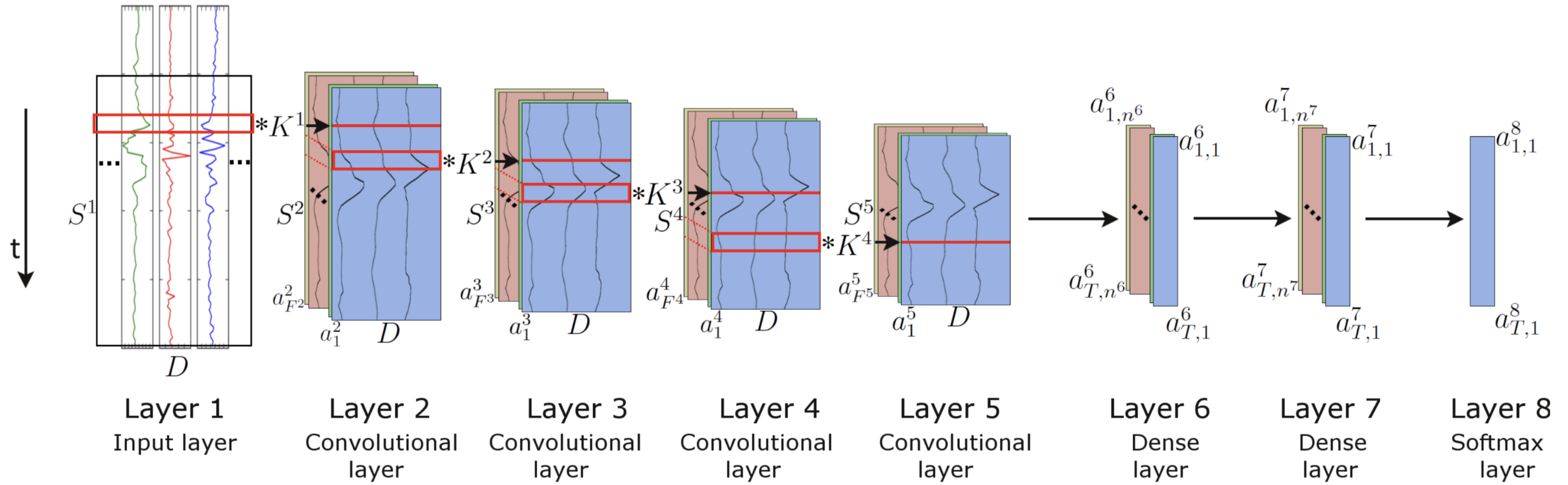
Using CNN_LSTM for Time Series Classification also Prediction

References: https://arxiv.org/abs/1411.4389 https://www.mdpi.com/1424-8220/16/1/115/html

Combine Deel CNN and LSTM



LSTM network models are a type of recurrent neural network that are able to learn and remember over long sequences of input data. They are intended for use with data that is comprised of long sequences of data, up to 200 to 400 time steps.

The CNN model learns to map a given window of signal data from each axis Accelerometer where the model reads across each window of data and prepares an internal representation of the window.

The CNN LSTM model will read subsequences of the main sequence in as blocks, extract features from each block, then allow the LSTM to interpret the features extracted from each block.

IMPORT LIBRARY

```
import numpy as np
import pandas as pd
import matplotlib
```

```
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn import metrics
from numpy import mean
from numpy import std #(standard deviation)
from tensorflow import keras
import os
from __future__ import print_function

#also Using KERAS FOR RNN (LSTM Cell)
from keras.models import Sequential
from keras.layers import Dense
import seaborn as sns
from scipy import stats
from pylab import rcParams
from sklearn import metrics
from sklearn.model_selection import train_test_split

#import for CNN_LSTM

from numpy import dstack
from keras.layers import Dropout,Flatten, Reshape,Dense, TimeDistributed
from keras.layers import LSTM
from keras.layers.convolutional import Conv2D,Conv1D
from keras.layers.convolutional import MaxPooling1D, MaxPooling2D
from keras.utils import to_categorical
from keras.utils import np_utils
from __future__ import absolute_import, division, print_function, unicode_literals
from sklearn.metrics import classification_report
#import for filter
from scipy import signal
```

```
!pip install h5py pyyaml
!pip install tf_nightly
```

IMPORT DATASET

```
from google.colab import files
upload = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving processingdatafilter1.csv to processingdatafilter1.csv

DATA PREPROCESSING

Low Pass Filter

```
dataset= pd.read_csv('VeryHIGH_movement.csv')
dataset
```

```
#@author: Yi Yu

D = 'processingdataset_Disaster_prevention.csv'

def butter_lowpass(cutoff, nyq_freq, order=4):
    normal_cutoff = float(cutoff) / nyq_freq
    b, a = signal.butter(order, normal_cutoff, btype='lowpass')
    return b, a

def butter_lowpass_filter(data, cutoff_freq, nyq_freq, order=4):
    b, a = butter_lowpass(cutoff_freq, nyq_freq, order=order)
    y = signal.filtfilt(b, a, data)
    return y

acceler = ['x1','y1','z1','x2','y2','z2']


data = pd.read_csv('VeryHIGH_movement.csv')
sample_rate = 50

for i in acceler :

  x = data[i]
  signal_length = len(x)

  # Filter signal x, result stored to y:
```

```
  cutoff_frequency = 0.5                                                                                         ######### CAN be CHANGED
  y = butter_lowpass_filter(x, cutoff_frequency, sample_rate/2)

  # Difference acts as a special high-pass from a reversed butterworth filter.
  diff = np.array(x)-np.array(y)

  plt.figure(figsize = (6,3))
  plt.plot(x, color='red', label="Original signal, {} samples".format(signal_length))

  plt.figure(figsize = (6,3))
  plt.plot(y, color='blue', label="Filtered low-pass with cutoff frequency of {} Hz".format(cutoff_frequency))

  plt.figure(figsize = (6,3))
  plt.plot(diff, color='gray', label="What has been removed")

  plt.legend()
  plt.show()

df= pd.DataFrame(data=y)
df.to_csv('low_pass_filter.csv')

  # Visualize

```

Moving Avrage

```
#@author: Yi-Yu

import warnings
from scipy import signal
import math

data = pd.read_csv('low_pass_filter.csv')

X = data['0']

window=[7]                                                        ######### CAN BE CHANGED

for i in window:
    rolling = X.rolling(window=i)
    rolling_mean = rolling.mean()
```

```
    rolling_mean = rolling.mean()
    True_rolling_mean1 = rolling_mean

    for j in range(0,i):

        True_rolling_mean1.iloc[j] = X[j]

plt.figure(figsize=(6,3))
plt.plot(True_rolling_mean1[:])
plt.show()

df= pd.DataFrame(data=True_rolling_mean1)
df.to_csv('after_moving_average.csv')
```

LABEL DATA

```
dataset= pd.read_csv('processingdatafilter1.csv')
dataset = dataset.iloc[:, 1:3]
dataset.size
```

```
13284
```

```
dataset.head()
```

```
#Look Backstep to create the dataset (Decreasing or increasing number of step depend of the raw dataset)
look_back_step = 10
total_size_of_dataset = dataset.size-look_back_step
threshold = 0.005
threshold1=0.04
threshold2=0.1

memory_of_label= list()
# we should adding more threshold to our model
```

```
for i in range(total_size_of_dataset):

  if(abs(dataset[look_back_step+i])-abs(dataset[i])>=threshold):
    memory_of_label.append('slow_movement')
  elif((abs(dataset[look_back_step+i])-abs(dataset[i])>=threshold1):
```

```
    memory_of_label.append('high')
  elif((abs(dataset[look_back_step+i])-abs(dataset[i])>=threshold2):
    memory_of_label.append('very_high')
  else:
    memory_of_label.append('stable')
```

```
#Reviewing data and saving the file
print(memory_of_label)
df= pd.DataFrame(data=memory_of_label)
df.to_csv('label.csv')
!ls
```

IMPORT DATA

```
# load a single file as a numpy array

df = pd.read_csv('processingdatafilter1.csv')
df.head(10)


```

	AcX	AcY	AcZ	Lable
0	1.706447	1.998895	1.716803	stable
1	1.706926	1.999394	1.717309	stable
2	1.707550	2.000055	1.717906	stable
3	1.708214	2.000779	1.718519	stable
4	1.708793	2.001436	1.719052	stable
5	1.709161	2.001892	1.719406	stable
6	1.709219	2.002036	1.719501	stable
7	1.708921	2.001808	1.719294	stable
8	1.708278	2.001214	1.718793	stable
9	1.707361	2.000328	1.718057	stable

PREPARING & PROCESSING INPUT TO MODEL

Transfer All data to Numeric Before Feeding to model

```
from sklearn import preprocessing
# Define column name of the label vector

LABEL = 'LableEncoder'
# Transform the labels from String to Integer via LabelEncoder
le = preprocessing.LabelEncoder()
# Add a new column to the existing DataFrame with the encoded values
df[LABEL] = le.fit_transform(df['Lable'].values.ravel())
df[LABEL]


'''
#The Simple way label by ourself

# Classify How many special object
df["Lable"].unique()
# Taking all the unique data to abitrary number
df["Lable"].astype("category").cat.codes
#Map the dataste into dictionary
Lable_class_dict={"stable":1, "movement":2, "highly movement ": 3, "SLOWMOVEMNT":4}
# ENCODE THEM INTO THE NUMBER
df['Lable'] = df['Lable'].map(Lable_class_dict)
Label= 'Lable'
df['Lable']
df.head()
'''
```

```
'\n#The Simple way label by ourself\n\n# Classify How many special object\ndf["Lable"].unique()\n# Taking all the unique data to abitra
```

** Look at my data**

[1]Cleaning Data [2] Inspect Data to See Corelation of Each Column & Statistic The data **[3]Split Data {Testing, Training} [4] Normaliz The Data to 0---->1

- How many rows are in the dataset?
- How many columns are in this dataset?
- What data types are the columns?
- Is the data complete? Are there nulls? Do we have to infer values?
- What is the definition of these columns?

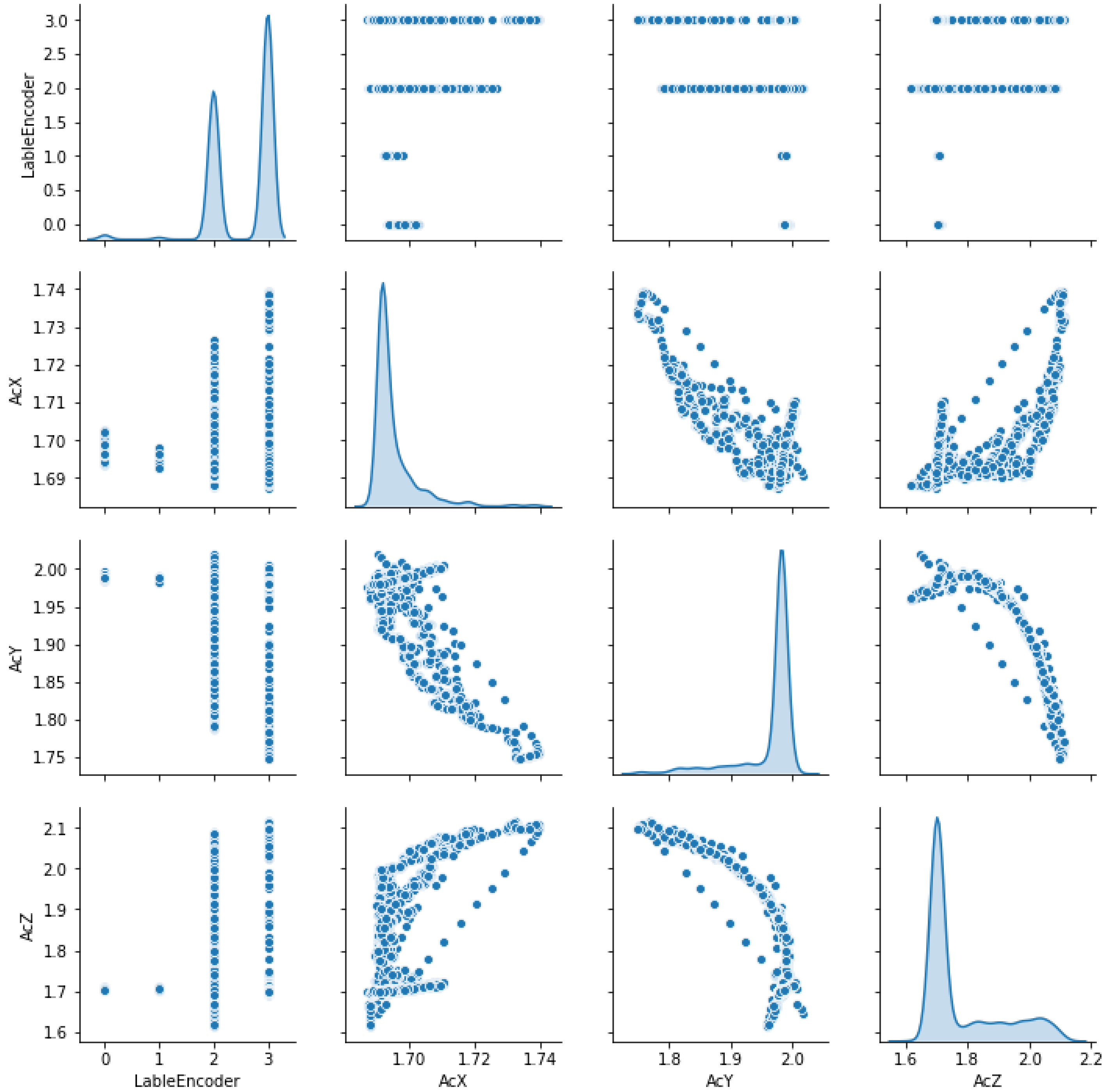
```
# CLEAN DATA
  #The Dataset contains a few Unkowns values
df = df.dropna()
df.isna().sum()


```

```
from sklearn.utils import shuffle
df = shuffle(df)
df.head()
```

```
#SPLITING DATASET TESTING & TRAINING
train_dataset = df.sample(frac=0.8,random_state=0)
test_dataset = df.drop(train_dataset.index)
```

```
#INSPECT THE DATA
sns.pairplot(train_dataset[["LableEncoder","AcX", "AcY", "AcZ", ]], diag_kind="kde")
#kde smooth histogram
```

```
<seaborn.axisgrid.PairGrid at 0x7ff8e08416a0>
```




```
df.head()
```

```
df.head()
```

	AcX	AcY	AcZ	Lable	LableEncoder
3461	1.698164	1.989683	1.707680	stable	3
726	1.691817	1.980501	1.698903	stable	3
3233	1.693018	1.982130	1.700442	stable	3
738	1.691337	1.979422	1.695606	movement	2
2680	1.693980	1.982295	1.700896	stable	3

```

#STATIC DATASET
train_stats = train_dataset.describe()
train_stats.pop("LableEncoder")
train_stats = train_stats.transpose()
train_stats

```

	count	mean	std	min	25%	50%	75%	max
AcX	5314.0	1.695448	0.006728	1.687465	1.691739	1.692999	1.696515	1.739507
AcY	5314.0	1.965533	0.045069	1.748218	1.978860	1.981919	1.985166	2.018903
AcZ	5314.0	1.780958	0.125235	1.614460	1.699983	1.703001	1.851358	2.114893

```
#NORMALIZE THE DATASET
'''
def norm(x):
  return (x - train_stats['mean']) / train_stats['std']
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)

'''

# Surpress warning for next 3 operation
pd.options.mode.chained_assignment = None  # default='warn'
train_dataset['AcX'] = train_dataset['AcX'] /train_dataset['AcX'].max()
train_dataset['AcY'] =train_dataset['AcY'] /train_dataset['AcY'].max()
train_dataset['AcZ'] =train_dataset['AcZ'] / train_dataset['AcZ'].max()
```

```
train_dataset['AcZ'] =train_dataset['AcZ'] / train_dataset['AcZ'].max()
# Round numbers
train_dataset = train_dataset.round({'AcX': 5, 'AcY': 5, 'AcZ': 5})

df['AcZ'] = df['AcZ'].astype('float32')
df['AcX'] = df['AcX'].astype('float32')
df['AcY'] = df['AcY'].astype('float32')
df['LableEncoder'] = df['LableEncoder'].astype('float32')
train_dataset.head()
```

	AcX	AcY	AcZ	Lable	LableEncoder
2659	0.97329	0.98161	0.80466	stable	3
1222	0.97419	0.98206	0.80309	stable	3
3718	0.97280	0.98194	0.80355	stable	3
5288	0.97316	0.98583	0.84561	movement	2
2772	0.97082	0.98068	0.80355	stable	3

RESHAPE DATA INTO SEGMENT AND 3DIMENSION

with 80 steps (see constant defined earlier). Taking into consideration the 20 Hz sampling rate, this equals to 4 second time intervals (calculation: 0.05 * 80 = 4). Besides reshaping the data, the function will also separate the features (x-acceleration, y-acceleration, z-acceleration) and the labels.

```
# The number of steps within one time segment
TIME_PERIODS = 2
# The steps to take from one segment to the next; if this value is equal to
# TIME_PERIODS, then there is no overlap between the segments
STEP_DISTANCE = 2


def create_segments_and_labels(df, time_steps, step, label_name):

    # x, y, z acceleration as features
    N_FEATURES = 3
    # Number of steps to advance in each iteration (for me, it should always
    # be equal to the time_steps in order to have no overlap between segments)
    # step = time_steps
```

```
    # step = time_steps
    segments = []
    labels = []
    for i in range(0, len(df) - time_steps, step):
        xs = df['AcX'].values[i: i + time_steps]
        ys = df['AcY'].values[i: i + time_steps]
        zs = df['AcZ'].values[i: i + time_steps]

        # Retrieve the most often used label in this segment

  # What is exactly the lable here findout to make sure Y lable = X train
        label = stats.mode(df['LableEncoder'][i: i + time_steps])[0][0]
        segments.append([xs, ys, zs])
        labels.append(label)

  # Bring the segments into a better shape
    reshaped_segments = np.asarray(segments, dtype= np.float32).reshape(-1, time_steps, N_FEATURES)
    labels = np.asarray(labels)

    return reshaped_segments, labels

x_train, y_train = create_segments_and_labels(train_dataset,
                                              TIME_PERIODS,
                                              STEP_DISTANCE,
                                           train_dataset)
```

```
# Here The Shape of X training and Y lable has to the same length if not something wrong
print('x_train shape: ', x_train.shape)
print(x_train.shape[0], 'training samples')
print('y_train shape: ', y_train.shape)
```

```
x_train shape:  (2656, 2, 3)
2656 training samples
y_train shape:  (2656,)
```

```
num_time_periods, num_axis = x_train.shape[1], x_train.shape[2]

num_classes = le.classes_.size
print(list(le.classes_))
```

```
['SLOWMOVEMNT', 'highly movement ', 'movement', 'stable']
```

```python
# Set input & output dimensions
input_shape = (num_time_periods*3)
x_train = x_train.reshape(x_train.shape[0], input_shape)
print('x_train shape:', x_train.shape)
print('input_shape:', input_shape)
```

```
x_train shape: (2656, 6)
input_shape: 6
```

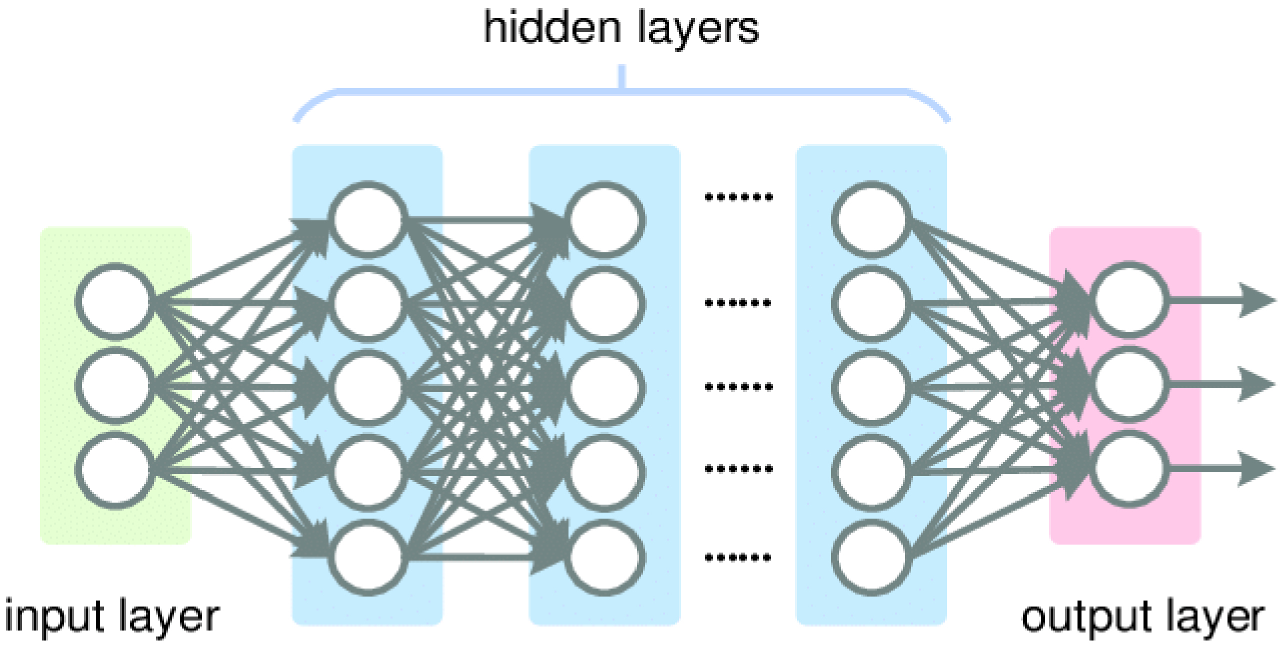
```python
y_train_hot = np_utils.to_categorical(y_train)
print('New y_train shape: ', y_train_hot.shape)
```

```
New y_train shape:  (2656, 4)
```

```python
x_train = x_train.astype('float32')
y_train = y_train.astype('float32')
```

The new method feeding the data to model

DEFINE CNN_LSTM MODEL




```
model = Sequential()
# Remark: since coreml cannot accept vector shapes of complex shape like
# prior feeding it into the network
model.add(Reshape((TIME_PERIODS, 3), input_shape=(input_shape,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_2 (Reshape)          (None, 2, 3)              0
_________________________________________________________________
dense_8 (Dense)              (None, 2, 100)            400
_________________________________________________________________
dense_9 (Dense)              (None, 2, 100)            10100
_________________________________________________________________
dense_10 (Dense)             (None, 2, 100)            10100
_________________________________________________________________
flatten_5 (Flatten)          (None, 200)               0
_________________________________________________________________
dense_11 (Dense)             (None, 4)                 804
=================================================================
Total params: 21,404
Trainable params: 21,404
Non-trainable params: 0
_________________________________________________________________
None
```

TRAINING MODEL

```
from keras.callbacks import History
history = History()
callbacks_list = [
    keras.callbacks.ModelCheckpoint(
        filepath='best_model.{epoch:02d}-{val_loss:.2f}.h5',
        monitor='val_loss', save_best_only=True),
    keras.callbacks.EarlyStopping(monitor='acc', patience=100), history]


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

# Hyper-parameters
BATCH_SIZE =30
EPOCHS = 100

```

```
# Enable validation to use ModelCheckpoint and EarlyStopping callbacks.
history = model.fit(x_train,
                      y_train_hot,
                      batch_size=BATCH_SIZE,
                      epochs=EPOCHS,
                      callbacks=callbacks_list,
                      validation_split=0.2,
                      verbose=1)
```

SECOND WAY VISUALIZE ALL TRAINING TESTING MODEL BETTER Visualize the model's training progress using the stats stored in the `history` object.

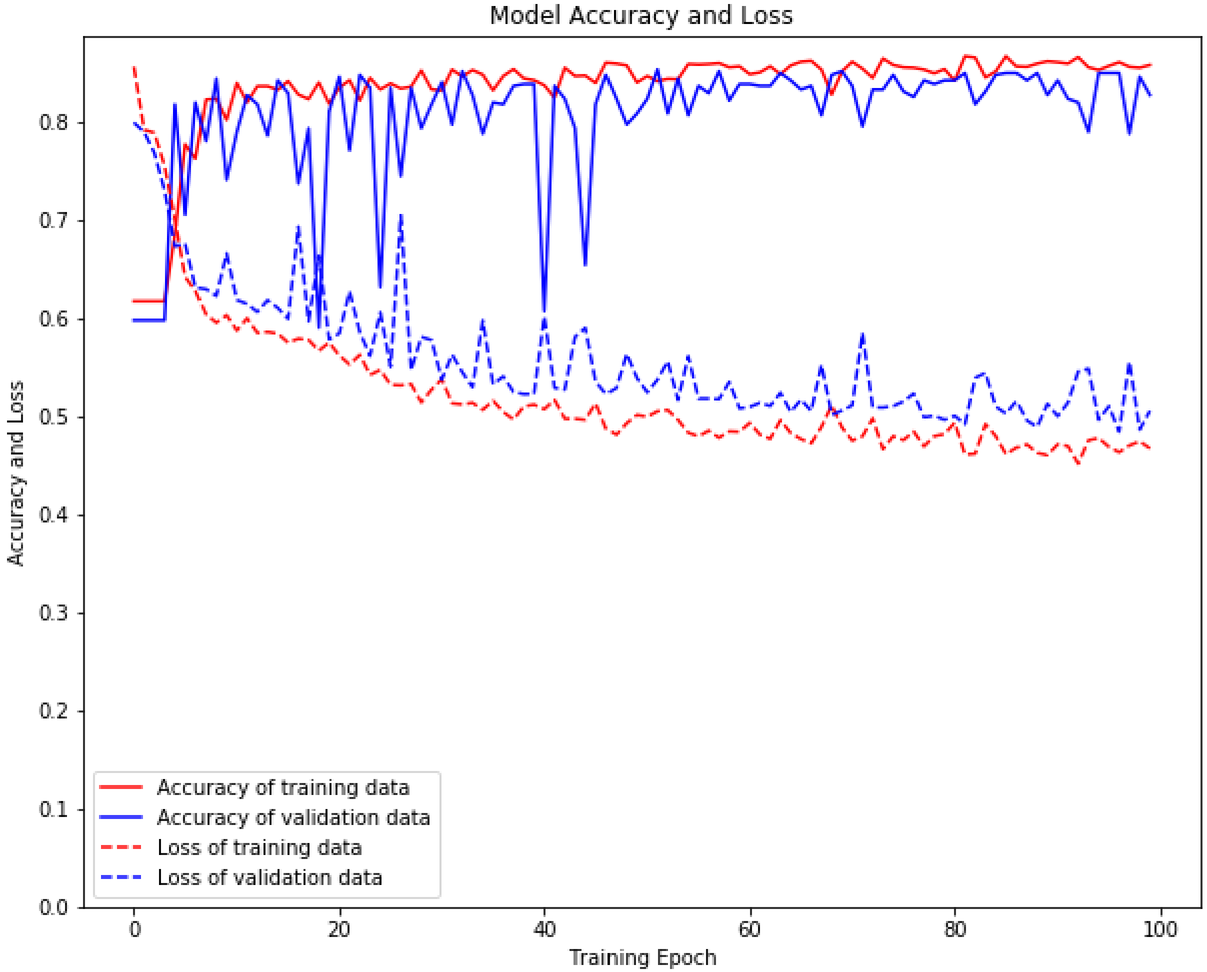
CHECKING GENERALIZATION

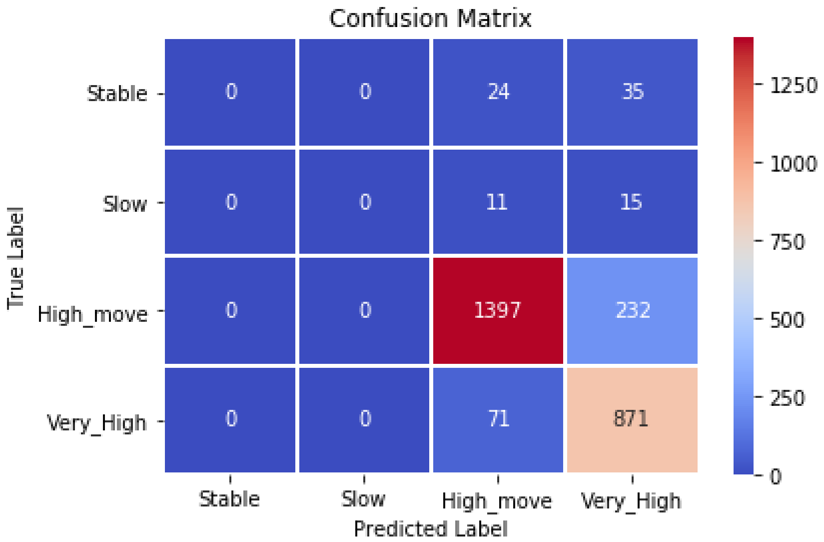
```
hist = pd.DataFrame(history.history)
hist['epochs'] = history.epoch
hist.tail()
```

	val_loss	val_acc	loss	acc	epochs
95	0.510161	0.849624	0.468636	0.856874	95
96	0.484145	0.849624	0.463130	0.860640	96
97	0.555710	0.787594	0.470055	0.855932	97
98	0.486225	0.845865	0.474244	0.854991	98
99	0.505399	0.827068	0.467077	0.857815	99

```
plt.figure(figsize=(10, 8))
plt.plot(history.history['acc'], 'r', label='Accuracy of training data')
plt.plot(history.history['val_acc'], 'b', label='Accuracy of validation data')
plt.plot(history.history['loss'], 'r--', label='Loss of training data')
plt.plot(history.history['val_loss'], 'b--', label='Loss of validation data')
plt.title('Model Accuracy and Loss')
plt.ylabel('Accuracy and Loss')
plt.xlabel('Training Epoch')
plt.ylim(0)
plt.legend()
plt.show()
```

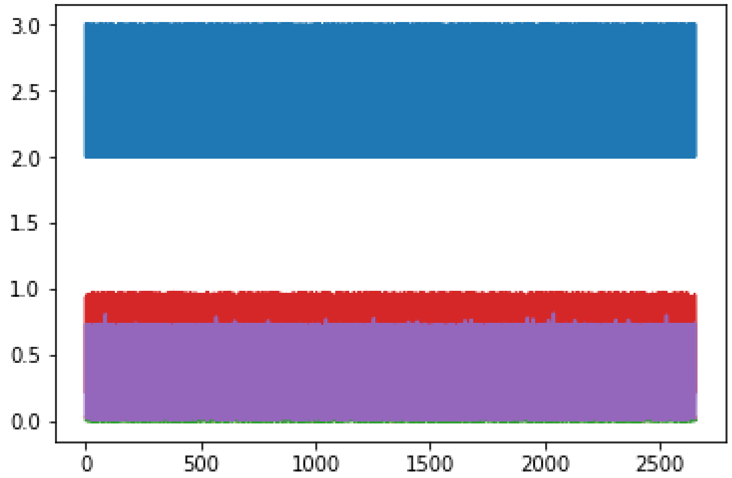
```
LABELS = ['Stable', 'Slow', 'High_move', 'Very_High']
from sklearn.metrics import confusion_matrix
def show_confusion_matrix(validations, predictions):

    matrix = metrics.confusion_matrix(validations, predictions)
    plt.figure(figsize=(6, 4))
    sns.heatmap(matrix,
                cmap='coolwarm',
                linecolor='white',
                linewidths=1,
                xticklabels=LABELS,
                yticklabels=LABELS,
                annot=True,
                fmt='d')
    plt.title ('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

# Print confusion matrix for training data
y_pred_train = model.predict(x_train)
# Take the class with the highest probability from the train predictions
max_y_pred_train = np.argmax(y_pred_train, axis=1)
max_y_train = np.argmax(y_train_hot, axis=1)
show_confusion_matrix(max_y_train,max_y_pred_train)
#print(classification_report(y_train, max_y_pred_train))


print(y_pred_train)
print(max_y_pred_train)
plt.plot(max_y_pred_train)
plt.plot(y_pred_train)
plt.show()
```





[[0.03512397 0.01204157 0.22422521 0.72860926]

```
[[0.03312597 0.01204157 0.22422521 0.72860926]
 [0.02075282 0.00777149 0.943353   0.0281227 ]
 [0.03510279 0.01200471 0.22544615 0.7274464 ]
 ...
 [0.01652309 0.00591087 0.9592032  0.01836277]
 [0.02544351 0.02095251 0.7463907  0.20721333]
 [0.03570118 0.01225461 0.22863567 0.7234086 ]]
[3 2 3 ... 2 2 3]
```




```
# The number of steps within one time segment
TIME_PERIODS = 2
# The steps to take from one segment to the next; if this value is equal to
# TIME_PERIODS, then there is no overlap between the segments
STEP_DISTANCE = 3


def create_segments_and_labels(df, time_steps, step, label_name):

    # x, y, z acceleration as features
    N_FEATURES = 3
    # Number of steps to advance in each iteration (for me, it should always
    # be equal to the time_steps in order to have no overlap between segments)
    # step = time_steps
    segments = []
    labels = []
    for i in range(0, len(df) - time_steps, step):
        xs = df['AcX'].values[i: i + time_steps]
        ys = df['AcY'].values[i: i + time_steps]
```

```
            zs = df['AcZ'].values[i: i + time_steps]

            # Retrieve the most often used label in this segment

      # What is exactly the lable here findout to make sure Y lable = X train
            label = stats.mode(df['LableEncoder'][i: i + time_steps])[0][0]
            segments.append([xs, ys, zs])
            labels.append(label)

      # Bring the segments into a better shape
        reshaped_segments = np.asarray(segments, dtype= np.float32).reshape(-1, time_steps, N_FEATURES)
        labels = np.asarray(labels)

        return reshaped_segments, labels

x_test, y_test= create_segments_and_labels(test_dataset,
                                              TIME_PERIODS,
                                              STEP_DISTANCE,
                                           test_dataset)
```

```
print(x_test.shape)
print(y_test.shape)

```

```
(442, 2, 3)
(442,)
```

```
num_time_periods, num_axis = x_test.shape[1], x_test.shape[2]
num_classes = le.classes_.size
print(list(le.classes_))
```

```
['SLOWMOVEMNT', 'highly movement ', 'movement', 'stable']
```

```
num_time_periods, num_sensor = x_test.shape[1], x_test.shape[2]
```

```
input_shape1 = num_time_periods*3
x_test = x_test.reshape(x_test.shape[0], input_shape1)
print('x_test shape:', x_test.shape)
print('input_shape1:', input_shape1)

```

```
x_test shape: (442, 6)
input_shape1: 6
```

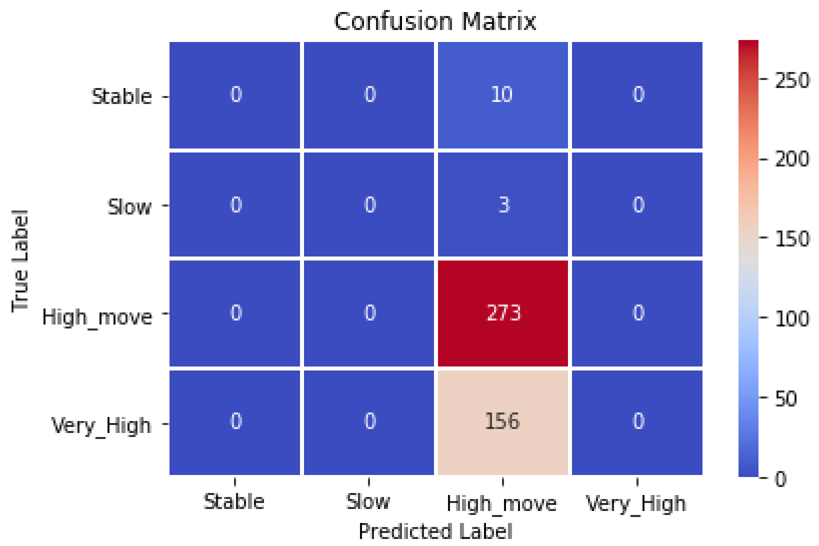
```
y_test_hot = np_utils.to_categorical(y_test, num_classes)
print('New y_test shape: ', y_test_hot.shape)

y_test = y_test.astype('float32')
x_test= x_test.astype('float32')
```

```
New y_test shape:  (442, 4)
```

CLASSIFICATION MODEL

```
from sklearn.metrics import confusion_matrix
LABELS = ['Stable', 'Slow', 'High_move', 'Very_High']
def show_confusion_matrix(validations, predictions):

    matrix = metrics.confusion_matrix(validations, predictions)
    plt.figure(figsize=(6, 4))
    sns.heatmap(matrix,
                cmap='coolwarm',
                linecolor='white',
                linewidths=1,
                xticklabels=LABELS,
                yticklabels=LABELS,
                annot=True,
                fmt='d')
    plt.title ('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

y_pred_test = model.predict(x_test)
# Take the class with the highest probability from the test predictions
max_y_pred_test = np.argmax(y_pred_test, axis=1)

max_y_test = np.argmax(y_test_hot, axis=1)
show_confusion_matrix(max_y_test, max_y_pred_test)
```

```
print(classification_report(max_y_test, max_y_pred_test))


```




```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        10
           1       0.00      0.00      0.00         3
           2       0.62      1.00      0.76       273
           3       0.00      0.00      0.00       156

    accuracy                           0.62       442
   macro avg       0.15      0.25      0.19       442
weighted avg       0.38      0.62      0.47       442

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-de
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-de
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-de
  'precision', 'predicted', average, warn_for)
```

PREDICTION MODEL

Prediction With LSTM-- CNN Model

```
from google.colab import files
upload = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable

```
df=pd.read_csv('AcZ.csv')
df.head()
```

	AcZ
0	1.716803
1	1.717309
2	1.717906
3	1.718519
4	1.719052

```
df.values.shape
```

```
(6642, 1)
```

```
train_dataset = df.sample(frac=0.8,random_state=0)
test_dataset = df.drop(train_dataset.index)
train_dataset.shape
```

```
(5314, 1)
```

```
x_train = train_dataset
y_train= test_dataset
```

```
# Here The Shape of X training and Y lable has to the same length if not something wrong
print('x_train shape: ', x_train.shape)
print(x_train.shape[0], 'training samples')
print('y_train shape: ', y_train.shape)
```

```
print("y_train shape: ", y_train.shape)
```

```
x_train shape:  (5314, 1)
5314 training samples
y_train shape:  (1328, 1)
```

```
x_train= x_train.values.ravel()
x_train.shape
```

```
(5314,)
```

```
def split_sequence(sequence, n_steps):
  X, y = list(), list()
  for i in range(len(sequence)):
    # find the end of this pattern
    end_ix = i + n_steps
    # check if we are beyond the sequence
    if end_ix > len(sequence)-1:
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    y.append(seq_y)
  return array(X), array(y)
```

```
from numpy import array
# choose a number of time steps
n_steps = 4
# split into samples
X, y = split_sequence(x_train, n_steps)

```

```
X.shape
```

```
(5310, 4)
```

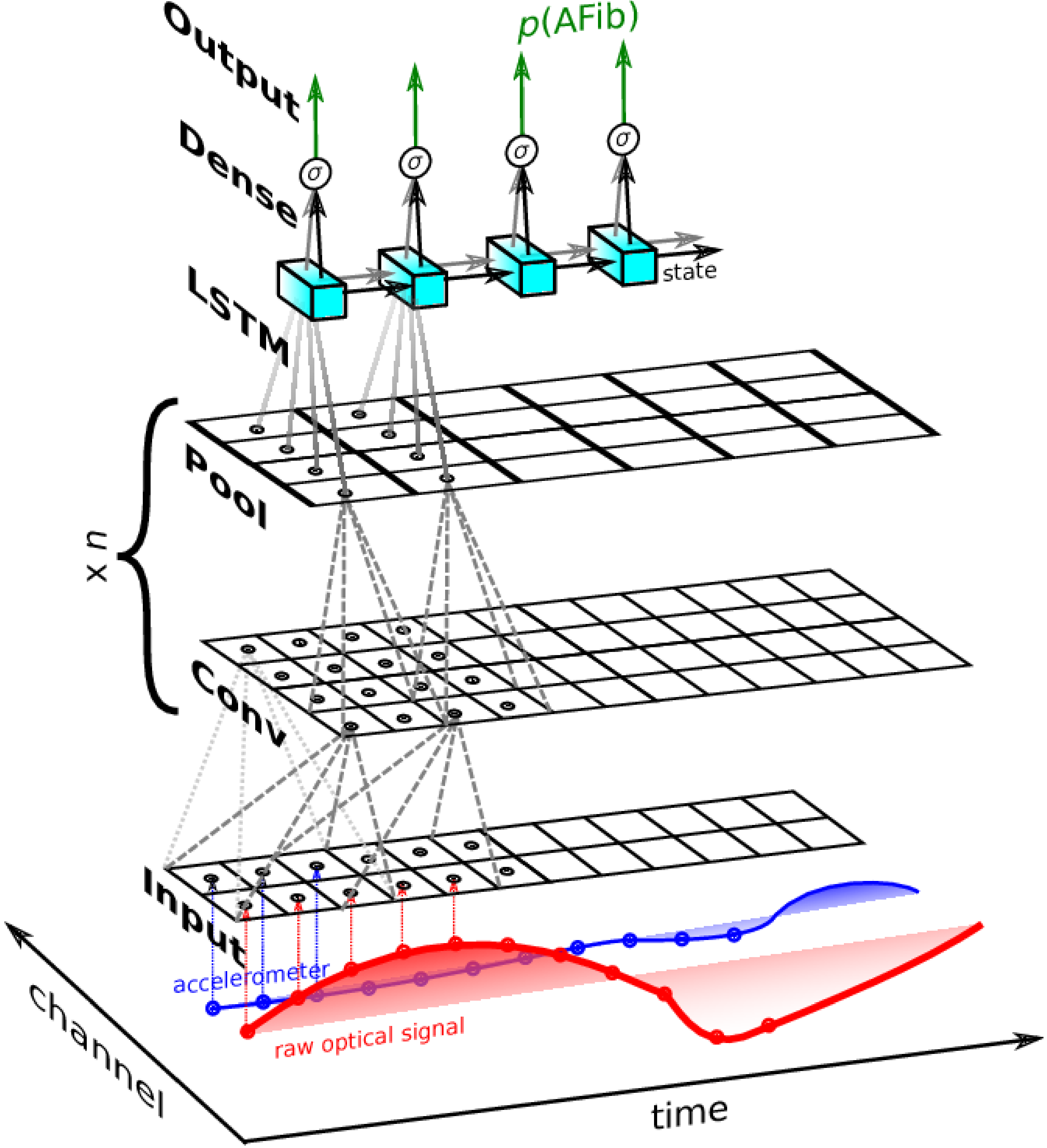
```
# Creating the input Shape with 4 Dimension
n_features = 1
n_seq = 2
n_steps = 2
```

```
X = X.reshape((X.shape[0], n_seq, n_steps, n_features))
X.shape
```

```
(5310, 2, 2, 1)
```

DEFINE CNN_LSTM Model

```

```



Output
p(AFib)
Dense
σ
LSTM
state
Pool
x n
Conv
Input
accelerometer
raw optical signal
channel
time

```
#Define the Model
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'), input_shape=(None, n_steps, n_features)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(100, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

```

```

```
#adding Check point and history to review the model accuracy
# include the epoch in the file name. (uses `str.format`)
from keras.callbacks import History
history = History()
checkpoint_path = "training_1.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

cp_callback = [tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True,period=10, ),
               keras.callbacks.EarlyStopping(monitor='loss', patience=20), history]


history = model.fit(X, y, callbacks = cp_callback, epochs=50, verbose=0)
```

```
hist = pd.DataFrame(history.history)
hist['epochs'] = history.epoch
hist.tail()
```

```
plt.figure(figsize=(10, 8))

plt.plot(history.history['loss'], 'r--', label='Loss of training data')


plt.ylabel(' Loss Value')
plt.xlabel('Training Epoch')
plt.ylim(0)
plt.legend()
plt.show()
```

```
x_test= test_dataset
x_test.shape
```

```
x_test= x_test.values.ravel()
x_test.shape
```

```
# choose a number of time steps
n_steps = 4
# split into samples
x_test, y = split_sequence(x_test, n_steps)
x_test.shape
```

```
# Creating the input Shape with 4 Dimension
n_features = 1
n_seq = 2
n_steps = 2
x_input = x_test.reshape((x_test.shape[0], n_seq, n_steps, n_features))
predict_result=  model.predict(x_input, verbose=0)
predict_result.shape

```

```
x_input
```

```
predict_result


```

```
 plt.figure(figsize = (10,8))
plt.plot(predict_result, 'r', label='Loss of training data')

```

```
plt.figure(figsize = (10,8))
plt.plot(x_test)
plt.show()

```

```
hist = pd.DataFrame(history.history)
hist['epochs'] = history.epoch
hist.tail()
```