Parkinson Symptom: Freezing of Gait (FOG) Prediction with PyTorch

In [1]:
```
import os
import gc
import random
import time

import json
from tqdm import tqdm
import glob
import numpy as np
import pandas as pd

import torch
import torch.nn as nn

from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

from sklearn.model_selection import train_test_split, StratifiedGroupKFold
from sklearn.metrics import accuracy_score, average_precision_score

import warnings
warnings.filterwarnings(action='ignore')
```

In [2]:
```
class Config:
    train_dir1 = "/input/tlvmc-parkinsons-freezing-gait-prediction/train/defog"
    train_dir2 = "/input/tlvmc-parkinsons-freezing-gait-prediction/train/tdcsfog"

    batch_size = 1024
    window_size = 32
    window_future = 8
    window_past = window_size - window_future

    wx = 8

    model_dropout = 0.2
    model_hidden = 512
    model_nblocks = 3

    lr = 0.00015
    num_epochs = 8
    device = 'cuda' if torch.cuda.is_available() else 'cpu'

    feature_list = ['AccV', 'AccML', 'AccAP']
    label_list = ['StartHesitation', 'Turn', 'Walking']


cfg = Config()
```

In [3]:
```
cfg.device
```

Out[3]:
```
'cuda'
```

Stratified Group K Fold

It's mentioned in the data that the subjects are different in the training and test sets, and there are also differences between the public and private splits of the test data. Therefore, we should use Stratified Group K Fold. However, since positive instances in the sequences are very scarce, we need to select the best fold that provides the optimal balance between positive and negative instances.

tdcsfog

In [4]:
```
# Analysis of positive instances in each fold of our CV folds

n1_sum = []
n2_sum = []
n3_sum = []
count = []


metadata = pd.read_csv("/input/copy-train-metadata/tdcsfog_metadata.csv")

for f in tqdm(metadata['Id']):
    fpath = f"/input/tlvmc-parkinsons-freezing-gait-prediction/train/tdcsfog/{f}.csv"
    df = pd.read_csv(fpath)

    n1_sum.append(np.sum(df['StartHesitation']))
    n2_sum.append(np.sum(df['Turn']))
    n3_sum.append(np.sum(df['Walking']))
    count.append(len(df))

print(f"32 files have positive values in all 3 classes")

metadata['n1_sum'] = n1_sum
metadata['n2_sum'] = n2_sum
metadata['n3_sum'] = n3_sum
metadata['count'] = count

sgkf = StratifiedGroupKFold(n_splits=5, random_state=42, shuffle=True)
for i, (train_index, valid_index) in enumerate(sgkf.split(X=metadata['Id'], y=[1]*len(
    print(f"Fold = {i}")
    train_ids = metadata.loc[train_index, 'Id']
    valid_ids = metadata.loc[valid_index, 'Id']

    print(f"Length of Train = {len(train_index)}, Length of Valid = {len(valid_index)}
    n1_sum = metadata.loc[train_index, 'n1_sum'].sum()
    n2_sum = metadata.loc[train_index, 'n2_sum'].sum()
    n3_sum = metadata.loc[train_index, 'n3_sum'].sum()
    print(f"Train classes: {n1_sum:,}, {n2_sum:,}, {n3_sum:,}")

    n1_sum = metadata.loc[valid_index, 'n1_sum'].sum()
    n2_sum = metadata.loc[valid_index, 'n2_sum'].sum()
    n3_sum = metadata.loc[valid_index, 'n3_sum'].sum()
    print(f"Valid classes: {n1_sum:,}, {n2_sum:,}, {n3_sum:,}")

# FOLD 2 is the most well balanced
# The actual train-test split (based on Fold 2)
```

```
metadata = pd.read_csv("/input/copy-train-metadata/tdcsfog_metadata.csv")
sgkf = StratifiedGroupKFold(n_splits=5, random_state=42, shuffle=True)
for i, (train_index, valid_index) in enumerate(sgkf.split(X=metadata['Id'], y=[1]*len(
    if i != 2:
        continue
    print(f"Fold = {i}")
    train_ids = metadata.loc[train_index, 'Id']
    valid_ids = metadata.loc[valid_index, 'Id']
    print(f"Length of Train = {len(train_ids)}, Length of Valid = {len(valid_ids)}")

    if i == 2:
        break

train_fpaths_tdcs = [f"/input/tlvmc-parkinsons-freezing-gait-prediction/train/tdcsfog/
valid_fpaths_tdcs = [f"/input/tlvmc-parkinsons-freezing-gait-prediction/train/tdcsfog/
```

```
100%|██████████| 833/833 [00:23<00:00, 34.91it/s]
32 files have positive values in all 3 classes
Fold = 0
Length of Train = 672, Length of Valid = 161
Train classes: 287,832, 1,462,652, 175,633
Valid classes: 16,958, 216,130, 32,205
Fold = 1
Length of Train = 613, Length of Valid = 220
Train classes: 51,748, 909,505, 65,242
Valid classes: 253,042, 769,277, 142,596
Fold = 2
Length of Train = 703, Length of Valid = 130
Train classes: 271,881, 1,332,746, 183,673
Valid classes: 32,909, 346,036, 24,165
Fold = 3
Length of Train = 649, Length of Valid = 184
Train classes: 303,710, 1,517,147, 205,196
Valid classes: 1,080, 161,635, 2,642
Fold = 4
Length of Train = 695, Length of Valid = 138
Train classes: 303,989, 1,493,078, 201,608
Valid classes: 801, 185,704, 6,230
Fold = 2
Length of Train = 703, Length of Valid = 130
```

defog

In [5]:

```
# Analysis of positive instances in each fold of our CV folds

n1_sum = []
n2_sum = []
n3_sum = []
count = []


metadata = pd.read_csv("/input/copy-train-metadata/defog_metadata.csv")
metadata['n1_sum'] = 0
metadata['n2_sum'] = 0
metadata['n3_sum'] = 0
metadata['count'] = 0
```

```python
for f in tqdm(metadata['Id']):
    fpath = f"/input/tlvmc-parkinsons-freezing-gait-prediction/train/defog/{f}.csv"
    if os.path.exists(fpath) == False:
        continue

    df = pd.read_csv(fpath)
    metadata.loc[metadata['Id'] == f, 'n1_sum'] = np.sum(df['StartHesitation'])
    metadata.loc[metadata['Id'] == f, 'n2_sum'] = np.sum(df['Turn'])
    metadata.loc[metadata['Id'] == f, 'n3_sum'] = np.sum(df['Walking'])
    metadata.loc[metadata['Id'] == f, 'count'] = len(df)

metadata = metadata[metadata['count'] > 0].reset_index()

sgkf = StratifiedGroupKFold(n_splits=5, random_state=42, shuffle=True)
for i, (train_index, valid_index) in enumerate(sgkf.split(X=metadata['Id'], y=[1]*len(
    print(f"Fold = {i}")
    train_ids = metadata.loc[train_index, 'Id']
    valid_ids = metadata.loc[valid_index, 'Id']

    print(f"Length of Train = {len(train_index)}, Length of Valid = {len(valid_index)}
    n1_sum = metadata.loc[train_index, 'n1_sum'].sum()
    n2_sum = metadata.loc[train_index, 'n2_sum'].sum()
    n3_sum = metadata.loc[train_index, 'n3_sum'].sum()
    print(f"Train classes: {n1_sum:,}, {n2_sum:,}, {n3_sum:,}")

    n1_sum = metadata.loc[valid_index, 'n1_sum'].sum()
    n2_sum = metadata.loc[valid_index, 'n2_sum'].sum()
    n3_sum = metadata.loc[valid_index, 'n3_sum'].sum()
    print(f"Valid classes: {n1_sum:,}, {n2_sum:,}, {n3_sum:,}")

# FOLD 2 is the most well balanced
# The actual train-test split (based on Fold 2)

sgkf = StratifiedGroupKFold(n_splits=5, random_state=42, shuffle=True)
for i, (train_index, valid_index) in enumerate(sgkf.split(X=metadata['Id'], y=[1]*len(
    if i != 1:
        continue
    print(f"Fold = {i}")
    train_ids = metadata.loc[train_index, 'Id']
    valid_ids = metadata.loc[valid_index, 'Id']
    print(f"Length of Train = {len(train_ids)}, Length of Valid = {len(valid_ids)}")

    if i == 2:
        break

train_fpaths_de = [f"/input/tlvmc-parkinsons-freezing-gait-prediction/train/defog/{_id
valid_fpaths_de = [f"/input/tlvmc-parkinsons-freezing-gait-prediction/train/defog/{_id
```

```
100%|██████████| 137/137 [00:30<00:00,  4.54it/s]
```

```
Fold = 0
Length of Train = 75, Length of Valid = 16
Train classes: 500, 428,683, 37,609
Valid classes: 0, 158,803, 60,910
Fold = 1
Length of Train = 65, Length of Valid = 26
Train classes: 216, 490,429, 84,955
Valid classes: 284, 97,057, 13,564
Fold = 2
Length of Train = 76, Length of Valid = 15
Train classes: 410, 488,634, 87,986
Valid classes: 90, 98,852, 10,533
Fold = 3
Length of Train = 70, Length of Valid = 21
Train classes: 435, 424,494, 88,800
Valid classes: 65, 162,992, 9,719
Fold = 4
Length of Train = 78, Length of Valid = 13
Train classes: 439, 517,704, 94,726
Valid classes: 61, 69,782, 3,793
Fold = 1
Length of Train = 65, Length of Valid = 26
```

In [6]:
```
train_fpaths = [(f, 'de') for f in train_fpaths_de] + [(f, 'tdcs') for f in train_fpat
valid_fpaths = [(f, 'de') for f in valid_fpaths_de] + [(f, 'tdcs') for f in valid_fpat
```

Dataset

We create our dataset for a specific time instance using a window that includes past and future accelerometer (Acc) readings. If any part of the window data is missing, we fill the gaps with zeros.

In [7]:
```
class FOGDataset(Dataset):
    def __init__(self, fpaths, scale=9.806, split="train"):
        super(FOGDataset, self).__init__()
        tm = time.time()
        self.split = split
        self.scale = scale

        self.fpaths = fpaths
        self.dfs = [self.read(f[0], f[1]) for f in fpaths]
        self.f_ids = [os.path.basename(f[0])[:-4] for f in self.fpaths]

        self.end_indices = []
        self.shapes = []
        _length = 0
        for df in self.dfs:
            self.shapes.append(df.shape[0])
            _length += df.shape[0]
            self.end_indices.append(_length)

        self.dfs = np.concatenate(self.dfs, axis=0).astype(np.float16)
        self.length = self.dfs.shape[0]

        shape1 = self.dfs.shape[1]

        self.dfs = np.concatenate([np.zeros((cfg.wx*cfg.window_past, shape1)), self.df
```

```
        print(f"Dataset initialized in {time.time() - tm} secs!")
        gc.collect()

    def read(self, f, _type):
        df = pd.read_csv(f)
        if self.split == "test":
            return np.array(df)

        if _type =="tdcs":
            df['Valid'] = 1
            df['Task'] = 1
            df['tdcs'] = 1
        else:
            df['tdcs'] = 0

        return np.array(df)

    def __getitem__(self, index):
        if self.split == "train":
            row_idx = random.randint(0, self.length-1) + cfg.wx*cfg.window_past
        elif self.split == "test":
            for i,e in enumerate(self.end_indices):
                if index >= e:
                    continue
                df_idx = i
                break

            row_idx_true = self.shapes[df_idx] - (self.end_indices[df_idx] - index)
            _id = self.f_ids[df_idx] + "_" + str(row_idx_true)
            row_idx = index + cfg.wx*cfg.window_past
        else:
            row_idx = index + cfg.wx*cfg.window_past

        #scale = 9.806 if self.dfs[row_idx, -1] == 1 else 1.0
        x = self.dfs[row_idx - cfg.wx*cfg.window_past : row_idx + cfg.wx*cfg.window_fu
        x = x[::cfg.wx, :][::-1, :]
        x = torch.tensor(x.astype('float'))#/scale

        t = self.dfs[row_idx, -3]*self.dfs[row_idx, -2]

        if self.split == "test":
            return _id, x, t

        y = self.dfs[row_idx, 4:7].astype('float')
        y = torch.tensor(y)

        return x, y, t

    def __len__(self):
        # return self.length
        if self.split == "train":
            return 5_000_000
        return self.length
```

In [8]:
```
gc.collect()
```

Out[8]:
```
23
```

Model

In [9]:
```
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )

class FOGModel(nn.Module):
    def __init__(self, p=cfg.model_dropout, dim=cfg.model_hidden, nblocks=cfg.model_nb
        super(FOGModel, self).__init__()
        self.dropout = nn.Dropout(p)
        self.in_layer = nn.Linear(cfg.window_size*3, dim)
        self.blocks = nn.Sequential(*[_block(dim, dim, p) for _ in range(nblocks)])
        self.out_layer = nn.Linear(dim, 3)

    def forward(self, x):
        x = x.view(-1, cfg.window_size*3)
        x = self.in_layer(x)
        for block in self.blocks:
            x = block(x)
        x = self.out_layer(x)
        return x
```

In [10]:
```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
```

Training

In [11]:
```
from torch.cuda.amp import GradScaler

def train_one_epoch(model, loader, optimizer, criterion):
    loss_sum = 0.
    scaler = GradScaler()

    model.train()
    for x,y,t in tqdm(loader):
        x = x.to(cfg.device).float()
        y = y.to(cfg.device).float()
        t = t.to(cfg.device).float()

        y_pred = model(x)
        loss = criterion(y_pred, y)
        loss = torch.mean(loss*t.unsqueeze(-1), dim=1)

        t_sum = torch.sum(t)
        if t_sum > 0:
            loss = torch.sum(loss)/t_sum
        else:
            loss = torch.sum(loss)*0.

        # loss.backward()
        scaler.scale(loss).backward()
        # optimizer.step()
```

```
        scaler.step(optimizer)
        scaler.update()

        optimizer.zero_grad()

        loss_sum += loss.item()

    print(f"Train Loss: {(loss_sum/len(loader)):.04f}")


def validation_one_epoch(model, loader, criterion):
    loss_sum = 0.
    y_true_epoch = []
    y_pred_epoch = []
    t_valid_epoch = []

    model.eval()
    for x,y,t in tqdm(loader):
        x = x.to(cfg.device).float()
        y = y.to(cfg.device).float()
        t = t.to(cfg.device).float()

        with torch.no_grad():
            y_pred = model(x)
            loss = criterion(y_pred, y)
            loss = torch.mean(loss*t.unsqueeze(-1), dim=1)

            t_sum = torch.sum(t)
            if t_sum > 0:
                loss = torch.sum(loss)/t_sum
            else:
                loss = torch.sum(loss)*0.

        loss_sum += loss.item()
        y_true_epoch.append(y.cpu().numpy())
        y_pred_epoch.append(y_pred.cpu().numpy())
        t_valid_epoch.append(t.cpu().numpy())

    y_true_epoch = np.concatenate(y_true_epoch, axis=0)
    y_pred_epoch = np.concatenate(y_pred_epoch, axis=0)

    t_valid_epoch = np.concatenate(t_valid_epoch, axis=0)
    y_true_epoch = y_true_epoch[t_valid_epoch > 0, :]
    y_pred_epoch = y_pred_epoch[t_valid_epoch > 0, :]

    scores = [average_precision_score(y_true_epoch[:,i], y_pred_epoch[:,i]) for i in r
    mean_score = np.mean(scores)
    print(f"Validation Loss: {(loss_sum/len(loader)):.04f}, Validation Score: {mean_sc

    return mean_score
```

In [12]:

```
model = FOGModel().to(cfg.device)
print(f"Number of parameters in model - {count_parameters(model):,}")

train_dataset = FOGDataset(train_fpaths, split="train")
valid_dataset = FOGDataset(valid_fpaths, split="valid")
print(f"lengths of datasets: train - {len(train_dataset)}, valid - {len(valid_dataset)

train_loader = DataLoader(train_dataset, batch_size=cfg.batch_size, num_workers=5, shu
```

```python
valid_loader = DataLoader(valid_dataset, batch_size=cfg.batch_size, num_workers=5)

optimizer = torch.optim.Adam(model.parameters(), lr=cfg.lr)
criterion = torch.nn.BCEWithLogitsLoss(reduction='none').to(cfg.device)
# sched = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.85)

max_score = 0.0

print("="*50)
for epoch in range(cfg.num_epochs):
    print(f"Epoch: {epoch}")
    train_one_epoch(model, train_loader, optimizer, criterion)
    score = validation_one_epoch(model, valid_loader, criterion)
    # sched.step()

    if score > max_score:
        max_score = score
        torch.save(model.state_dict(), "best_model_state.h5")
        print("Saving Model ...")

    print("="*50)

gc.collect()
```

```
Number of parameters in model - 842,243
Dataset initialized in 47.77522015571594 secs!
Dataset initialized in 14.32553482055664 secs!
lengths of datasets: train - 5000000, valid - 4984740
==================================================
Epoch: 0
100%|██████████| 4883/4883 [01:34<00:00, 51.46it/s]
Train Loss: 0.1761
100%|██████████| 4868/4868 [01:15<00:00, 64.52it/s]
Validation Loss: 0.0820, Validation Score: 0.245, ClassWise: 0.047,0.658,0.029
Saving Model ...
==================================================
Epoch: 1
100%|██████████| 4883/4883 [01:31<00:00, 53.12it/s]
Train Loss: 0.1539
100%|██████████| 4868/4868 [01:14<00:00, 65.65it/s]
Validation Loss: 0.0848, Validation Score: 0.238, ClassWise: 0.038,0.643,0.032
==================================================
Epoch: 2
100%|██████████| 4883/4883 [01:32<00:00, 52.66it/s]
Train Loss: 0.1459
100%|██████████| 4868/4868 [01:14<00:00, 65.33it/s]
Validation Loss: 0.0899, Validation Score: 0.238, ClassWise: 0.040,0.637,0.036
==================================================
Epoch: 3
100%|██████████| 4883/4883 [01:32<00:00, 52.63it/s]
Train Loss: 0.1405
100%|██████████| 4868/4868 [01:15<00:00, 64.33it/s]
Validation Loss: 0.0946, Validation Score: 0.242, ClassWise: 0.035,0.639,0.051
==================================================
Epoch: 4
100%|██████████| 4883/4883 [01:33<00:00, 52.19it/s]
Train Loss: 0.1363
```

```
100%|██████████| 4868/4868 [01:15<00:00, 64.89it/s]
Validation Loss: 0.0926, Validation Score: 0.243, ClassWise: 0.044,0.640,0.046
==================================================
Epoch: 5
100%|██████████| 4883/4883 [01:32<00:00, 52.67it/s]
Train Loss: 0.1325
100%|██████████| 4868/4868 [01:14<00:00, 65.09it/s]
Validation Loss: 0.0958, Validation Score: 0.244, ClassWise: 0.038,0.648,0.048
==================================================
Epoch: 6
100%|██████████| 4883/4883 [01:32<00:00, 52.76it/s]
Train Loss: 0.1294
100%|██████████| 4868/4868 [01:16<00:00, 63.92it/s]
Validation Loss: 0.0960, Validation Score: 0.243, ClassWise: 0.038,0.645,0.045
==================================================
Epoch: 7
100%|██████████| 4883/4883 [01:33<00:00, 52.33it/s]
Train Loss: 0.1269
100%|██████████| 4868/4868 [01:16<00:00, 63.89it/s]
Validation Loss: 0.1016, Validation Score: 0.244, ClassWise: 0.034,0.650,0.050
==================================================
```

Out[12]:
```
0
```

Result

In [13]:
```
model = FOGModel().to(cfg.device)
model.load_state_dict(torch.load("/working/best_model_state.h5"))
model.eval()

test_defog_paths = glob.glob("/input/tlvmc-parkinsons-freezing-gait-prediction/test/de
test_tdcsfog_paths = glob.glob("/input/tlvmc-parkinsons-freezing-gait-prediction/test/
test_fpaths = [(f, 'de') for f in test_defog_paths] + [(f, 'tdcs') for f in test_tdcsf

test_dataset = FOGDataset(test_fpaths, split="test")
test_loader = DataLoader(test_dataset, batch_size=cfg.batch_size, num_workers=5)

ids = []
preds = []

for _id, x, _ in tqdm(test_loader):
    x = x.to(cfg.device).float()
    with torch.no_grad():
        y_pred = model(x)*0.1

    ids.extend(_id)
    preds.extend(list(np.nan_to_num(y_pred.cpu().numpy())))
```

```
Dataset initialized in 0.5296854972839355 secs!
100%|██████████| 280/280 [00:03<00:00, 85.93it/s]
```

In [14]:
```
sample_result = pd.read_csv("/input/tlvmc-parkinsons-freezing-gait-prediction/sample_r
sample_result.shape
```

Out[14]:
```
(286370, 4)
```

In [15]:
```
preds = np.array(preds)
submission = pd.DataFrame({'Id': ids, 'StartHesitation': np.round(preds[:,0],5), \
                           'Turn': np.round(preds[:,1],5), 'Walking': np.round(preds[:

submission = pd.merge(sample_submission[['Id']], submission, how='left', on='Id').fill
submission.to_csv("result.csv", index=False)
```

In [16]:
```
print(result.shape)
result.head()
```

```
(286370, 4)
```

Out[16]:

	Id	StartHesitation	Turn	Walking
0	003f117e14_0	-0.51943	-0.39609	-0.57060
1	003f117e14_1	-0.51950	-0.39636	-0.57089
2	003f117e14_2	-0.51870	-0.39566	-0.56989
3	003f117e14_3	-0.51897	-0.39604	-0.57041
4	003f117e14_4	-0.51876	-0.39579	-0.57034