In [1]:

```
import pandas as pd
import numpy as np

from tensorflow import keras
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout

from kerastuner import RandomSearch, BayesianOptimization
from kerastuner.engine.hyperparameters import HyperParameters
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.compat.v1.keras.layers import CuDNNLSTM

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Utility function to print the confusion matrix
def confusion_matrix_user(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

Data

In [3]:

```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

In [4]:

```
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]
```

In [5]:

```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subs
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 sign
    return np.transpose(signals_data, (1, 2, 0))
```

In [6]:

```
def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummie
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()
```

In [7]:

```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test
```

In [8]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

Splitting Train and Test

In [9]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [10]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

Defining function which returns the tuned model architecture

In [11]:

```
def c_model(hp):

    model = Sequential()

    model.add(CuDNNLSTM(units = hp.Int('units_1', 32, 256, step = 32), input_
                        kernel_initializer= 'he_normal',
                        return_sequences = True))

    model.add(Dropout(hp.Float('dropout_1', 0.2, 0.8, step = 0.1)))

    model.add(CuDNNLSTM(units = hp.Int('units_2', 32, 256, step = 32),
                        kernel_initializer= 'he_normal'))

    model.add(Dropout(hp.Float('dropout_2', 0.2, 0.8, step = 0.1)))

    for i in range(hp.Int('num_layers', 0, 2, step = 1)):
        model.add(Dense(units=hp.Int('dense_' + str(i), 32, 512, step = 32),
                        kernel_initializer= 'he_normal',
                        activation='relu'))

    model.add(Dropout(hp.Float('dropout_3', 0.0, 0.8, step = 0.1)))

    model.add(Dense(n_classes, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
              optimizer= hp.Choice('optimizer_name', ['adam','adagrad','adade
              metrics=['accuracy'])

    return model
```

Tuning the model's architecture and Hyper parameter using Keras Tuner

In [12]:

```
class MyTuner(RandomSearch):
    def run_trial(self, trial, *args, **kwargs):
    # You can add additional HyperParameters for preprocessing and custom tra
    # via overriding `run_trial`
        kwargs['batch_size'] = trial.hyperparameters.Int('batch_size', 32, 25
        kwargs['epochs'] = trial.hyperparameters.Int('epochs', 5, 30, step=5)
        super(MyTuner, self).run_trial(trial, *args, **kwargs)

# Uses same arguments as the BayesianOptimization Tuner.
tuner = MyTuner(c_model,
                objective='val_accuracy',
                max_trials=10,
                executions_per_trial=10,
                directory='Random tuner',
                project_name='optimized_model')

# Don't pass epochs or batch_size here, let the Tuner tune them.
tuner.search(X_train,
             Y_train,
             validation_split=0.3,
             verbose=2,
             use_multiprocessing=True)

tuner.search_space_summary()

tuner.results_summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In Tenso
rFlow 2.x, dropout() uses dropout rate instead of keep_pro
b. Please ensure that this is intended.
Train on 5146 samples, validate on 2206 samples
Epoch 1/5
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In Tenso
rFlow 2.x, dropout() uses dropout rate instead of keep_pro
b. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In Tenso
rFlow 2.x, dropout() uses dropout rate instead of keep_pro
b. Please ensure that this is intended.
5146/5146 - 20s - loss: 1.0901 - accuracy: 0.5286 - val_los
s: 0.6732 - val_accuracy: 0.7049
Epoch 2/5
5146/5146 - 4s - loss: 0.7284 - accuracy: 0.6901 - val_los
s: 0.4658 - val_accuracy: 0.9039
Epoch 3/5
5146/5146 - 4s - loss: 0.5776 - accuracy: 0.7798 - val_los
s: 0.3909 - val_accuracy: 0.9116
```

Best hyper parameters

In [13]:

```
best_hp = tuner.get_best_hyperparameters()[0]
best_hp.values
```

Out[13]:

```
{'units_1': 224,
 'dropout_1': 0.8000000000000003,
 'units_2': 96,
 'dropout_2': 0.30000000000000004,
 'num_layers': 1,
 'dropout_3': 0.6000000000000001,
 'optimizer_name': 'adam',
 'batch_size': 64,
 'epochs': 20,
 'dense_0': 288}
```

Best Model's Architecture

In [14]:

```
best_model = tuner.get_best_models(num_models=1)[0]
best_model.summary()
```

```
Model: "sequential"
______________________________________________________________
___
Layer (type)                 Output Shape              Param #
==============================================================
===
cu_dnnlstm (CuDNNLSTM)       (None, 128, 224)          210560
______________________________________________________________
___
dropout (Dropout)            (None, 128, 224)          0
______________________________________________________________
___
cu_dnnlstm_1 (CuDNNLSTM)     (None, 96)                123648
______________________________________________________________
___
dropout_1 (Dropout)          (None, 96)                0
______________________________________________________________
___
dense (Dense)                (None, 288)               27936
______________________________________________________________
___
dropout_2 (Dropout)          (None, 288)               0
______________________________________________________________
___
dense_1 (Dense)              (None, 6)                 1734
==============================================================
===
Total params: 363,878
Trainable params: 363,878
Non-trainable params: 0
______________________________________________________________
___
```

Best Model on Test data

In [38]:

```
best_model.fit(X_train, Y_train,
          batch_size=194,
          epochs=5,
          validation_data=(X_test, Y_test),
          verbose = 2)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/5
7352/7352 - 1s - loss: 0.1056 - accuracy: 0.9684 - val_loss:
0.4614 - val_accuracy: 0.9342
Epoch 2/5
7352/7352 - 1s - loss: 0.0651 - accuracy: 0.9788 - val_loss:
0.5997 - val_accuracy: 0.9328
Epoch 3/5
7352/7352 - 1s - loss: 0.0472 - accuracy: 0.9807 - val_loss:
0.6456 - val_accuracy: 0.9270
Epoch 4/5
7352/7352 - 1s - loss: 0.0386 - accuracy: 0.9838 - val_loss:
0.5644 - val_accuracy: 0.9386
Epoch 5/5
7352/7352 - 1s - loss: 0.0269 - accuracy: 0.9897 - val_loss:
0.5855 - val_accuracy: 0.9410
```

Out[38]:

```
<tensorflow.python.keras.callbacks.History at 0x147894e6348>
```

In [39]:

```
score = best_model.evaluate(X_test, Y_test, verbose = 2)
```

```
2947/1 - 1s - loss: 0.2927 - accuracy: 0.9410
```

In [40]:

```
print('\n\nThe test result obtained By fine-tuning the model is :\nLoss = {}\
```

```
The test result obtained By fine-tuning the model is :
Loss = 0.5854996841915335
Accuracy is : 0.9409568905830383
```

In [41]:

```
# Confusion Matrix
confusion_matrix_user(Y_test, best_model.predict(X_test, batch_size=192))
```

Out[41]:

Pred	LAYING	SITTING	STANDING	WALKING	WALKING_
True					
LAYING	537	0	0	0	
SITTING	0	418	69	0	
STANDING	0	45	487	0	
WALKING	0	0	0	463	
WALKING_DOWNSTAIRS	0	0	0	1	
WALKING_UPSTAIRS	0	0	0	12	

Summary

- Performed hyper parameter tuning with different architectures using Keras Tuner
- Without hyper parameter tuning and a single layer LSTM, an Accuracy of 90 % was obtained
- By performing hyper parameter tuning and an advance architecture, best Accuracy observed is 94.09 %
- Tuning the model resulted in an improvement of model's accuracy by 4.09 %