*#-\*- coding:utf-8 -\*-***import** pandas **as** pd  
**import** numpy **as** np  
**import** tensorflow **as** tf  
**import** matplotlib.pyplot **as** plt  
**from** tensorflow.keras.layers **import** Dense, Conv2D, MaxPooling2D, Dropout, LSTMCell, Softmax, ReLU, Embedding, SimpleRNNCell, SimpleRNN  
*#import tensorflow.compat.v1 as tf  
#tf.disable\_v2\_behavior()***from** tensorflow **import** keras  
**from** sklearn **import** model\_selection  
**from** tensorflow.keras **import** layers  
**import** os  
**import** random  
os.environ[**'TF\_CPP\_MIN\_LOG\_LEVEL'**] = **'2'  
assert** tf.\_\_version\_\_.startswith(**'2.'**)  
tf.random.set\_seed(22)  
np.random.seed(22)  
  
**def** read\_data(file\_path):  
 column\_names = [**'user-id'**, **'activity'**, **'timestamp'**, **'x-axis'**, **'y-axis'**, **'z-axis'**]  
 data = pd.read\_csv(file\_path, header=**None**, names=column\_names)  
 **return** data  
  
**def** feature\_normalize(dataset):  
 mu = np.mean(dataset, axis=0)  
 sigma = np.std(dataset, axis=0)  
 **return** (dataset - mu) / sigma  
  
  
data = read\_data(**"C:/Users/xx412/PycharmProjects/pythonProject/WISDM.txt"**)  
data = data.dropna(axis=0, how=**'any'**)  
  
i = 0  
**for** x **in** data[**'x-axis'**]:  
 **if** x == 0:  
 data = data.drop(labels=i, axis=0)  
 i = i + 1  
data.reset\_index(drop=**True**, inplace=**True**)  
  
data[**'activity'**] = data[**'activity'**].replace(**'Walking'**, **'0'**)  
data[**'activity'**] = data[**'activity'**].replace(**'Jogging'**, **'1'**)  
data[**'activity'**] = data[**'activity'**].replace(**'Upstairs'**, **'2'**)  
data[**'activity'**] = data[**'activity'**].replace(**'Downstairs'**, **'3'**)  
data[**'activity'**] = data[**'activity'**].replace(**'Sitting'**, **'4'**)  
data[**'activity'**] = data[**'activity'**].replace(**'Standing'**, **'5'**)  
  
data[**'activity'**] = data[**'activity'**].astype(np.float)  
  
  
data[**'x-axis'**] = feature\_normalize(data[**'x-axis'**])  
data[**'y-axis'**] = feature\_normalize(data[**'y-axis'**])  
data[**'z-axis'**] = feature\_normalize(data[**'z-axis'**])  
print(data.shape)  
data.to\_csv(**'data.txt'**)  
  
data0 = pd.DataFrame(columns=(**'user-id'**, **'activity'**, **'timestamp'**, **'x-axis'**, **'y-axis'**, **'z-axis'**))  
data1 = pd.DataFrame(columns=(**'user-id'**, **'activity'**, **'timestamp'**, **'x-axis'**, **'y-axis'**, **'z-axis'**))  
data2 = pd.DataFrame(columns=(**'user-id'**, **'activity'**, **'timestamp'**, **'x-axis'**, **'y-axis'**, **'z-axis'**))  
data3 = pd.DataFrame(columns=(**'user-id'**, **'activity'**, **'timestamp'**, **'x-axis'**, **'y-axis'**, **'z-axis'**))  
data4 = pd.DataFrame(columns=(**'user-id'**, **'activity'**, **'timestamp'**, **'x-axis'**, **'y-axis'**, **'z-axis'**))  
data5 = pd.DataFrame(columns=(**'user-id'**, **'activity'**, **'timestamp'**, **'x-axis'**, **'y-axis'**, **'z-axis'**))  
**def** data\_classify(new\_data, act):  
 n = 0  
 m = 0  
 **for** x **in** data[**'activity'**]:  
 **if** x == act:  
 new\_data.loc[n] = data.loc[m]  
 n = n+1  
 m += 1  
data\_classify(data0, 0)  
data\_classify(data1, 1)  
data\_classify(data2, 2)  
data\_classify(data3, 3)  
data\_classify(data4, 4)  
data\_classify(data5, 5)  
  
  
**def** to\_90(data):  
 segments = np.zeros((1, 3))  
 labels = np.zeros((1))  
 c = 0  
 **for** i **in** data.index:  
 segments = np.vstack([segments, np.array(  
 [data.loc[c, **'x-axis'**], data.loc[c, **'y-axis'**],  
 data.loc[c, **'z-axis'**]])])  
 labels = np.vstack([labels, np.array([data[**"activity"**].loc[c]])])  
 c += 1  
 labels = np.delete(labels, 0, 0).astype(int)  
 segments = np.delete(segments, 0, 0)  
 **return** segments, labels  
  
segments0, labels0 = to\_90(data0)  
segments1, labels1 = to\_90(data1)  
segments2, labels2 = to\_90(data2)  
segments3, labels3 = to\_90(data3)  
segments4, labels4 = to\_90(data4)  
segments5, labels5 = to\_90(data5)  
  
segments = np.vstack([segments0, segments1, segments2,  
 segments3, segments4, segments5])  
labels = np.vstack([labels0, labels1, labels2,  
 labels3, labels4, labels5])  
x\_train, x\_test, y\_train, y\_test = model\_selection.train\_test\_split(segments, labels, test\_size=0.33, random\_state=0)  
  
print(x\_train)  
print(x\_test)  
print(y\_train)  
print(y\_test)  
**def** tongji(daten):  
 mask = np.unique(daten)  
 tmp = {}  
 **for** v **in** mask:  
 tmp[v] = np.sum(daten == v)  
 print(**"mask值为："**)  
 print(mask)  
 print(**"Statistische Ergebnisse："**)  
 print(tmp)  
  
tongji(y\_train)  
tongji(y\_test)  
print(x\_train.shape[1])  
print(y\_train.shape)  
  
  
  
  
  
model = tf.keras.Sequential()  
  
model.add(layers.Dense(3, activation=**'relu'**, input\_shape=(x\_train.shape[1],)))  
model.add(layers.Dense(50, activation=**'relu'**))  
model.add(layers.Dense(np.unique(y\_train).size \* 4, activation=**'relu'**))  
model.add(layers.Dense(np.unique(y\_train).size, activation=**'softmax'**))  
  
y\_train = tf.keras.utils.to\_categorical(y\_train, 6)  
y\_test = tf.keras.utils.to\_categorical(y\_test, 6)  
  
  
model.compile(optimizer=**'adam'**,  
 loss=**'categorical\_crossentropy'**,  
 metrics=[**'accuracy'**])  
  
  
history = model.fit(x\_train,  
 y\_train,  
 batch\_size=16,  
 epochs=1500,  
 validation\_data=(x\_test, y\_test),  
 verbose=1)  
eval\_result = model.evaluate(x\_test, y\_test)  
predictions = model.predict(x\_test)  
test\_score = model.evaluate(x\_test, y\_test)  
  
  
accuracy = history.history[**'accuracy'**]  
loss = history.history[**'loss'**]  
val\_loss = history.history[**'val\_loss'**]  
val\_accuracy = history.history[**'val\_accuracy'**]  
epochs = np.arange(len(accuracy)) + 1  
  
plt.subplot(211)  
plt.title(**f'Test accuracy: {**round(test\_score[1], 3)**}'**)  
plt.plot(epochs, accuracy, label=**'Accuracy'**)  
plt.plot(epochs, val\_accuracy, label=**'Validate accuracy'**)  
plt.grid(**True**)  
plt.legend()  
  
plt.subplot(212)  
plt.title(**f'Test loss: {**round(test\_score[0], 3)**}'**)  
plt.grid(**True**)  
  
print(**'Loss:'**, eval\_result[0])  
print(**'Accuracy:'**, eval\_result[1]\*100)  
  
model.save(**'./model.h5'**)  
  
  
loss = history.history[**'loss'**]  
val\_loss = history.history[**'val\_loss'**]  
epochs = range(1, len(loss) + 1)  
plt.plot(epochs, loss, **'g.'**, label=**'Training loss'**)  
plt.plot(epochs, val\_loss, **'b'**, label=**'Validation loss'**)  
plt.title(**'Training and validation loss'**)  
plt.xlabel(**'Epochs'**)  
plt.ylabel(**'Loss'**)  
plt.legend()  
plt.show()  
  
  
converter = tf.lite.TFLiteConverter.from\_keras\_model(model)  
converter.optimizations = [tf.lite.Optimize.OPTIMIZE\_FOR\_SIZE]  
tflite\_model = converter.convert()  
print(tflite\_model)  
open(**'model.tflite'**, **'wb'**).write(tflite\_model)  
  
**def** hex\_to\_c\_arrary(hex\_data, var\_name):  
 c\_str = **''** c\_str += **'#ifndef '** + var\_name.upper() + **'\_H\n'** c\_str += **'#define '** + var\_name.upper() + **'\_H\n\n'** c\_str += **'\nunsigned int'** + var\_name + **'\_len='** +str(len(hex\_data)) + **';\n'** c\_str += **'const unsigned char '** + var\_name + **'[] = {'** hex\_array = []  
 **for** i, val **in** enumerate(hex\_data):  
 hex\_str = format(val, **'#04x'**)  
  
 **if**(i+1) < len(hex\_data):  
 hex\_str += **','  
 if**(i+1) % 12 == 0:  
 hex\_str += **'\n'** hex\_array.append(hex\_str)  
  
 c\_str += **'\n'** + format(**' '**.join(hex\_array)) + **'\n};\n\n'** c\_str += **'#endif //'** + var\_name.upper() + **'\_H'  
  
 return** c\_str  
c\_modek\_name = **'model'  
with** open(c\_modek\_name + **'.h'**, **'w'**) **as** file:  
 file.write(hex\_to\_c\_arrary(tflite\_model, c\_modek\_name))