In [1]:
```
# Importing Libraries
```

In [1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout

from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras.layers import LSTM,BatchNormalization
import matplotlib.pyplot as plt
import seaborn as sns
```

```
Using TensorFlow backend.
```

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(Y_true, Y_pred):
    #Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, 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'F:\\Python\\Appliedai\\Human recognition\\HAR\\UCI_HAR_Da
taset\\{subset}\\Inertial Signals\\{signal}_{subset}.txt'
        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 signa
ls)
    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
s.html)
    """
    filename = f'F:\\Python\\Appliedai\\Human recognition\\HAR\\UCI_HAR_Datase
t\\{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]:
```
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

In [9]:
```
# Configuring a session
session_conf = tf.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [10]:
```
# Import Keras
from keras import backend as K
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [11]:
```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [12]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [13]:
```
# number f time steps = 128 , at every time step the input dim is 9, correspon
ding to the 9 time series data
# there are 7352 total number of windows/time series and each window correspon
ds to 1 time series which is 128 timesteps long
# and at each timestep there are 9 readings .
# for each time series we have to predict what is the class label from 1 to 6
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
```

In [14]:
```
#function to plot Categorical Crossentropy Loss VS No. of epochs plot
def plt_dynamic(x, vy, ty):
    plt.figure(figsize=(10,6))
    plt.plot(x, ty, 'b', label="Train Loss")
    plt.plot(x, vy, 'r', label="Validation/Test Loss")
    plt.title('\nCategorical_crossentropy Loss VS Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Categorical Crossentropy Loss-Train and Test loss')
    plt.legend()
    plt.grid()
    plt.show()
```

Defining the Architecture of LSTM

Model with 1 LSTM layer with 32 hidden units and Dropout

In [39]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [102]:
```
import warnings
warnings.filterwarnings("ignore")
from keras.layers import LSTM,BatchNormalization

# Initializing parameters
epochs = 20
batch_size = 32
n_hidden = 32

# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
model.add(BatchNormalization())
# Adding a dropout layer , to avoid overfitting
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_6 (LSTM)                (None, 32)                5376      
_________________________________________________________________
batch_normalization_5 (Batch (None, 32)                128       
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 198       
=================================================================
Total params: 5,702
Trainable params: 5,638
Non-trainable params: 64
_________________________________________________________________
```

In [42]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [43]:
```
import warnings
warnings.filterwarnings("ignore")

# Compiling the model  , loss is categorical_crossentropy , as the problem is
 multi-class classification
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=[
'accuracy'])
# Training the model
history = model.fit(X_train, Y_train, batch_size=batch_size, validation_data=(
X_test, Y_test), epochs=epochs)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/20
7352/7352 [==============================] - 25s 3ms/step - loss: 1.1224 - ac
c: 0.5545 - val_loss: 0.7203 - val_acc: 0.7177
Epoch 2/20
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7228 - ac
c: 0.6902 - val_loss: 0.9391 - val_acc: 0.5850
Epoch 3/20
7352/7352 [==============================] - 24s 3ms/step - loss: 0.5412 - ac
c: 0.7790 - val_loss: 0.6111 - val_acc: 0.7665
Epoch 4/20
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4082 - ac
c: 0.8455 - val_loss: 0.5761 - val_acc: 0.8320
Epoch 5/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.3278 - ac
c: 0.8803 - val_loss: 0.5876 - val_acc: 0.8694
Epoch 6/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2590 - ac
c: 0.9094 - val_loss: 0.5522 - val_acc: 0.8721
Epoch 7/20
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2263 - ac
c: 0.9246 - val_loss: 0.5993 - val_acc: 0.8633
Epoch 8/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1970 - ac
c: 0.9291 - val_loss: 0.4673 - val_acc: 0.9046
Epoch 9/20
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2259 - ac
c: 0.9257 - val_loss: 0.4603 - val_acc: 0.9013
Epoch 10/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1816 - ac
c: 0.9376 - val_loss: 0.6288 - val_acc: 0.8721
Epoch 11/20
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1676 - ac
c: 0.9382 - val_loss: 0.4961 - val_acc: 0.8989
Epoch 12/20
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1664 - ac
c: 0.9393 - val_loss: 0.4992 - val_acc: 0.9023
Epoch 13/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1631 - ac
c: 0.9400 - val_loss: 0.4950 - val_acc: 0.8968
Epoch 14/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1539 - ac
c: 0.9419 - val_loss: 0.4357 - val_acc: 0.9070
Epoch 15/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1528 - ac
c: 0.9448 - val_loss: 0.4963 - val_acc: 0.9084
Epoch 16/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1530 - ac
c: 0.9449 - val_loss: 0.5005 - val_acc: 0.9162
Epoch 17/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1385 - ac
c: 0.9470 - val_loss: 0.4614 - val_acc: 0.9080
Epoch 18/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1582 - ac
c: 0.9422 - val_loss: 0.7532 - val_acc: 0.8778
Epoch 19/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1429 - ac
```

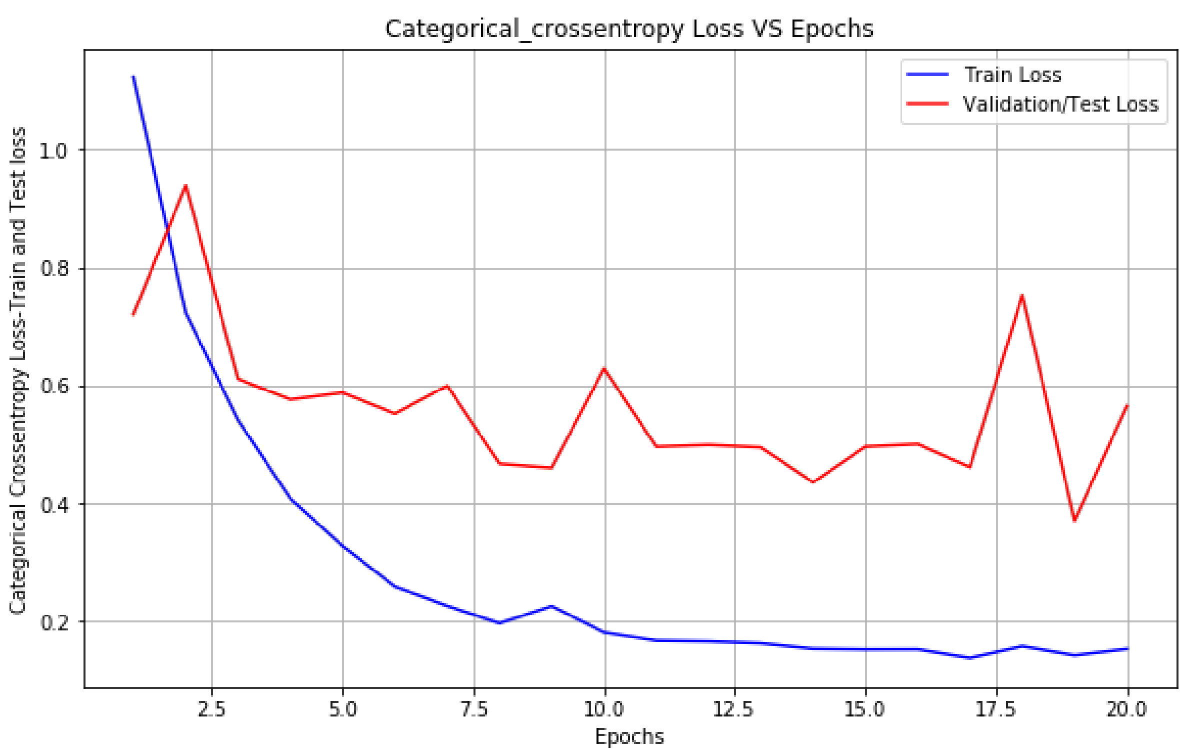
```
c: 0.9434 - val_loss: 0.3707 - val_acc: 0.9199
Epoch 20/20
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1536 - ac
c: 0.9448 - val_loss: 0.5651 - val_acc: 0.8982
```

Plotting the error plot

In [48]:

```
import matplotlib.pyplot as plt
# Evaluating the model on test data
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Plotting the results
# list of epoch numbers
x = list(range(1, epochs+1))

# we will get val_loss and val_acc only when you pass the paramter validation_
data
# val_loss : validation loss
vy = history.history['val_loss']

# Training loss
ty = history.history['loss']
# calling the dynamic function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.5651354040048198
Test accuracy: 0.8982015609093994
```



Confusion matrix

In [49]:
```
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 521        0         0        0                   0
SITTING                  0      367        97        0                   1
STANDING                 0       55       461        0                   0
WALKING                  0        0         0      437                   3
WALKING_DOWNSTAIRS       0        0         0        0                 410
WALKING_UPSTAIRS         0        0         0        6                  14

Pred                WALKING_UPSTAIRS
True
LAYING                            16
SITTING                           26
STANDING                          16
WALKING                           56
WALKING_DOWNSTAIRS                10
WALKING_UPSTAIRS                 451
```

- With a simple 2 layer architecture we got 89.82% accuracy and a multi class log-loss of 0.56 which is categorical cross entropy.
- We can further imporve the performace with Hyperparameter tuning

Model with 1 LSTM layer with 64 hidden units and Dropout

In [52]:
```
import warnings
warnings.filterwarnings("ignore")

# Initializing parameters
epochs = 30
batch_size = 32

# Initiliazing the sequential model
model1 = Sequential()
# Configuring the parameters
model1.add(LSTM(64, input_shape=(timesteps, input_dim)))
model1.add(BatchNormalization())
# Adding a dropout layer , to avoid overfitting
model1.add(Dropout(0.7))
# Adding a dense output layer with sigmoid activation
model1.add(Dense(n_classes, activation='softmax'))
model1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 64)                18944
_________________________________________________________________
batch_normalization_4 (Batch (None, 64)                256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 390
=================================================================
Total params: 19,590
Trainable params: 19,462
Non-trainable params: 128
_________________________________________________________________
```

In [55]:
```
# Reference -   https://machinelearningmastery.com/how-to-stop-training-deep-n
eural-networks-at-the-right-time-using-early-stopping/
# https://keras.io/callbacks/
import warnings
warnings.filterwarnings("ignore")
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

# Compiling the model  , loss is categorical_crossentropy , as the problem is
 multi-class classification
model1.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=[
'accuracy'])

# specifying the filepath to store the best model
filepath = "HAR_bestmodel_LSTM.hdf5"
# early stopping
es = EarlyStopping(monitor='val_acc', mode='max', verbose=1, patience=1)
# model checkpoint to save the model with best accuracy
mc = ModelCheckpoint(filepath, monitor='val_acc', mode='max', verbose=1, save_
best_only=True)

# Training the model
history = model1.fit(X_train, Y_train, batch_size=batch_size, validation_data=
(X_test, Y_test), epochs=epochs, verbose=2, callbacks=[es, mc])
```

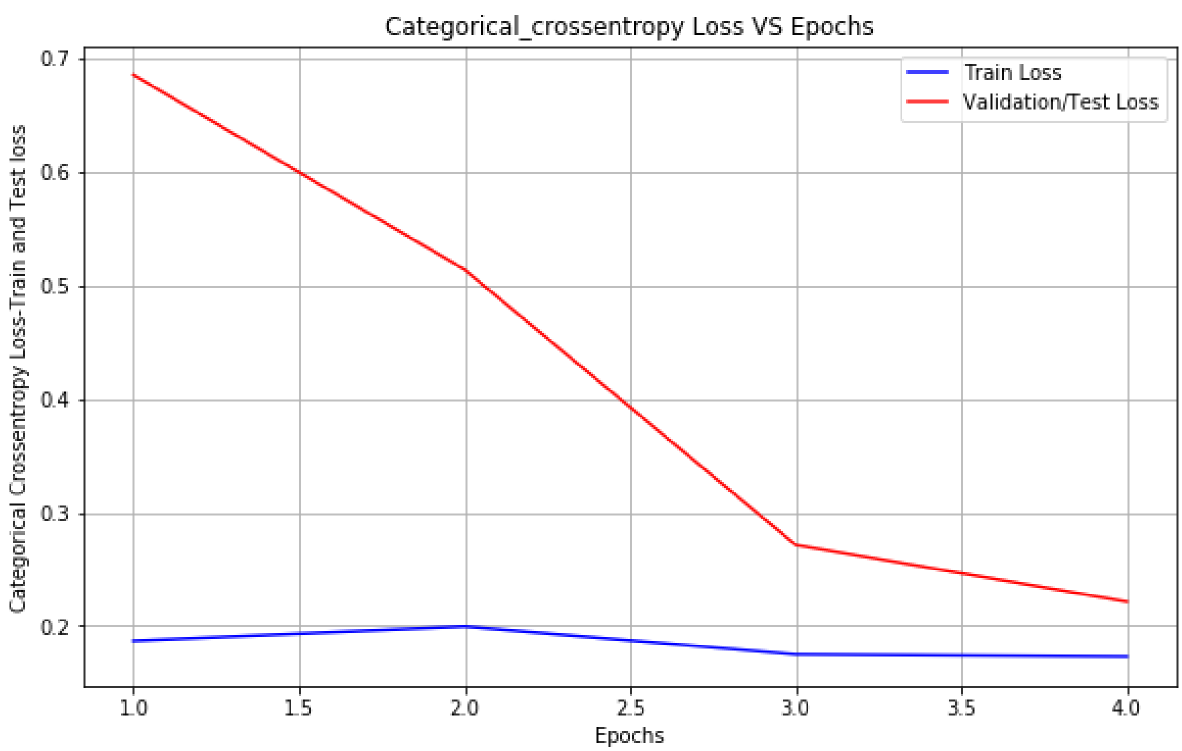
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 36s - loss: 0.1869 - acc: 0.9339 - val_loss: 0.6852 - val_acc: 0.8595

Epoch 00001: val_acc improved from -inf to 0.85952, saving model to HAR_bestm
odel_LSTM.hdf5
Epoch 2/30
 - 34s - loss: 0.1994 - acc: 0.9350 - val_loss: 0.5138 - val_acc: 0.9060

Epoch 00002: val_acc improved from 0.85952 to 0.90601, saving model to HAR_be
stmodel_LSTM.hdf5
Epoch 3/30
 - 36s - loss: 0.1750 - acc: 0.9359 - val_loss: 0.2713 - val_acc: 0.9328

Epoch 00003: val_acc improved from 0.90601 to 0.93281, saving model to HAR_be
stmodel_LSTM.hdf5
Epoch 4/30
 - 34s - loss: 0.1731 - acc: 0.9403 - val_loss: 0.2215 - val_acc: 0.9165

Epoch 00004: val_acc did not improve from 0.93281
Epoch 00004: early stopping
```

Plotting the error plot

In [59]:
```
#function to plot Categorical Crossentropy Loss VS No. of epochs plot
def plt_dynamic(x, vy, ty):
    plt.figure(figsize=(10,6))
    plt.plot(x, ty, 'b', label="Train Loss")
    plt.plot(x, vy, 'r', label="Validation/Test Loss")
    plt.title('\nCategorical_crossentropy Loss VS Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Categorical Crossentropy Loss-Train and Test loss')
    plt.legend()
    plt.grid()
    plt.show()
```

In [64]:
```
# Reference -  https://machinelearningmastery.com/how-to-stop-training-deep-ne
ural-networks-at-the-right-time-using-early-stopping/

from keras.models import load_model

# Saving the best model
saved_model = load_model('HAR_bestmodel_LSTM.hdf5')

# Evaluating the model on test data
score1 = saved_model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score1[0])
print('Test accuracy:', score1[1])

# Plotting the results
# list of epoch numbers
x = list(range(1, 5))

# we will get val_loss and val_acc only when you pass the paramter validation_
data
# val_loss : validation loss
vy = history.history['val_loss']

# Training loss
ty = history.history['loss']
# calling the dynamic function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.2712926466724943
Test accuracy: 0.9328130302002036
```



Confusion Matrix

In [99]:
```
# Utility function to print the confusion matrix
import seaborn as sns
def confusion_matrix(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)])
    plt.figure(1,figsize=(10,8))
    df=pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pred l
abels'])
    sns.heatmap(df, annot=True,cmap='RdPu',fmt='g')
    return pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pr
ed labels'])
```

In [100]:
```
Y_pred = saved_model.predict(X_test)
cm = confusion_matrix(Y_test, Y_pred)
```

Observation :-

With a simple 1 LSTM layer with 64 hidden layers and dropout layer architecture we got 93.28% accuracy and a multi class log-loss of 0.27 which is categorical cross entropy.

Model with 2 LSTM layers with 32 hidden units and Dropout

In [21]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [22]:
```
import warnings
warnings.filterwarnings("ignore")
from keras.layers import LSTM,BatchNormalization

# Initializing parameters
epochs = 30
batch_size = 32

# Initiliazing the sequential model
model2 = Sequential()
# Adding LSTM and Configuring the parameters
model2.add(LSTM(32, input_shape=(timesteps, input_dim), return_sequences=True
))
model2.add(BatchNormalization())
# Adding a dropout layer , to avoid overfitting
model2.add(Dropout(0.7))
# Adding LSTM layer
model2.add(LSTM(32))
# Adding dropout
model2.add(Dropout(0.7))
# Adding a dense output layer with sigmoid activation
model2.add(Dense(n_classes, activation='softmax'))
model2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_5 (LSTM)                (None, 128, 32)           5376      
_________________________________________________________________
batch_normalization_2 (Batch (None, 128, 32)           128       
_________________________________________________________________
dropout_3 (Dropout)          (None, 128, 32)           0         
_________________________________________________________________
lstm_6 (LSTM)                (None, 32)                8320      
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 198       
=================================================================
Total params: 14,022
Trainable params: 13,958
Non-trainable params: 64
_________________________________________________________________
```

In [25]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [28]:
```
# Reference -   https://machinelearningmastery.com/how-to-stop-training-deep-n
eural-networks-at-the-right-time-using-early-stopping/
# https://keras.io/callbacks/
import warnings
warnings.filterwarnings("ignore")
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.models import load_model

# Compiling the model  , loss is categorical_crossentropy , as the problem is
 multi-class classification
model2.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=[
'accuracy'])

# specifying the filepath to store the best model
filepath = "HAR_model2_LSTM.hdf5"
# early stopping
es2 = EarlyStopping(monitor='val_acc', mode='max', verbose=1, patience=20)
# model checkpoint to save the model with best accuracy
mc2 = ModelCheckpoint(filepath, monitor='val_acc', mode='max', verbose=1, save
_best_only=True)

# Training the model
history = model2.fit(X_train, Y_train, batch_size=batch_size, validation_data=
(X_test, Y_test), epochs=epochs, verbose=2, callbacks=[es2, mc2])

# Saving the best model
saved_model2 = load_model('HAR_model2_LSTM.hdf5')
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 55s - loss: 0.2233 - acc: 0.9290 - val_loss: 0.3942 - val_acc: 0.8894

Epoch 00001: val_acc improved from -inf to 0.88938, saving model to HAR_model
2_LSTM.hdf5
Epoch 2/30
 - 51s - loss: 0.2160 - acc: 0.9252 - val_loss: 0.3732 - val_acc: 0.9043

Epoch 00002: val_acc improved from 0.88938 to 0.90431, saving model to HAR_mo
del2_LSTM.hdf5
Epoch 3/30
 - 53s - loss: 0.2175 - acc: 0.9279 - val_loss: 0.4823 - val_acc: 0.8819

Epoch 00003: val_acc did not improve from 0.90431
Epoch 4/30
 - 50s - loss: 0.2216 - acc: 0.9285 - val_loss: 0.5732 - val_acc: 0.8765

Epoch 00004: val_acc did not improve from 0.90431
Epoch 5/30
 - 51s - loss: 0.1842 - acc: 0.9391 - val_loss: 0.3775 - val_acc: 0.9023

Epoch 00005: val_acc did not improve from 0.90431
Epoch 6/30
 - 49s - loss: 0.1985 - acc: 0.9323 - val_loss: 0.3977 - val_acc: 0.8955

Epoch 00006: val_acc did not improve from 0.90431
Epoch 7/30
 - 51s - loss: 0.1880 - acc: 0.9329 - val_loss: 0.3587 - val_acc: 0.9145

Epoch 00007: val_acc improved from 0.90431 to 0.91449, saving model to HAR_mo
del2_LSTM.hdf5
Epoch 8/30
 - 51s - loss: 0.1833 - acc: 0.9374 - val_loss: 0.3075 - val_acc: 0.9209

Epoch 00008: val_acc improved from 0.91449 to 0.92094, saving model to HAR_mo
del2_LSTM.hdf5
Epoch 9/30
 - 51s - loss: 0.1910 - acc: 0.9339 - val_loss: 0.8432 - val_acc: 0.8507

Epoch 00009: val_acc did not improve from 0.92094
Epoch 10/30
 - 51s - loss: 0.1901 - acc: 0.9342 - val_loss: 0.3820 - val_acc: 0.9087

Epoch 00010: val_acc did not improve from 0.92094
Epoch 11/30
 - 49s - loss: 0.1681 - acc: 0.9397 - val_loss: 0.5707 - val_acc: 0.8965

Epoch 00011: val_acc did not improve from 0.92094
Epoch 12/30
 - 51s - loss: 0.1675 - acc: 0.9388 - val_loss: 0.3673 - val_acc: 0.9104

Epoch 00012: val_acc did not improve from 0.92094
Epoch 13/30
 - 51s - loss: 0.1783 - acc: 0.9361 - val_loss: 0.4292 - val_acc: 0.9080

Epoch 00013: val_acc did not improve from 0.92094
```

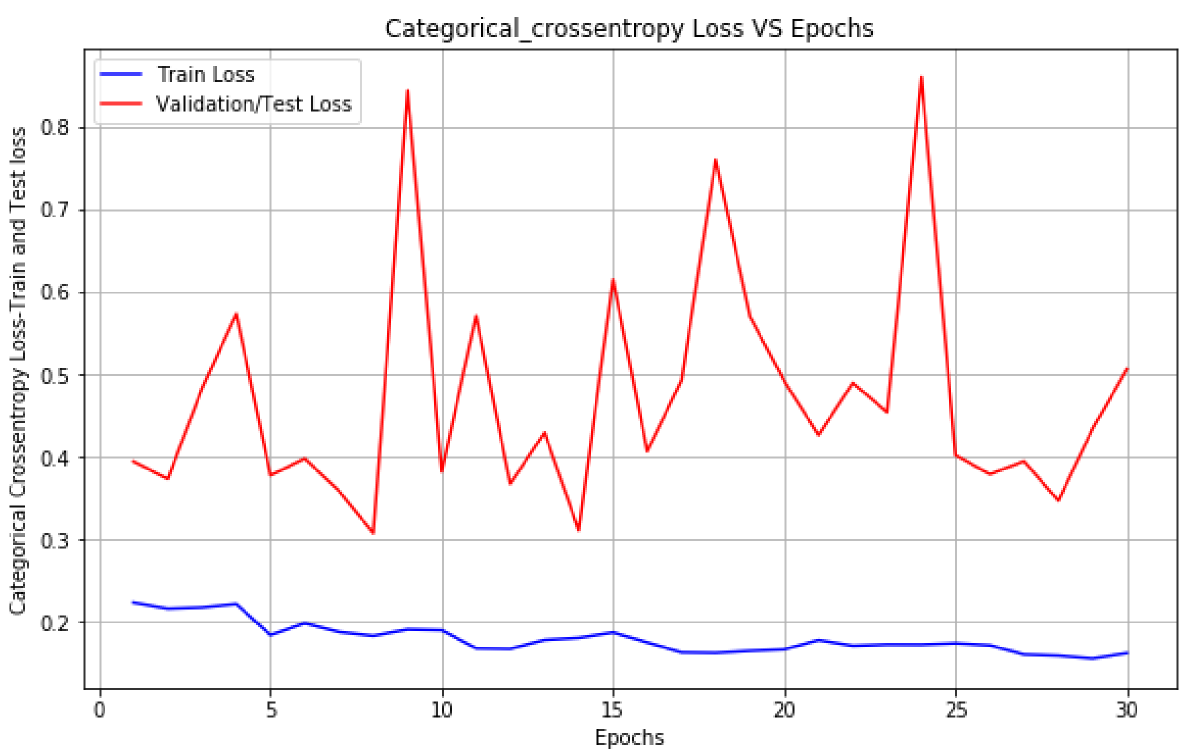
```
Epoch 14/30
 - 50s - loss: 0.1808 - acc: 0.9391 - val_loss: 0.3107 - val_acc: 0.9186

Epoch 00014: val_acc did not improve from 0.92094
Epoch 15/30
 - 50s - loss: 0.1873 - acc: 0.9363 - val_loss: 0.6144 - val_acc: 0.8911

Epoch 00015: val_acc did not improve from 0.92094
Epoch 16/30
 - 52s - loss: 0.1753 - acc: 0.9388 - val_loss: 0.4066 - val_acc: 0.9101

Epoch 00016: val_acc did not improve from 0.92094
Epoch 17/30
 - 51s - loss: 0.1632 - acc: 0.9369 - val_loss: 0.4926 - val_acc: 0.8945

Epoch 00017: val_acc did not improve from 0.92094
Epoch 18/30
 - 51s - loss: 0.1626 - acc: 0.9363 - val_loss: 0.7597 - val_acc: 0.8524

Epoch 00018: val_acc did not improve from 0.92094
Epoch 19/30
 - 52s - loss: 0.1651 - acc: 0.9391 - val_loss: 0.5696 - val_acc: 0.8880

Epoch 00019: val_acc did not improve from 0.92094
Epoch 20/30
 - 52s - loss: 0.1668 - acc: 0.9380 - val_loss: 0.4914 - val_acc: 0.9067

Epoch 00020: val_acc did not improve from 0.92094
Epoch 21/30
 - 50s - loss: 0.1780 - acc: 0.9382 - val_loss: 0.4263 - val_acc: 0.9006

Epoch 00021: val_acc did not improve from 0.92094
Epoch 22/30
 - 55s - loss: 0.1712 - acc: 0.9403 - val_loss: 0.4894 - val_acc: 0.9023

Epoch 00022: val_acc did not improve from 0.92094
Epoch 23/30
 - 47s - loss: 0.1727 - acc: 0.9378 - val_loss: 0.4539 - val_acc: 0.9084

Epoch 00023: val_acc did not improve from 0.92094
Epoch 24/30
 - 50s - loss: 0.1726 - acc: 0.9370 - val_loss: 0.8598 - val_acc: 0.8687

Epoch 00024: val_acc did not improve from 0.92094
Epoch 25/30
 - 53s - loss: 0.1742 - acc: 0.9338 - val_loss: 0.4019 - val_acc: 0.9162

Epoch 00025: val_acc did not improve from 0.92094
Epoch 26/30
 - 53s - loss: 0.1718 - acc: 0.9399 - val_loss: 0.3789 - val_acc: 0.9189

Epoch 00026: val_acc did not improve from 0.92094
Epoch 27/30
 - 52s - loss: 0.1606 - acc: 0.9378 - val_loss: 0.3946 - val_acc: 0.9203

Epoch 00027: val_acc did not improve from 0.92094
Epoch 28/30
```

```
 - 52s - loss: 0.1593 - acc: 0.9389 - val_loss: 0.3468 - val_acc: 0.9247

Epoch 00028: val_acc improved from 0.92094 to 0.92467, saving model to HAR_mo
del2_LSTM.hdf5
Epoch 29/30
 - 51s - loss: 0.1558 - acc: 0.9430 - val_loss: 0.4337 - val_acc: 0.9145

Epoch 00029: val_acc did not improve from 0.92467
Epoch 30/30
 - 51s - loss: 0.1623 - acc: 0.9378 - val_loss: 0.5061 - val_acc: 0.9094

Epoch 00030: val_acc did not improve from 0.92467
```

Plotting the error plot

In [34]:
```
# Reference -  https://machinelearningmastery.com/how-to-stop-training-deep-ne
ural-networks-at-the-right-time-using-early-stopping/

from keras.models import load_model
import matplotlib.pyplot as plt

# Evaluating the model on test data
score2 = saved_model2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score2[0])
print('Test accuracy:', score2[1])

# Plotting the results
# list of epoch numbers
x = list(range(1, epochs+1))

# we will get val_loss and val_acc only when you pass the paramter validation_
data
# val_loss : validation loss
vy = history.history['val_loss']

# Training loss
ty = history.history['loss']
# calling the dynamic function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.34682589094722394
Test accuracy: 0.9246691550729556
```



Confusion matrix

In [35]:
```
# Utility function to print the confusion matrix
import seaborn as sns
def confusion_matrix(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)])
    plt.figure(1,figsize=(10,8))
    df=pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pred l
abels'])
    sns.heatmap(df, annot=True,cmap='RdPu',fmt='g')
    return pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pr
ed labels'])
```

In [36]:
```
Y_pred = saved_model2.predict(X_test)
cm = confusion_matrix(Y_test, Y_pred)
```

Observation :-

With a simple 2 LSTM layer with 32 hidden layers and dropout layer architecture we got 92.46% accuracy and a multi class log-loss of 0.34 which is categorical cross entropy.

Model with 2 LSTM layers with 64 hidden units and Dropout

In [48]:
```
import warnings
warnings.filterwarnings("ignore")

# Initializing parameters
epochs = 20
batch_size = 32

# Initiliazing the sequential model
model3 = Sequential()
# Adding LSTM and Configuring the parameters
model3.add(LSTM(64, input_shape=(timesteps, input_dim), return_sequences=True
))
model3.add(BatchNormalization())
# Adding a dropout layer , to avoid overfitting
model3.add(Dropout(0.8))
# Adding LSTM layer
model3.add(LSTM(64))
# Adding dropout
model3.add(Dropout(0.8))
# Adding a dense output layer with sigmoid activation
model3.add(Dense(n_classes, activation='sigmoid'))
model3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_19 (LSTM)               (None, 128, 64)           18944     
_________________________________________________________________
batch_normalization_10 (Batc (None, 128, 64)           256       
_________________________________________________________________
dropout_17 (Dropout)         (None, 128, 64)           0         
_________________________________________________________________
lstm_20 (LSTM)               (None, 64)                33024     
_________________________________________________________________
dropout_18 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_9 (Dense)              (None, 6)                 390       
=================================================================
Total params: 52,614
Trainable params: 52,486
Non-trainable params: 128
_________________________________________________________________
```

In [49]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [50]:
```
# Reference -   https://machinelearningmastery.com/how-to-stop-training-deep-n
eural-networks-at-the-right-time-using-early-stopping/
# https://keras.io/callbacks/
import warnings
warnings.filterwarnings("ignore")

# Compiling the model  , loss is categorical_crossentropy , as the problem is
 multi-class classification
model3.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=[
'accuracy'])

# specifying the filepath to store the best model
filepath = "HAR_model3_LSTM.hdf5"
# early stopping
es3 = EarlyStopping(monitor='val_acc', mode='max', verbose=1, patience=15)
# model checkpoint to save the model with best accuracy
mc3 = ModelCheckpoint(filepath, monitor='val_acc', mode='max', verbose=1, save
_best_only=True)

# Training the model
history = model3.fit(X_train, Y_train, batch_size=batch_size, validation_data=
(X_test, Y_test), epochs=epochs, verbose=2, callbacks=[es3, mc3])

# Saving the best model
saved_model3 = load_model('HAR_model3_LSTM.hdf5')
```

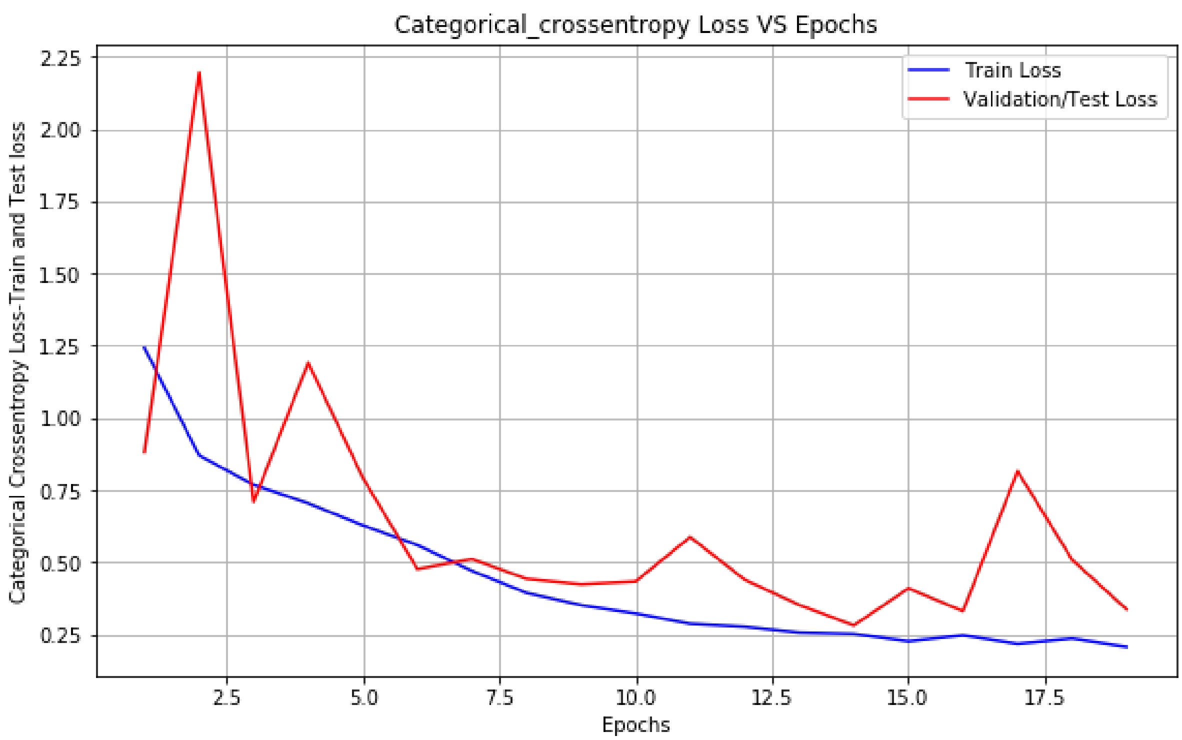
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/20
 - 80s - loss: 1.2419 - acc: 0.5537 - val_loss: 0.8814 - val_acc: 0.6634

Epoch 00001: val_acc improved from -inf to 0.66339, saving model to HAR_model
3_LSTM.hdf5
Epoch 2/20
 - 74s - loss: 0.8702 - acc: 0.6318 - val_loss: 2.1946 - val_acc: 0.4157

Epoch 00002: val_acc did not improve from 0.66339
Epoch 3/20
 - 74s - loss: 0.7685 - acc: 0.6602 - val_loss: 0.7095 - val_acc: 0.7160

Epoch 00003: val_acc improved from 0.66339 to 0.71598, saving model to HAR_mo
del3_LSTM.hdf5
Epoch 4/20
 - 73s - loss: 0.7044 - acc: 0.7013 - val_loss: 1.1912 - val_acc: 0.6535

Epoch 00004: val_acc did not improve from 0.71598
Epoch 5/20
 - 74s - loss: 0.6283 - acc: 0.7462 - val_loss: 0.7932 - val_acc: 0.7248

Epoch 00005: val_acc improved from 0.71598 to 0.72480, saving model to HAR_mo
del3_LSTM.hdf5
Epoch 6/20
 - 76s - loss: 0.5603 - acc: 0.8047 - val_loss: 0.4771 - val_acc: 0.8629

Epoch 00006: val_acc improved from 0.72480 to 0.86291, saving model to HAR_mo
del3_LSTM.hdf5
Epoch 7/20
 - 75s - loss: 0.4708 - acc: 0.8517 - val_loss: 0.5119 - val_acc: 0.8531

Epoch 00007: val_acc did not improve from 0.86291
Epoch 8/20
 - 75s - loss: 0.3948 - acc: 0.8825 - val_loss: 0.4434 - val_acc: 0.8660

Epoch 00008: val_acc improved from 0.86291 to 0.86597, saving model to HAR_mo
del3_LSTM.hdf5
Epoch 9/20
 - 75s - loss: 0.3524 - acc: 0.8999 - val_loss: 0.4244 - val_acc: 0.8711

Epoch 00009: val_acc improved from 0.86597 to 0.87106, saving model to HAR_mo
del3_LSTM.hdf5
Epoch 10/20
 - 417s - loss: 0.3234 - acc: 0.9053 - val_loss: 0.4341 - val_acc: 0.8802

Epoch 00010: val_acc improved from 0.87106 to 0.88022, saving model to HAR_mo
del3_LSTM.hdf5
Epoch 11/20
 - 96s - loss: 0.2879 - acc: 0.9154 - val_loss: 0.5878 - val_acc: 0.8626

Epoch 00011: val_acc did not improve from 0.88022
Epoch 12/20
 - 92s - loss: 0.2776 - acc: 0.9154 - val_loss: 0.4402 - val_acc: 0.8826

Epoch 00012: val_acc improved from 0.88022 to 0.88259, saving model to HAR_mo
del3_LSTM.hdf5
```

```
Epoch 13/20
 - 90s - loss: 0.2577 - acc: 0.9199 - val_loss: 0.3528 - val_acc: 0.8890

Epoch 00013: val_acc improved from 0.88259 to 0.88904, saving model to HAR_mo
del3_LSTM.hdf5
Epoch 14/20
 - 74s - loss: 0.2528 - acc: 0.9185 - val_loss: 0.2821 - val_acc: 0.9169

Epoch 00014: val_acc improved from 0.88904 to 0.91686, saving model to HAR_mo
del3_LSTM.hdf5
Epoch 15/20
 - 74s - loss: 0.2270 - acc: 0.9264 - val_loss: 0.4104 - val_acc: 0.8877

Epoch 00015: val_acc did not improve from 0.91686
Epoch 16/20
 - 74s - loss: 0.2478 - acc: 0.9195 - val_loss: 0.3316 - val_acc: 0.8996

Epoch 00016: val_acc did not improve from 0.91686
Epoch 17/20
 - 75s - loss: 0.2184 - acc: 0.9301 - val_loss: 0.8158 - val_acc: 0.8276

Epoch 00017: val_acc did not improve from 0.91686
Epoch 18/20
 - 75s - loss: 0.2368 - acc: 0.9293 - val_loss: 0.5092 - val_acc: 0.8748

Epoch 00018: val_acc did not improve from 0.91686
Epoch 19/20
 - 75s - loss: 0.2081 - acc: 0.9313 - val_loss: 0.3375 - val_acc: 0.9060

Epoch 00019: val_acc did not improve from 0.91686
Epoch 20/20
 - 75s - loss: nan - acc: 0.7035 - val_loss: nan - val_acc: 0.1683

Epoch 00020: val_acc did not improve from 0.91686
```

Plotting the error plot

In [51]:
```
# Evaluating the model on test data
score3 = saved_model3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score3[0])
print('Test accuracy:', score3[1])

# Plotting the results
# list of epoch numbers
x = list(range(1, epochs+1))

# we will get val_loss and val_acc only when you pass the paramter validation_
data
# val_loss : validation loss
vy = history.history['val_loss']

# Training loss
ty = history.history['loss']
# calling the dynamic function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.28211334944713906
Test accuracy: 0.9168646080760094
```



Confusion matrix

In [52]:
```
# Utility function to print the confusion matrix
import seaborn as sns
def confusion_matrix(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)])
    plt.figure(1,figsize=(10,8))
    df=pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pred l
abels'])
    sns.heatmap(df, annot=True,cmap='RdPu',fmt='g')
    return pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pr
ed labels'])
```

In [53]:
```
Y_pred = saved_model3.predict(X_test)
cm = confusion_matrix(Y_test, Y_pred)
```

Observation :-

With a architecture of 2 LSTM layers and with 64 hidden layers and dropout of 0.8, we got 91.68% accuracy and a multi class log-loss of 0.28 which is categorical cross entropy.

Bi-Directional LSTM model with 32 hidden units and Dropout

In [55]:
```
import warnings
warnings.filterwarnings("ignore")
from keras.layers import Bidirectional

# Initializing parameters
epochs = 30
batch_size = 32

# Initiliazing the sequential model
model4 = Sequential()
# Adding LSTM and Configuring the parameters
model4.add(Bidirectional(LSTM(32, return_sequences=True), input_shape=(timeste
ps, input_dim), merge_mode='concat'))
model4.add(BatchNormalization())
# Adding a dropout layer , to avoid overfitting
model4.add(Dropout(0.7))
# Adding LSTM layer
model4.add(LSTM(32))
# Adding dropout
model4.add(Dropout(0.7))
# Adding a dense output layer with sigmoid activation
model4.add(Dense(n_classes, activation='softmax'))
model4.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_1 (Bidirection (None, 128, 64)           10752
_________________________________________________________________
batch_normalization_11 (Batc (None, 128, 64)           256
_________________________________________________________________
dropout_19 (Dropout)         (None, 128, 64)           0
_________________________________________________________________
lstm_22 (LSTM)               (None, 32)                12416
_________________________________________________________________
dropout_20 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 6)                 198
=================================================================
Total params: 23,622
Trainable params: 23,494
Non-trainable params: 128
_________________________________________________________________
```

In [56]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [57]:
```
# Reference -   https://machinelearningmastery.com/how-to-stop-training-deep-n
eural-networks-at-the-right-time-using-early-stopping/
# https://keras.io/callbacks/
# https://keras.io/layers/wrappers/

import warnings
warnings.filterwarnings("ignore")

# Compiling the model  , loss is categorical_crossentropy , as the problem is
 multi-class classification
model4.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=[
'accuracy'])

# specifying the filepath to store the best model
filepath = "HAR_model4_LSTM.hdf5"
# early stopping
es4 = EarlyStopping(monitor='val_acc', mode='max', verbose=1, patience=20)
# model checkpoint to save the model with best accuracy
mc4 = ModelCheckpoint(filepath, monitor='val_acc', mode='max', verbose=1, save
_best_only=True)

# Training the model
history = model4.fit(X_train, Y_train, batch_size=batch_size, validation_data=
(X_test, Y_test), epochs=epochs, verbose=2, callbacks=[es4, mc4])

# Saving the best model
saved_model4 = load_model('HAR_model4_LSTM.hdf5')
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 88s - loss: 1.1461 - acc: 0.5445 - val_loss: 0.7891 - val_acc: 0.6685

Epoch 00001: val_acc improved from -inf to 0.66848, saving model to HAR_model
4_LSTM.hdf5
Epoch 2/30
 - 79s - loss: 0.7385 - acc: 0.6994 - val_loss: 0.5878 - val_acc: 0.7788

Epoch 00002: val_acc improved from 0.66848 to 0.77876, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 3/30
 - 75s - loss: 0.5622 - acc: 0.7886 - val_loss: 0.8105 - val_acc: 0.7299

Epoch 00003: val_acc did not improve from 0.77876
Epoch 4/30
 - 75s - loss: 0.4429 - acc: 0.8508 - val_loss: 0.7568 - val_acc: 0.7475

Epoch 00004: val_acc did not improve from 0.77876
Epoch 5/30
 - 75s - loss: 0.3470 - acc: 0.8920 - val_loss: 0.6392 - val_acc: 0.8096

Epoch 00005: val_acc improved from 0.77876 to 0.80964, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 6/30
 - 92s - loss: 0.3279 - acc: 0.8979 - val_loss: 0.4440 - val_acc: 0.8755

Epoch 00006: val_acc improved from 0.80964 to 0.87547, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 7/30
 - 86s - loss: 0.2645 - acc: 0.9129 - val_loss: 0.4083 - val_acc: 0.8860

Epoch 00007: val_acc improved from 0.87547 to 0.88599, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 8/30
 - 88s - loss: 0.2536 - acc: 0.9183 - val_loss: 0.4009 - val_acc: 0.8907

Epoch 00008: val_acc improved from 0.88599 to 0.89074, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 9/30
 - 86s - loss: 0.2419 - acc: 0.9232 - val_loss: 0.4556 - val_acc: 0.8829

Epoch 00009: val_acc did not improve from 0.89074
Epoch 10/30
 - 102s - loss: 0.2247 - acc: 0.9268 - val_loss: 0.5533 - val_acc: 0.8721

Epoch 00010: val_acc did not improve from 0.89074
Epoch 11/30
 - 95s - loss: 0.2311 - acc: 0.9256 - val_loss: 0.3988 - val_acc: 0.8880

Epoch 00011: val_acc did not improve from 0.89074
Epoch 12/30
 - 92s - loss: 0.2155 - acc: 0.9320 - val_loss: 0.6315 - val_acc: 0.8609

Epoch 00012: val_acc did not improve from 0.89074
Epoch 13/30
 - 104s - loss: 0.2029 - acc: 0.9293 - val_loss: 0.4851 - val_acc: 0.8860
```

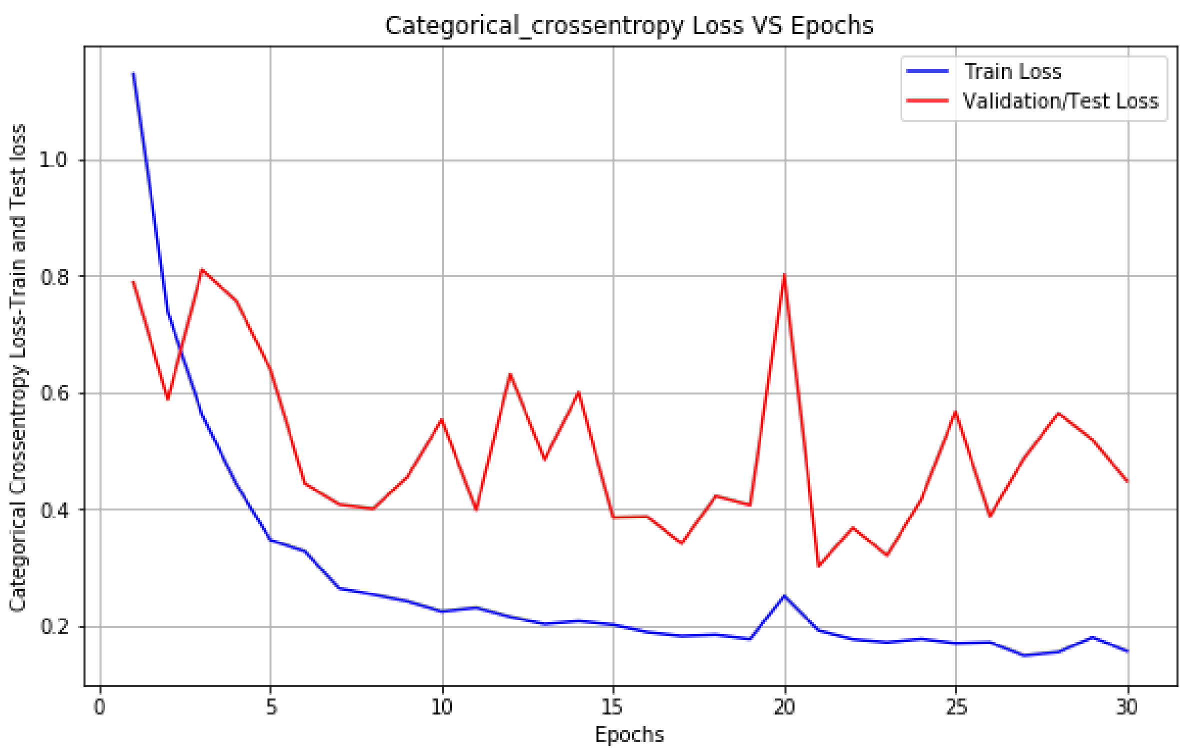
```
Epoch 00013: val_acc did not improve from 0.89074
Epoch 14/30
 - 98s - loss: 0.2082 - acc: 0.9319 - val_loss: 0.6002 - val_acc: 0.8164

Epoch 00014: val_acc did not improve from 0.89074
Epoch 15/30
 - 91s - loss: 0.2018 - acc: 0.9291 - val_loss: 0.3857 - val_acc: 0.9016

Epoch 00015: val_acc improved from 0.89074 to 0.90159, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 16/30
 - 76s - loss: 0.1892 - acc: 0.9361 - val_loss: 0.3871 - val_acc: 0.8890

Epoch 00016: val_acc did not improve from 0.90159
Epoch 17/30
 - 76s - loss: 0.1826 - acc: 0.9347 - val_loss: 0.3412 - val_acc: 0.9077

Epoch 00017: val_acc improved from 0.90159 to 0.90770, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 18/30
 - 84s - loss: 0.1849 - acc: 0.9361 - val_loss: 0.4224 - val_acc: 0.8918

Epoch 00018: val_acc did not improve from 0.90770
Epoch 19/30
 - 85s - loss: 0.1773 - acc: 0.9363 - val_loss: 0.4070 - val_acc: 0.9036

Epoch 00019: val_acc did not improve from 0.90770
Epoch 20/30
 - 87s - loss: 0.2513 - acc: 0.9310 - val_loss: 0.8024 - val_acc: 0.8554

Epoch 00020: val_acc did not improve from 0.90770
Epoch 21/30
 - 87s - loss: 0.1920 - acc: 0.9368 - val_loss: 0.3026 - val_acc: 0.9257

Epoch 00021: val_acc improved from 0.90770 to 0.92569, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 22/30
 - 90s - loss: 0.1770 - acc: 0.9363 - val_loss: 0.3681 - val_acc: 0.9158

Epoch 00022: val_acc did not improve from 0.92569
Epoch 23/30
 - 89s - loss: 0.1717 - acc: 0.9353 - val_loss: 0.3208 - val_acc: 0.9274

Epoch 00023: val_acc improved from 0.92569 to 0.92738, saving model to HAR_mo
del4_LSTM.hdf5
Epoch 24/30
 - 101s - loss: 0.1776 - acc: 0.9362 - val_loss: 0.4170 - val_acc: 0.8982

Epoch 00024: val_acc did not improve from 0.92738
Epoch 25/30
 - 91s - loss: 0.1700 - acc: 0.9378 - val_loss: 0.5672 - val_acc: 0.8958

Epoch 00025: val_acc did not improve from 0.92738
Epoch 26/30
 - 88s - loss: 0.1718 - acc: 0.9395 - val_loss: 0.3876 - val_acc: 0.9030
```

```
Epoch 00026: val_acc did not improve from 0.92738
Epoch 27/30
 - 79s - loss: 0.1491 - acc: 0.9456 - val_loss: 0.4873 - val_acc: 0.9057

Epoch 00027: val_acc did not improve from 0.92738
Epoch 28/30
 - 74s - loss: 0.1551 - acc: 0.9426 - val_loss: 0.5643 - val_acc: 0.9040

Epoch 00028: val_acc did not improve from 0.92738
Epoch 29/30
 - 73s - loss: 0.1804 - acc: 0.9392 - val_loss: 0.5189 - val_acc: 0.9023

Epoch 00029: val_acc did not improve from 0.92738
Epoch 30/30
 - 73s - loss: 0.1568 - acc: 0.9392 - val_loss: 0.4485 - val_acc: 0.9206

Epoch 00030: val_acc did not improve from 0.92738
```

Plotting the error plot

In [58]:
```
# Evaluating the model on test data
score4 = saved_model4.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score4[0])
print('Test accuracy:', score4[1])

# Plotting the results
# list of epoch numbers
x = list(range(1, epochs+1))

# we will get val_loss and val_acc only when you pass the paramter validation_
data
# val_loss : validation loss
vy = history.history['val_loss']

# Training loss
ty = history.history['loss']
# calling the dynamic function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.32075892655986304
Test accuracy: 0.9273837801153716
```



Confusion matrix

In [59]:
```
# Utility function to print the confusion matrix
import seaborn as sns
def confusion_matrix(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)])
    plt.figure(1,figsize=(10,8))
    df=pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pred l
abels'])
    sns.heatmap(df, annot=True,cmap='RdPu',fmt='g')
    return pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pr
ed labels'])
```

In [60]:
```
Y_pred = saved_model4.predict(X_test)
cm = confusion_matrix(Y_test, Y_pred)
```

Observation :-

With a architecture of Bi-Directional LSTM model with 32 hidden units and Dropout of 0.7, we got 92.73% accuracy and a multi class log-loss of 0.32 which is categorical cross entropy.

Bi-Directional LSTM model with 64 hidden units and Dropout

In [15]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [16]:
```
import warnings
warnings.filterwarnings("ignore")
from keras.layers import Bidirectional

# Initializing parameters
epochs = 30
batch_size = 32

# Initiliazing the sequential model
model5 = Sequential()
# Adding LSTM and Configuring the parameters
model5.add(Bidirectional(LSTM(64, return_sequences=True), input_shape=(timeste
ps, input_dim), merge_mode='concat'))
model5.add(BatchNormalization())
# Adding a dropout layer , to avoid overfitting
model5.add(Dropout(0.7))
# Adding LSTM layer
model5.add(LSTM(64))
# Adding dropout
model5.add(Dropout(0.7))
# Adding a dense output layer with sigmoid activation
model5.add(Dense(n_classes, activation='softmax'))
model5.summary()
```

```
WARNING:tensorflow:From I:\Python\Anaconda3\lib\site-packages\tensorflow\pyth
on\framework\op_def_library.py:263: colocate_with (from tensorflow.python.fra
mework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From I:\Python\Anaconda3\lib\site-packages\keras\backend\t
ensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_op
s) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
bidirectional_1 (Bidirection (None, 128, 128)          37888     
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128)          512       
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 128)          0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                49408     
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 390       
=================================================================
Total params: 88,198
Trainable params: 87,942
Non-trainable params: 256
_________________________________________________________________
```

In [17]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [18]:
```
# Reference -   https://machinelearningmastery.com/how-to-stop-training-deep-n
eural-networks-at-the-right-time-using-early-stopping/
# https://keras.io/callbacks/
# https://keras.io/layers/wrappers/

import warnings
warnings.filterwarnings("ignore")

# Compiling the model  , loss is categorical_crossentropy , as the problem is
 multi-class classification
model5.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac
curacy'])

# specifying the filepath to store the best model
filepath = "HAR_model5_LSTM.hdf5"
# early stopping
es5 = EarlyStopping(monitor='val_acc', mode='max', verbose=1, patience=20)
# model checkpoint to save the model with best accuracy
mc5 = ModelCheckpoint(filepath, monitor='val_acc', mode='max', verbose=1, save
_best_only=True)

# Training the model
history = model5.fit(X_train, Y_train, batch_size=batch_size, validation_data=
(X_test, Y_test), epochs=epochs, verbose=2, callbacks=[es5, mc5])

# Saving the best model
saved_model5 = load_model('HAR_model5_LSTM.hdf5')
```

```
WARNING:tensorflow:From I:\Python\Anaconda3\lib\site-packages\tensorflow\pyth
on\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is de
precated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 125s - loss: 0.9334 - acc: 0.6255 - val_loss: 0.5767 - val_acc: 0.7693

Epoch 00001: val_acc improved from -inf to 0.76926, saving model to HAR_model
5_LSTM.hdf5
Epoch 2/30
 - 111s - loss: 0.4787 - acc: 0.8264 - val_loss: 0.3236 - val_acc: 0.8860

Epoch 00002: val_acc improved from 0.76926 to 0.88599, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 3/30
 - 111s - loss: 0.2785 - acc: 0.9053 - val_loss: 0.5888 - val_acc: 0.8035

Epoch 00003: val_acc did not improve from 0.88599
Epoch 4/30
 - 112s - loss: 0.2536 - acc: 0.9136 - val_loss: 0.3555 - val_acc: 0.8860

Epoch 00004: val_acc did not improve from 0.88599
Epoch 5/30
 - 111s - loss: 0.1990 - acc: 0.9267 - val_loss: 0.4107 - val_acc: 0.8877

Epoch 00005: val_acc improved from 0.88599 to 0.88768, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 6/30
 - 112s - loss: 0.2419 - acc: 0.9142 - val_loss: 0.3230 - val_acc: 0.9030

Epoch 00006: val_acc improved from 0.88768 to 0.90295, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 7/30
 - 113s - loss: 0.1862 - acc: 0.9351 - val_loss: 0.3239 - val_acc: 0.9043

Epoch 00007: val_acc improved from 0.90295 to 0.90431, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 8/30
 - 115s - loss: 0.2032 - acc: 0.9293 - val_loss: 0.3037 - val_acc: 0.8962

Epoch 00008: val_acc did not improve from 0.90431
Epoch 9/30
 - 117s - loss: 0.1676 - acc: 0.9388 - val_loss: 0.3222 - val_acc: 0.9141

Epoch 00009: val_acc improved from 0.90431 to 0.91415, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 10/30
 - 116s - loss: 0.1639 - acc: 0.9382 - val_loss: 0.3339 - val_acc: 0.8972

Epoch 00010: val_acc did not improve from 0.91415
Epoch 11/30
 - 116s - loss: 0.1757 - acc: 0.9343 - val_loss: 0.3877 - val_acc: 0.9131

Epoch 00011: val_acc did not improve from 0.91415
Epoch 12/30
```

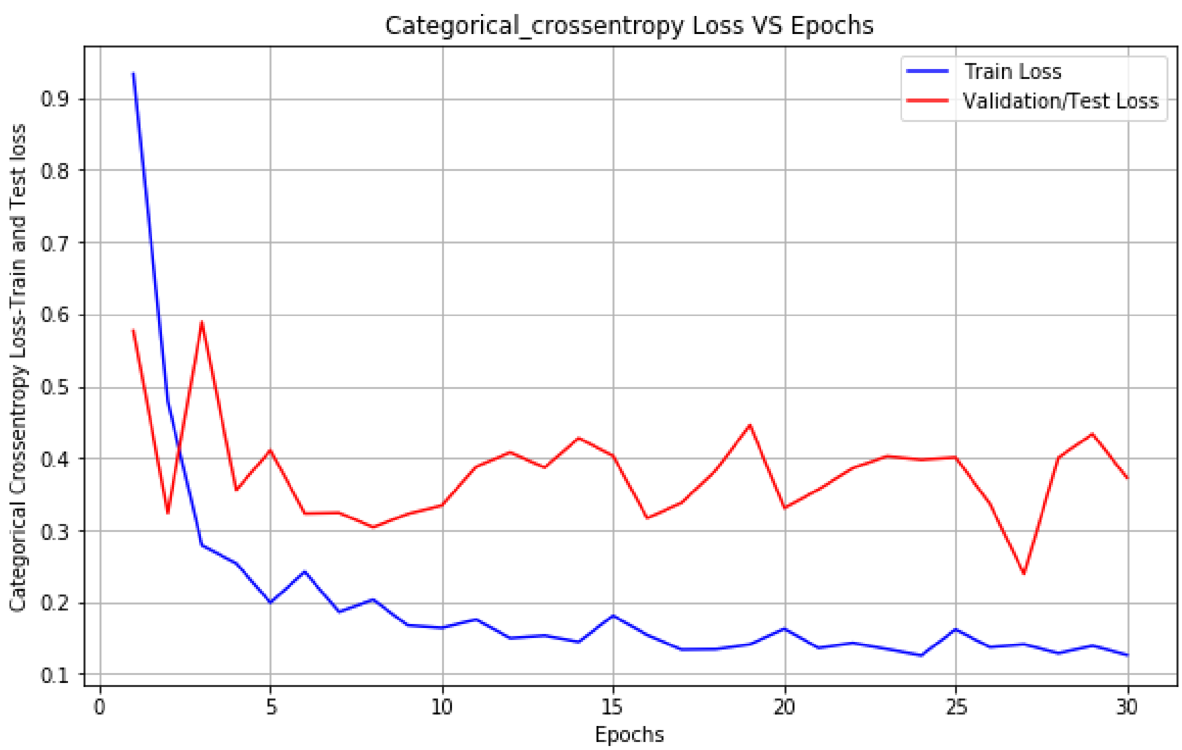
```
 - 115s - loss: 0.1498 - acc: 0.9426 - val_loss: 0.4077 - val_acc: 0.9148

Epoch 00012: val_acc improved from 0.91415 to 0.91483, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 13/30
 - 2091s - loss: 0.1534 - acc: 0.9431 - val_loss: 0.3867 - val_acc: 0.9148

Epoch 00013: val_acc did not improve from 0.91483
Epoch 14/30
 - 114s - loss: 0.1446 - acc: 0.9407 - val_loss: 0.4276 - val_acc: 0.8751

Epoch 00014: val_acc did not improve from 0.91483
Epoch 15/30
 - 111s - loss: 0.1809 - acc: 0.9361 - val_loss: 0.4030 - val_acc: 0.8958

Epoch 00015: val_acc did not improve from 0.91483
Epoch 16/30
 - 111s - loss: 0.1542 - acc: 0.9448 - val_loss: 0.3164 - val_acc: 0.9043

Epoch 00016: val_acc did not improve from 0.91483
Epoch 17/30
 - 111s - loss: 0.1338 - acc: 0.9448 - val_loss: 0.3378 - val_acc: 0.9158

Epoch 00017: val_acc improved from 0.91483 to 0.91585, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 18/30
 - 112s - loss: 0.1343 - acc: 0.9475 - val_loss: 0.3826 - val_acc: 0.9114

Epoch 00018: val_acc did not improve from 0.91585
Epoch 19/30
 - 113s - loss: 0.1412 - acc: 0.9456 - val_loss: 0.4458 - val_acc: 0.7944

Epoch 00019: val_acc did not improve from 0.91585
Epoch 20/30
 - 116s - loss: 0.1628 - acc: 0.9343 - val_loss: 0.3304 - val_acc: 0.9074

Epoch 00020: val_acc did not improve from 0.91585
Epoch 21/30
 - 117s - loss: 0.1361 - acc: 0.9463 - val_loss: 0.3563 - val_acc: 0.9135

Epoch 00021: val_acc did not improve from 0.91585
Epoch 22/30
 - 121s - loss: 0.1430 - acc: 0.9427 - val_loss: 0.3860 - val_acc: 0.9060

Epoch 00022: val_acc did not improve from 0.91585
Epoch 23/30
 - 131s - loss: 0.1346 - acc: 0.9444 - val_loss: 0.4024 - val_acc: 0.9046

Epoch 00023: val_acc did not improve from 0.91585
Epoch 24/30
 - 128s - loss: 0.1257 - acc: 0.9494 - val_loss: 0.3976 - val_acc: 0.9141

Epoch 00024: val_acc did not improve from 0.91585
Epoch 25/30
 - 137s - loss: 0.1618 - acc: 0.9421 - val_loss: 0.4009 - val_acc: 0.8853

Epoch 00025: val_acc did not improve from 0.91585
```

```
Epoch 26/30
 - 137s - loss: 0.1373 - acc: 0.9465 - val_loss: 0.3363 - val_acc: 0.9172

Epoch 00026: val_acc improved from 0.91585 to 0.91720, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 27/30
 - 129s - loss: 0.1412 - acc: 0.9455 - val_loss: 0.2387 - val_acc: 0.9247

Epoch 00027: val_acc improved from 0.91720 to 0.92467, saving model to HAR_mo
del5_LSTM.hdf5
Epoch 28/30
 - 120s - loss: 0.1287 - acc: 0.9493 - val_loss: 0.4005 - val_acc: 0.9114

Epoch 00028: val_acc did not improve from 0.92467
Epoch 29/30
 - 123s - loss: 0.1393 - acc: 0.9460 - val_loss: 0.4334 - val_acc: 0.9080

Epoch 00029: val_acc did not improve from 0.92467
Epoch 30/30
 - 126s - loss: 0.1263 - acc: 0.9482 - val_loss: 0.3727 - val_acc: 0.9084

Epoch 00030: val_acc did not improve from 0.92467
```

Plotting the error plot

In [19]:
```
# Evaluating the model on test data
score5 = saved_model5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score5[0])
print('Test accuracy:', score5[1])

# Plotting the results
# list of epoch numbers
x = list(range(1, epochs+1))

# we will get val_loss and val_acc only when you pass the paramter validation_
data
# val_loss : validation loss
vy = history.history['val_loss']

# Training loss
ty = history.history['loss']
# calling the dynamic function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.23872518167037293
Test accuracy: 0.9246691550729556
```



Confusion matrix

In [20]:
```
# Utility function to print the confusion matrix
import seaborn as sns
def confusion_matrix(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)])
    plt.figure(1,figsize=(10,8))
    df=pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pred l
abels'])
    sns.heatmap(df, annot=True,cmap='RdPu',fmt='g')
    return pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pr
ed labels'])
```

In [21]:
```
Y_pred = saved_model5.predict(X_test)
cm = confusion_matrix(Y_test, Y_pred)
```

Observation :-

With a architecture of Bi-Directional LSTM model with 64 hidden units and Dropout of 0.7, we got 92.46% accuracy and a multi class log-loss of 0.23 which is categorical cross entropy.

Bi-Directional LSTM model with 48 and 64 hidden units and Dropout

In [26]:
```
import warnings
warnings.filterwarnings("ignore")
from keras.layers import Bidirectional

# Initializing parameters
epochs = 30
batch_size = 32

# Initiliazing the sequential model
model6 = Sequential()
# Adding LSTM and Configuring the parameters
model6.add(Bidirectional(LSTM(48, return_sequences=True), input_shape=(timeste
ps, input_dim), merge_mode='concat'))
model6.add(BatchNormalization())
# Adding a dropout layer , to avoid overfitting
model6.add(Dropout(0.4))
# Adding LSTM layer
model6.add(LSTM(64))
# Adding dropout
model6.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model6.add(Dense(n_classes, activation='softmax'))
model6.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
bidirectional_4 (Bidirection (None, 128, 96)           22272     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128, 96)           384       
_________________________________________________________________
dropout_7 (Dropout)          (None, 128, 96)           0         
_________________________________________________________________
lstm_8 (LSTM)                (None, 64)                41216     
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 390       
=================================================================
Total params: 64,262
Trainable params: 64,070
Non-trainable params: 192
_________________________________________________________________
```

In [27]:
```
# Reference -   https://machinelearningmastery.com/how-to-stop-training-deep-n
eural-networks-at-the-right-time-using-early-stopping/
# https://keras.io/callbacks/
# https://keras.io/layers/wrappers/

import warnings
warnings.filterwarnings("ignore")

# Compiling the model  , loss is categorical_crossentropy , as the problem is
 multi-class classification
model6.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=[
'accuracy'])

# specifying the filepath to store the best model
filepath = "HAR_model6_LSTM.hdf5"
# early stopping
es6 = EarlyStopping(monitor='val_acc', mode='max', verbose=1, patience=20)
# model checkpoint to save the model with best accuracy
mc6 = ModelCheckpoint(filepath, monitor='val_acc', mode='max', verbose=1, save
_best_only=True)

# Training the model
history = model6.fit(X_train, Y_train, batch_size=batch_size, validation_data=
(X_test, Y_test), epochs=epochs, verbose=2, callbacks=[es6, mc6])

# Saving the best model
saved_model6 = load_model('HAR_model6_LSTM.hdf5')
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 124s - loss: 0.6080 - acc: 0.7764 - val_loss: 0.4049 - val_acc: 0.8558

Epoch 00001: val_acc improved from -inf to 0.85579, saving model to HAR_model
6_LSTM.hdf5
Epoch 2/30
 - 118s - loss: 0.2499 - acc: 0.9128 - val_loss: 0.3157 - val_acc: 0.8792

Epoch 00002: val_acc improved from 0.85579 to 0.87920, saving model to HAR_mo
del6_LSTM.hdf5
Epoch 3/30
 - 118s - loss: 0.1737 - acc: 0.9343 - val_loss: 0.2889 - val_acc: 0.9053

Epoch 00003: val_acc improved from 0.87920 to 0.90533, saving model to HAR_mo
del6_LSTM.hdf5
Epoch 4/30
 - 127s - loss: 0.1656 - acc: 0.9369 - val_loss: 0.2348 - val_acc: 0.9192

Epoch 00004: val_acc improved from 0.90533 to 0.91924, saving model to HAR_mo
del6_LSTM.hdf5
Epoch 5/30
 - 118s - loss: 0.1479 - acc: 0.9416 - val_loss: 0.3567 - val_acc: 0.8843

Epoch 00005: val_acc did not improve from 0.91924
Epoch 6/30
 - 112s - loss: 0.1433 - acc: 0.9437 - val_loss: 0.2659 - val_acc: 0.9332

Epoch 00006: val_acc improved from 0.91924 to 0.93315, saving model to HAR_mo
del6_LSTM.hdf5
Epoch 7/30
 - 117s - loss: 0.1334 - acc: 0.9480 - val_loss: 0.2890 - val_acc: 0.9220

Epoch 00007: val_acc did not improve from 0.93315
Epoch 8/30
 - 116s - loss: 0.1320 - acc: 0.9465 - val_loss: 0.4114 - val_acc: 0.9091

Epoch 00008: val_acc did not improve from 0.93315
Epoch 9/30
 - 119s - loss: 0.1322 - acc: 0.9479 - val_loss: 0.2787 - val_acc: 0.9264

Epoch 00009: val_acc did not improve from 0.93315
Epoch 10/30
 - 106s - loss: 0.1290 - acc: 0.9490 - val_loss: 0.2644 - val_acc: 0.9203

Epoch 00010: val_acc did not improve from 0.93315
Epoch 11/30
 - 101s - loss: 0.1280 - acc: 0.9502 - val_loss: 0.3716 - val_acc: 0.9087

Epoch 00011: val_acc did not improve from 0.93315
Epoch 12/30
 - 99s - loss: 0.1290 - acc: 0.9493 - val_loss: 0.3439 - val_acc: 0.9104

Epoch 00012: val_acc did not improve from 0.93315
Epoch 13/30
 - 100s - loss: 0.1287 - acc: 0.9516 - val_loss: 0.3770 - val_acc: 0.9138
```

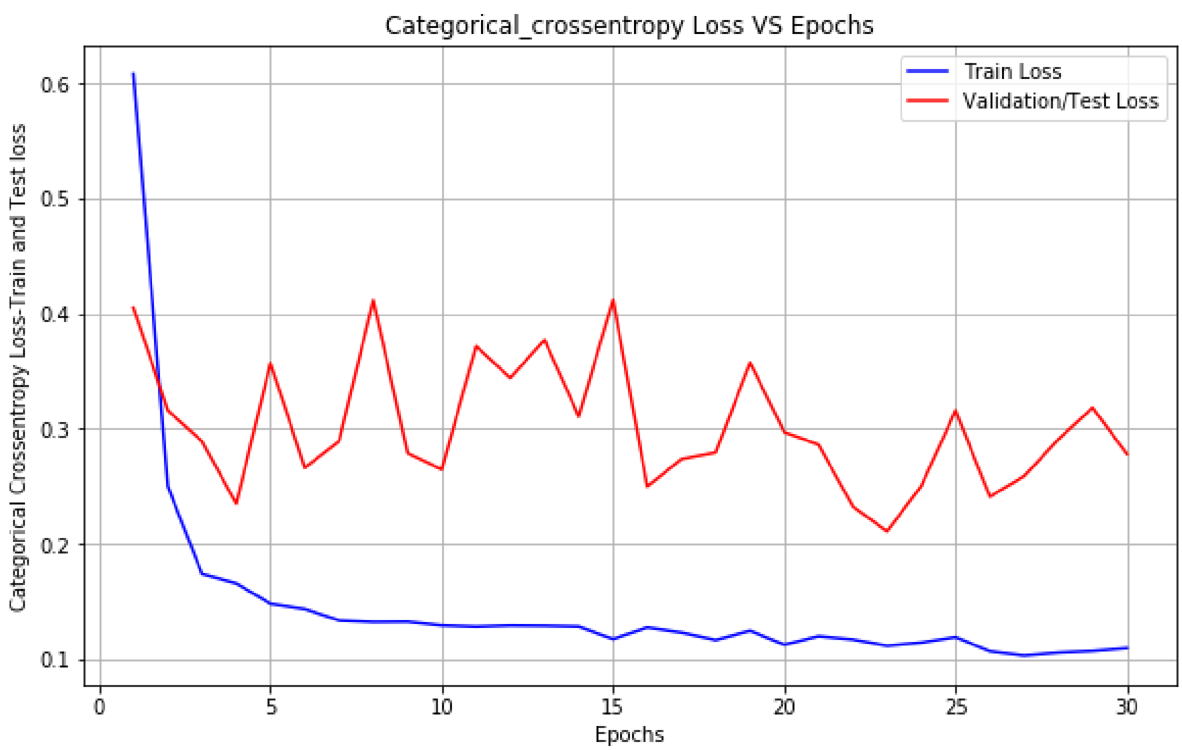
```
Epoch 00013: val_acc did not improve from 0.93315
Epoch 14/30
 - 100s - loss: 0.1281 - acc: 0.9483 - val_loss: 0.3104 - val_acc: 0.9094

Epoch 00014: val_acc did not improve from 0.93315
Epoch 15/30
 - 109s - loss: 0.1172 - acc: 0.9528 - val_loss: 0.4117 - val_acc: 0.8951

Epoch 00015: val_acc did not improve from 0.93315
Epoch 16/30
 - 107s - loss: 0.1273 - acc: 0.9512 - val_loss: 0.2497 - val_acc: 0.9206

Epoch 00016: val_acc did not improve from 0.93315
Epoch 17/30
 - 107s - loss: 0.1229 - acc: 0.9502 - val_loss: 0.2735 - val_acc: 0.9226

Epoch 00017: val_acc did not improve from 0.93315
Epoch 18/30
 - 108s - loss: 0.1164 - acc: 0.9523 - val_loss: 0.2792 - val_acc: 0.9348

Epoch 00018: val_acc improved from 0.93315 to 0.93485, saving model to HAR_mo
del6_LSTM.hdf5
Epoch 19/30
 - 109s - loss: 0.1245 - acc: 0.9512 - val_loss: 0.3572 - val_acc: 0.9067

Epoch 00019: val_acc did not improve from 0.93485
Epoch 20/30
 - 103s - loss: 0.1122 - acc: 0.9513 - val_loss: 0.2968 - val_acc: 0.9213

Epoch 00020: val_acc did not improve from 0.93485
Epoch 21/30
 - 101s - loss: 0.1197 - acc: 0.9527 - val_loss: 0.2861 - val_acc: 0.9243

Epoch 00021: val_acc did not improve from 0.93485
Epoch 22/30
 - 99s - loss: 0.1168 - acc: 0.9529 - val_loss: 0.2323 - val_acc: 0.9165

Epoch 00022: val_acc did not improve from 0.93485
Epoch 23/30
 - 100s - loss: 0.1114 - acc: 0.9540 - val_loss: 0.2107 - val_acc: 0.9427

Epoch 00023: val_acc improved from 0.93485 to 0.94265, saving model to HAR_mo
del6_LSTM.hdf5
Epoch 24/30
 - 100s - loss: 0.1141 - acc: 0.9516 - val_loss: 0.2502 - val_acc: 0.9301

Epoch 00024: val_acc did not improve from 0.94265
Epoch 25/30
 - 100s - loss: 0.1189 - acc: 0.9533 - val_loss: 0.3157 - val_acc: 0.9213

Epoch 00025: val_acc did not improve from 0.94265
Epoch 26/30
 - 100s - loss: 0.1066 - acc: 0.9563 - val_loss: 0.2409 - val_acc: 0.9399

Epoch 00026: val_acc did not improve from 0.94265
Epoch 27/30
 - 109s - loss: 0.1030 - acc: 0.9570 - val_loss: 0.2586 - val_acc: 0.9298
```

```
Epoch 00027: val_acc did not improve from 0.94265
Epoch 28/30
 - 112s - loss: 0.1055 - acc: 0.9572 - val_loss: 0.2906 - val_acc: 0.9233

Epoch 00028: val_acc did not improve from 0.94265
Epoch 29/30
 - 102s - loss: 0.1069 - acc: 0.9561 - val_loss: 0.3181 - val_acc: 0.9192

Epoch 00029: val_acc did not improve from 0.94265
Epoch 30/30
 - 102s - loss: 0.1093 - acc: 0.9543 - val_loss: 0.2780 - val_acc: 0.9270

Epoch 00030: val_acc did not improve from 0.94265
```

Ploting the error plot

In [28]:
```
# Evaluating the model on test data
score6 = saved_model6.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score6[0])
print('Test accuracy:', score6[1])

# Plotting the results
# list of epoch numbers
x = list(range(1, epochs+1))

# we will get val_loss and val_acc only when you pass the paramter validation_
data
# val_loss : validation loss
vy = history.history['val_loss']

# Training loss
ty = history.history['loss']
# calling the dynamic function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.21072815701344155
Test accuracy: 0.9426535459789617
```



Confusion matrix

In [29]:
```
# Utility function to print the confusion matrix
import seaborn as sns
def confusion_matrix(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)])
    plt.figure(1,figsize=(10,8))
    df=pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pred l
abels'])
    sns.heatmap(df, annot=True,cmap='RdPu',fmt='g')
    return pd.crosstab(Y_true, Y_pred, rownames=['True labels'], colnames=['Pr
ed labels'])
```

In [30]:
```
Y_pred = saved_model6.predict(X_test)
cm = confusion_matrix(Y_test, Y_pred)
```

Observation :-

With a architecture of Bi-Directional LSTM model with 48 and 64 hidden units and Dropout of 0.4 and 0.5, we got 94.26% accuracy and a multi class log-loss of 0.21 which is categorical cross entropy.

Models Summarization

In [37]:
```
from pandas import DataFrame
from pandas import DataFrame
HAR = {'Model':['1-LSTM layer with 32 hidden units','1 LSTM layer with 64 hidd
en units','2 LSTM layers with 32 hidden units',
               '2 LSTM layers with 64 hidden units','Bi-Directional LSTM model
with 32 hidden units',
               'Bi-Directional LSTM model with 64 hidden units','Bi-Directiona
l LSTM model with 48 and 64 hidden units'],
               'Dropout':['0.5','0.7','0.7','0.8','0.7','0.7','0.4,0.5'],
               'Activation':['softmax','softmax','softmax','Sigmoid','softmax'
,'softmax','softmax'],
               'Optimizer':['rmsprop','rmsprop','rmsprop','rmsprop','rmsprop',
'adam','rmsprop'],
               'Loss':['categorical_crossentropy','categorical_crossentropy',
'categorical_crossentropy','categorical_crossentropy',
                      'categorical_crossentropy','categorical_crossentropy','c
ategorical_crossentropy'],
               'Training accuracy':['0.94','0.935','0.93','0.918','0.93','0.9
4','0.95'],
               'Test accuracy':['0.898','0.932','0.924','0.916','0.927','0.92
4','0.94']}
```

In [38]:
```
Final_conclusions = DataFrame(HAR)
Final_conclusions
```

Out[38]:

	Model	Dropout	Activation	Optimizer	Loss	Training accuracy	Tes accuracy
0	1-LSTM layer with 32 hidden units	0.5	softmax	rmsprop	categorical_crossentropy	0.94	0.898
1	1 LSTM layer with 64 hidden units	0.7	softmax	rmsprop	categorical_crossentropy	0.935	0.932
2	2 LSTM layers with 32 hidden units	0.7	softmax	rmsprop	categorical_crossentropy	0.93	0.924
3	2 LSTM layers with 64 hidden units	0.8	Sigmoid	rmsprop	categorical_crossentropy	0.918	0.916
4	Bi-Directional LSTM model with 32 hidden units	0.7	softmax	rmsprop	categorical_crossentropy	0.93	0.927
5	Bi-Directional LSTM model with 64 hidden units	0.7	softmax	adam	categorical_crossentropy	0.94	0.924
6	Bi-Directional LSTM model with 48 and 64 hidde...	0.4,0.5	softmax	rmsprop	categorical_crossentropy	0.95	0.94

The best accuracy I got after training Bi-Directional LSTM model on Raw data is 0.947

Conclusions:-

From the above observations we can observe,

1. All the above models are trained on Raw data.
2. Model with Bi-Directional LSTM with 48 and 64 hidden units and dropouts of 0.4 and 0.5 gave good accuracy of 0.9427.
3. All other models have almost similar accuracy which is above 90%, even with different LSTM layes, different dropout rates and different number of hidden units.
4. With each epoch the accuracy increased.
5. I used sigmoid, softmax activations and rmsprop, adam optimizers for LSTM and Bi-Directional LSTM.
6. Bi-Directional LSTM models worked well than the simple LSTM models and gave good accuracy than the other models.
7. The Raw data with deep learning models gave good accuracy which is as good as the accuracy of expert engineered features with classical Machine learning models.
8. If we train Deep learning models on expert engineered features , we can get still more better accuracy similar to classical Machine learning models.