```
import tensorflow as tf
tf.test.gpu_device_name()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
```

```
!ln -sf /opt/bin/nvidia-smi /usr/bin/nvidia-smi
!pip install gputil
!pip install psutil
!pip install humanize
import psutil
import humanize
import os
import GPUtil as GPU
GPUs = GPU.getGPUs()
# XXX: only one GPU on Colab and isn’t guaranteed
gpu = GPUs[0]
def printm():
 process = psutil.Process(os.getpid())
 print("Gen RAM Free: " + humanize.naturalsize( psutil.virtual_memory().available ), " | Proc
 print("GPU RAM Free: {0:.0f}MB | Used: {1:.0f}MB | Util {2:3.0f}% | Total {3:.0f}MB".format(
printm()
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
from __future__ import print_function
import numpy as np
import glob
np.random.seed(1337)
```

```
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.layers import Input, LSTM
from tensorflow.python.keras.utils import np_utils
from tensorflow.python.keras.callbacks import EarlyStopping

import tensorflow as tf

import matplotlib.pyplot as plt
%matplotlib inline

import datetime
now = datetime.datetime.now

batch_size = 10

# length of data sequences
n_timesteps = 240
# dimension of data sequences
```

```
n_dim = 117
```

1. Loading Data

```
def reading_files():
    all_files_corrected = glob.glob("/content/drive/My Drive/DeepLearningProject/Segmented Mo
    all_files_incorrected = glob.glob("/content/drive/My Drive/DeepLearningProject/Segmented
    return all_files_corrected,all_files_incorrected


all_files_corrected , all_files_incorrected =reading_files()


all_files_corrected = sorted(all_files_corrected)
all_files_incorrected = sorted(all_files_incorrected)


import csv
import numpy as np

def load_data(corrected_file, incorrected_file):
    f = open( corrected_file )
    csv_f = csv.reader(f)
    X_Corr = list(csv_f)

    # Convert the input sequences into numpy arrays
    train_input1 = np.asarray(X_Corr)
    n_dim = 117
    data_correct = np.zeros((90,240,n_dim))
    for i in range(len(train_input1)//n_dim):
          data_correct[i,:,:] = np.transpose(train_input1[n_dim*i:n_dim*(i+1),:])

    f = open( incorrected_file )
    csv_f = csv.reader(f)
    X_Incor = list(csv_f)

    # Convert the input sequences into numpy arrays
    train_input2 = np.asarray(X_Incor)
    n_dim = 117
    data_incorrect = np.zeros((90,240,n_dim))
    for i in range(len(train_input2)//n_dim):
          data_incorrect[i,:,:] = np.transpose(train_input2[n_dim*i:n_dim*(i+1),:])

    return data_correct, data_incorrect


exercises_names = [ "deep squat",
                   "hurdle step",
                   "inline lunge",
                   "side lunge",
                   "sit to stand",
```

```
                    "standing active straight leg raise",
                    "standing shoulder abduction",
                    "standing shoulder extension",
                    "standing shoulder internal-external rotation",
                    "standing shoulder scaption"]


print(len(all_files_incorrected),len(all_files_corrected),len(exercises_names))


def data_setting(index):
  X_correct, X_incorrect = load_data(all_files_corrected[index],all_files_incorrected[index])
  data_correct = np.zeros((X_correct.shape[0],n_timesteps+100,n_dim))
  for i in range(X_correct.shape[0]):
      data_correct[i,:,:] = np.concatenate((np.concatenate((np.tile(X_correct[i,0,:],[50, 1])

  data_incorrect = np.zeros((X_incorrect.shape[0],n_timesteps+100,n_dim))
  for i in range(X_incorrect.shape[0]):
      data_incorrect[i,:,:] = np.concatenate((np.concatenate((np.tile(X_incorrect[i,0,:],[50,
  return data_correct,data_incorrect
```

2. Adding Frames

```
def adding_frames(X_correct,X_incorrect, exercise_num ):
    # Add 50 time frames at the beginning and end of sequences
  # The autoencoder has difficulties with the beginning and ending frames
  data_correct = np.zeros((X_correct.shape[0],n_timesteps+100,n_dim))
  for i in range(X_correct.shape[0]):
      data_correct[i,:,:] = np.concatenate((np.concatenate((np.tile(X_correct[i,0,:],[50, 1])

  data_incorrect = np.zeros((X_incorrect.shape[0],n_timesteps+100,n_dim))
  for i in range(X_incorrect.shape[0]):
      data_incorrect[i,:,:] = np.concatenate((np.concatenate((np.tile(X_incorrect[i,0,:],[50,

  # Plot the first sequences of correct and incorrect data
  plt.figure(figsize = (12,6))
  # plt.title()
  ax = plt.subplot(1,2,1)
  ax.set_title(' Sequence of correct data  for exercise \n'+exercises_names[exercise_num] ,fo
  plt.plot(data_correct[0])
  plt.ylim([-1,1])
  ax2 = plt.subplot(1,2,2)
  ax2.set_title(' Sequence of incorrect data for exercise \n'+exercises_names[exercise_num] ,
  plt.plot(data_incorrect[0])
  plt.ylim([-1,1])
  plt.tight_layout()
  plt.savefig("/content/drive/My Drive/DeepLearningProject/Figures/pos_m0"+str(exercise_num+1
  plt.show()
```

3. Ploting seqences of correct and incorrect

```
def sequence_of_the_exercises(files_corrected,files_incorrected , exercise_num):
  X_correct, X_incorrect = load_data( files_corrected , files_incorrected)
  print(X_correct.shape, 'correct sequences')
  print(X_incorrect.shape, 'incorrect sequences')

  # Plot the first sequences of correct and incorrect data
  plt.figure(figsize = (12,6))
  plt.subplot(1,2,1)
  plt.plot(X_correct[0])
  plt.ylim([-1,1])
  plt.subplot(1,2,2)
  plt.plot(X_incorrect[0])
  plt.ylim([-1,1])
  plt.tight_layout()
  plt.show()
  # adding_frames(X_correct,X_incorrect ,exercise_num )
```

4. Model Building

```
# Encoder layers
input_seq = Input(shape=(n_timesteps+100,n_dim))
encoded1 = LSTM(30,return_sequences = True)(input_seq)
encoded2 = LSTM(10,return_sequences = True)(encoded1)
# Encoded representation of the input, 340x4 vector
encoded = LSTM(4,return_sequences = True)(encoded2)
# Decoder layers
decoded1 = LSTM(10,return_sequences = True)(encoded)
decoded2 = LSTM(30,return_sequences = True)(decoded1)
decoded = LSTM(n_dim, return_sequences = True)(decoded2)

# The model maps an input to its reconstruction
autoencoder = Model(inputs=input_seq, outputs=decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.summary()


def running_models(index):
  data_correct , data_incorrect = data_setting(index)
  import random
  trainidx = random.sample(range(0,data_correct.shape[0]),63)
  valididx = np.setdiff1d(np.arange(0,90,1),trainidx)
  train_data = data_correct[trainidx,:,:]
  valid_data = data_correct[valididx,:,:]
  import os
```

```
import os
# Directory where the checkpoints will be saved
# checkpoint_dir = '/content/drive/My Drive/DeepLearningProject/trainingcheckpoints/m0'+str
# # Name of the checkpoint files
# checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

# checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
#     filepath=checkpoint_prefix,
#     save_weights_only=True)

# Train an autoencoder on the correct data sequences

# Measure the training time
t = now()

# Request to stop before reaching the number of epochs if the validation loss does not decr
early_stopping = EarlyStopping(monitor='val_loss', patience = 1000)

history = autoencoder.fit(train_data, train_data, epochs = 10000, batch_size = batch_size,
                validation_data=(valid_data, valid_data), verbose = 0, callbacks = [early_s

print('Training time: %s' % (now() - t))
loss(history,index)

# Encode and decode sequences to check the model performance
decoded_seqs = autoencoder.predict(data_correct)
decode_Sequence_Plot(decoded_seqs,index,data_correct)
  # Create an encoder model, that maps an input to its encoded representation
encoder = Model(inputs=input_seq, outputs=encoded)

# Test the encoder model
encoded_seqs = encoder.predict(data_correct)

encoded_sequence_plot(encoded_seqs,index,data_correct)
time_frame_autencoder_plot(encoded_seqs,index,data_correct)

# Remove the added first and last 50 frames
encoded_seqs = encoded_seqs[:,50:-50,:]

print(encoded_seqs.shape, 'encoded sequences shape')
# Reshape the encoded sequences, because savetxt saves two dimensional data
seqs = encoded_seqs.reshape(encoded_seqs.shape[0],encoded_seqs.shape[1]*encoded_seqs.shape[
print(seqs.shape, 'encoded sequences shape for saving')
# Save the data in the file 'Autoencoder_Output_Correct.csv'
np.savetxt('/content/drive/My Drive/DeepLearningProject/Segmented Movements/Data_Proccessed
#files.download('/content/drive/My Drive/DeepLearningProject/Segmented Movements/Data_Procc

# Reduce the dimensionality of the incorrect sequences
encoded_seqs_incorrect = encoder.predict(data_incorrect)

# Remove the added first and last 50 frames
encoded_seqs_incorrect = encoded_seqs_incorrect[:,50:-50,:]
```

```
  print(encoded_seqs_incorrect.shape, 'encoded incorrect sequences shape')
  # Reshape the encoded sequences, because savetxt saves only tow dimensional data
  seqs_incorrect = encoded_seqs_incorrect.reshape(encoded_seqs_incorrect.shape[0],encoded_seq
  print(seqs_incorrect.shape, 'encoded incorrect sequences shape for saving')
  # Save the incorrect data in the file 'Autoencoder_Output_Incorrect.csv'
  np.savetxt('/content/drive/My Drive/DeepLearningProject/Segmented Movements/Data_Proccessed
  #files.download('/content/drive/My Drive/DeepLearningProject/Segmented Movements/Data_Procc
```

5 . Ploting loss , encoders output, decoder output and autoenc

```
def loss(history , index):
  from google.colab import files
  plt.figure()
  plt.subplot(121)
  plt.plot(history.history['loss'])
  plt.title('Loss for \n'+str(exercises_names[index]))
  plt.subplot(122)
  plt.plot(history.history['val_loss'])
  plt.title('Validation Loss for \n'+str(exercises_names[index]))
  plt.tight_layout()
  plt.savefig("/content/drive/My Drive/DeepLearningProject/Figures/loss_pos_m0"+str(index+1)+
  plt.show()
  #files.download('/content/drive/My Drive/DeepLearningProject/Figures/loss_m0'+str(index+1)+
  # Print the resulting training and validation loss values
  print(history.history['loss'][-1])
  print(history.history['val_loss'][-1])


def decode_Sequence_Plot(decoded_seqs,index,data_correct):

  # Plot the results
  n = 2  # how many sequences we will display
  plt.figure(figsize = (12,6))
  for i in range(n):
      # display original sequences
      plt.subplot(n, 2, 2*i+1)
      plt.plot(data_correct[i])
      plt.title('original sequences \n'+str(exercises_names[index]))
      # display reconstruction
      plt.subplot(n, 2, 2*i+2)
      plt.plot(decoded_seqs[i])
      plt.title('decoded sequences \n'+str(i+1)+" : "+str(exercises_names[index]))
  plt.tight_layout()
  plt.savefig("/content/drive/My Drive/DeepLearningProject/Figures/original_seq_vs_decoded_se
  #files.download("/content/drive/My Drive/DeepLearningProject/Figures/original_seq_vs_decode
  plt.show()
```

```
def encoded_sequence_plot(encoded_seqs,index,data_correct):
  # Plot the results
  n = 2  # how many sequences we will display
  plt.figure(figsize = (12,6))
  for i in range(n):
      # display original sequences
      plt.subplot(n, 2, 2*i+1)
      plt.plot(data_correct[i])
      plt.title('original sequences \n'+str(exercises_names[index]))
      # display reconstruction
      plt.subplot(n, 2, 2*i+2)
      plt.plot(encoded_seqs[i])
      plt.title('encoded sequences \n'+str(i+1)+" :"+str(exercises_names[index]))
  plt.tight_layout()
  plt.savefig("/content/drive/My Drive/DeepLearningProject/Figures/original_seq_vs_encoded_se
  #files.download("/content/drive/My Drive/DeepLearningProject/Figures/original_seq_vs_encode
  plt.show()


def time_frame_autencoder_plot(encoded_seqs,index,data_correct):
  plt.figure(figsize = (14,14))
  for i in range(data_correct.shape[0]):
      plt.subplot(4,1,1)
      plt.plot(encoded_seqs[i,50:-50,0])
      plt.xlabel('Time Frame',fontsize=12)
      plt.ylabel('Angle (Degrees)',fontsize=12)
      plt.subplot(4,1,2)
      plt.plot(encoded_seqs[i,50:-50,1])
      plt.xlabel('Time Frame',fontsize=12)
      plt.ylabel('Angle (Degrees)',fontsize=12)
      plt.subplot(4,1,3)
      plt.plot(encoded_seqs[i,50:-50,2])
      plt.xlabel('Time Frame',fontsize=12)
      plt.ylabel('Angle (Degrees)',fontsize=12)
      plt.subplot(4,1,4)
      plt.plot(encoded_seqs[i,50:-50,3])
      plt.xlabel('Time Frame',fontsize=12)
      plt.ylabel('Angle (Degrees)',fontsize=12)
  plt.tight_layout()
  plt.savefig("/content/drive/My Drive/DeepLearningProject/Figures/autoencoder_output_pos_m0"
  #files.download("/content/drive/My Drive/DeepLearningProject/Figures/autoencoder_output_m0"
  plt.show()
```

. 6 Running Models

```
for i in range(0,10):
  running_models(i)
```