1D-CNN-1sec

December 2, 2021

1 Libraries

Import your libraries https://towardsdatascience.com/how-to-use-convolutional-neural-networks-for-time-series-classification-56b1b0a07a57

[2]:
```
import torch
import os
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
from pylab import rcParams
import math
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,␣
↪precision_score, recall_score
from torch import nn, optim
import torch.nn.functional as F
```

```
using gpu 0
```

2 NN = torch.load(MODEL_PATH)

3 Importing Data

Import the CSV file with Actions, Sum and Div as a Dataframe called df. Fill the empty values of Action with 0. Replace NaN values with 0. Delete first 100 rows.

[3]:
```
#load in df
Player = 15
Game = 2
```

```
df = pd.read_csv('matrix_Player_' + str(Player) + '_game_' + str(Game) +␣
↪'_QuarterSplit.csv')

#Delete first 99 rows, so df starts at timeLine == 1.00
df = df.drop(columns=['Unnamed: 0'])
df = df.iloc[99: , :]

df.head()
```

[3]:

	frAcc	frRoAcc	frDispl	frRoAng	frSpeed	timeLine	\
99	-6.4188	2828.5	3314.5	-87.889	-0.72304	3582.0	
100	-3.9796	2828.5	3313.6	-88.288	-0.78723	3582.0	
101	-1.2585	2828.5	3312.7	-88.327	-0.82702	3582.0	
102	-9.4697	2828.4	3311.8	-89.588	-0.83961	3582.0	
103	-11.3390	2828.4	3310.9	-91.083	-0.93430	3582.0	

	frameRotationalSpeedX	frameRotationalSpeedY	frameRotationalSpeedZ	\
99	-0.17111	8.5322	-88.441	
100	1.04300	7.6440	-88.914	
101	0.49000	7.4200	-88.970	
102	0.35000	7.4760	-90.230	
103	0.68600	7.2940	-91.742	

	wheelRotationalSpeedX	wheelRotationalSpeedY	wheelRotationalSpeedZ	\
99	-45.080	98.752	-12.127	
100	-56.028	92.540	-12.838	
101	-63.008	86.987	-13.331	
102	-64.064	81.032	-14.630	
103	-79.380	86.940	-20.370	

	frRoSpeed	Filt_WheelX	Filt_FrameZ	Action	Quarter
99	-39.9170	41.224349	-204.464236	0.0	1
100	-3.9105	42.053609	-204.540434	0.0	1
101	-126.0700	42.833437	-204.182064	0.0	1
102	-149.5100	43.544112	-203.399449	0.0	1
103	-239.6600	44.165948	-202.205326	0.0	1

4 Pre Preprocessing

[4]:
```
# Fill NaN with 0
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df = df.fillna(0)


#Convert Sprinting to 1
```

```
df.Action.replace({'Sprinting': 1},inplace=True)
```

5 Data Preparating

Split data into a train and test set

[5]:
```
train = df[df.Quarter != 4]
test = df[df.Quarter == 4]

def rounddown(x):
    return (int(math.ceil(x / 100.0)) * 100) - 100

X_train = train[['wheelRotationalSpeedX','frameRotationalSpeedZ','frAcc']]
X_train = X_train.iloc[0:rounddown(len(X_train))]
y_train = train[['Action']]
y_train = y_train.iloc[0:rounddown(len(y_train))]

y_train = np.array_split(y_train, int(len(y_train)/100))
y_train = np.asarray(y_train)

y = []
for i in range (0,len(y_train)):
    y.append(y_train[i].max())
y_train = np.asarray(y)


X_test = test[['wheelRotationalSpeedX','frameRotationalSpeedZ','frAcc']]
X_test = X_test.iloc[0:rounddown(len(X_test))]

y_test = test[['Action']]
y_test = y_test.iloc[0:rounddown(len(y_test))]
y_test = np.array_split(y_test, int(len(y_test)/100))
y_test = np.asarray(y_test)

y = []
for i in range (0,len(y_test)):
    y.append(y_test[i].max())
y_test = np.asarray(y)
```

5.1 Slicing the train and test sets into windows of 1 sec (100 samples per window)

[6]:
```
size_batch = 100
channels = 3
```

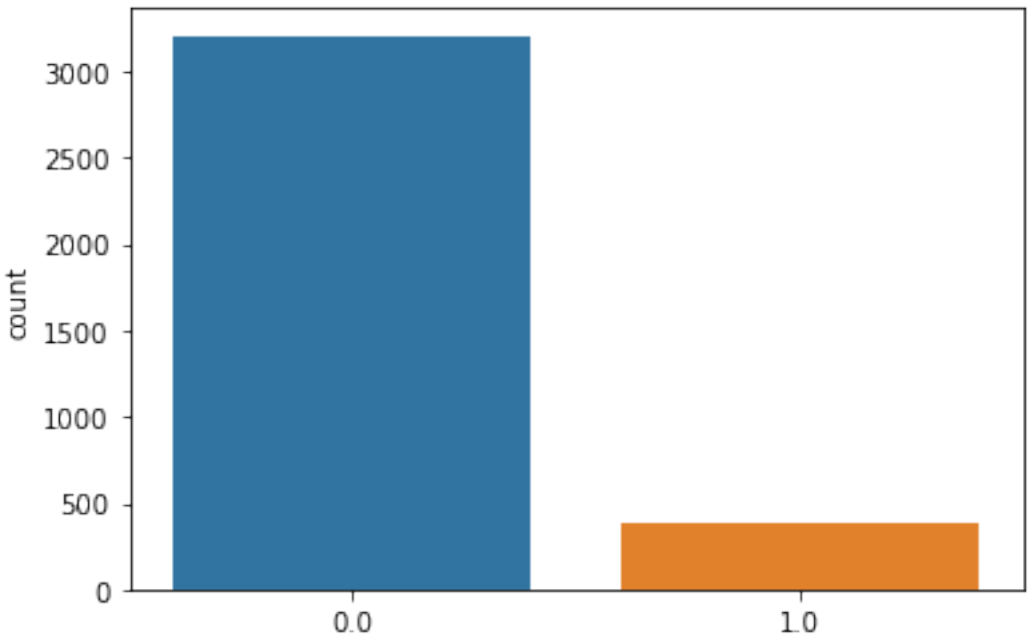
[7]:
```
A1 = X_train['wheelRotationalSpeedX'].to_numpy().reshape(int(len(X_train)/
 ↪size_batch),size_batch)
A2 = X_train['frameRotationalSpeedZ'].to_numpy().reshape(int(len(X_train)/
 ↪size_batch),size_batch)
A3 = X_train['frAcc'].to_numpy().reshape(int(len(X_train)/
 ↪size_batch),size_batch)

X_train = np.stack((A1,A2,A3),axis=1)
X_train = X_train.reshape([int(len(X_train)),channels,size_batch])
print(X_train.shape)
```

```
(3594, 3, 100)
```

[8]:
```
A1 = X_test['wheelRotationalSpeedX'].to_numpy().reshape(int(len(X_test)/
 ↪size_batch),size_batch)
A2 = X_test['frameRotationalSpeedZ'].to_numpy().reshape(int(len(X_test)/
 ↪size_batch),size_batch)
A3 = X_test['frAcc'].to_numpy().reshape(int(len(X_test)/size_batch),size_batch)

X_test = np.stack((A1,A2,A3),axis=1)
X_test = X_test.reshape([int(len(X_test)),channels,size_batch])

sns.countplot(y_train)
```

```
/opt/jupyterhub/anaconda/lib/python3.9/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[8]:
```
<AxesSubplot:ylabel='count'>
```



5.2 Balancing the data (Copy all positives and paste them after X_train)

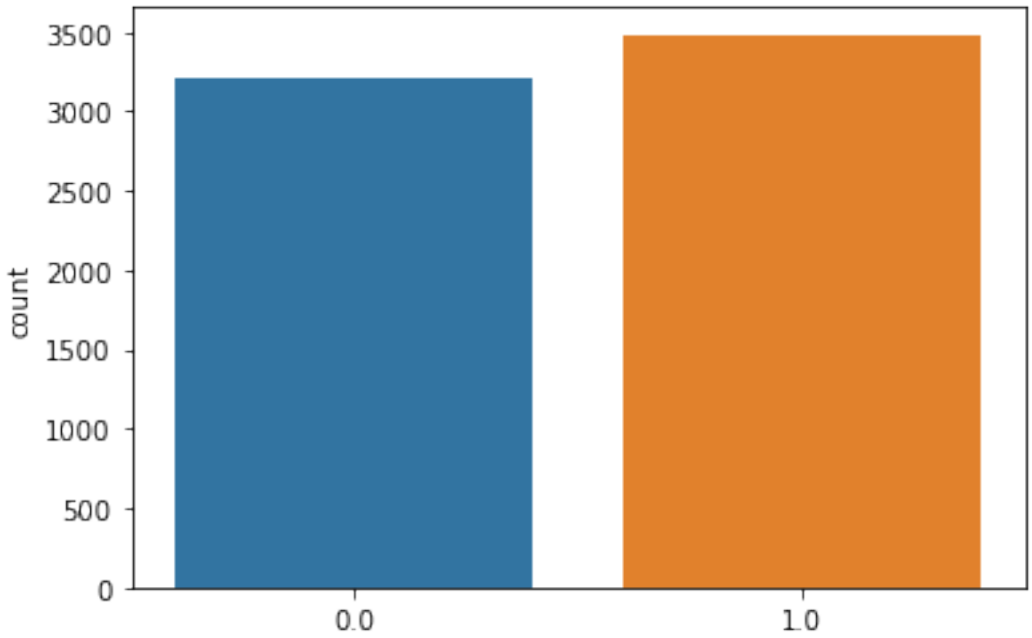
[9]:
```
X_train_Resample = pd.DataFrame()
y_train_Resample = pd.DataFrame()

for i in range(0,len(y_train)):
    if y_train[i] == 1:
        X_train_Resample = X_train_Resample.append(pd.DataFrame(X_train[i]))

        y_train_Resample = y_train_Resample.append(pd.DataFrame([1]))
```

[10]:
```
X_train_Resample = X_train_Resample.to_numpy()
X_train_Resample = X_train_Resample.reshape([int(len(X_train_Resample)/
 ↪channels),channels,size_batch])

y_train_Resample = y_train_Resample.to_numpy().reshape([len(y_train_Resample)])

for i in range(0,3):
    X_train_Resample = np.concatenate((X_train_Resample, X_train_Resample),␣
 ↪axis=0)
    y_train_Resample = np.concatenate((y_train_Resample, y_train_Resample),␣
 ↪axis=0)

X_train_Resample = np.concatenate((X_train, X_train_Resample), axis=0)
```

```
y_train_Resample = np.concatenate((y_train, y_train_Resample), axis=0)

sns.countplot(y_train_Resample)
```

```
/opt/jupyterhub/anaconda/lib/python3.9/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[10]: <AxesSubplot:ylabel='count'>



5.3 Convert the X_train, X_test, y_train, y_test to Tensors

[11]:
```
X_train = torch.from_numpy(X_train).float()
y_train = torch.squeeze(torch.from_numpy(y_train).float())

X_test = torch.from_numpy(X_test).float()
y_test = torch.squeeze(torch.from_numpy(y_test).float())

X_train_Resample = torch.from_numpy(X_train_Resample).float()
y_train_Resample = torch.squeeze(torch.from_numpy(y_train_Resample).float())
```