1D-CNN-1sec-Dataloader

December 6, 2021

1 Libraries

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

[22]:
```
import torch
import os
import math
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from torch.utils.data import DataLoader, TensorDataset
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from torch import nn, optim
import torch.nn.functional as F
```

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.

[23]:
```
#load in df
Player = 15
Game = 2
Quarter = 4
```

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

df_y = pd.read_csv('Player_15_Game2_Sprints_Q1234.csv')

df_y = df_y[['Action','Quarter','timeLine']]

df_y = df_y.iloc[::2]

#Delete first 99 rows, so df starts at timeLine == 1.00
df['wheelRotationalSpeedX_Diff'] = np.insert(np.diff(df.
↪wheelRotationalSpeedX,n=1),0,0)

df.head()
```

[23]:

	Unnamed: 0	frAcc	frRoAcc	frDispl	frRoAng	frSpeed	timeLine	\
0	358099	17.91200	2828.7	3394.9	81.515	0.23278	3581.0	
1	358100	7.14760	2828.7	3395.7	81.397	0.41190	3581.0	
2	358101	-0.89061	2828.8	3396.5	82.233	0.48338	3581.0	
3	358102	4.17050	2828.8	3397.3	74.342	0.47447	3581.0	
4	358103	1.15710	2828.8	3398.0	74.332	0.51618	3581.0	

	frameRotationalSpeedX	frameRotationalSpeedY	frameRotationalSpeedZ	\
0	7.3500	-8.9367	80.033	
1	4.0600	-8.1200	80.010	
2	-17.5700	-0.4200	81.340	
3	-1.6940	-10.5000	72.576	
4	-2.0922	-6.1600	73.111	

	wheelRotationalSpeedX	wheelRotationalSpeedY	wheelRotationalSpeedZ	\
0	-32.737	-39.114	45.033	
1	-1.050	-15.960	38.570	
2	10.780	-32.340	45.640	
3	16.485	-41.601	51.163	
4	23.847	-34.619	54.281	

	frRoSpeed	Filt_WheelX	Filt_FrameZ	Action	Quarter	\
0	-11.8030	67.674366	8.849375	0.0	1	
1	83.6110	65.140780	8.214781	0.0	1	
2	-789.0700	62.446959	7.721630	0.0	1	
3	-1.0099	59.592743	7.380241	0.0	1	
4	94.2010	56.579696	7.199759	0.0	1	

	wheelRotationalSpeedX_Diff
0	0.000
1	31.687
2	11.830

```
3                    5.705
4                    7.362
```

4 Pre Preprocessing

[24]:
```
#Convert Sprinting to 1
df.Action.replace({'Sprinting': 1},inplace=True)
```

5 Data Preparating

Split data into a train and test set

[25]:
```
train = df[df.Quarter != Quarter]
test = df[df.Quarter == Quarter]

y_train = df_y[df_y.Quarter != Quarter]
y_test = df_y[df_y.Quarter == Quarter]
y_test = y_test.iloc[:-1 , :]

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 = y_train[['Action']]

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

y_test = y_test[['Action']]
```

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

[26]:
```
size_batch = 100
channels = 3
```

[27]:
```
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)
```

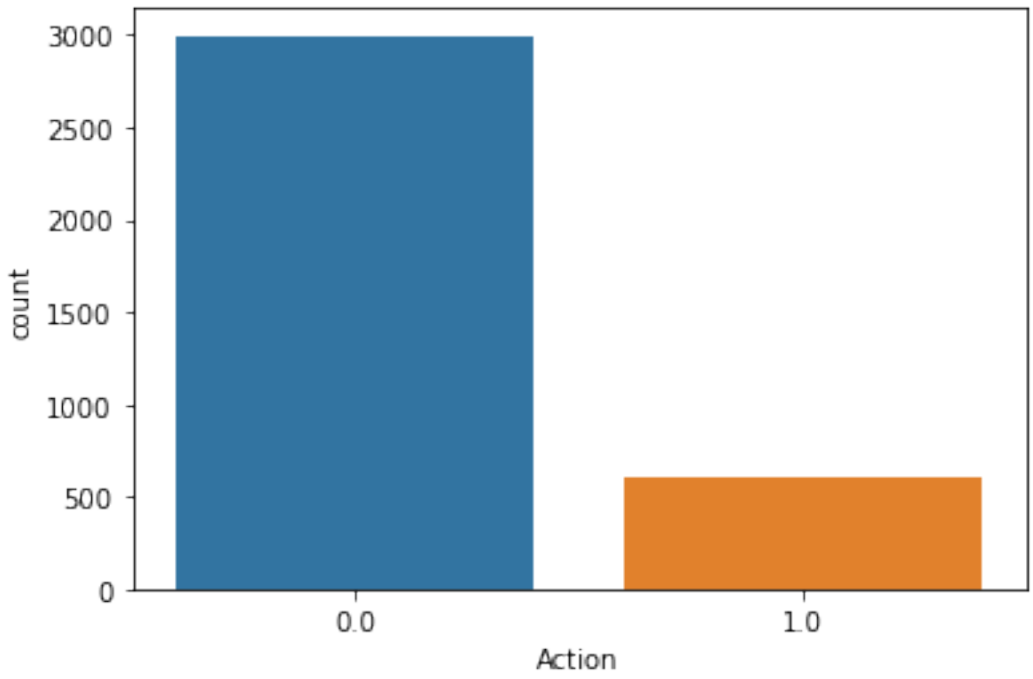
```
A3 = X_train['frAcc'].to_numpy().reshape(int(len(X_train)/
 ↪size_batch),size_batch)

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)
```

```
(3595, 3, 100)
```

[28]:
```
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.Action)
```

```
/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(
```

[28]:
```
<AxesSubplot:xlabel='Action', ylabel='count'>
```



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

[29]:
```
X_train_Resample = pd.DataFrame()
y_train_Resample = pd.DataFrame()

y_train = y_train.squeeze()
y_train = y_train.to_numpy()

print(y_train)

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]))
```

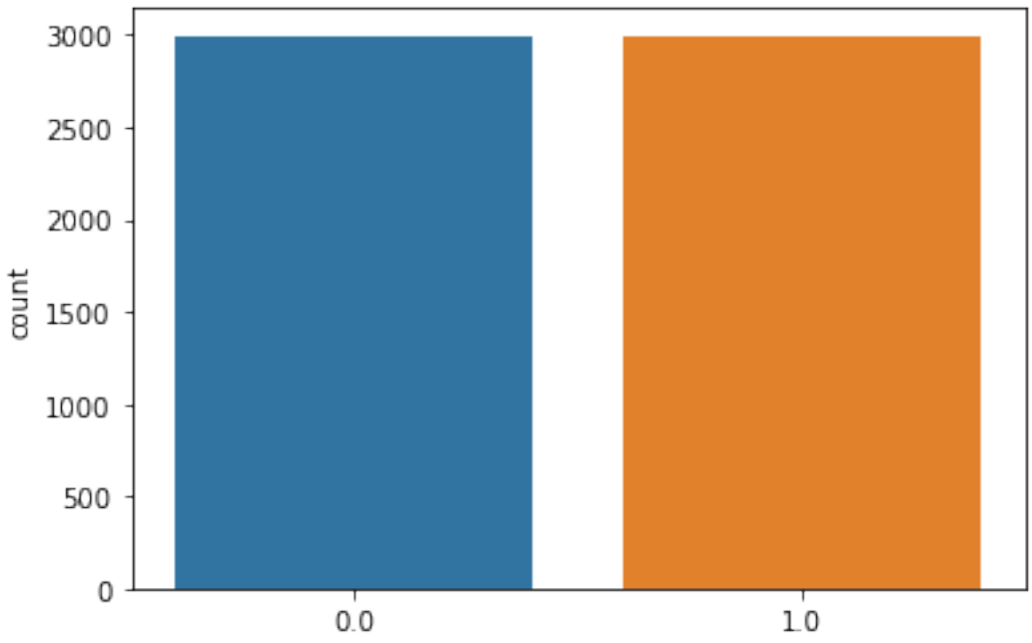
```
[0. 0. 0. … 0. 0. 0.]
```

[30]:
```
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(0,2):
    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)

print(y_train_Resample)

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)
```

```
[1 1 1 … 1 1 1]

/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(
```

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



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

[31]:
```
X_train = torch.from_numpy(X_train_Resample).float()
y_train = torch.squeeze(torch.from_numpy(y_train_Resample).float())

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

[32]:
```
train_ds = TensorDataset(X_train, y_train)
train_dl = DataLoader(train_ds, batch_size=64, num_workers = 4, pin_memory =␣
↪True)
```

6 CNN def

Define the Convolutional Neural Network

[33]:
```
class model(nn.Module):
    def __init__(self,n_features,kernel_size):
        super(model, self).__init__()
        #Use Padding to get good results
        self.conv1 = nn.Conv1d(n_features, 36, kernel_size=kernel_size, stride␣
↪= 1, padding='same') #3 input channels, 18 output channels
        self.conv2 = nn.Conv1d(36, 78, kernel_size=kernel_size, stride = 1,␣
↪padding='same') #18 input channels from previous Conv. layer, 36 out
        self.conv2_drop = nn.Dropout2d() #dropout


        self.fc1 = nn.Linear(78, 54) #Fully-connected classifier layer
        self.fc2 = nn.Linear(54, 19) #Fully-connected classifier layer
        self.fc3 = nn.Linear(19,1) #Fully-connected classifier layer

    def forward(self, x):
        x = F.relu(F.max_pool1d(self.conv1(x), 10)) #Use bigger Pool1d for model
        x = F.relu(F.max_pool1d(self.conv2_drop(self.conv2(x)), 10))

        #point A
        x = x.transpose(1, 2)

        #point B
        x = self.fc1(x)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return torch.sigmoid(self.fc3(x))
```

```
    def post_forward(self, y):
        return torch.round(y)

CNN = model(X_train.shape[1],50)
```

7 Training options

[34]:
```
criterion = nn.BCELoss()
optimizer = optim.Adam(CNN.parameters(), lr=0.001)
```

8 Training the NN on the GPU

[36]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

X_test = X_test.to(device)
y_test = y_test.to(device)

CNN = CNN.to(device)
criterion = criterion.to(device)
```

9 Define functions for calculating Accuracy of model

[37]:
```
def calculate_accuracy(y_true, y_pred):
    return (y_true == y_pred).sum().float() / len(y_true)
```

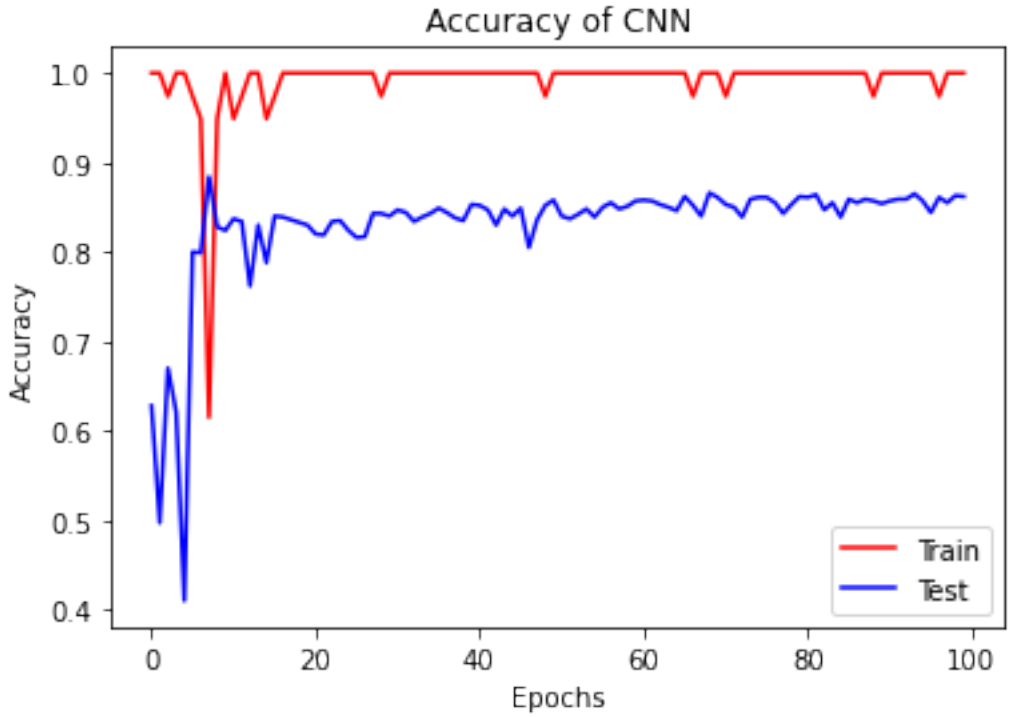
10 For loop through different epochs

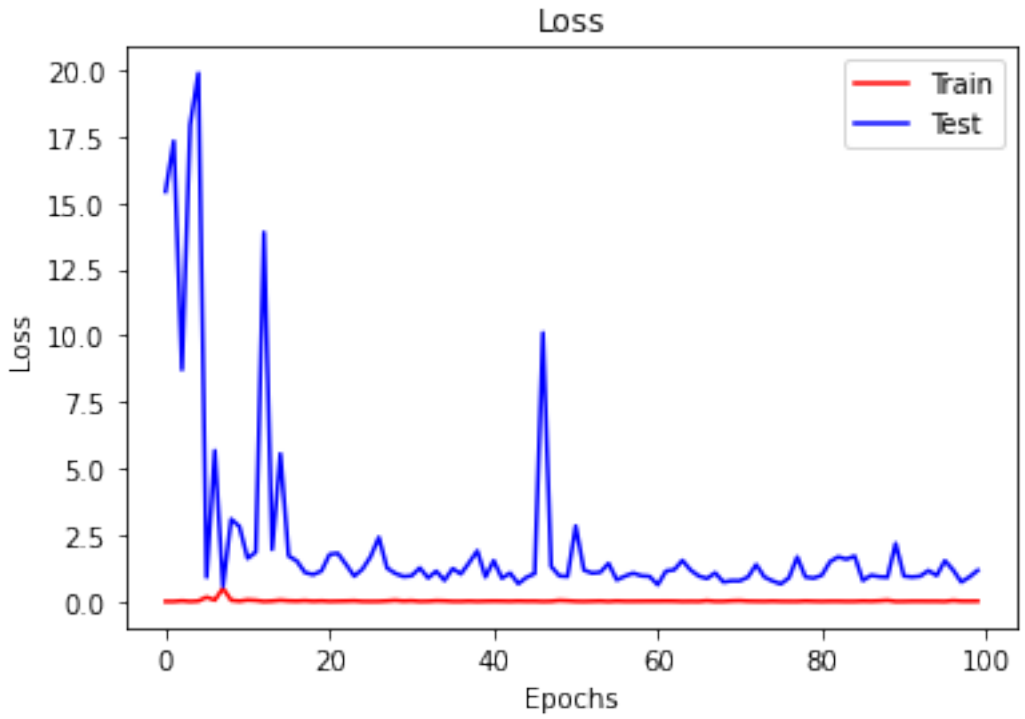
[38]:
```
def round_tensor(t, decimal_places=3):
    return round(t.item(), decimal_places)

Results_acc = pd.DataFrame(columns = ['Epoch', 'Acc_train', 'Acc_test'])
Results_loss = pd.DataFrame(columns = ['Epoch', 'Loss_train', 'Loss_test'])
Results_recall = pd.DataFrame(columns = ['Epoch', 'Recall_test'])
Results_prec = pd.DataFrame(columns = ['Epoch', 'Prec_test'])

for epoch in range(100):

    for x, y in train_dl:
```

```
        x, y = x.to(device), y.to(device)

        optimizer.zero_grad()
        y_pred = CNN(x)
        y_pred = y_pred.squeeze()
        train_loss = criterion(y_pred, y)
        train_loss.backward()
        optimizer.step()

    if epoch % 1 == 0:

        train_acc = calculate_accuracy(y, CNN.post_forward(y_pred))
        train_loss = criterion(y_pred,y)

        y_test_pred = CNN(X_test)
        y_test_pred = torch.squeeze(y_test_pred)
        test_loss = criterion(y_test_pred, y_test)
        test_acc = calculate_accuracy(y_test, CNN.post_forward(y_test_pred))

        Confusion = confusion_matrix(y_test.cpu(),y_test_pred.ge(.5).view(-1).
↪cpu())
        test_recall = Confusion[1][1]/(Confusion[1][1] + Confusion[1][0])
        test_prec = Confusion[1][1]/(Confusion[0][1] + Confusion[1][1])

        Acc = {'Epoch': epoch, 'Acc_train': round_tensor(train_acc), 'Acc_test':
↪ round_tensor(test_acc)}
        Loss = {'Epoch': epoch, 'Loss_train': round_tensor(train_loss),␣
↪'Loss_test': round_tensor(test_loss)}
        Recall = {'Epoch': epoch, 'Recall_test': round_tensor(test_recall)}
        Prec = {'Epoch': epoch, 'Prec_test': round_tensor(test_prec)}

        Results_acc = Results_acc.append(Acc, ignore_index=True)
        Results_loss = Results_loss.append(Loss, ignore_index=True)
        Results_recall = Results_recall.append(Recall, ignore_index=True)
        Results_prec = Results_prec.append(Prec, ignore_index=True)
```

[39]:
```
plt.plot(Results_acc.Epoch,Results_acc.Acc_train,'r',Results_acc.
 ↪Epoch,Results_acc.Acc_test,'b')
plt.legend(['Train','Test'])
plt.title('Accuracy of CNN')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.show()

plt.plot(Results_loss.Epoch,Results_loss.Loss_train,'r',Results_loss.
 ↪Epoch,Results_loss.Loss_test,'b')
plt.legend(['Train','Test'])
```

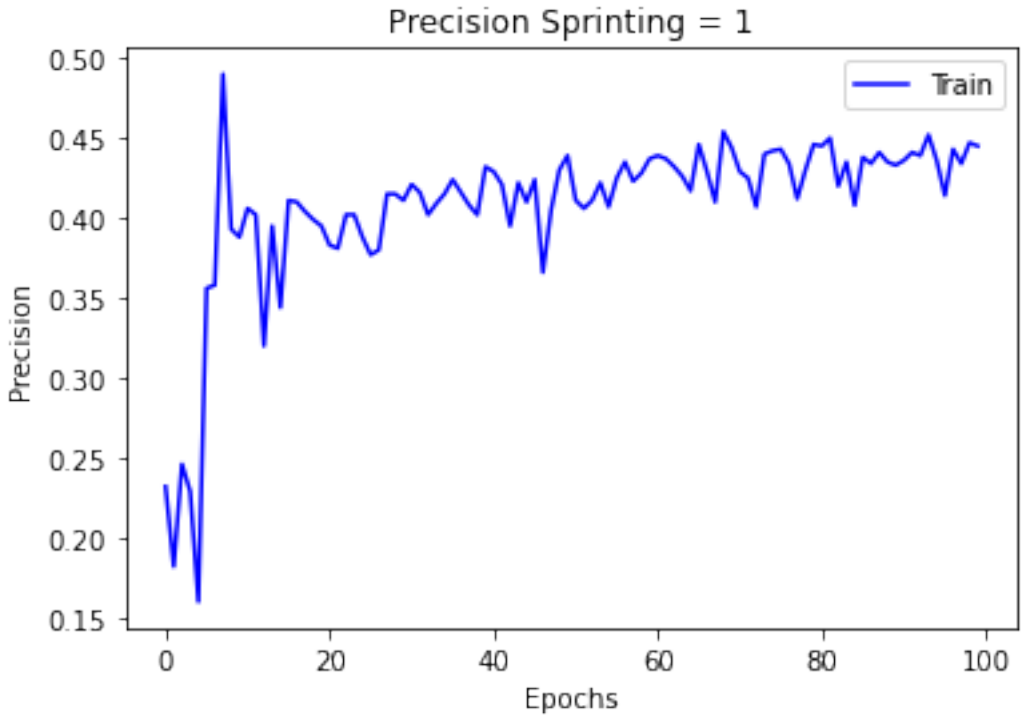
```
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.show()
```





[40]:
```
plt.plot(Results_recall.Epoch , Results_recall.Recall_test,'b')
plt.legend(['Test'])
plt.title('Recall Sprinting = 1')
plt.ylabel('Recall')
plt.xlabel('Epochs')
plt.show()

plt.plot(Results_prec.Epoch,Results_prec.Prec_test,'b')
plt.legend(['Train'])
plt.title('Precision Sprinting = 1')
plt.ylabel('Precision')
plt.xlabel('Epochs')
plt.show()
```


Recall Sprinting = 1
Test
Recall
Epochs
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
20
40
60
80
100


Precision Sprinting = 1
Train
Precision
Epochs
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
20
40
60
80
100

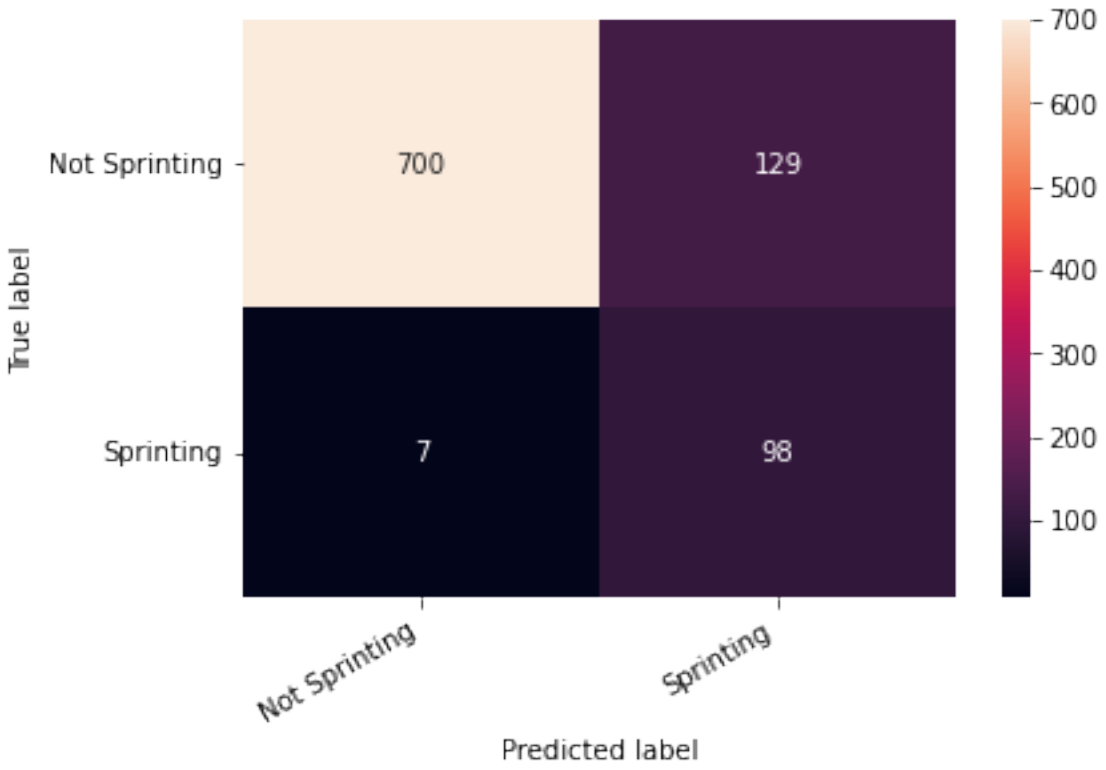
11 Validate/Tune Model

Validate results of the model (Precision/Recall). Tune the parameters of the model to achieve better results

[41]:
```
classes = ['Not Sprinting', 'Sprinting']
y_pred = CNN(X_test)
y_pred = y_pred.ge(.5).view(-1).cpu()
y_test = y_test.cpu()
print(classification_report(y_test, y_pred, target_names=classes))
```

```
              precision    recall  f1-score   support

Not Sprinting       0.99      0.84      0.91       829
    Sprinting       0.43      0.93      0.59       105

     accuracy                           0.85       934
    macro avg       0.71      0.89      0.75       934
 weighted avg       0.93      0.85      0.88       934
```

[42]:
```
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=classes, columns=classes)
hmap = sns.heatmap(df_cm, annot=True, fmt="d")
hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0, ha='right')
hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30, ha='right')
plt.ylabel('True label')
plt.xlabel('Predicted label');
```



12 Closing Notebook

[ ]:
```
%%javascript
Jupyter.notebook.session.delete();
```

```
<IPython.core.display.Javascript object>
```