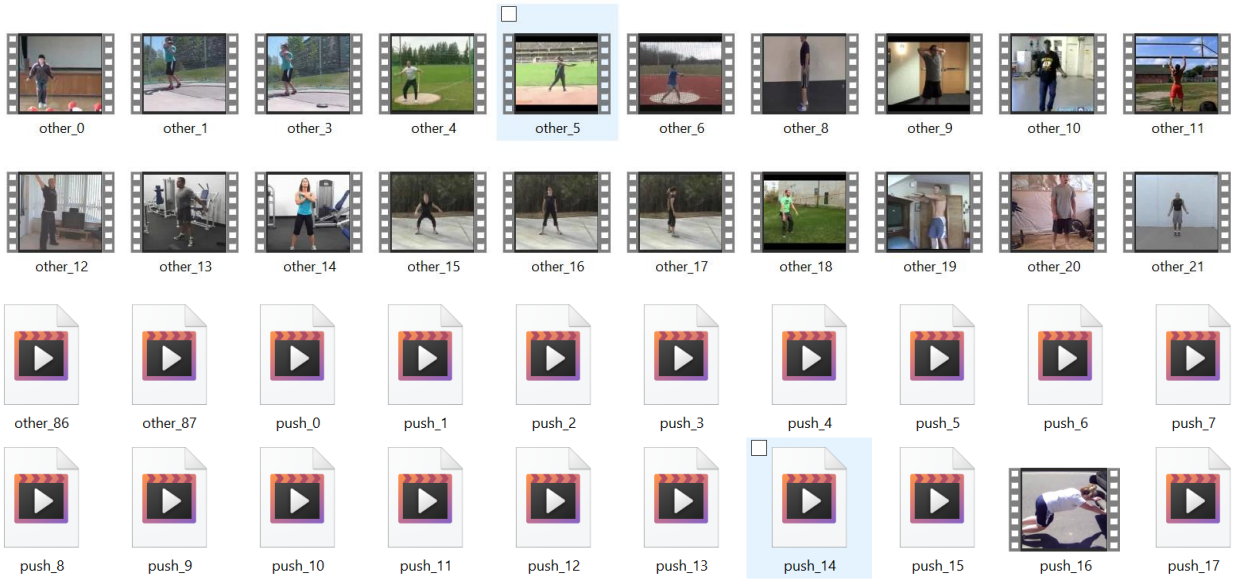
In this report a brief of my previous models, their problems and my solutions are explained besides my final model with detail.

Dataset:

I could find 30 short videos of pushing from HMDB dataset and I recorded 40 more videos of pushing by myself. I order to keep the balance, I got 72 videos from UCF dataset in which the user is doing any other activities rather than pushing.

Labeling:

In order to save time and avoid manual labeling, each video in my dataset is mono- action (either pushing or other). Hence the video name is representative of the frames' labels. (Figure_1)




**First and second model (Ref: Report 4)** → I developed two models: 1. CNN and 2. CNN+LSTM. In this step I only had 50 videos of pushing and 50 for others. And I only used video frames for training my data. In this step I did not know how to use Openpose and I used a function to capture my videos and get frames. I used VGG for transfer learning in my CNN model.

**Problems of my CNN model:** It did not consider time series and hence not converged into a good accuracy. Also CNN alone was not enough to detect figure's landmark, hence openpose was needed. Also my dataset was not enough.

**Problems of my CNN+LSTM model:** I could not use transfer learning in this step for my model. The model was not stable and accuracy changed a lot. Time series was considered in this model, but still the CNN part was not strong enough to detect the landmark features, and hence openpose was needed.

**Third model (Ref: Report 5):** → In this project I used openpose json files to get landmark features. I developed a sequential model with one LSTM layer which considered time series data.

**Problems of my model in project 05:** Although the model was very simple but I got a good accuracy. However the model still had some problems in testing new data. Specially when a figure mimic pushing action but not really touching/pushing an object. Or when I did not have enough light.

**Forth model (Ref: Report 6):** → In this project I developed an API model with two inputs. 1.json input and 2. Frame inputs. In this step, I used openpose to get both json files and captured images of each video. The model consisted of two branches: 1) Conv3D and 2) LSTM
both models are flattened and concatenated to get merged and in the end, a Dense classifier is used to detect whether the action is push (1) or other (0).

**Problems of my API model** : Although the model get run with very few data It could not get run with many data as input and I received OOM error (Out Of Memory) in Google Colab. Hence I need a generator to generate data by batches in order to avoid OOM error. Also a simple Conv3D is not strong enough and Transfer learning was needed.

# Project 08:

In this project I tried to solve the problems that I had in previous models:

1. Using both Json files and images as Input
2. Using API model with Transfer Learning
3. Using Generator to generate data by batches
4. Adding more dataset (40 more videos are recorded by me→20 for pushes and 20 for other)

**Challenge1:**
I could not use the default Keras ImageDataGenerator(), since this generator does not consider timeseries. Also, I could not use the batchsize as timesteps because each video results in different number of frames due to the length of the video, So, padding was needed to fix the number of timesteps during the data generation and the default Keras ImageDataGenerator() does not allow padding. Keras has another generator for timeseries data named as "TimeseriesGenerator", however the input data need to be 2D. Also, the problem of padding data during generation still exists. Hence I scripted a generator where I can handle timesteps and padding while generating data.

**Challenge2:**
Unlike model.fit() that can accept multiple inputs, model.fit_generator() cannot accept multiple generators by default. Hence a function is added to make it work with multiple generators.
Source_code: https://github.com/keras-team/keras/issues/8130

**Challenge 3:**
Sometimes some of the frames of my videos did not have a figure. In openpose, videos with no figure returns jsons with empty people object. I had to waive these frames when reading jsons. Although this worked pretty well if I only use json as input, it can not solve the problem if I use frames as well as another input. To solve this problem I forced the code to not only waive the json files where there is no figure but also return the names of those frames. Later when I read the frame images I added an if statement to also waive those frames with the mentioned file names.

**Challenge 4:**
The generator needed to read json files and images by batches. However, since padding problem exists (explained in challenge 1) I solved the problem by calling the frames by their video_name through passing a dictionary where key is the video name.

**Challenge 5:**
My datasets are videos of Other and pushing. The model will read all other first and then pushing. Hence, I needed to shuffle my data but if I shuffle my data (json and images) directly, the sequences will be ruined. So I shuffle the video names and then call my data (json and images) by video name. Also the validation split is done before shuffling to make sure of the balance between splitting data into train and validation list.

**Challenge 6:**
Transfer learning models (VGG16, Inception, MobileNet, … ) all have 3D input shape. However my Image input is 4D considering timesteps. The challenge is solved through TimeDistributed layer.

## Project 8 detail:

## Reading Json files with Generator
Although Json files are generally light enough and may not need generator. However, since the model is an API model and it is fit as model.fit_generator(), both input needs to be read in the same way.

Notes about generator:
1. The generator, generates data by batchsize (n-timesteps) and hence save memory
2. The generator needs to return a tuple of (x_train , target)
3. For some reason generator in Keras needs While loop (Stackoverflow)
4. A generator by itself is an object and only could be called through a for loop

Splitting data for train and validation using video name ( **Challenge 5**)

```
# input = vid_list
# output = train_list and val_list shuffled by name
def get_train_val_list(vid_list, seed=6, split=0.8):
    other = []
    push = []
    for vid_name in vid_list:
        if 'other' in vid_name:
            other.append(0)
        elif 'push' in vid_name:
            push.append(0)

    split_other = int(len(other) * split)
    split_push = int(len(push) * split)

    train_vid_list = vid_list[0:split_other] + vid_list[len(other):len(other) + split_push]
    val_vid_list = vid_list[split_other:len(other)] + vid_list[len(other) + split_push:]

    random.Random(seed).shuffle(train_vid_list)
    random.Random(seed).shuffle(val_vid_list)

    return train_vid_list, val_vid_list
```

Detecting the frames with no figure ( **Challenge 3**)

```
# input = json_file directory
# output = a dic of file_name where json people are empty.
def get_invalid_frames(json_dir):
    no_fig_files = {}
    for file in os.listdir(json_dir):
        file_path = json_dir + '/' + file
        temp = json.load(open(file_path))
        if len(temp['people']) == 0:
            parts = file.split('_')
            img_name = parts[0] + '_' + parts[1] + '_' + parts[2] + '_rendered.png' #change to png
            file_name = get_file_name(file)
            no_fig_files.setdefault(file_name, []).append(img_name)

    return no_fig_files
```

Reading Landmark features ( **challenge 4**)

```
# input = json_file directory , video_name (e.g: other_0)
# output = list of list of features (vector of 75 size) relating to all frames corresponding to video_name
def read_data_from_landmarks(json_dir,vid_name):
    landmarks = []
    for file in os.listdir(json_dir):
        if vid_name == get_file_name(file):
            file_path = json_dir + '/' + file
            temp = json.load(open(file_path))
            if len(temp['people']) == 0:
                pass
            else:
                value = temp['people'][0]['pose_keypoints_2d']
                norm_value = [float(i) / sum(value) for i in value]     # normalize the landmark features
                landmarks.append(norm_value)

    return landmarks
```

Landmark Generator (**Challenge 1**)

```
#### Land_mark generator which yeild a tuple of padded_landmarks and corresponding label
# This generator will later be called in model.fite_generator()
def landmark_generator(vid_list, json_dir, pad_len=n_timesteps):
    while True:
        for vid_name in vid_list:
            landmark = read_data_from_landmarks(json_dir, vid_name)
            padded = pad_sequences([landmark], dtype='float32', maxlen=pad_len, padding='post')
            batch = (padded, add_first_dim(encode_label(vid_name)))
            yield batch
```

## Reading captured frames with Generator (**Challenge 3 and Challenge 4**)

```
# input = image_file
# output = img_array
def read_img(img_file):
    img = cv2.imread(img_file)
    img = cv2.resize(img, dsize=(size, size), interpolation=cv2.INTER_CUBIC)
    return img


# input= files in frame_dir and vid_name
# output = arrays of all frames of one video, exluding invalid frames that are waived already in r
def get_array_of_frames(frame_dir, vid_name, json_dir):
    images = []
    invalid_frames = get_invalid_frames(json_dir)     # invalid frames of json needs to be waived
    for file in os.listdir(frame_dir):
        if vid_name == get_file_name(file):

            if vid_name in invalid_frames:
                if file in invalid_frames[vid_name]:
                    continue    #pass

            array = read_img(frame_dir + "/" + file)
            norm_array = normalized_value(array)
            images.append(norm_array)

    return images
```

Image Generator (**Challenge 1**)

```
#### image generator which yeild a tuple of padded_arrays of images and corresponding label
# This generator will later be called in model.fite_generator()
def image_generator(vid_list, frame_dir, json_dir, pad_len=n_timesteps):
    while True:
        for vid_name in vid_list:
            frames = get_array_of_frames(frame_dir, vid_name, json_dir)
            padded = pad_sequences([frames], dtype='float32', maxlen=pad_len, padding='post')
            batch = (padded, add_first_dim(encode_label(vid_name)))
            yield batch
```

# Model Configuration

Model API using Transfer Learning (using MobileNet) for Image inputs (**Challenge 6**)

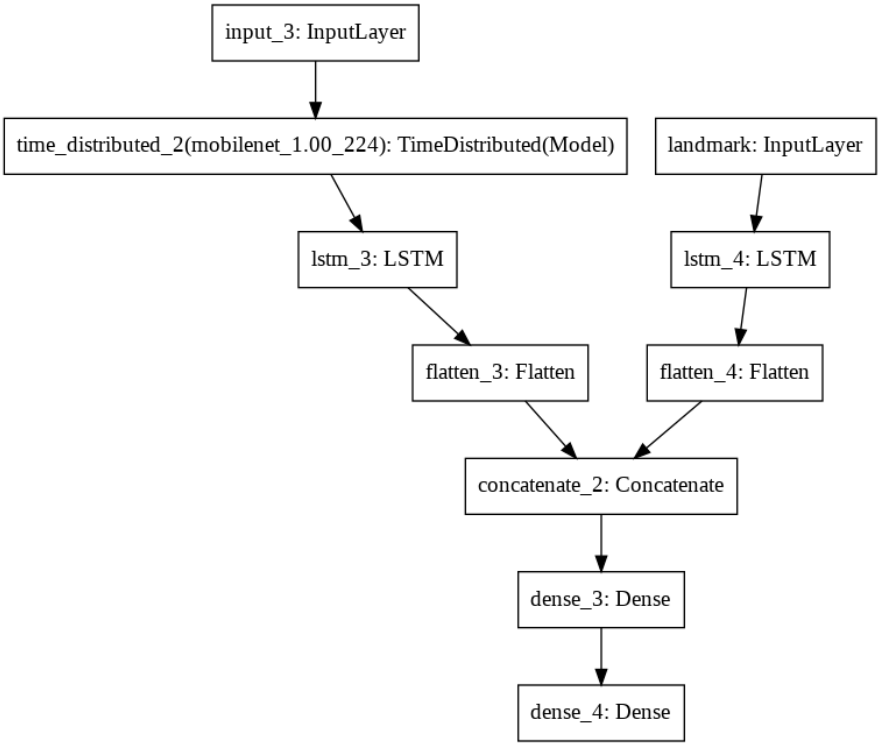
```
# Json
landmark_input = Input(shape=(max_frame, vector_size),
                       name='landmark')
lstm1 = LSTM(units=cells, dropout=0.1, recurrent_dropout=0.5, return_sequences=True)(landmark_input)
flat1 = Flatten()(lstm1)

# image
image_input = Input(shape=(max_frame, size, size, 3), name = 'image')
base_model = MobileNet(input_shape=(size, size, 3),
                       include_top=False,
                       weights='imagenet',
                       input_tensor=None,
                       pooling='avg',
                       classes=2)

im = TimeDistributed(base_model)(image_input)
im = LSTM(units=cells, return_sequences=True)(im)
flat2 = Flatten()(im)

merge = concatenate([flat1, flat2])
#dr = Dropout(0.5)(merge)

hidden = Dense(1, activation='relu')(merge)
output = Dense(1, activation='sigmoid')(hidden)    # softmax
```

```
model = Model([landmark_input,image_input], output)
print(model.summary())

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Model Parameters:

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
image (InputLayer)           (None, 50, 168, 168, 0
_________________________________________________________________________________________________
landmark (InputLayer)        (None, 50, 75)            0
_________________________________________________________________________________________________
time_distributed_3 (TimeDistrib (None, 50, 1024)       3228864     image[0][0]
_________________________________________________________________________________________________
lstm_6 (LSTM)                (None, 50, 2)             624         landmark[0][0]
_________________________________________________________________________________________________
lstm_7 (LSTM)                (None, 50, 2)             8216        time_distributed_3[0][0]
_________________________________________________________________________________________________
flatten_5 (Flatten)          (None, 100)               0           lstm_6[0][0]
_________________________________________________________________________________________________
flatten_6 (Flatten)          (None, 100)               0           lstm_7[0][0]
_________________________________________________________________________________________________
concatenate_3 (Concatenate)  (None, 200)               0           flatten_5[0][0]
                                                                   flatten_6[0][0]
_________________________________________________________________________________________________
dense_5 (Dense)              (None, 1)                 201         concatenate_3[0][0]
_________________________________________________________________________________________________
dense_6 (Dense)              (None, 1)                 2           dense_5[0][0]
=================================================================================================
Total params: 3,237,907
Trainable params: 3,216,019
Non-trainable params: 21,888
_________________________________________________________________________________________________
None
```

Model Architecture:



Calling the input generators:

```
[12] base_dir = "/content/drive/My Drive/Google_Colab"
     json_dir = base_dir + '/' + 'openpose_json'
     frame_dir = base_dir + '/' + 'video_frames'
     video_dir = base_dir + '/' + 'video'

     vid_list = get_vidlist (video_dir)
     train_list, val_list = get_train_val_list(vid_list, split=0.8)

     # calling json generator for train and validation
     train_json_generator = landmark_generator(train_list, json_dir)
     val_json_generator = landmark_generator(val_list, json_dir)

     # calling image generator for train and validation
     train_img_generator = image_generator(train_list, frame_dir,json_dir)
     val_img_generator = image_generator(val_list, frame_dir,json_dir)

     # calling generate_multiple_generator to get train_gen and val_gen
     train_gen = generate_multiple_generator(train_json_generator,train_img_generator)
     val_gen = generate_multiple_generator(val_json_generator, val_img_generator)
```

Creating multiple generator to be used in model.fit_generator() (**Challenge2** )

```
[11] def generate_multiple_generator(genX1, genX2):
         while True:
             X1i = next(genX1)
             X2i = next(genX2)
             yield [X1i[0], X2i[0]], X1i[1]
```

Fit the model with 2 generators:

```
epochs = 5

if __name__ == '__main__':
    history=model.fit_generator(generator = train_gen,
                        steps_per_epoch=len(train_list),
                        epochs = epochs,
                        validation_data = val_gen,
                        validation_steps = len(val_list),
                        use_multiprocessing=False,
                        shuffle=False)   #, workers = 2     and use_multiprocessing=True for Colab
```

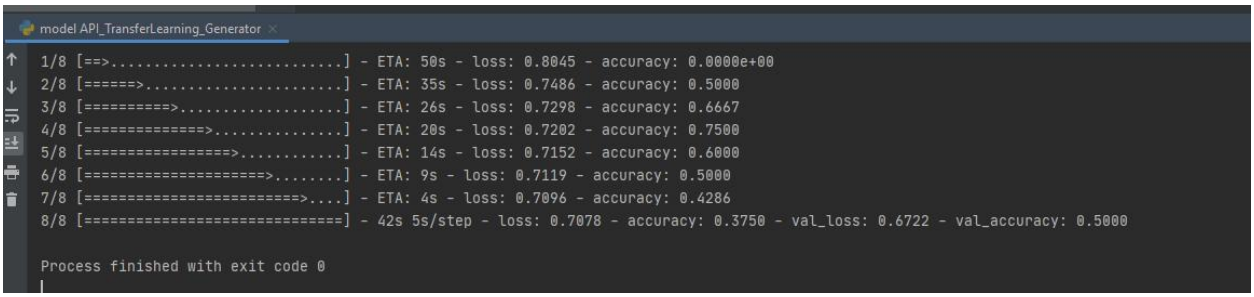
**Result:**

Although I have generator The code cannot be run in Google Colab and get “Stop Iteration” error.

```
Epoch 1/5
/usr/local/lib/python3.6/dist-packages/keras/utils/data_utils.py:718: UserWarning: An input could not be retrieved. It could be because a worker has died.We do not ha
  UserWarning)
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-28-d25f12be69be> in <module>()
      9                         validation_steps = len(val_list),
     10                         use_multiprocessing=False,
---> 11                         shuffle=False)   #, workers = 2     and use_multiprocessing=True for Colab
     12

                                 2 frames
/usr/local/lib/python3.6/dist-packages/keras/engine/training_generator.py in fit_generator(model, generator, steps_per_epoch, epochs, verbose, callbacks, validation_d
    183             batch_index = 0
    184             while steps_done < steps_per_epoch:
--> 185                 generator_output = next(output_generator)
    186
    187                 if not hasattr(generator_output, '__len__'):

StopIteration:
```

I searched the error and try to change my hyperparameters but got the same error.

However in order to check the code, I ran it on my laptop (CPU) but I needed to highly reduce the number of train and validation (5 video for train and 1 for validation). I aslo highly reduced number of frames to consider ( instead of 125 , I consider only 10 frames) and considering only one cell for LSTM layer. And running the model for 8 epochs.
It is obvious that the accuracy is not high in this case but it shows that the model can get run without error. The problem is related with Google Colab which unfortunately I could not figure out yet!




So, in conclusion,  my best model is still my model submitted in project 05 with 96 accuracy.

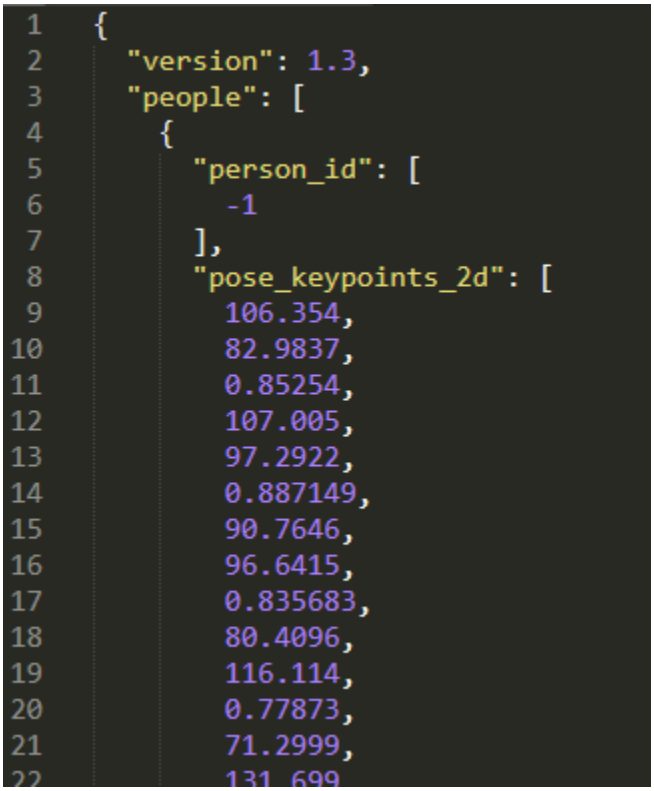
The report is pasted here again:

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'

```python
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.

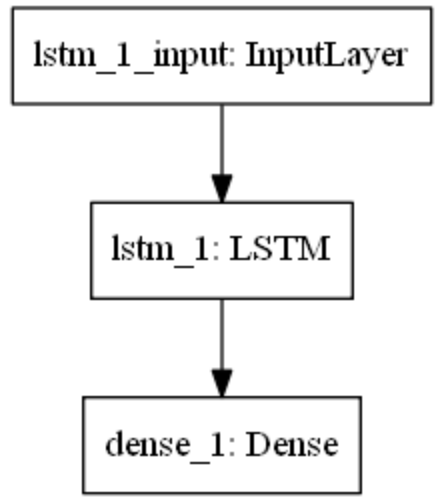
1. Input_shape (n_timesteps,n_features) → this should match the shape of my x_train (100, 125,75)
   Note that 100 is number of my videos and it will be later fed during fed through batch_size
2. return_sequences = True → This Argument plays crucial role in model training and keeping the time_steps in 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)

```python
## 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)
```

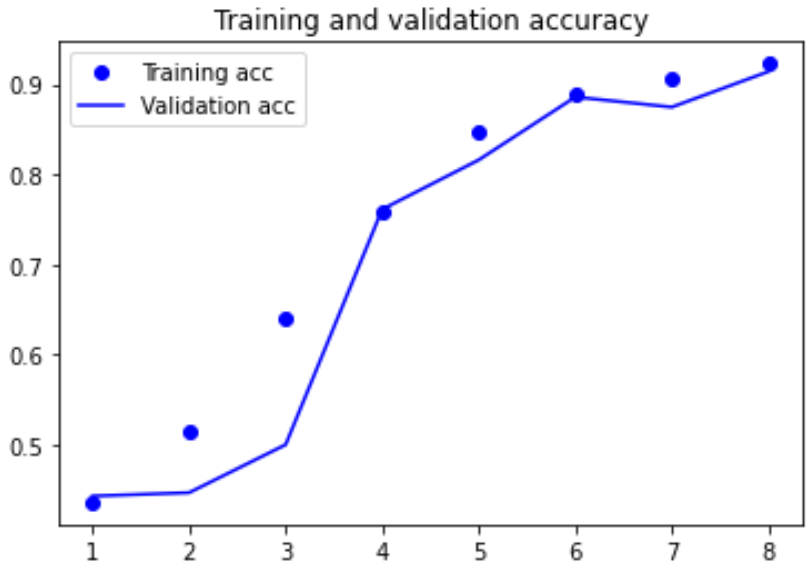
```
Epoch 1/8
17/17 [==============================] - 1s 70ms/step - loss: 4.9410 - accuracy: 0.4357 - val_loss: 1.1827 - val_accuracy: 0.4427
Epoch 2/8
17/17 [==============================] - 1s 57ms/step - loss: 0.9198 - accuracy: 0.5145 - val_loss: 0.8950 - val_accuracy: 0.4464
Epoch 3/8
17/17 [==============================] - 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
```

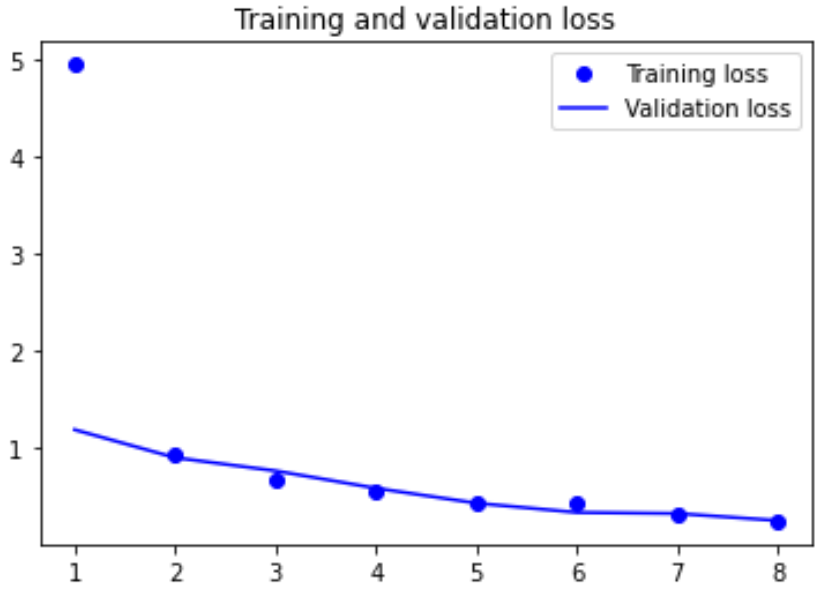
Saving the model:

```
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:

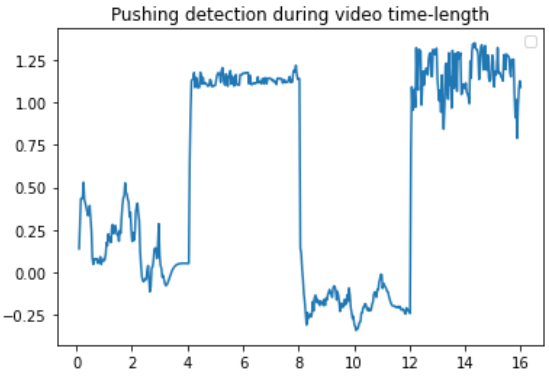
In total I had 20 short videos for test (10 pushing and 10 other). Each sample test consists 4 videos made up of 2 pushing and 2 other that are coupled as one video sample and given as x_test to evaluate the model on unseen samples. For example sample 1video consists: other+pushing+other+pushing

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:

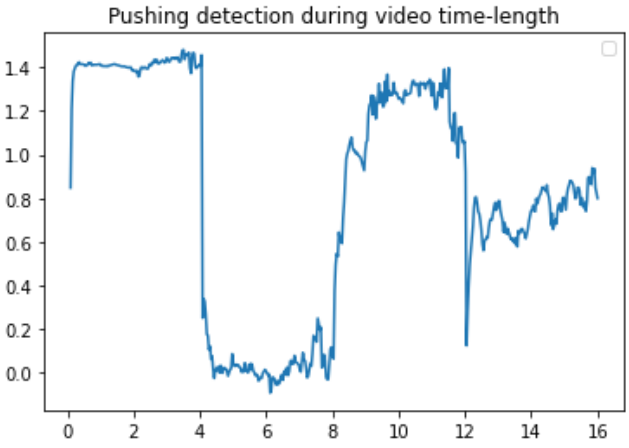
```
[119] scores = loaded_model.evaluate(x_test, y_test,batch_size=1)
      print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1]*100))
```

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


Results sample 3:

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

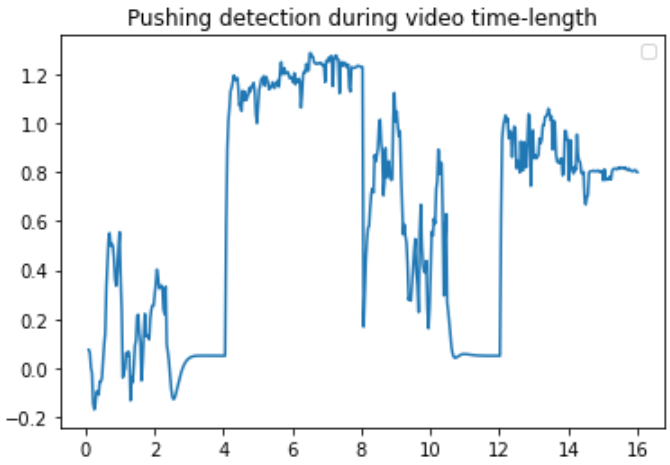
```
4/4 [==============================] - 0s 9ms/step - loss: 0.3724 - accuracy: 0.7540
accuracy: 75.40%
```




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%
```