Human_Activity_Recognition

April 18, 2020

[1]:
```
pip install -U keras-tuner
```

```
Collecting keras-tuner
  Downloading https://files.pythonhosted.org/packages/a7/f7/4b41b6832abf4c
9bef71a664dc563adb25afc5812831667c6db572b1a261/keras-tuner-1.0.1.tar.gz (54kB)
     |                  | 61kB 3.5MB/s
Requirement already satisfied, skipping upgrade: future in
/usr/local/lib/python3.6/dist-packages (from keras-tuner) (0.16.0)
Requirement already satisfied, skipping upgrade: numpy in
/usr/local/lib/python3.6/dist-packages (from keras-tuner) (1.18.2)
Requirement already satisfied, skipping upgrade: tabulate in
/usr/local/lib/python3.6/dist-packages (from keras-tuner) (0.8.7)
Collecting terminaltables
  Downloading https://files.pythonhosted.org/packages/9b/c4/4a21174f32f8a7e11047
98c445dacdc1d4df86f2f26722767034e4de4bff/terminaltables-3.1.0.tar.gz
Collecting colorama
  Downloading https://files.pythonhosted.org/packages/c9/dc/45cdef1b4d119eb96316
b3117e6d5708a08029992b2fee2c143c7a0a5cc5/colorama-0.4.3-py2.py3-none-any.whl
Requirement already satisfied, skipping upgrade: tqdm in
/usr/local/lib/python3.6/dist-packages (from keras-tuner) (4.38.0)
Requirement already satisfied, skipping upgrade: requests in
/usr/local/lib/python3.6/dist-packages (from keras-tuner) (2.21.0)
Requirement already satisfied, skipping upgrade: scipy in
/usr/local/lib/python3.6/dist-packages (from keras-tuner) (1.4.1)
Requirement already satisfied, skipping upgrade: scikit-learn in
/usr/local/lib/python3.6/dist-packages (from keras-tuner) (0.22.2.post1)
Requirement already satisfied, skipping upgrade: chardet<3.1.0,>=3.0.2 in
/usr/local/lib/python3.6/dist-packages (from requests->keras-tuner) (3.0.4)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in
/usr/local/lib/python3.6/dist-packages (from requests->keras-tuner) (2020.4.5.1)
Requirement already satisfied, skipping upgrade: idna<2.9,>=2.5 in
/usr/local/lib/python3.6/dist-packages (from requests->keras-tuner) (2.8)
Requirement already satisfied, skipping upgrade: urllib3<1.25,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests->keras-tuner) (1.24.3)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in
/usr/local/lib/python3.6/dist-packages (from scikit-learn->keras-tuner) (0.14.1)
Building wheels for collected packages: keras-tuner, terminaltables
  Building wheel for keras-tuner (setup.py) … done
```

```
  Created wheel for keras-tuner: filename=keras_tuner-1.0.1-cp36-none-any.whl
size=73200
sha256=395d213972c517ee2516f9480d3d03344b8543957dc8e941afb8d63854c3f4e3
  Stored in directory: /root/.cache/pip/wheels/b9/cc/62/52716b70dd90f3db12519233
c3a93a5360bc672da1a10ded43
  Building wheel for terminaltables (setup.py) … done
  Created wheel for terminaltables: filename=terminaltables-3.1.0-cp36-none-
any.whl size=15356
sha256=450c5e3d9d2a0caef46beea52a689c89a4e79b10e2a5526f511ecae2d38b5490
  Stored in directory: /root/.cache/pip/wheels/30/6b/50/6c75775b681fb36cdfac7f19
799888ef9d8813aff9e379663e
Successfully built keras-tuner terminaltables
Installing collected packages: terminaltables, colorama, keras-tuner
Successfully installed colorama-0.4.3 keras-tuner-1.0.1 terminaltables-3.1.0
```

[2]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten, Dense,␣
 ↪Dropout,BatchNormalization,LSTM,Conv1D,MaxPooling1D
from kerastuner import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters
import time
from tensorflow.keras.layers import Flatten
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.preprocessing import StandardScaler
from keras import regularizers, optimizers
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
Using TensorFlow backend.
```

[0]:
```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.random.set_seed(42)
```

[4]:
```
print(tf.__version__)
```

```
2.2.0-rc3
```

Load Data

[5]:
```
cd /content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)
```

```
/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)
```

[0]:
```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

[0]:
```
def data():
    ''' This function is to load the data'''
    SIGNALS =␣
 ↪["body_acc_x","body_acc_y","body_acc_z","body_gyro_x","body_gyro_y","body_gyro_z","total_ac
    signals_data = []
    for signal in SIGNALS:
        filename = f'/content/drive/My Drive/HumanActivityRecognition.zip␣
 ↪(Unzipped Files)/HAR/UCI_HAR_Dataset/train/Inertial Signals/{signal}_train.
 ↪txt'
        signals_data.append(pd.read_csv(filename, delim_whitespace=True,␣
 ↪header=None).values)
    X_train = np.transpose(signals_data, (1, 2, 0))

    signals_data = []
    for signal in SIGNALS:
        filename = f'/content/drive/My Drive/HumanActivityRecognition.zip␣
 ↪(Unzipped Files)/HAR/UCI_HAR_Dataset/test/Inertial Signals/{signal}_test.txt'
        signals_data.append(
            pd.read_csv(filename, delim_whitespace=True, header=None).values )
    X_test = np.transpose(signals_data, (1, 2, 0))

    filename = f'/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped␣
 ↪Files)/HAR/UCI_HAR_Dataset/train/y_train.txt'
    y = pd.read_csv(filename, delim_whitespace=True, header=None)[0]
    Y_train = pd.get_dummies(y).values

    filename = f'/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped␣
 ↪Files)/HAR/UCI_HAR_Dataset/test/y_test.txt'
    y = pd.read_csv(filename, delim_whitespace=True, header=None)[0]
    Y_test = pd.get_dummies(y).values

    return X_train , Y_train , X_test , Y_test
```

[0]:
```
# Loading the train and test data
X_train_full, Y_train_full,X_test_full, Y_test_full = data()
```

[9]:
```
timesteps = len(X_train_full[0])
input_dim = len(X_train_full[0][0])
n_classes = 6
```

```python
print(timesteps)
print(input_dim)
print(len(X_train_full))
```

```
128
9
7352
```

[0]:
```python
'''def best_hyperparameters(hp):
  model = Sequential()

  model.add(LSTM(units=hp.
↪Int('units',min_value=95,max_value=130,step=5),input_shape=(timesteps,␣
↪input_dim),return_sequences=True))

  model.add(BatchNormalization())

  model.add(Dropout(0.7))

  model.add(LSTM(50))

  model.add(Dropout(0.70))
  #model.add(BatchNormalization())

  # model.add(Dropout(hp.Float('dropout',min_value=0.1,max_value=0.9,step=0.
↪1)))

  model.add(Flatten())

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

  model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',␣
↪values=[1e-1,1e-2,1e-3,1e-4,1e-5])),loss='categorical_crossentropy',metrics=['accuracy'])␣
↪

  return model'''
```

[0]:
```
"def best_hyperparameters(hp):\n  model = Sequential()\n\n  model.add(LSTM(units
=hp.Int('units',min_value=95,max_value=130,step=5),input_shape=(timesteps,
input_dim),return_sequences=True)) \n  \n  model.add(BatchNormalization()) \n\n
model.add(Dropout(0.7)) \n\n  model.add(LSTM(50))\n  \n
model.add(Dropout(0.70))\n  #model.add(BatchNormalization()) \n\n  #
model.add(Dropout(hp.Float('dropout',min_value=0.1,max_value=0.9,step=0.1)))
\n\n  model.add(Flatten())\n  \n  model.add(Dense(n_classes,
activation='sigmoid'))\n\n
model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', values=
[1e-1,1e-2,1e-3,1e-4,1e-5])),loss='categorical_crossentropy',metrics=['accuracy'
```

```
]) \n\n  return model"
```

[0]:
```
#␣
↪tuner_search=RandomSearch(best_hyperparameters,objective='val_accuracy',max_trials=5,direct
↪activity")
```

[0]:
```
# tuner_search.search(X_train,Y_train,epochs=15,batch_size␣
↪=32,validation_split=0.2)
```

[0]:
```
from keras.regularizers import L1L2
reg = L1L2(0.01, 0.01)
model = Sequential()
model.add(LSTM(100, input_shape=(timesteps, input_dim),␣
↪kernel_initializer='glorot_normal' , return_sequences=True,␣
↪bias_regularizer=reg))
model.add(BatchNormalization())
model.add(Dropout(0.70))
model.add(LSTM(50))
model.add(Dropout(0.70))
model.add(Dense(n_classes, activation='sigmoid'))
print("Model Summary: ")
model.summary()
```

```
Model Summary:
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 128, 100)          44000
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 100)          400
_________________________________________________________________
dropout (Dropout)            (None, 128, 100)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                30200
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 6)                 306
=================================================================
Total params: 74,906
Trainable params: 74,706
Non-trainable params: 200
_________________________________________________________________
```

[0]:
```
# Compiling the model
```

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

[0]:
```
# Training the model
model.fit(X_train_full,Y_train_full,batch_size=16,validation_data=(X_test_full,␣
→Y_test_full),epochs=50,verbose=1)
```

```
Epoch 1/50
460/460 [==============================] - 32s 69ms/step - loss: 1.7053 -
accuracy: 0.5660 - val_loss: 1.0362 - val_accuracy: 0.7004
Epoch 2/50
460/460 [==============================] - 31s 67ms/step - loss: 0.6516 -
accuracy: 0.7482 - val_loss: 0.2428 - val_accuracy: 0.8734
Epoch 3/50
460/460 [==============================] - 31s 67ms/step - loss: 0.1678 -
accuracy: 0.8796 - val_loss: 0.0876 - val_accuracy: 0.8989
Epoch 4/50
460/460 [==============================] - 31s 67ms/step - loss: 0.1177 -
accuracy: 0.9189 - val_loss: 0.1405 - val_accuracy: 0.8643
Epoch 5/50
460/460 [==============================] - 31s 67ms/step - loss: 0.1071 -
accuracy: 0.9242 - val_loss: 0.0993 - val_accuracy: 0.8985
Epoch 6/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0943 -
accuracy: 0.9327 - val_loss: 0.0793 - val_accuracy: 0.9155
Epoch 7/50
460/460 [==============================] - 31s 68ms/step - loss: 0.0914 -
accuracy: 0.9312 - val_loss: 0.1342 - val_accuracy: 0.8897
Epoch 8/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0846 -
accuracy: 0.9365 - val_loss: 0.0978 - val_accuracy: 0.9067
Epoch 9/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0782 -
accuracy: 0.9388 - val_loss: 0.1317 - val_accuracy: 0.8955
Epoch 10/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0808 -
accuracy: 0.9377 - val_loss: 0.0812 - val_accuracy: 0.9158
Epoch 11/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0746 -
accuracy: 0.9378 - val_loss: 0.0920 - val_accuracy: 0.9216
Epoch 12/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0745 -
accuracy: 0.9423 - val_loss: 0.1266 - val_accuracy: 0.8999
Epoch 13/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0721 -
accuracy: 0.9400 - val_loss: 0.0892 - val_accuracy: 0.9145
Epoch 14/50
```

```
460/460 [==============================] - 31s 67ms/step - loss: 0.0691 -
accuracy: 0.9433 - val_loss: 0.1345 - val_accuracy: 0.9070
Epoch 15/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0698 -
accuracy: 0.9448 - val_loss: 0.1304 - val_accuracy: 0.9114
Epoch 16/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0714 -
accuracy: 0.9430 - val_loss: 0.1673 - val_accuracy: 0.8860
Epoch 17/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0695 -
accuracy: 0.9438 - val_loss: 0.1283 - val_accuracy: 0.9006
Epoch 18/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0691 -
accuracy: 0.9426 - val_loss: 0.1198 - val_accuracy: 0.9101
Epoch 19/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0677 -
accuracy: 0.9412 - val_loss: 0.1241 - val_accuracy: 0.9233
Epoch 20/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0702 -
accuracy: 0.9422 - val_loss: 0.1030 - val_accuracy: 0.9162
Epoch 21/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0667 -
accuracy: 0.9444 - val_loss: 0.0875 - val_accuracy: 0.9240
Epoch 22/50
460/460 [==============================] - 31s 68ms/step - loss: 0.0694 -
accuracy: 0.9479 - val_loss: 0.1100 - val_accuracy: 0.9108
Epoch 23/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0667 -
accuracy: 0.9453 - val_loss: 0.1173 - val_accuracy: 0.9121
Epoch 24/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0671 -
accuracy: 0.9453 - val_loss: 0.1362 - val_accuracy: 0.9192
Epoch 25/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0644 -
accuracy: 0.9449 - val_loss: 0.1236 - val_accuracy: 0.9131
Epoch 26/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0707 -
accuracy: 0.9427 - val_loss: 0.1104 - val_accuracy: 0.9240
Epoch 27/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0654 -
accuracy: 0.9455 - val_loss: 0.1410 - val_accuracy: 0.9084
Epoch 28/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0671 -
accuracy: 0.9450 - val_loss: 0.1181 - val_accuracy: 0.9206
Epoch 29/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0688 -
accuracy: 0.9450 - val_loss: 0.1315 - val_accuracy: 0.9155
Epoch 30/50
```

```
460/460 [==============================] - 31s 67ms/step - loss: 0.0635 -
accuracy: 0.9487 - val_loss: 0.0923 - val_accuracy: 0.9223
Epoch 31/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0627 -
accuracy: 0.9495 - val_loss: 0.0994 - val_accuracy: 0.9186
Epoch 32/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0642 -
accuracy: 0.9474 - val_loss: 0.1225 - val_accuracy: 0.9148
Epoch 33/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0682 -
accuracy: 0.9453 - val_loss: 0.0932 - val_accuracy: 0.9209
Epoch 34/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0638 -
accuracy: 0.9475 - val_loss: 0.1241 - val_accuracy: 0.9169
Epoch 35/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0619 -
accuracy: 0.9509 - val_loss: 0.1454 - val_accuracy: 0.9158
Epoch 36/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0685 -
accuracy: 0.9475 - val_loss: 0.1177 - val_accuracy: 0.9182
Epoch 37/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0655 -
accuracy: 0.9470 - val_loss: 0.1473 - val_accuracy: 0.9196
Epoch 38/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0688 -
accuracy: 0.9476 - val_loss: 0.1026 - val_accuracy: 0.9186
Epoch 39/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0630 -
accuracy: 0.9512 - val_loss: 0.1233 - val_accuracy: 0.9284
Epoch 40/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0599 -
accuracy: 0.9524 - val_loss: 0.1423 - val_accuracy: 0.9182
Epoch 41/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0639 -
accuracy: 0.9455 - val_loss: 0.1243 - val_accuracy: 0.9175
Epoch 42/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0646 -
accuracy: 0.9494 - val_loss: 0.1305 - val_accuracy: 0.9138
Epoch 43/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0654 -
accuracy: 0.9509 - val_loss: 0.1227 - val_accuracy: 0.9298
Epoch 44/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0625 -
accuracy: 0.9460 - val_loss: 0.1305 - val_accuracy: 0.9192
Epoch 45/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0623 -
accuracy: 0.9523 - val_loss: 0.1199 - val_accuracy: 0.9240
Epoch 46/50
```

```
460/460 [==============================] - 31s 67ms/step - loss: 0.0615 -
accuracy: 0.9482 - val_loss: 0.1428 - val_accuracy: 0.9104
Epoch 47/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0644 -
accuracy: 0.9510 - val_loss: 0.1200 - val_accuracy: 0.9206
Epoch 48/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0611 -
accuracy: 0.9497 - val_loss: 0.1490 - val_accuracy: 0.9158
Epoch 49/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0609 -
accuracy: 0.9498 - val_loss: 0.1452 - val_accuracy: 0.9148
Epoch 50/50
460/460 [==============================] - 31s 67ms/step - loss: 0.0615 -
accuracy: 0.9489 - val_loss: 0.1342 - val_accuracy: 0.9175
```

[0]: <tensorflow.python.keras.callbacks.History at 0x7efc9a5eda20>

[0]:
```python
print()
scores = model.evaluate(X_test_full, Y_test_full, verbose=0)
print("Test Accuracy: %f%%" % (scores[1]*100))
print()
```

```
Test Accuracy: 91.754329%
```

[0]:
```python
def plot_confusion_matrix_lstm(y_test, y_predict):
    result = confusion_matrix(y_test, y_predict)

    plt.figure(figsize=(12, 10))
    sns.heatmap(result,
                xticklabels= list(ACTIVITIES.values()),
                yticklabels=list(ACTIVITIES.values()),
                annot=True, fmt="d");
    plt.title("Confusion matrix")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

[0]:
```python
# 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)])
```

```
    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'])
# Confusion Matrix
confusion_matrix(Y_test_full, model.predict(X_test_full))
```

[0]:
```
Pred                LAYING  SITTING  …  WALKING_DOWNSTAIRS  WALKING_UPSTAIRS
True                                 …
LAYING                 537        0  …                   0                 0
SITTING                  0      382  …                   0                 2
STANDING                 0       85  …                   0                 0
WALKING                  0        0  …                  28                 3
WALKING_DOWNSTAIRS       0        0  …                 415                 5
WALKING_UPSTAIRS         0        3  …                   8               459

[6 rows x 6 columns]
```

Divide and Conquer

[0]:
```
def data_train():
    ''' This function is to load the data'''
    SIGNALS =␣
 ↪["body_acc_x","body_acc_y","body_acc_z","body_gyro_x","body_gyro_y","body_gyro_z","total_ac
    signals_data = []
    for signal in SIGNALS:
        filename = f'/content/drive/My Drive/HumanActivityRecognition.zip␣
 ↪(Unzipped Files)/HAR/UCI_HAR_Dataset/train/Inertial Signals/{signal}_train.
 ↪txt'
        signals_data.append(pd.read_csv(filename, delim_whitespace=True,␣
 ↪header=None).values)
    X_train = np.transpose(signals_data, (1, 2, 0))

    return X_train

def data_test():
    ''' This function is to load the data'''
    SIGNALS =␣
 ↪["body_acc_x","body_acc_y","body_acc_z","body_gyro_x","body_gyro_y","body_gyro_z","total_ac
    signals_data = []
    for signal in SIGNALS:
        filename = f'/content/drive/My Drive/HumanActivityRecognition.zip␣
 ↪(Unzipped Files)/HAR/UCI_HAR_Dataset/test/Inertial Signals/{signal}_test.txt'
        signals_data.append(pd.read_csv(filename, delim_whitespace=True,␣
 ↪header=None).values)
    X_test = np.transpose(signals_data, (1, 2, 0))

    return X_test
```

```
```

[0]:
```
def load_y_static_dynamic(subset):
  filename = f'/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped␣
↪Files)/HAR/UCI_HAR_Dataset/{subset}/y_{subset}.txt'
  y = pd.read_csv(filename, delim_whitespace=True, header=None)[0]
  Y_train = pd.get_dummies(y).values
  y[y<=3] = 0 # Dynamic activities
  y[y>3] = 1  # Static activities
  return pd.get_dummies(y).values
```

[0]:
```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = data_train(), data_test()
    y_train, y_test = load_y_static_dynamic('train'),␣
↪load_y_static_dynamic('test')

    return X_train, X_test, y_train, y_test
```

[13]:
```
x_train, x_test, y_train, y_test = load_data()
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

[13]: ((7352, 128, 9), (2947, 128, 9), (7352, 2), (2947, 2))

[14]:
```
timesteps = len(x_train[0])
input_dim = len(x_train[0][0])

print(timesteps)
print(input_dim)
print(len(x_train))
```

```
128
9
7352
```

[15]:
```
model1= Sequential()
model1.add(Conv1D(filters= 32, kernel_size= 3, activation= 'relu',␣
↪kernel_initializer= 'he_normal',kernel_regularizer = regularizers.l2(0.1),␣
↪input_shape=(timesteps, input_dim)))
model1.add(Dropout(0.5))
model1.add(MaxPooling1D(pool_size= 2))
model1.add(Flatten())
model1.add(Dense(units= 2, activation= 'softmax'))
model1.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 126, 32)           896

_________________________________________________________________
dropout (Dropout)            (None, 126, 32)           0

_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 63, 32)            0

_________________________________________________________________
flatten (Flatten)            (None, 2016)              0

_________________________________________________________________
dense (Dense)                (None, 2)                 4034
=================================================================
Total params: 4,930
Trainable params: 4,930
Non-trainable params: 0

_________________________________________________________________
```

[0]:
```
# Compiling the model
model1.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

[17]:
```
# Initializing parameters
epochs = 10
batch_size = 16
# Training the model
history1= model1.fit(x_train, y_train, batch_size=batch_size,␣
 ↪validation_data=(x_test, y_test), epochs=epochs)
```

```
Epoch 1/10
460/460 [==============================] - 3s 5ms/step - loss: 2.4361 -
accuracy: 0.9744 - val_loss: 0.5410 - val_accuracy: 0.9854
Epoch 2/10
460/460 [==============================] - 2s 5ms/step - loss: 0.1865 -
accuracy: 0.9977 - val_loss: 0.1080 - val_accuracy: 0.9949
Epoch 3/10
460/460 [==============================] - 2s 5ms/step - loss: 0.0502 -
accuracy: 0.9977 - val_loss: 0.0714 - val_accuracy: 0.9959
Epoch 4/10
460/460 [==============================] - 2s 5ms/step - loss: 0.0376 -
accuracy: 0.9980 - val_loss: 0.0709 - val_accuracy: 0.9908
Epoch 5/10
460/460 [==============================] - 2s 5ms/step - loss: 0.0320 -
accuracy: 0.9980 - val_loss: 0.0527 - val_accuracy: 0.9966
Epoch 6/10
460/460 [==============================] - 2s 5ms/step - loss: 0.0264 -
accuracy: 0.9992 - val_loss: 0.0491 - val_accuracy: 0.9946
```

```
Epoch 7/10
460/460 [==============================] - 2s 5ms/step - loss: 0.0249 -
accuracy: 0.9982 - val_loss: 0.0415 - val_accuracy: 0.9969
Epoch 8/10
460/460 [==============================] - 2s 5ms/step - loss: 0.0230 -
accuracy: 0.9985 - val_loss: 0.0566 - val_accuracy: 0.9932
Epoch 9/10
460/460 [==============================] - 2s 5ms/step - loss: 0.0214 -
accuracy: 0.9981 - val_loss: 0.0591 - val_accuracy: 0.9959
Epoch 10/10
460/460 [==============================] - 2s 5ms/step - loss: 0.0199 -
accuracy: 0.9986 - val_loss: 0.0307 - val_accuracy: 0.9973
```

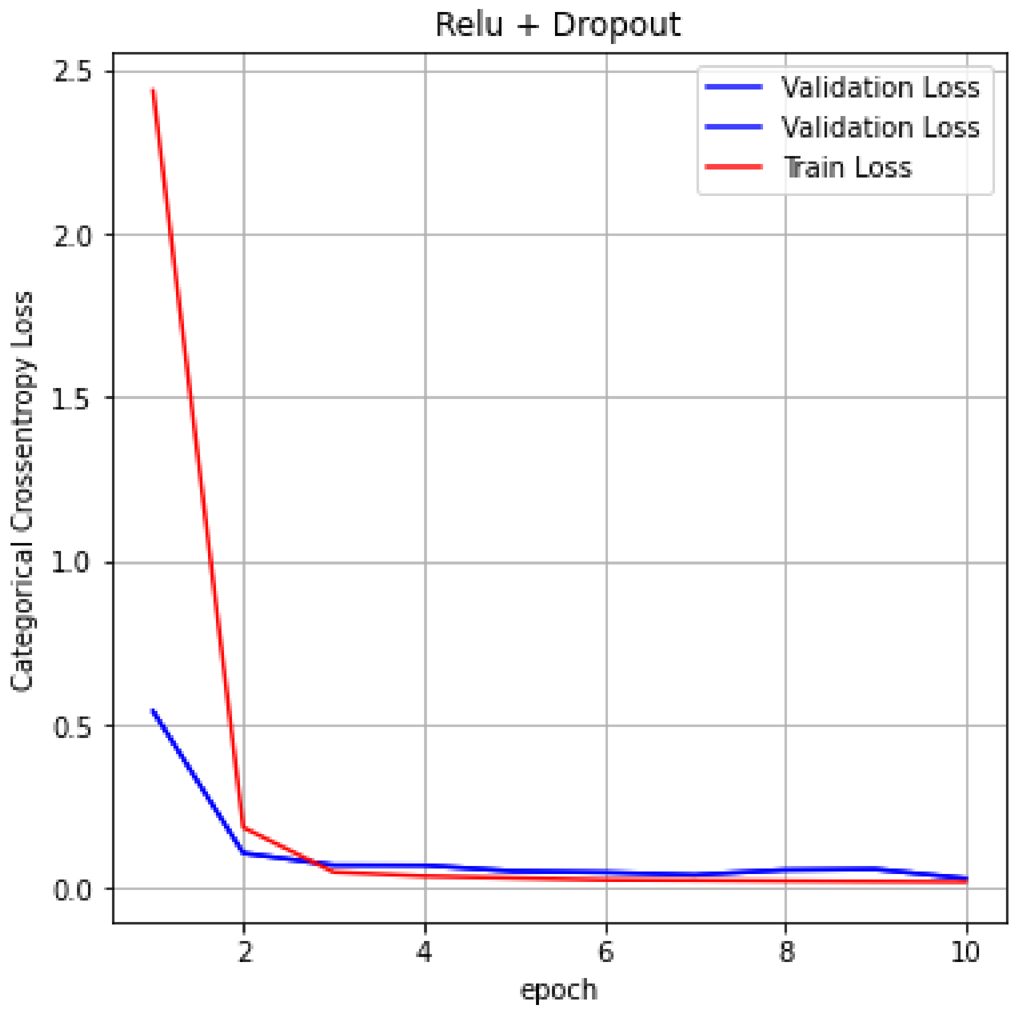
[18]:
```
score1 = model1.evaluate(x_test, y_test)
```

```
93/93 [==============================] - 0s 3ms/step - loss: 0.0306 - accuracy:
0.9973
```

[0]:
```
def plt_dynamic(x, vy, ty, ax, color = 'b'):
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, ty, 'r', label = 'Train Loss')
    plt.grid()
    plt.legend()
    fig.canvas.draw()
```

[20]:
```
fig, ax = plt.subplots(1,1, figsize = (6, 6))
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')
plt.title('Relu + Dropout')

# list of epoch numbers: epoch = 10
x = list(range(1,10+1))
vy = history1.history['val_loss']
ty = history1.history['loss']
plt_dynamic(x, vy, ty, ax)
```

[20]:



[21]:
```
import pickle
model1.save('model1')
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/resource_variable_ops.py:1817: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: model1/assets
```

Static Activities

[0]:
```
def data_load(subset):
    ''' This function is to load the data'''
```

```
    SIGNALS =␣
↪["body_acc_x","body_acc_y","body_acc_z","body_gyro_x","body_gyro_y","body_gyro_z","total_ac
    signals_data = []
    for signal in SIGNALS:
        filename = f'/content/drive/My Drive/HumanActivityRecognition.zip␣
↪(Unzipped Files)/HAR/UCI_HAR_Dataset/{subset}/Inertial Signals/
↪{signal}_{subset}.txt'
        signals_data.append(pd.read_csv(filename, delim_whitespace=True,␣
↪header=None).values)
    data = np.transpose(signals_data, (1, 2, 0))
    return data
```

[0]:

```
# Labelling the classes in 'y' after OHE

label = {0:'SITTING', 1:'STANDING', 2:'LAYING'}

def load_y_static(subset):
  filename = f'/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped␣
↪Files)/HAR/UCI_HAR_Dataset/{subset}/y_{subset}.txt'
  y = pd.read_csv(filename, delim_whitespace=True, header=None)[0]
  y_subset = y>3
  y = y[y_subset]
  return pd.get_dummies(y).values,y_subset
```

[0]:

```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = data_load('train'), data_load('test')
    y_train, y_train1 = load_y_static('train')
    y_test, y_test1 =  load_y_static('test')

    X_train = X_train[y_train1]

    X_test = X_test[y_test1]

    return X_train, X_test, y_train, y_test
```

[25]:

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

print('X_train shape is: ',X_train.shape)
print('Y_train shape is: ',Y_train.shape)
print('X_test shape is: ',X_test.shape)
print('Y_test shape is: ',Y_test.shape)
```

```
X_train shape is:  (4067, 128, 9)
Y_train shape is:  (4067, 3)
X_test shape is:  (1560, 128, 9)
Y_test shape is:  (1560, 3)
```

[26]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])


print('Timesteps:', timesteps)
print('Input Dim:', input_dim)
print('No. of Train datapoints:', len(X_train))
```

```
Timesteps: 128
Input Dim: 9
No. of Train datapoints: 4067
```

[27]:

```
model_s= Sequential()
model_s.add(Conv1D(filters= 64, kernel_size= 5, activation= 'relu',␣
 ↪kernel_initializer= 'he_uniform', input_shape=(timesteps, input_dim)))
model_s.add(Conv1D(filters= 64, kernel_size= 5, activation= 'relu',␣
 ↪kernel_initializer= 'he_uniform'))
model_s.add(MaxPooling1D(pool_size= 2, padding= 'same'))
model_s.add(Dropout(0.40))
model_s.add(Conv1D(filters= 32, kernel_size= 5, activation= 'relu',␣
 ↪kernel_initializer= 'he_uniform'))
model_s.add(Conv1D(filters= 32, kernel_size= 5, activation= 'relu',␣
 ↪kernel_initializer= 'he_uniform',))

# https://stackoverflow.com/a/49089027/10219869
# https://stackoverflow.com/a/58498450/10219869

model_s.add(MaxPooling1D(pool_size= 2, padding= 'same'))
model_s.add(BatchNormalization())
model_s.add(Dropout(0.40))
model_s.add(Flatten())
model_s.add(Dense(units= 100, activation= 'relu'))
model_s.add(BatchNormalization())
model_s.add(Dropout(0.40))
model_s.add(Dense(units= 3, activation= 'softmax'))
model_s.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 124, 64)           2944
_________________________________________________________________
```

```
conv1d_2 (Conv1D)            (None, 120, 64)          20544
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 60, 64)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 64)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 56, 32)           10272
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 52, 32)           5152
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 26, 32)           0
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 32)           128
_________________________________________________________________
dropout_2 (Dropout)          (None, 26, 32)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 832)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)              83300
_________________________________________________________________
batch_normalization_1 (Batch (None, 100)              400
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)              0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                303
=================================================================
Total params: 123,043
Trainable params: 122,779
Non-trainable params: 264
_________________________________________________________________
```

[0]:

```
# Compiling the model
model_s.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

[29]:

```
# Initializing parameters
epochs =100
batch_size =20
# Training the model
history_s= model_s.fit(X_train, Y_train, batch_size=batch_size,␣
 ↪validation_data=(X_test, Y_test), epochs=epochs)
```

```
Epoch 1/100
204/204 [==============================] - 2s 10ms/step - loss: 0.3618 -
accuracy: 0.8648 - val_loss: 0.3792 - val_accuracy: 0.8679
Epoch 2/100
204/204 [==============================] - 2s 8ms/step - loss: 0.2646 -
accuracy: 0.8921 - val_loss: 0.3223 - val_accuracy: 0.8859
```

```
Epoch 3/100
204/204 [==============================] - 2s 8ms/step - loss: 0.2446 -
accuracy: 0.9014 - val_loss: 0.3441 - val_accuracy: 0.8686
Epoch 4/100
204/204 [==============================] - 2s 8ms/step - loss: 0.2315 -
accuracy: 0.9093 - val_loss: 0.2821 - val_accuracy: 0.8917
Epoch 5/100
204/204 [==============================] - 2s 9ms/step - loss: 0.2249 -
accuracy: 0.9026 - val_loss: 0.2798 - val_accuracy: 0.8923
Epoch 6/100
204/204 [==============================] - 2s 8ms/step - loss: 0.2095 -
accuracy: 0.9112 - val_loss: 0.2947 - val_accuracy: 0.8917
Epoch 7/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1985 -
accuracy: 0.9120 - val_loss: 0.4637 - val_accuracy: 0.7250
Epoch 8/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1991 -
accuracy: 0.9147 - val_loss: 0.3748 - val_accuracy: 0.7705
Epoch 9/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1907 -
accuracy: 0.9112 - val_loss: 0.3474 - val_accuracy: 0.8827
Epoch 10/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1754 -
accuracy: 0.9203 - val_loss: 0.3375 - val_accuracy: 0.8763
Epoch 11/100
204/204 [==============================] - 2s 9ms/step - loss: 0.1956 -
accuracy: 0.9154 - val_loss: 0.3326 - val_accuracy: 0.8885
Epoch 12/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1635 -
accuracy: 0.9240 - val_loss: 0.3918 - val_accuracy: 0.8699
Epoch 13/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1947 -
accuracy: 0.9162 - val_loss: 0.3130 - val_accuracy: 0.9006
Epoch 14/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1580 -
accuracy: 0.9277 - val_loss: 0.3041 - val_accuracy: 0.8891
Epoch 15/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1851 -
accuracy: 0.9176 - val_loss: 0.3466 - val_accuracy: 0.8974
Epoch 16/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1597 -
accuracy: 0.9253 - val_loss: 0.3321 - val_accuracy: 0.8455
Epoch 17/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1570 -
accuracy: 0.9314 - val_loss: 0.3255 - val_accuracy: 0.8904
Epoch 18/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1482 -
accuracy: 0.9262 - val_loss: 0.3069 - val_accuracy: 0.8936
```

```
Epoch 19/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1467 -
accuracy: 0.9368 - val_loss: 0.3417 - val_accuracy: 0.8410
Epoch 20/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1355 -
accuracy: 0.9380 - val_loss: 0.4114 - val_accuracy: 0.8718
Epoch 21/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1309 -
accuracy: 0.9422 - val_loss: 0.3806 - val_accuracy: 0.8962
Epoch 22/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1345 -
accuracy: 0.9432 - val_loss: 0.3956 - val_accuracy: 0.8878
Epoch 23/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1203 -
accuracy: 0.9425 - val_loss: 0.4944 - val_accuracy: 0.8590
Epoch 24/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1209 -
accuracy: 0.9491 - val_loss: 0.4382 - val_accuracy: 0.8750
Epoch 25/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1171 -
accuracy: 0.9489 - val_loss: 0.5230 - val_accuracy: 0.8833
Epoch 26/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1035 -
accuracy: 0.9528 - val_loss: 0.4884 - val_accuracy: 0.8750
Epoch 27/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1168 -
accuracy: 0.9498 - val_loss: 0.3873 - val_accuracy: 0.8929
Epoch 28/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1127 -
accuracy: 0.9501 - val_loss: 0.4350 - val_accuracy: 0.9064
Epoch 29/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1018 -
accuracy: 0.9577 - val_loss: 0.4536 - val_accuracy: 0.9000
Epoch 30/100
204/204 [==============================] - 2s 8ms/step - loss: 0.1034 -
accuracy: 0.9577 - val_loss: 0.3705 - val_accuracy: 0.8885
Epoch 31/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0981 -
accuracy: 0.9572 - val_loss: 0.4651 - val_accuracy: 0.8808
Epoch 32/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0884 -
accuracy: 0.9631 - val_loss: 0.4097 - val_accuracy: 0.8962
Epoch 33/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0937 -
accuracy: 0.9612 - val_loss: 0.3760 - val_accuracy: 0.9038
Epoch 34/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0817 -
accuracy: 0.9641 - val_loss: 0.4294 - val_accuracy: 0.8917
```

```
Epoch 35/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0826 -
accuracy: 0.9666 - val_loss: 0.5202 - val_accuracy: 0.8872
Epoch 36/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0840 -
accuracy: 0.9619 - val_loss: 0.5858 - val_accuracy: 0.8679
Epoch 37/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0716 -
accuracy: 0.9702 - val_loss: 0.5628 - val_accuracy: 0.8788
Epoch 38/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0779 -
accuracy: 0.9666 - val_loss: 0.5540 - val_accuracy: 0.9032
Epoch 39/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0721 -
accuracy: 0.9683 - val_loss: 0.4252 - val_accuracy: 0.9167
Epoch 40/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0762 -
accuracy: 0.9683 - val_loss: 0.4747 - val_accuracy: 0.9013
Epoch 41/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0725 -
accuracy: 0.9712 - val_loss: 0.4844 - val_accuracy: 0.8974
Epoch 42/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0749 -
accuracy: 0.9698 - val_loss: 0.7187 - val_accuracy: 0.8718
Epoch 43/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0611 -
accuracy: 0.9739 - val_loss: 0.4535 - val_accuracy: 0.9147
Epoch 44/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0643 -
accuracy: 0.9717 - val_loss: 0.6184 - val_accuracy: 0.8987
Epoch 45/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0626 -
accuracy: 0.9742 - val_loss: 0.4271 - val_accuracy: 0.8737
Epoch 46/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0563 -
accuracy: 0.9779 - val_loss: 0.4767 - val_accuracy: 0.9058
Epoch 47/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0609 -
accuracy: 0.9781 - val_loss: 0.4441 - val_accuracy: 0.9038
Epoch 48/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0565 -
accuracy: 0.9771 - val_loss: 0.5007 - val_accuracy: 0.9064
Epoch 49/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0494 -
accuracy: 0.9803 - val_loss: 0.4703 - val_accuracy: 0.8859
Epoch 50/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0399 -
accuracy: 0.9857 - val_loss: 0.5312 - val_accuracy: 0.8949
```

```
Epoch 51/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0596 -
accuracy: 0.9764 - val_loss: 0.4503 - val_accuracy: 0.9192
Epoch 52/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0477 -
accuracy: 0.9803 - val_loss: 0.6705 - val_accuracy: 0.9032
Epoch 53/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0489 -
accuracy: 0.9823 - val_loss: 0.5595 - val_accuracy: 0.8897
Epoch 54/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0483 -
accuracy: 0.9818 - val_loss: 0.4750 - val_accuracy: 0.9000
Epoch 55/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0561 -
accuracy: 0.9793 - val_loss: 0.5473 - val_accuracy: 0.8917
Epoch 56/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0472 -
accuracy: 0.9828 - val_loss: 0.5218 - val_accuracy: 0.9096
Epoch 57/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0387 -
accuracy: 0.9848 - val_loss: 0.5581 - val_accuracy: 0.9122
Epoch 58/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0426 -
accuracy: 0.9843 - val_loss: 0.5903 - val_accuracy: 0.8872
Epoch 59/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0336 -
accuracy: 0.9889 - val_loss: 0.6742 - val_accuracy: 0.8833
Epoch 60/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0458 -
accuracy: 0.9833 - val_loss: 0.5150 - val_accuracy: 0.8955
Epoch 61/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0469 -
accuracy: 0.9843 - val_loss: 0.4853 - val_accuracy: 0.9038
Epoch 62/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0508 -
accuracy: 0.9835 - val_loss: 0.5122 - val_accuracy: 0.9090
Epoch 63/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0377 -
accuracy: 0.9875 - val_loss: 0.4954 - val_accuracy: 0.8859
Epoch 64/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0326 -
accuracy: 0.9872 - val_loss: 0.5599 - val_accuracy: 0.8962
Epoch 65/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0392 -
accuracy: 0.9862 - val_loss: 0.5793 - val_accuracy: 0.8968
Epoch 66/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0370 -
accuracy: 0.9852 - val_loss: 0.5392 - val_accuracy: 0.8942
```

```
Epoch 67/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0273 -
accuracy: 0.9899 - val_loss: 0.5820 - val_accuracy: 0.8904
Epoch 68/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0327 -
accuracy: 0.9880 - val_loss: 0.5235 - val_accuracy: 0.9013
Epoch 69/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0387 -
accuracy: 0.9880 - val_loss: 0.6118 - val_accuracy: 0.9064
Epoch 70/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0465 -
accuracy: 0.9857 - val_loss: 0.5888 - val_accuracy: 0.9045
Epoch 71/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0458 -
accuracy: 0.9838 - val_loss: 0.5338 - val_accuracy: 0.8923
Epoch 72/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0385 -
accuracy: 0.9860 - val_loss: 0.5074 - val_accuracy: 0.9218
Epoch 73/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0321 -
accuracy: 0.9897 - val_loss: 0.6245 - val_accuracy: 0.8936
Epoch 74/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0435 -
accuracy: 0.9855 - val_loss: 0.6730 - val_accuracy: 0.8942
Epoch 75/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0304 -
accuracy: 0.9911 - val_loss: 0.7452 - val_accuracy: 0.8679
Epoch 76/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0372 -
accuracy: 0.9867 - val_loss: 0.5693 - val_accuracy: 0.8833
Epoch 77/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0427 -
accuracy: 0.9857 - val_loss: 0.5557 - val_accuracy: 0.9096
Epoch 78/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0418 -
accuracy: 0.9880 - val_loss: 0.5122 - val_accuracy: 0.8968
Epoch 79/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0569 -
accuracy: 0.9779 - val_loss: 0.4199 - val_accuracy: 0.9071
Epoch 80/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0281 -
accuracy: 0.9904 - val_loss: 0.5035 - val_accuracy: 0.9013
Epoch 81/100
204/204 [==============================] - 2s 9ms/step - loss: 0.0231 -
accuracy: 0.9916 - val_loss: 0.4881 - val_accuracy: 0.9064
Epoch 82/100
204/204 [==============================] - 2s 9ms/step - loss: 0.0208 -
accuracy: 0.9934 - val_loss: 0.4814 - val_accuracy: 0.9147
```

```
Epoch 83/100
204/204 [==============================] - 2s 9ms/step - loss: 0.0360 -
accuracy: 0.9865 - val_loss: 0.4840 - val_accuracy: 0.9096
Epoch 84/100
204/204 [==============================] - 2s 9ms/step - loss: 0.0198 -
accuracy: 0.9934 - val_loss: 0.6188 - val_accuracy: 0.8872
Epoch 85/100
204/204 [==============================] - 2s 9ms/step - loss: 0.0306 -
accuracy: 0.9894 - val_loss: 0.5949 - val_accuracy: 0.8949
Epoch 86/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0424 -
accuracy: 0.9872 - val_loss: 0.6443 - val_accuracy: 0.8897
Epoch 87/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0428 -
accuracy: 0.9857 - val_loss: 0.6167 - val_accuracy: 0.8808
Epoch 88/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0268 -
accuracy: 0.9911 - val_loss: 0.4552 - val_accuracy: 0.9064
Epoch 89/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0179 -
accuracy: 0.9941 - val_loss: 0.5015 - val_accuracy: 0.8974
Epoch 90/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0161 -
accuracy: 0.9951 - val_loss: 0.5779 - val_accuracy: 0.9090
Epoch 91/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0566 -
accuracy: 0.9855 - val_loss: 0.4685 - val_accuracy: 0.8987
Epoch 92/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0276 -
accuracy: 0.9904 - val_loss: 0.6556 - val_accuracy: 0.8904
Epoch 93/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0340 -
accuracy: 0.9892 - val_loss: 0.6625 - val_accuracy: 0.8897
Epoch 94/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0305 -
accuracy: 0.9939 - val_loss: 0.4868 - val_accuracy: 0.9115
Epoch 95/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0335 -
accuracy: 0.9889 - val_loss: 0.6124 - val_accuracy: 0.8942
Epoch 96/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0204 -
accuracy: 0.9929 - val_loss: 0.6752 - val_accuracy: 0.8737
Epoch 97/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0199 -
accuracy: 0.9939 - val_loss: 0.6532 - val_accuracy: 0.8949
Epoch 98/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0210 -
accuracy: 0.9929 - val_loss: 0.6500 - val_accuracy: 0.8974
```

```
Epoch 99/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0133 -
accuracy: 0.9953 - val_loss: 0.6491 - val_accuracy: 0.9115
Epoch 100/100
204/204 [==============================] - 2s 8ms/step - loss: 0.0263 -
accuracy: 0.9909 - val_loss: 0.6171 - val_accuracy: 0.9103
```

[30]:
```
model_s.evaluate(X_test, Y_test)
```

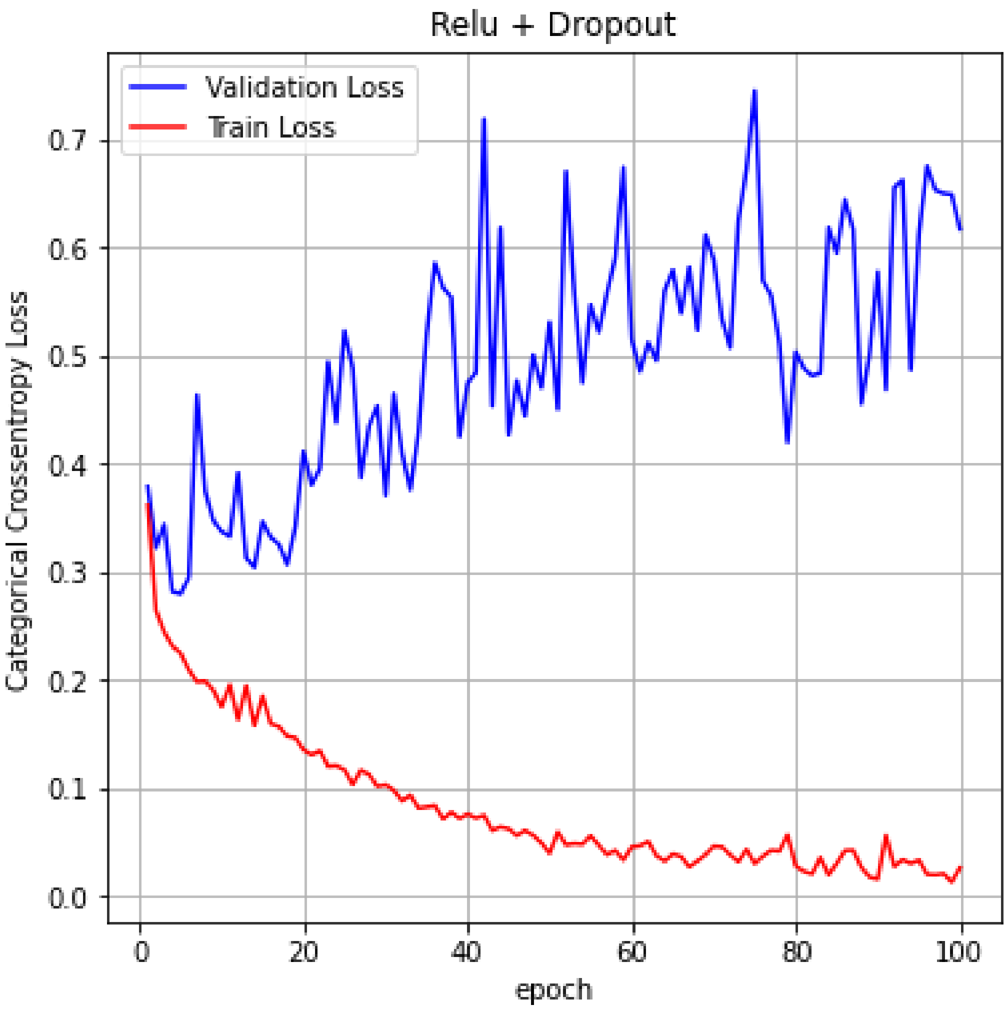
```
49/49 [==============================] - 0s 5ms/step - loss: 0.6140 - accuracy:
0.9103
```

[30]: [0.6139529347419739, 0.9102563858032227]

[0]:
```
def plt_dynamic(x, vy, ty, ax, color = 'b'):
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, ty, 'r', label = 'Train Loss')
    plt.grid()
    plt.legend()
    fig.canvas.draw()
```

[32]:
```
fig, ax = plt.subplots(1,1, figsize = (6, 6))
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')
plt.title('Relu + Dropout')

# list of epoch numbers: epoch = 100
x = list(range(1,epochs+1))
vy = history_s.history['val_loss']
ty = history_s.history['loss']
plt_dynamic(x, vy, ty, ax)
```

[32]:



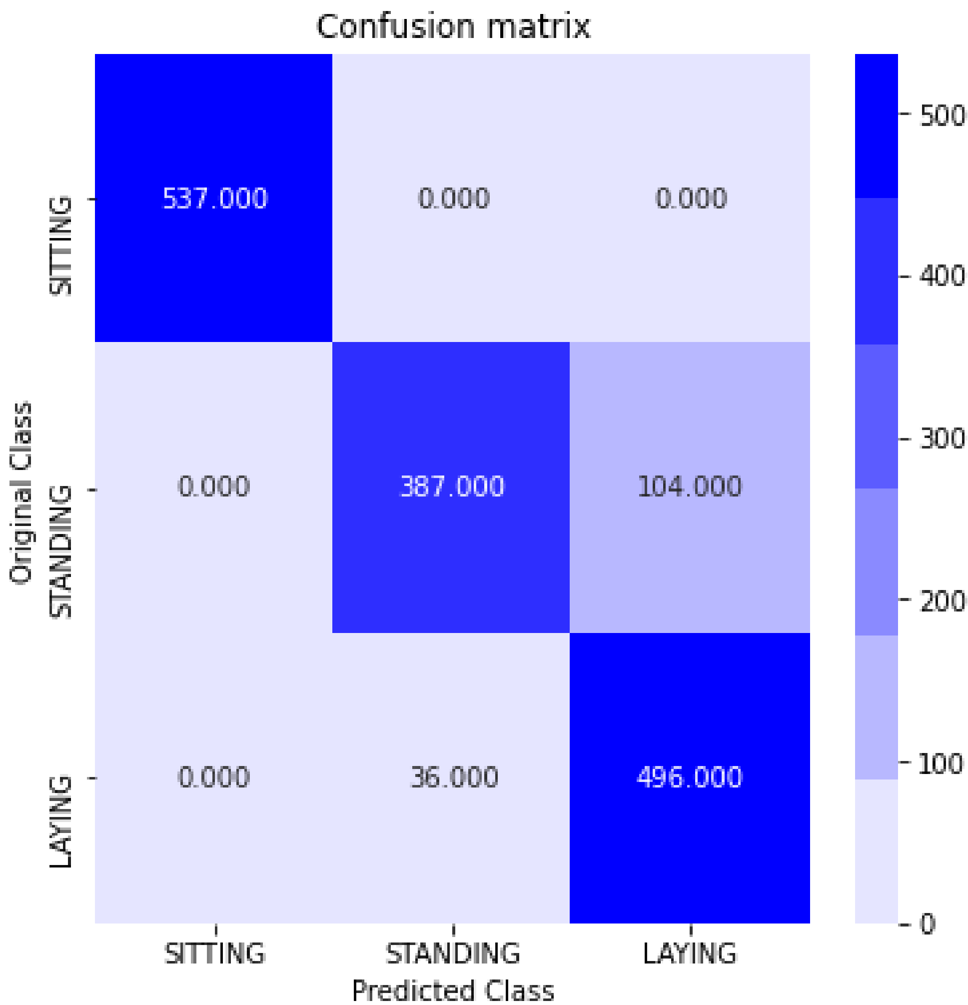
[0]:

```
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([label[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([label[y] for y in np.argmax(Y_pred, axis=1)])
    c_m = pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])

    plt.figure(figsize= (6, 6))
    c_m = sns.heatmap(c_m, annot=True, cmap= sns.light_palette("blue"), fmt=".
↪3f", xticklabels=label.values(), yticklabels= label.values())
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")


    return c_m
```

[34]:
```
confusion_matrix(Y_test, model_s.predict(X_test))
```

[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa250099c50>

[34]:



[35]:
```
import pickle
model_s.save('model_s')
```

INFO:tensorflow:Assets written to: model_s/assets

Dynamic Activities

[0]:
```
# Labelling the classes in 'y' after OHE

label = {0:'WALKING', 1:'WALKING_UPSTAIRS', 2:'WALKING_DOWNSTAIRS'}

def load_y_Dynamic(subset):
```

```
  filename = f'/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped␣
↪Files)/HAR/UCI_HAR_Dataset/{subset}/y_{subset}.txt'
  y = pd.read_csv(filename, delim_whitespace=True, header=None)[0]
  y_subset = y<=3
  y = y[y_subset]
  return pd.get_dummies(y).values,y_subset
```

[0]:
```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = data_load('train'), data_load('test')
    y_train, y_train1 = load_y_Dynamic('train')
    y_test, y_test1 =  load_y_Dynamic('test')

    X_train = X_train[y_train1]

    X_test = X_test[y_test1]

    return X_train, X_test, y_train, y_test
```

[38]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()

print('X_train shape is: ',X_train.shape)
print('Y_train shape is: ',Y_train.shape)
print('X_test shape is: ',X_test.shape)
print('Y_test shape is: ',Y_test.shape)
```

```
X_train shape is:  (3285, 128, 9)
Y_train shape is:  (3285, 3)
X_test shape is:  (1387, 128, 9)
Y_test shape is:  (1387, 3)
```

[39]:
```
input_dim = len(X_train[0][0])

print('Timesteps:', timesteps)
print('Input Dim:', input_dim)
print('No. of Train datapoints:', len(X_train))
```

```
Timesteps: 128
Input Dim: 9
No. of Train datapoints: 3285
```

[40]:
```
model_d= Sequential()
```

```
model_d.add(Conv1D(filters= 64, kernel_size= 5, activation= 'relu',␣
 ↪kernel_initializer= 'he_uniform',
                  input_shape=(timesteps, input_dim)))
model_d.add(Conv1D(filters= 64, kernel_size= 5, activation= 'relu',␣
 ↪kernel_initializer= 'he_uniform'))
model_d.add(MaxPooling1D(pool_size= 2, padding= 'same'))
model_d.add(Dropout(0.40))
model_d.add(Conv1D(filters= 32, kernel_size= 5, activation= 'relu',␣
 ↪kernel_initializer= 'he_uniform'))
model_d.add(Conv1D(filters= 32, kernel_size= 5, activation= 'relu',␣
 ↪kernel_initializer= 'he_uniform',))

# https://stackoverflow.com/a/49089027/10219869
# https://stackoverflow.com/a/58498450/10219869
model_d.add(MaxPooling1D(pool_size= 2, padding= 'same'))
model_d.add(BatchNormalization())
model_d.add(Dropout(0.40))
model_d.add(Flatten())
model_d.add(Dense(units= 100, activation= 'relu'))
model_d.add(BatchNormalization())
model_d.add(Dropout(0.40))
model_d.add(Dense(units= 3, activation= 'softmax'))
model_d.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_5 (Conv1D)            (None, 124, 64)           2944
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 120, 64)           20544
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 60, 64)            0
_________________________________________________________________
dropout_4 (Dropout)          (None, 60, 64)            0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 56, 32)            10272
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 52, 32)            5152
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 26, 32)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 26, 32)            128
_________________________________________________________________
dropout_5 (Dropout)          (None, 26, 32)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 832)               0
```

```
_________________________________________________________________
dense_3 (Dense)              (None, 100)               83300

_________________________________________________________________
batch_normalization_3 (Batch (None, 100)               400

_________________________________________________________________
dropout_6 (Dropout)          (None, 100)               0

_________________________________________________________________
dense_4 (Dense)              (None, 3)                 303
=================================================================
Total params: 123,043
Trainable params: 122,779
Non-trainable params: 264

_________________________________________________________________
```

[0]:

```
# Compiling the model
model_d.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

[42]:

```
# Initializing parameters
epochs =100
batch_size =20
# Training the model
history_d= model_d.fit(X_train, Y_train, batch_size=batch_size,␣
 ↪validation_data=(X_test, Y_test), epochs=epochs)
```

```
Epoch 1/100
165/165 [==============================] - 2s 10ms/step - loss: 1.1062 -
accuracy: 0.5723 - val_loss: 0.9853 - val_accuracy: 0.6222
Epoch 2/100
165/165 [==============================] - 1s 8ms/step - loss: 0.2503 -
accuracy: 0.9096 - val_loss: 0.7093 - val_accuracy: 0.7549
Epoch 3/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0826 -
accuracy: 0.9726 - val_loss: 0.3539 - val_accuracy: 0.8745
Epoch 4/100
165/165 [==============================] - 2s 9ms/step - loss: 0.0512 -
accuracy: 0.9857 - val_loss: 0.1500 - val_accuracy: 0.9416
Epoch 5/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0384 -
accuracy: 0.9930 - val_loss: 0.0685 - val_accuracy: 0.9748
Epoch 6/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0283 -
accuracy: 0.9896 - val_loss: 0.0892 - val_accuracy: 0.9690
Epoch 7/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0254 -
accuracy: 0.9912 - val_loss: 0.0634 - val_accuracy: 0.9733
Epoch 8/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0194 -
```

```
accuracy: 0.9930 - val_loss: 0.0823 - val_accuracy: 0.9712
Epoch 9/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0204 -
accuracy: 0.9948 - val_loss: 0.2586 - val_accuracy: 0.9164
Epoch 10/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0299 -
accuracy: 0.9915 - val_loss: 0.1549 - val_accuracy: 0.9466
Epoch 11/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0368 -
accuracy: 0.9875 - val_loss: 0.2452 - val_accuracy: 0.9373
Epoch 12/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0222 -
accuracy: 0.9939 - val_loss: 0.2536 - val_accuracy: 0.9178
Epoch 13/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0357 -
accuracy: 0.9881 - val_loss: 0.1340 - val_accuracy: 0.9503
Epoch 14/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0182 -
accuracy: 0.9939 - val_loss: 0.0814 - val_accuracy: 0.9769
Epoch 15/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0056 -
accuracy: 0.9991 - val_loss: 0.0944 - val_accuracy: 0.9668
Epoch 16/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0075 -
accuracy: 0.9976 - val_loss: 0.1121 - val_accuracy: 0.9632
Epoch 17/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0111 -
accuracy: 0.9957 - val_loss: 0.1214 - val_accuracy: 0.9690
Epoch 18/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0186 -
accuracy: 0.9954 - val_loss: 0.2404 - val_accuracy: 0.9279
Epoch 19/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0137 -
accuracy: 0.9951 - val_loss: 0.1193 - val_accuracy: 0.9546
Epoch 20/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0091 -
accuracy: 0.9967 - val_loss: 0.4119 - val_accuracy: 0.8882
Epoch 21/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0174 -
accuracy: 0.9945 - val_loss: 0.0876 - val_accuracy: 0.9784
Epoch 22/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0328 -
accuracy: 0.9948 - val_loss: 0.0680 - val_accuracy: 0.9841
Epoch 23/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0410 -
accuracy: 0.9860 - val_loss: 0.0802 - val_accuracy: 0.9683
Epoch 24/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0255 -
```

```
accuracy: 0.9927 - val_loss: 0.2028 - val_accuracy: 0.9445
Epoch 25/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0094 -
accuracy: 0.9960 - val_loss: 0.0608 - val_accuracy: 0.9820
Epoch 26/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0075 -
accuracy: 0.9982 - val_loss: 0.0867 - val_accuracy: 0.9712
Epoch 27/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0071 -
accuracy: 0.9982 - val_loss: 0.1124 - val_accuracy: 0.9676
Epoch 28/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0083 -
accuracy: 0.9979 - val_loss: 0.0686 - val_accuracy: 0.9719
Epoch 29/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0062 -
accuracy: 0.9982 - val_loss: 0.0368 - val_accuracy: 0.9841
Epoch 30/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0049 -
accuracy: 0.9991 - val_loss: 0.1020 - val_accuracy: 0.9640
Epoch 31/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0049 -
accuracy: 0.9997 - val_loss: 0.0396 - val_accuracy: 0.9877
Epoch 32/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0032 -
accuracy: 0.9991 - val_loss: 0.0475 - val_accuracy: 0.9849
Epoch 33/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0091 -
accuracy: 0.9979 - val_loss: 0.0780 - val_accuracy: 0.9697
Epoch 34/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0096 -
accuracy: 0.9976 - val_loss: 0.0739 - val_accuracy: 0.9740
Epoch 35/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0085 -
accuracy: 0.9973 - val_loss: 0.1377 - val_accuracy: 0.9776
Epoch 36/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0186 -
accuracy: 0.9939 - val_loss: 0.0633 - val_accuracy: 0.9813
Epoch 37/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0041 -
accuracy: 0.9991 - val_loss: 0.0589 - val_accuracy: 0.9755
Epoch 38/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0465 -
accuracy: 0.9970 - val_loss: 0.1433 - val_accuracy: 0.9791
Epoch 39/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0264 -
accuracy: 0.9927 - val_loss: 0.0367 - val_accuracy: 0.9877
Epoch 40/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0132 -
```

```
accuracy: 0.9963 - val_loss: 0.0346 - val_accuracy: 0.9863
Epoch 41/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0243 -
accuracy: 0.9921 - val_loss: 0.0561 - val_accuracy: 0.9762
Epoch 42/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0137 -
accuracy: 0.9945 - val_loss: 0.0519 - val_accuracy: 0.9791
Epoch 43/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0065 -
accuracy: 0.9973 - val_loss: 0.1025 - val_accuracy: 0.9668
Epoch 44/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0100 -
accuracy: 0.9979 - val_loss: 0.0578 - val_accuracy: 0.9755
Epoch 45/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0202 -
accuracy: 0.9979 - val_loss: 0.1024 - val_accuracy: 0.9712
Epoch 46/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0125 -
accuracy: 0.9970 - val_loss: 0.0673 - val_accuracy: 0.9748
Epoch 47/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0203 -
accuracy: 0.9954 - val_loss: 0.0266 - val_accuracy: 0.9899
Epoch 48/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0024 -
accuracy: 0.9994 - val_loss: 0.0284 - val_accuracy: 0.9928
Epoch 49/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0033 -
accuracy: 0.9994 - val_loss: 0.0220 - val_accuracy: 0.9928
Epoch 50/100
165/165 [==============================] - 2s 9ms/step - loss: 0.0049 -
accuracy: 0.9985 - val_loss: 0.0437 - val_accuracy: 0.9856
Epoch 51/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0102 -
accuracy: 0.9982 - val_loss: 0.0456 - val_accuracy: 0.9863
Epoch 52/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0093 -
accuracy: 0.9976 - val_loss: 0.0502 - val_accuracy: 0.9834
Epoch 53/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0096 -
accuracy: 0.9976 - val_loss: 0.1020 - val_accuracy: 0.9676
Epoch 54/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0098 -
accuracy: 0.9988 - val_loss: 0.0822 - val_accuracy: 0.9748
Epoch 55/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0106 -
accuracy: 0.9948 - val_loss: 0.0587 - val_accuracy: 0.9820
Epoch 56/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0035 -
```

```
accuracy: 0.9982 - val_loss: 0.1049 - val_accuracy: 0.9683
Epoch 57/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0028 -
accuracy: 0.9994 - val_loss: 0.0583 - val_accuracy: 0.9841
Epoch 58/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0014 -
accuracy: 0.9997 - val_loss: 0.0502 - val_accuracy: 0.9834
Epoch 59/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0071 -
accuracy: 0.9979 - val_loss: 0.0250 - val_accuracy: 0.9921
Epoch 60/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0014 -
accuracy: 0.9997 - val_loss: 0.0382 - val_accuracy: 0.9877
Epoch 61/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0071 -
accuracy: 0.9985 - val_loss: 0.0738 - val_accuracy: 0.9726
Epoch 62/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0041 -
accuracy: 0.9991 - val_loss: 0.1250 - val_accuracy: 0.9762
Epoch 63/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0066 -
accuracy: 0.9982 - val_loss: 0.0968 - val_accuracy: 0.9740
Epoch 64/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0138 -
accuracy: 0.9948 - val_loss: 0.2894 - val_accuracy: 0.9402
Epoch 65/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0095 -
accuracy: 0.9967 - val_loss: 0.0374 - val_accuracy: 0.9885
Epoch 66/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0056 -
accuracy: 0.9979 - val_loss: 0.0494 - val_accuracy: 0.9834
Epoch 67/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0047 -
accuracy: 0.9985 - val_loss: 0.0822 - val_accuracy: 0.9748
Epoch 68/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0014 -
accuracy: 0.9997 - val_loss: 0.0916 - val_accuracy: 0.9784
Epoch 69/100
165/165 [==============================] - 1s 8ms/step - loss: 5.0931e-04 -
accuracy: 1.0000 - val_loss: 0.0699 - val_accuracy: 0.9784
Epoch 70/100
165/165 [==============================] - 1s 8ms/step - loss: 7.8539e-04 -
accuracy: 0.9997 - val_loss: 0.0510 - val_accuracy: 0.9856
Epoch 71/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0015 -
accuracy: 0.9997 - val_loss: 0.0999 - val_accuracy: 0.9805
Epoch 72/100
165/165 [==============================] - 1s 8ms/step - loss: 5.6427e-04 -
```

```
accuracy: 1.0000 - val_loss: 0.0671 - val_accuracy: 0.9849
Epoch 73/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0033 -
accuracy: 0.9988 - val_loss: 0.0870 - val_accuracy: 0.9805
Epoch 74/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0152 -
accuracy: 0.9960 - val_loss: 0.0986 - val_accuracy: 0.9798
Epoch 75/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0072 -
accuracy: 0.9982 - val_loss: 0.0855 - val_accuracy: 0.9769
Epoch 76/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0049 -
accuracy: 0.9988 - val_loss: 0.0424 - val_accuracy: 0.9827
Epoch 77/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0143 -
accuracy: 0.9991 - val_loss: 0.0389 - val_accuracy: 0.9870
Epoch 78/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0134 -
accuracy: 0.9970 - val_loss: 0.2074 - val_accuracy: 0.9430
Epoch 79/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0227 -
accuracy: 0.9979 - val_loss: 0.0488 - val_accuracy: 0.9841
Epoch 80/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0151 -
accuracy: 0.9948 - val_loss: 0.0366 - val_accuracy: 0.9913
Epoch 81/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0021 -
accuracy: 0.9991 - val_loss: 0.0248 - val_accuracy: 0.9928
Epoch 82/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0024 -
accuracy: 0.9994 - val_loss: 0.0299 - val_accuracy: 0.9899
Epoch 83/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0080 -
accuracy: 0.9991 - val_loss: 0.0607 - val_accuracy: 0.9827
Epoch 84/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0166 -
accuracy: 0.9948 - val_loss: 0.1378 - val_accuracy: 0.9596
Epoch 85/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0051 -
accuracy: 0.9985 - val_loss: 0.1307 - val_accuracy: 0.9625
Epoch 86/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0043 -
accuracy: 0.9994 - val_loss: 0.0929 - val_accuracy: 0.9697
Epoch 87/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0021 -
accuracy: 0.9994 - val_loss: 0.1296 - val_accuracy: 0.9676
Epoch 88/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0065 -
```

```
accuracy: 0.9985 - val_loss: 0.0858 - val_accuracy: 0.9805
Epoch 89/100
165/165 [==============================] - 1s 8ms/step - loss: 9.1152e-04 -
accuracy: 0.9997 - val_loss: 0.0822 - val_accuracy: 0.9762
Epoch 90/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0014 -
accuracy: 0.9994 - val_loss: 0.0801 - val_accuracy: 0.9755
Epoch 91/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0013 -
accuracy: 0.9994 - val_loss: 0.1277 - val_accuracy: 0.9690
Epoch 92/100
165/165 [==============================] - 1s 9ms/step - loss: 0.0018 -
accuracy: 0.9994 - val_loss: 0.1663 - val_accuracy: 0.9640
Epoch 93/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0010 -
accuracy: 0.9997 - val_loss: 0.0354 - val_accuracy: 0.9856
Epoch 94/100
165/165 [==============================] - 1s 9ms/step - loss: 7.4337e-04 -
accuracy: 0.9997 - val_loss: 0.0607 - val_accuracy: 0.9849
Epoch 95/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0016 -
accuracy: 0.9994 - val_loss: 0.0337 - val_accuracy: 0.9841
Epoch 96/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0012 -
accuracy: 0.9994 - val_loss: 0.0436 - val_accuracy: 0.9885
Epoch 97/100
165/165 [==============================] - 1s 9ms/step - loss: 5.3329e-04 -
accuracy: 0.9997 - val_loss: 0.0431 - val_accuracy: 0.9870
Epoch 98/100
165/165 [==============================] - 1s 8ms/step - loss: 5.8772e-04 -
accuracy: 1.0000 - val_loss: 0.0499 - val_accuracy: 0.9856
Epoch 99/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0063 -
accuracy: 0.9988 - val_loss: 0.1438 - val_accuracy: 0.9712
Epoch 100/100
165/165 [==============================] - 1s 8ms/step - loss: 0.0046 -
accuracy: 0.9988 - val_loss: 0.0717 - val_accuracy: 0.9740
```

[43]:
```python
model_d.evaluate(X_test, Y_test)
```

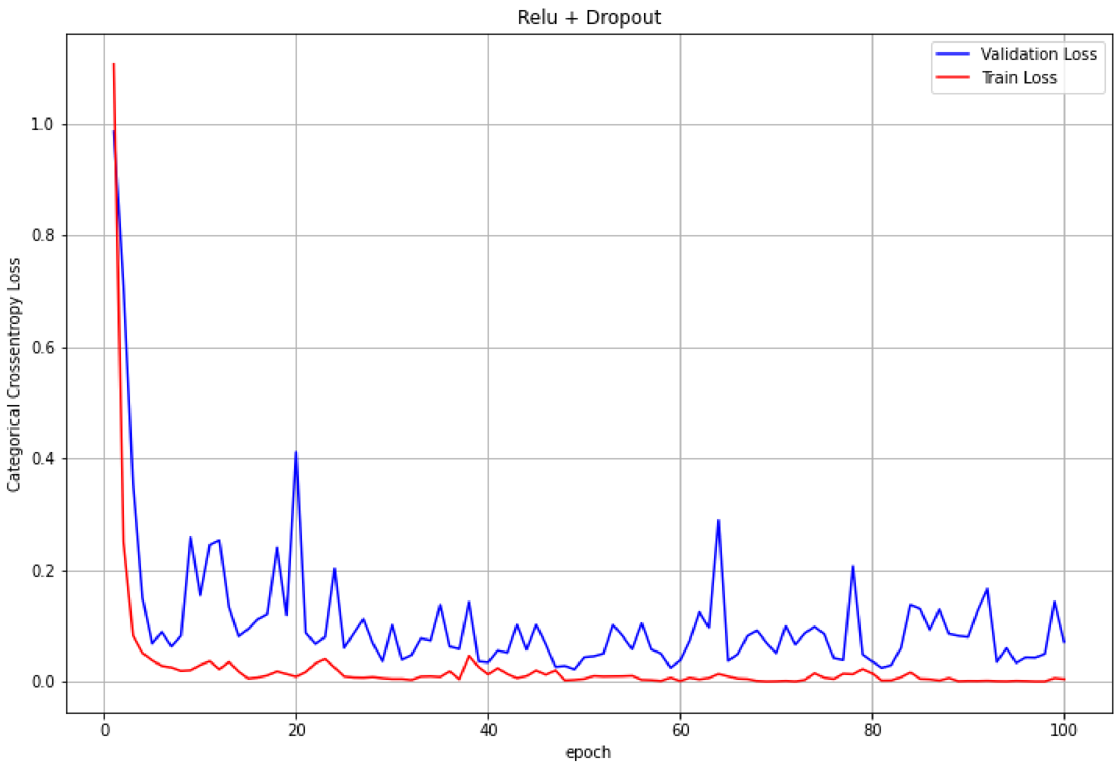
```
44/44 [==============================] - 0s 5ms/step - loss: 0.0713 - accuracy:
0.9740
```

[43]:
```
[0.07126739621162415, 0.974044680595398]
```

[44]:
```python
fig, ax = plt.subplots(1,1, figsize = (12, 8))
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')
```

```python
plt.title('Relu + Dropout')

# list of epoch numbers: epoch = 100
x = list(range(1,100+1))
vy = history_d.history['val_loss']
ty = history_d.history['loss']
plt_dynamic(x, vy, ty, ax)
```

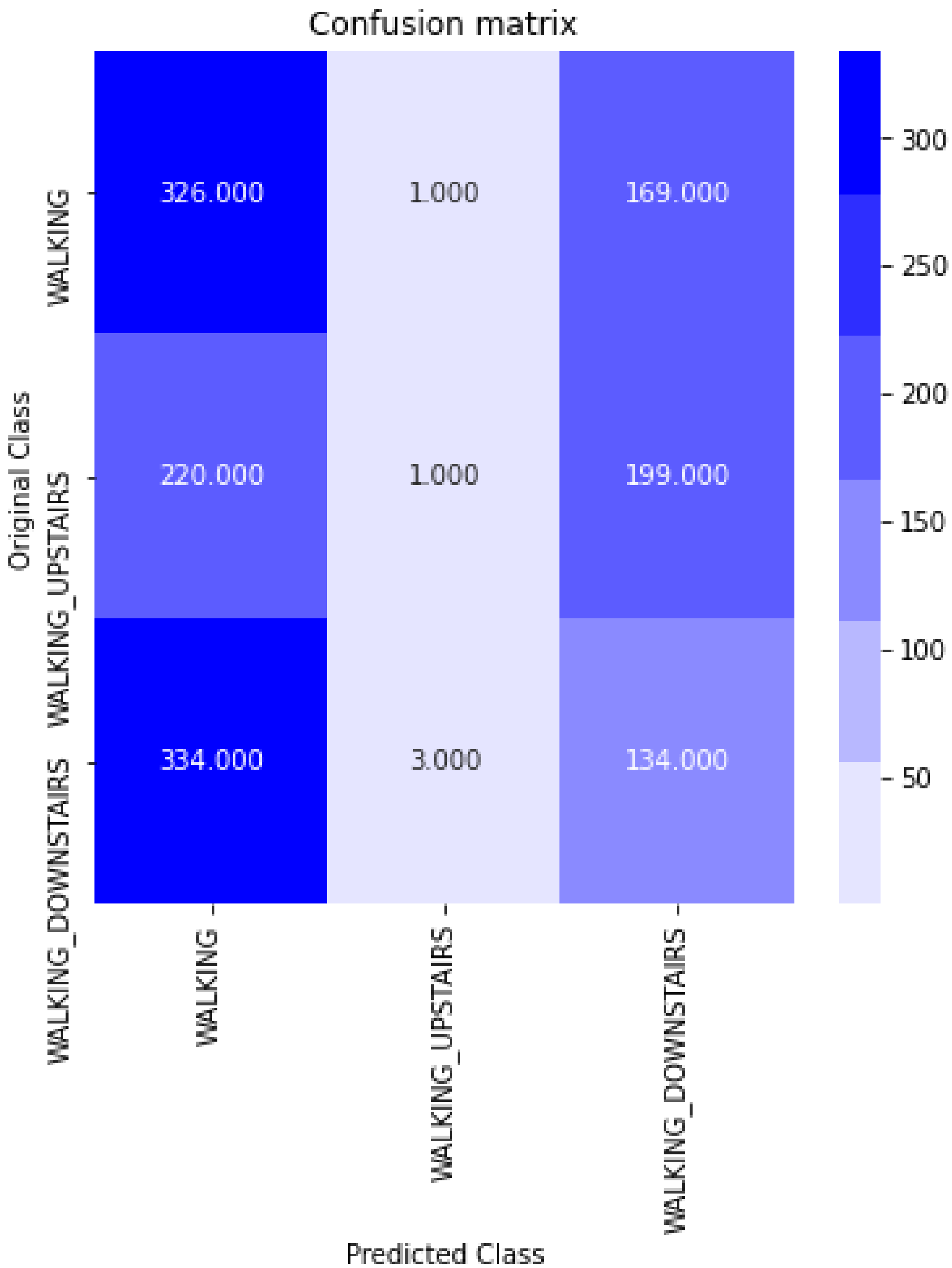
[44]:



[45]:
```python
confusion_matrix(Y_test, model_s.predict(X_test))
```

[45]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa1ea7d8cc0>

[45]:



[46]:
```
model_d.save('model_d')
```

```
INFO:tensorflow:Assets written to: model_d/assets
```

[47]:
```
print('X_train shape is: ',X_train_full.shape)
print('Y_train shape is: ',Y_train_full.shape)
print('X_test shape is: ',X_test_full.shape)
```

```
print('Y_test shape is: ',Y_test_full.shape)
```

```
X_train shape is:  (7352, 128, 9)
Y_train shape is:  (7352, 6)
X_test shape is:  (2947, 128, 9)
Y_test shape is:  (2947, 6)
```

[48]:
```
timesteps = len(X_train_full[0])
input_dim = len(X_train_full[0][0])
print(timesteps)
print(input_dim)
print(len(X_train_full))
```

```
128
9
7352
```

[0]:
```
predict_binary = model1.predict(X_test_full)
f_predict_binary = np.argmax(predict_binary, axis=1)
```

[0]:
```
X_dynamic= X_test_full[f_predict_binary==0]
```

[74]:
```
X_dynamic[0][0]
```

[74]:
```
array([-0.03167277, -0.08279653, -0.06853677, -0.6562038 ,  0.5171189 ,
       -0.2085947 ,  0.9455455 , -0.4315797 , -0.05446144])
```

[0]:
```
predict_dynamic = model_d.predict(X_dynamic)
f_predict_dynamic = np.argmax(predict_dynamic,axis=1)
```

[81]:
```
f_predict_dynamic+1
```

[81]:
```
array([1, 1, 1, …, 2, 2, 2])
```

[0]:
```

```

[0]:
```
def predict(X):
    ##predicting whether dynamic or static
    predict_binary = model1.predict(X)
    f_predict_binary = np.argmax(predict_binary, axis=1)

    #static data filter
    X_static = X[f_predict_binary==1]

    #dynamic data filter
    X_dynamic = X[f_predict_binary==0]
```

```
    #predicting static activities
    predict_static = model_s.predict(X_static)
    f_predict_static = np.argmax(predict_static,axis=1)

    #adding 3 because need to get inital prediction lable as output
    f_predict_static = f_predict_static + 3

    #predicting dynamic activites
    predict_dynamic = model_d.predict(X_dynamic)
    f_predict_dynamic = np.argmax(predict_dynamic,axis=1)

    # lable of dynamic activites is given as fellowes {0: 'WALKING',1:␣
 ↪'WALKING_UPSTAIRS',2: 'WALKING_DOWNSTAIRS',3: 'SITTING',4: 'STANDING',5:␣
 ↪'LAYING'},so No need add any prediction lable as output.
    f_predict_dynamic = f_predict_dynamic

    ##appending final output to one list in the same sequence of input data
    i,j = 0,0
    final_predict = []

    for q_p in f_predict_binary:
        if q_p == 1:
            final_predict.append(f_predict_static[i])
            i = i + 1
        else:
            final_predict.append(f_predict_dynamic[j])
            j = j + 1

    return final_predict
```

[93]:
```
from sklearn.metrics import accuracy_score

train_pred = predict(X_train_full)
test_pred = predict(X_test_full)

print('Accuracy of train data',accuracy_score(np.
 ↪argmax(Y_train_full,axis=1),train_pred))
print('Accuracy of validation data',accuracy_score(np.
 ↪argmax(Y_test_full,axis=1),test_pred))
```

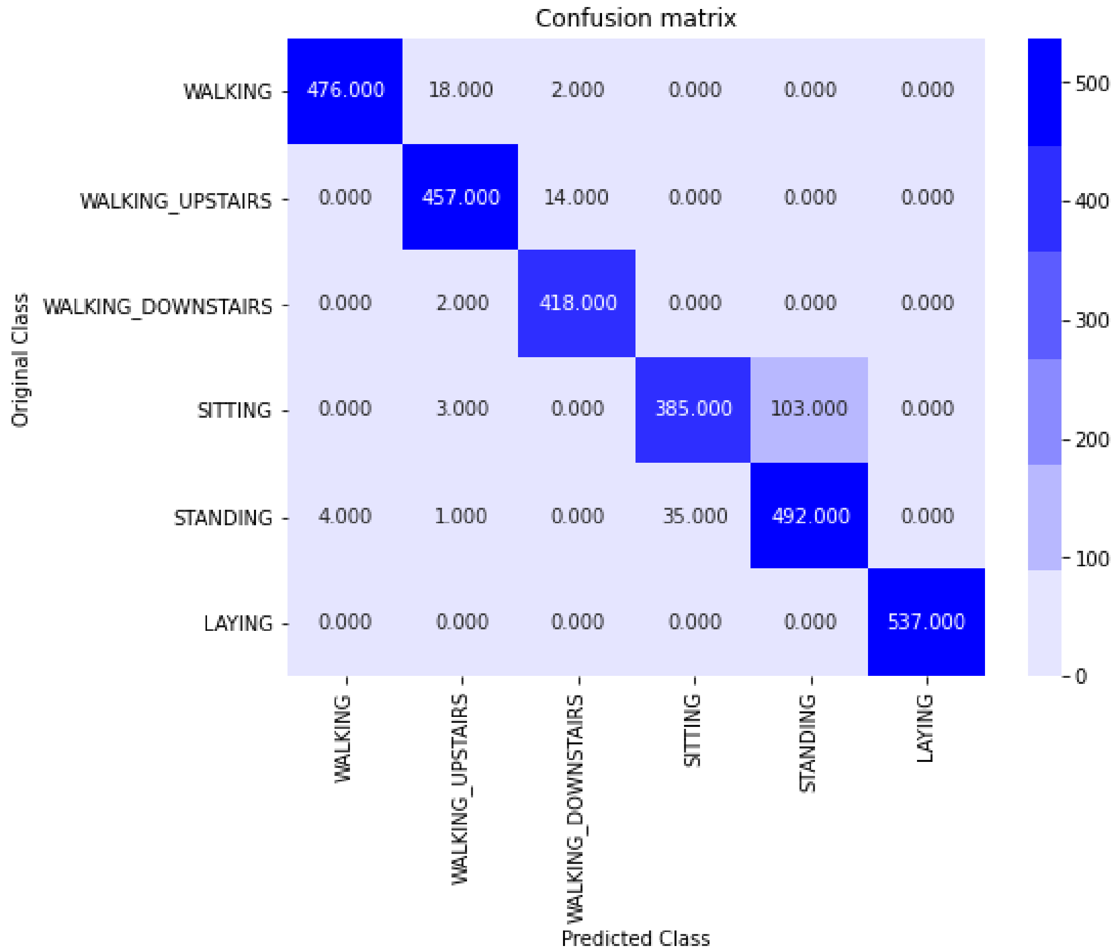
```
Accuracy of train data 0.9949673558215452
Accuracy of validation data 0.9382422802850356
```

[95]:
```
ACTIVITIES = {0: 'WALKING',1: 'WALKING_UPSTAIRS',2: 'WALKING_DOWNSTAIRS',3:␣
 ↪'SITTING',4: 'STANDING',5: 'LAYING'}
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(np.argmax(Y_test_full,axis=1), test_pred)
```

```
plt.figure(figsize= (8, 6))
sns.heatmap(cm, annot=True, cmap= sns.light_palette("blue"), fmt=".3f",␣
 ↪xticklabels=ACTIVITIES.values(),
                      yticklabels= ACTIVITIES.values())
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.title("Confusion matrix")
```

[95]: Text(0.5, 1.0, 'Confusion matrix')

[95]:



[96]:
```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ['Rank', 'Model', "Test Accuracy"]

x.add_row([1, "Divide + Conquer Model", "0.9949673558215452"])
x.add_row([2, "32 LSTM Base Model", "0.9382422802850356"])
```

```python
print(x)
```

```
+------+------------------------+--------------------+
| Rank |         Model          |   Test Accuracy    |
+------+------------------------+--------------------+
|  1   | Divide + Conquer Model | 0.9949673558215452 |
|  2   |   32 LSTM Base Model   | 0.9382422802850356 |
+------+------------------------+--------------------+
```