Final

June 8, 2025

[ ]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset, random_split
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_class_weight
from imblearn.over_sampling import RandomOverSampler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```python
WINDOW = 30
LABELS = [0, 1, 2, 3, 4]
channels = ['ACC_X', 'ACC_Y', 'ACC_Z', 'TEMP', 'EDA', 'HR']
cols = ['ACC_X', 'ACC_Y', 'ACC_Z', 'TEMP', 'EDA', 'HR', 'Sleep_Stage']
stage_map = {"W": 0, "N1": 1, "N2": 2, "N3": 3, "R": 4}
subjects_train = ['S002', 'S008']
subject_val = 'S005'
subjects = ['S002', 'S005', 'S008']

dfs = []
for subject in ['S002', 'S005', 'S008']:
    file_path = f'/Users/veeralpatel/ECE284FinalProject/data/
  ↪{subject}_PSG_df_updated.csv'
    df = pd.read_csv(file_path, usecols=cols)
    df['Subject'] = subject  # Add subject ID for tracking
    df['Sleep_Stage'] = df['Sleep_Stage'].map(stage_map)
    df = df.dropna(subset=['Sleep_Stage'])
    df['Sleep_Stage'] = df['Sleep_Stage'].astype(int)
    dfs.append(df)


full_df = pd.concat(dfs, ignore_index=True)
# print(full_df['Subject'].value_counts())
# print(full_df['Sleep_Stage'].value_counts())
```

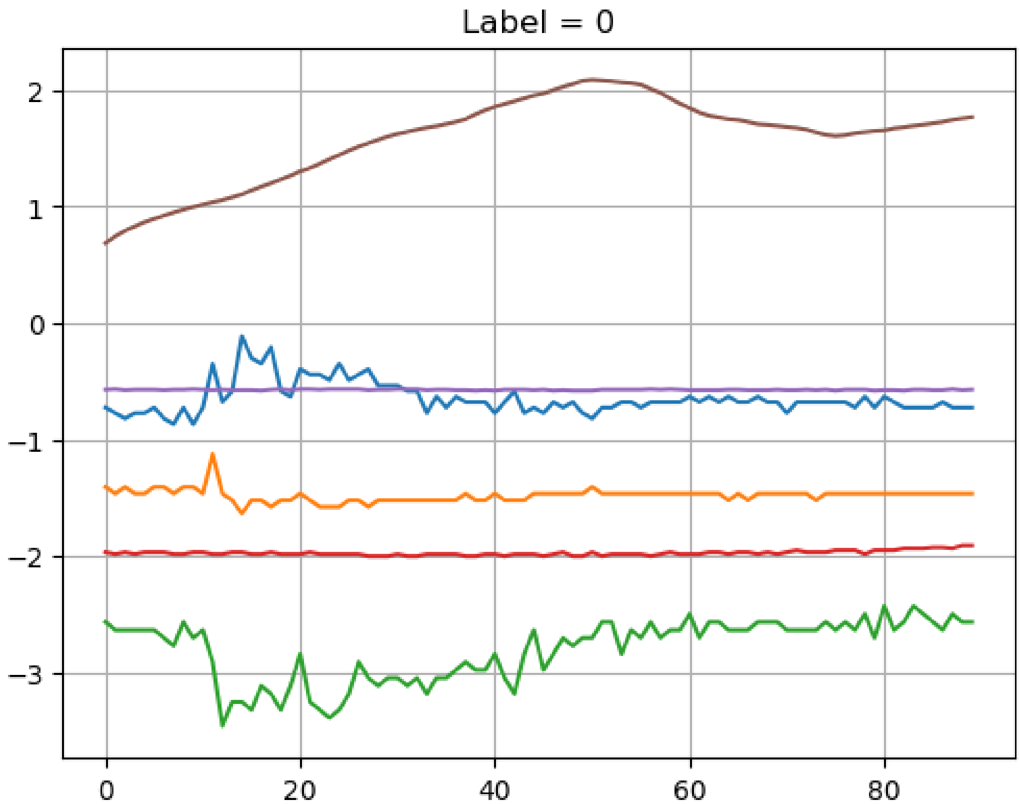
```
Subject
S008    2243997
S005    2198997
S002    1964127
Name: count, dtype: int64
Sleep_Stage
2    3693000
0    1115991
4     911130
1     405000
3     282000
Name: count, dtype: int64
```

[ ]:
```python
def load_and_preprocess(subject):
    path = f'/Users/veeralpatel/ECE284FinalProject/data/
 ↪{subject}_PSG_df_updated.csv'
    df = pd.read_csv(path, usecols=channels + ['Sleep_Stage'])
    df = df.iloc[::100].reset_index(drop=True)  # Downsample from 100Hz → 1Hz
    df['Sleep_Stage'] = df['Sleep_Stage'].map(stage_map)
    df = df.dropna(subset=['Sleep_Stage'])
    df['Sleep_Stage'] = df['Sleep_Stage'].astype(int)
    for col in channels:
        df[col] = (df[col] - df[col].mean()) / df[col].std()
    return df

def extract_contextual_epochs(df, window=30):
    X_segments, y_segments = [], []
    for i in range(window, len(df) - 2*window, window):
        prev_chunk = df.iloc[i-window:i]
        curr_chunk = df.iloc[i:i+window]
        next_chunk = df.iloc[i+window:i+2*window]

        if curr_chunk["Sleep_Stage"].nunique() == 1:
            combined = pd.concat([prev_chunk, curr_chunk, next_chunk])
            X_segments.append(combined[channels].values)
            y_segments.append(curr_chunk["Sleep_Stage"].iloc[0])

    return np.stack(X_segments), np.array(y_segments)


X_all, y_all = [], []
for subject in subjects:
    df = load_and_preprocess(subject)
    X_seg, y_seg = extract_contextual_epochs(df)

    X_all.append(X_seg)
    y_all.append(y_seg)
```

```
X = np.concatenate(X_all)
y = np.concatenate(y_all)
X_train, X_val, y_train, y_val = train_test_split(X, y, stratify=y, test_size=0.
↪2, random_state=42) #SPLITTING DSRET HERE

#over-sampling
X_flat = X_train.reshape(len(X_train), -1)
ros = RandomOverSampler(random_state=42)
X_resampled, y_train = ros.fit_resample(X_flat, y_train)
X_train = X_resampled.reshape(-1, WINDOW, len(channels))


class SleepDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.long)
    def __len__(self): return len(self.X)
    def __getitem__(self, idx): return self.X[idx], self.y[idx]
train_loader = DataLoader(SleepDataset(X_train, y_train), batch_size=32,␣
↪shuffle=True)
val_loader = DataLoader(SleepDataset(X_val, y_val), batch_size=32)

class_weights = compute_class_weight('balanced', classes=np.unique(y_train),␣
↪y=y_train)
class_weights = torch.tensor(class_weights, dtype=torch.float32)

print(f"  Train: {len(X_train)}, Val: {len(X_val)}")
print("  Class weights:", class_weights)

print("Unique labels:", np.unique(y))  # should print [0, 1, 2, 3, 4]
from collections import Counter
print("Full dataset:", Counter(y))
print("Train:", Counter(y_train))
print("Val:", Counter(y_val))
print("X shape:", X.shape)              # should be (num_epochs, 30, 6)
print("X_train shape:", X_train.shape) # (after resample)
print("X_val shape:", X_val.shape)
print("Sample input:", X_train[0])
assert not np.isnan(X).any(), "X has NaNs!"
assert not np.isnan(y).any(), "y has NaNs!"
import matplotlib.pyplot as plt
plt.plot(X[0])  # plot one 30s epoch across 6 channels
plt.title(f"Label = {y[0]}")
plt.grid(True)
plt.show()
from collections import Counter
```

```python
print("Resampled Train Class Balance:", Counter(y_train))
```

```
 Train: 13275, Val: 349
 Class weights: tensor([1., 1., 1., 1., 1.])
Unique labels: [0 1 2 3 4]
Full dataset: Counter({2: 1107, 4: 269, 0: 256, 3: 72, 1: 39})
Train: Counter({4: 885, 0: 885, 2: 885, 3: 885, 1: 885})
Val: Counter({2: 222, 4: 54, 0: 51, 3: 14, 1: 8})
X shape: (1743, 90, 6)
X_train shape: (13275, 30, 6)
X_val shape: (349, 90, 6)
Sample input: [[-0.21425795  0.21438903  1.75161913  0.66231614 -0.4390945
-0.66302252]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.44659038 -0.67091766]
 [-0.21425795  0.24171465  1.75161913  0.66231614 -0.44659038 -0.67793555]
 [-0.21425795  0.21438903  1.75161913  0.68134642 -0.43159862 -0.68407621]
 [-0.21425795  0.21438903  1.75161913  0.68134642 -0.43159862 -0.68846239]
 [-0.21425795  0.24171465  1.75161913  0.66231614 -0.44659038 -0.69284858]
 [-0.21425795  0.21438903  1.75161913  0.68134642 -0.44659038 -0.69723476]
 [-0.21425795  0.21438903  1.75161913  0.68134642 -0.4390945  -0.698112  ]
 [-0.21425795  0.21438903  1.75161913  0.68134642 -0.43159862 -0.69986647]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.44659038 -0.69986647]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.4390945  -0.698112  ]
 [-0.21425795  0.24171465  1.75161913  0.68134642 -0.4390945  -0.69723476]
 [-0.21425795  0.21438903  1.75161913  0.64328585 -0.44659038 -0.69372581]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.44659038 -0.68933963]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.43159862 -0.68495345]
 [-0.21425795  0.21438903  1.75161913  0.64328585 -0.4390945  -0.68056726]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.44659038 -0.67530384]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.4390945  -0.66916318]
 [-0.21425795  0.21438903  1.75161913  0.68134642 -0.4390945  -0.664777  ]
 [-0.21425795  0.21438903  1.75161913  0.64328585 -0.43159862 -0.66039081]
 [-0.21425795  0.24171465  1.75161913  0.68134642 -0.43159862 -0.65600463]
 [-0.21425795  0.24171465  1.75161913  0.66231614 -0.4390945  -0.65161844]
 [-0.21425795  0.21438903  1.75161913  0.68134642 -0.4390945  -0.64898673]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.44659038 -0.64547779]
 [-0.21425795  0.21438903  1.75161913  0.64328585 -0.4390945  -0.64284608]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.4390945  -0.6410916 ]
 [-0.21425795  0.24171465  1.75161913  0.66231614 -0.4390945  -0.64021437]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.43159862 -0.63845989]
 [-0.21425795  0.21438903  1.75161913  0.64328585 -0.42410274 -0.63670542]
 [-0.21425795  0.21438903  1.75161913  0.66231614 -0.4390945  -0.63582818]]
```




```
Resampled Train Class Balance: Counter({4: 885, 0: 885, 2: 885, 3: 885, 1: 885})
```

[71]:
```
class SleepDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.long)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

dataset = SleepDataset(X, y)
train_len = int(0.8 * len(dataset))
train_set, val_set = random_split(dataset, [train_len, len(dataset) -
 ↪train_len])
train_loader = DataLoader(train_set, batch_size=32, shuffle=True)
val_loader = DataLoader(val_set, batch_size=32)
```

```python
class CNN_BiLSTM_Model(nn.Module):
    def __init__(self, input_channels=6, num_classes=5):
        super(CNN_BiLSTM_Model, self).__init__()
        self.conv1 = nn.Conv1d(input_channels, 64, kernel_size=5)
        self.bn1 = nn.BatchNorm1d(64)
        self.pool1 = nn.MaxPool1d(kernel_size=2)

        self.conv2 = nn.Conv1d(64, 128, kernel_size=3)
        self.bn2 = nn.BatchNorm1d(128)

        self.norm = nn.LayerNorm(128)
        self.lstm = nn.LSTM(128, 128, batch_first=True, bidirectional=True,␣
 ↪dropout=0.3)

        self.fc1 = nn.Linear(256, 64)
        self.dropout = nn.Dropout(0.6)
        self.fc2 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = x.permute(0, 2, 1)
        x = self.pool1(F.relu(self.bn1(self.conv1(x))))
        x = F.relu(self.bn2(self.conv2(x)))
        x = x.permute(0, 2, 1)
        x = self.norm(x)
        x, _ = self.lstm(x)
        x = x[:, -1, :]
        x = self.dropout(F.relu(self.fc1(x)))
        return self.fc2(x)
```

[72]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F

model = CNN_BiLSTM_Model(input_channels=6, num_classes=5)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss(weight=class_weights, label_smoothing=0.1)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',␣
 ↪patience=5, factor=0.5)

EPOCHS = 50
best_val_loss = float('inf')

for epoch in range(EPOCHS):
    # === Train ===
    model.train()
    train_loss, train_correct, total = 0, 0, 0
```

```
    for xb, yb in train_loader:
        optimizer.zero_grad()
        out = model(xb)
        loss = criterion(out, yb)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        train_correct += (out.argmax(1) == yb).sum().item()
        total += yb.size(0)

    train_acc = train_correct / total

    # === Validate ===
    model.eval()
    val_loss, val_correct, total = 0, 0, 0
    all_preds, all_labels = [], []

    with torch.no_grad():
        for xb, yb in val_loader:
            out = model(xb)
            loss = criterion(out, yb)
            preds = out.argmax(1)

            val_loss += loss.item()
            val_correct += (preds == yb).sum().item()
            total += yb.size(0)

            all_preds.extend(preds.tolist())
            all_labels.extend(yb.tolist())

    val_acc = val_correct / total
    avg_val_loss = val_loss / len(val_loader)
    scheduler.step(avg_val_loss)

    print(f"Epoch {epoch+1:02d} | Train Acc: {train_acc:.3f} | Val Acc: {val_acc:.3f} | Val Loss: {avg_val_loss:.4f}")

    # Save best model
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(), "best_model.pt")
        print("  Saved best model")
```

```
/opt/homebrew/anaconda3/lib/python3.12/site-packages/torch/nn/modules/rnn.py:123: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.3 and num_layers=1
```

```
  warnings.warn(
Epoch 01 | Train Acc: 0.663 | Val Acc: 0.719 | Val Loss: 0.9543
 Saved best model
Epoch 02 | Train Acc: 0.743 | Val Acc: 0.765 | Val Loss: 0.8624
 Saved best model
Epoch 03 | Train Acc: 0.759 | Val Acc: 0.759 | Val Loss: 0.8664
Epoch 04 | Train Acc: 0.769 | Val Acc: 0.788 | Val Loss: 0.8109
 Saved best model
Epoch 05 | Train Acc: 0.779 | Val Acc: 0.774 | Val Loss: 0.8076
 Saved best model
Epoch 06 | Train Acc: 0.794 | Val Acc: 0.774 | Val Loss: 0.8085
Epoch 07 | Train Acc: 0.796 | Val Acc: 0.819 | Val Loss: 0.7692
 Saved best model
Epoch 08 | Train Acc: 0.798 | Val Acc: 0.828 | Val Loss: 0.7574
 Saved best model
Epoch 09 | Train Acc: 0.780 | Val Acc: 0.828 | Val Loss: 0.7530
 Saved best model
Epoch 10 | Train Acc: 0.811 | Val Acc: 0.811 | Val Loss: 0.7583
Epoch 11 | Train Acc: 0.830 | Val Acc: 0.828 | Val Loss: 0.7196
 Saved best model
Epoch 12 | Train Acc: 0.824 | Val Acc: 0.819 | Val Loss: 0.7137
 Saved best model
Epoch 13 | Train Acc: 0.821 | Val Acc: 0.828 | Val Loss: 0.7270
Epoch 14 | Train Acc: 0.822 | Val Acc: 0.848 | Val Loss: 0.7119
 Saved best model
Epoch 15 | Train Acc: 0.839 | Val Acc: 0.854 | Val Loss: 0.7099
 Saved best model
Epoch 16 | Train Acc: 0.846 | Val Acc: 0.848 | Val Loss: 0.7516
Epoch 17 | Train Acc: 0.815 | Val Acc: 0.857 | Val Loss: 0.7011
 Saved best model
Epoch 18 | Train Acc: 0.834 | Val Acc: 0.860 | Val Loss: 0.7171
Epoch 19 | Train Acc: 0.846 | Val Acc: 0.857 | Val Loss: 0.7027
Epoch 20 | Train Acc: 0.856 | Val Acc: 0.885 | Val Loss: 0.6696
 Saved best model
Epoch 21 | Train Acc: 0.862 | Val Acc: 0.842 | Val Loss: 0.6781
Epoch 22 | Train Acc: 0.848 | Val Acc: 0.880 | Val Loss: 0.6711
Epoch 23 | Train Acc: 0.872 | Val Acc: 0.888 | Val Loss: 0.6568
 Saved best model
Epoch 24 | Train Acc: 0.865 | Val Acc: 0.871 | Val Loss: 0.6635
Epoch 25 | Train Acc: 0.852 | Val Acc: 0.874 | Val Loss: 0.6613
Epoch 26 | Train Acc: 0.847 | Val Acc: 0.840 | Val Loss: 0.7332
Epoch 27 | Train Acc: 0.834 | Val Acc: 0.825 | Val Loss: 0.7129
Epoch 28 | Train Acc: 0.856 | Val Acc: 0.834 | Val Loss: 0.7193
Epoch 29 | Train Acc: 0.868 | Val Acc: 0.851 | Val Loss: 0.6972
Epoch 30 | Train Acc: 0.877 | Val Acc: 0.862 | Val Loss: 0.6719
Epoch 31 | Train Acc: 0.888 | Val Acc: 0.877 | Val Loss: 0.6539
 Saved best model
```

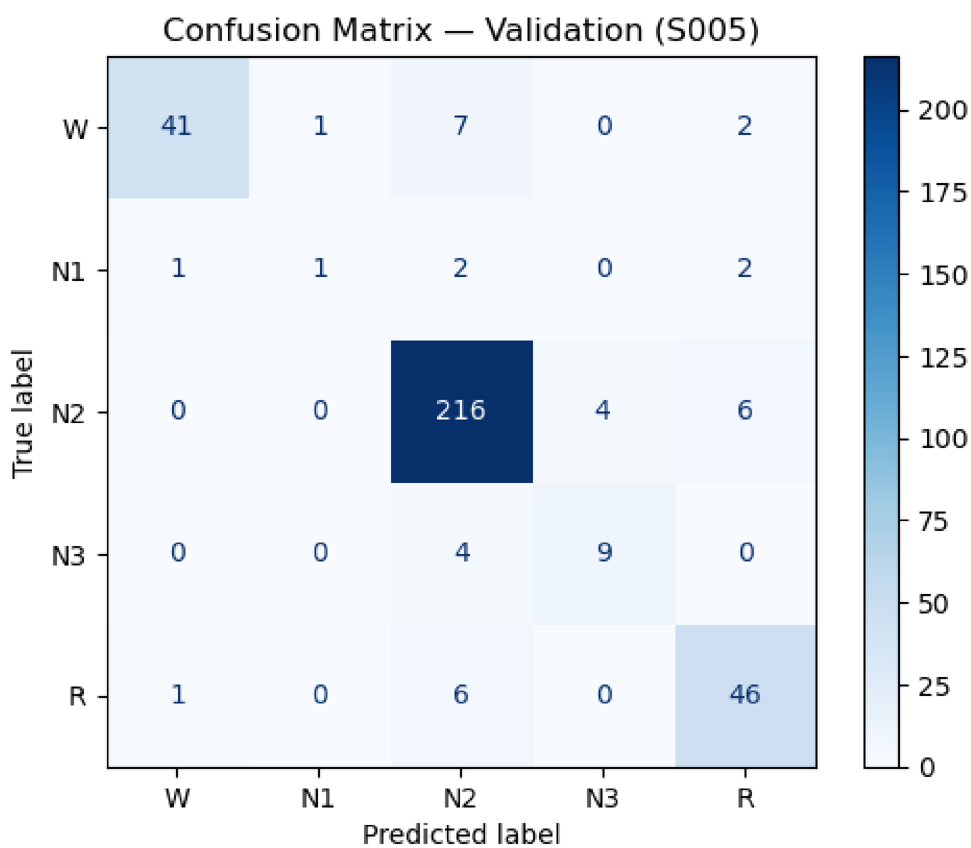
```
Epoch 32 | Train Acc: 0.874 | Val Acc: 0.874 | Val Loss: 0.6454
  Saved best model
Epoch 33 | Train Acc: 0.882 | Val Acc: 0.865 | Val Loss: 0.6431
  Saved best model
Epoch 34 | Train Acc: 0.886 | Val Acc: 0.897 | Val Loss: 0.6422
  Saved best model
Epoch 35 | Train Acc: 0.890 | Val Acc: 0.877 | Val Loss: 0.6501
Epoch 36 | Train Acc: 0.889 | Val Acc: 0.885 | Val Loss: 0.6293
  Saved best model
Epoch 37 | Train Acc: 0.890 | Val Acc: 0.880 | Val Loss: 0.6474
Epoch 38 | Train Acc: 0.892 | Val Acc: 0.883 | Val Loss: 0.6487
Epoch 39 | Train Acc: 0.895 | Val Acc: 0.883 | Val Loss: 0.6378
Epoch 40 | Train Acc: 0.903 | Val Acc: 0.885 | Val Loss: 0.6387
Epoch 41 | Train Acc: 0.900 | Val Acc: 0.877 | Val Loss: 0.6459
Epoch 42 | Train Acc: 0.892 | Val Acc: 0.888 | Val Loss: 0.6330
Epoch 43 | Train Acc: 0.889 | Val Acc: 0.905 | Val Loss: 0.6264
  Saved best model
Epoch 44 | Train Acc: 0.912 | Val Acc: 0.897 | Val Loss: 0.6346
Epoch 45 | Train Acc: 0.908 | Val Acc: 0.897 | Val Loss: 0.6363
Epoch 46 | Train Acc: 0.895 | Val Acc: 0.894 | Val Loss: 0.6445
Epoch 47 | Train Acc: 0.900 | Val Acc: 0.883 | Val Loss: 0.6364
Epoch 48 | Train Acc: 0.899 | Val Acc: 0.891 | Val Loss: 0.6284
Epoch 49 | Train Acc: 0.905 | Val Acc: 0.894 | Val Loss: 0.6303
Epoch 50 | Train Acc: 0.908 | Val Acc: 0.897 | Val Loss: 0.6254
  Saved best model
```

[76]:
```
import scipy.stats as stats
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import torch
model.eval()
correct, total = 0, 0
all_preds, all_labels = [], []

with torch.no_grad():
    for xb, yb in val_loader:
        out = model(xb)
        preds = out.argmax(1)
        all_preds.extend(preds.tolist())
        all_labels.extend(yb.tolist())
        correct += (preds == yb).sum().item()
        total += yb.size(0)

def smooth_predictions(preds, window=5):
    return np.array([
```

```python
        stats.mode(preds[max(0, i - window//2): i + window//2 + 1],␣
 ↪keepdims=True)[0][0]
        for i in range(len(preds))
    ])

smoothed_preds = smooth_predictions(np.array(all_preds), window=5)

from sklearn.metrics import classification_report
print("  Smoothed Prediction Classification Report:")
print(classification_report(all_labels, smoothed_preds, target_names=["W",␣
 ↪"N1", "N2", "N3", "R"]))

print(f"  Validation Accuracy: {correct/total:.4f}")
print(classification_report(all_labels, all_preds, target_names=["W", "N1",␣
 ↪"N2", "N3", "R"], zero_division=0))
cm = confusion_matrix(all_labels, all_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["W", "N1",␣
 ↪"N2", "N3", "R"])
disp.plot(cmap="Blues")
plt.title("Confusion Matrix - Validation (S005)")
plt.show()

from sklearn.metrics import classification_report
print(classification_report(all_labels, all_preds, zero_division=0))
torch.save(model.state_dict(), "dreamt.pt")
```

```
 Smoothed Prediction Classification Report:
              precision    recall  f1-score   support

           W       0.38      0.18      0.24        51
          N1       0.00      0.00      0.00         6
          N2       0.68      0.93      0.79       226
          N3       0.00      0.00      0.00        13
           R       0.53      0.15      0.24        53

    accuracy                           0.65       349
   macro avg       0.32      0.25      0.25       349
weighted avg       0.58      0.65      0.58       349

 Validation Accuracy: 0.8968
              precision    recall  f1-score   support

           W       0.95      0.80      0.87        51
          N1       0.50      0.17      0.25         6
          N2       0.92      0.96      0.94       226
          N3       0.69      0.69      0.69        13
           R       0.82      0.87      0.84        53
```

```
    accuracy                           0.90       349
   macro avg       0.78      0.70      0.72       349
weighted avg       0.89      0.90      0.89       349

/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



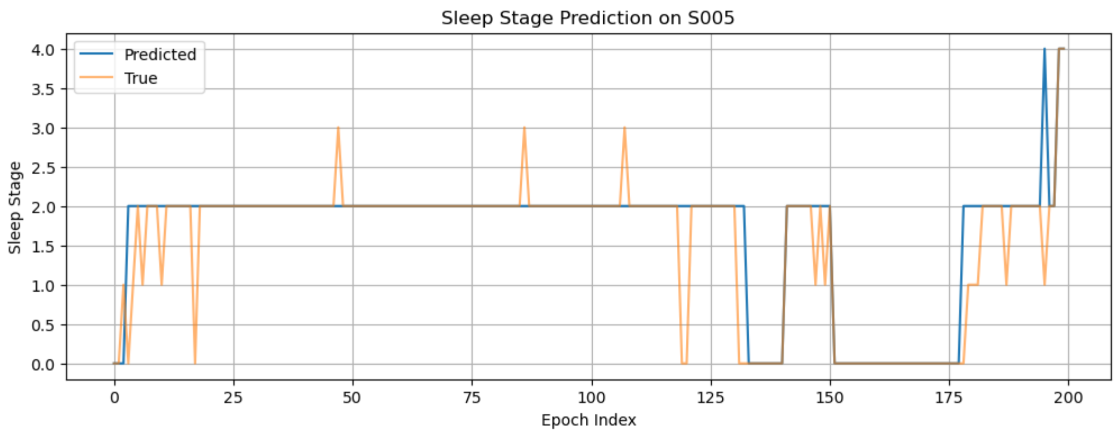
```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87        51
           1       0.50      0.17      0.25         6
           2       0.92      0.96      0.94       226
           3       0.69      0.69      0.69        13
           4       0.82      0.87      0.84        53

    accuracy                           0.90       349
   macro avg       0.78      0.70      0.72       349
weighted avg       0.89      0.90      0.89       349
```

[79]:
```python
# === 1. Load + downsample ===
df = pd.read_csv('/Users/veeralpatel/ECE284FinalProject/data/
↪S005_PSG_df_updated.csv', usecols=[
    'ACC_X', 'ACC_Y', 'ACC_Z', 'TEMP', 'EDA', 'HR', 'Sleep_Stage'
])

stage_map = {"W": 0, "N1": 1, "N2": 2, "N3": 3, "R": 4}
df['Sleep_Stage'] = df['Sleep_Stage'].map(stage_map)
df = df.dropna(subset=['Sleep_Stage'])
df['Sleep_Stage'] = df['Sleep_Stage'].astype(int)

# === 2. Normalize + downsample to 1Hz ===
features = ["ACC_X", "ACC_Y", "ACC_Z", "TEMP", "EDA", "HR"]
for col in features:
    df[col] = (df[col] - df[col].mean()) / df[col].std()

df = df.iloc[::100].reset_index(drop=True)  # from 100Hz → 1Hz
WINDOW = 90  # 3-epoch context (30s * 3)

# === 3. Contextual extraction to match training ===
X_segments, y_segments = [], []
for i in range(30, len(df) - 60, 30):
    prev = df.iloc[i - 30:i]
    curr = df.iloc[i:i + 30]
    next = df.iloc[i + 30:i + 60]

    if curr["Sleep_Stage"].nunique() == 1:
        combined = pd.concat([prev, curr, next])
        X_segments.append(combined[features].values)  # (90, 6)
        y_segments.append(curr["Sleep_Stage"].iloc[0])

X = np.stack(X_segments)
y = np.array(y_segments)
```

```python
X_tensor = torch.tensor(X, dtype=torch.float32)
y_tensor = torch.tensor(y, dtype=torch.long)


# === 5. Predict ===
with torch.no_grad():
    y_pred = model(X_tensor).argmax(dim=1)

# === 6. Evaluation ===
print("  Classification Report on S005:")
print(classification_report(y_tensor, y_pred, target_names=["W", "N1", "N2",␣
↪"N3", "R"]))

# === 7. Optional visualization ===
plt.figure(figsize=(12, 4))
plt.plot(y_pred[:200], label="Predicted")
plt.plot(y_tensor[:200], label="True", alpha=0.6)
plt.title("Sleep Stage Prediction on S005")
plt.ylabel("Sleep Stage")
plt.xlabel("Epoch Index")
plt.legend()
plt.grid(True)
plt.show()
```

```
  Classification Report on S005:
              precision    recall  f1-score   support

           W       0.90      0.71      0.79        92
          N1       0.40      0.05      0.09        39
          N2       0.88      0.98      0.93       478
          N3       0.00      0.00      0.00         5
           R       0.82      0.88      0.85       116

    accuracy                           0.87       730
   macro avg       0.60      0.52      0.53       730
weighted avg       0.84      0.87      0.85       730


/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



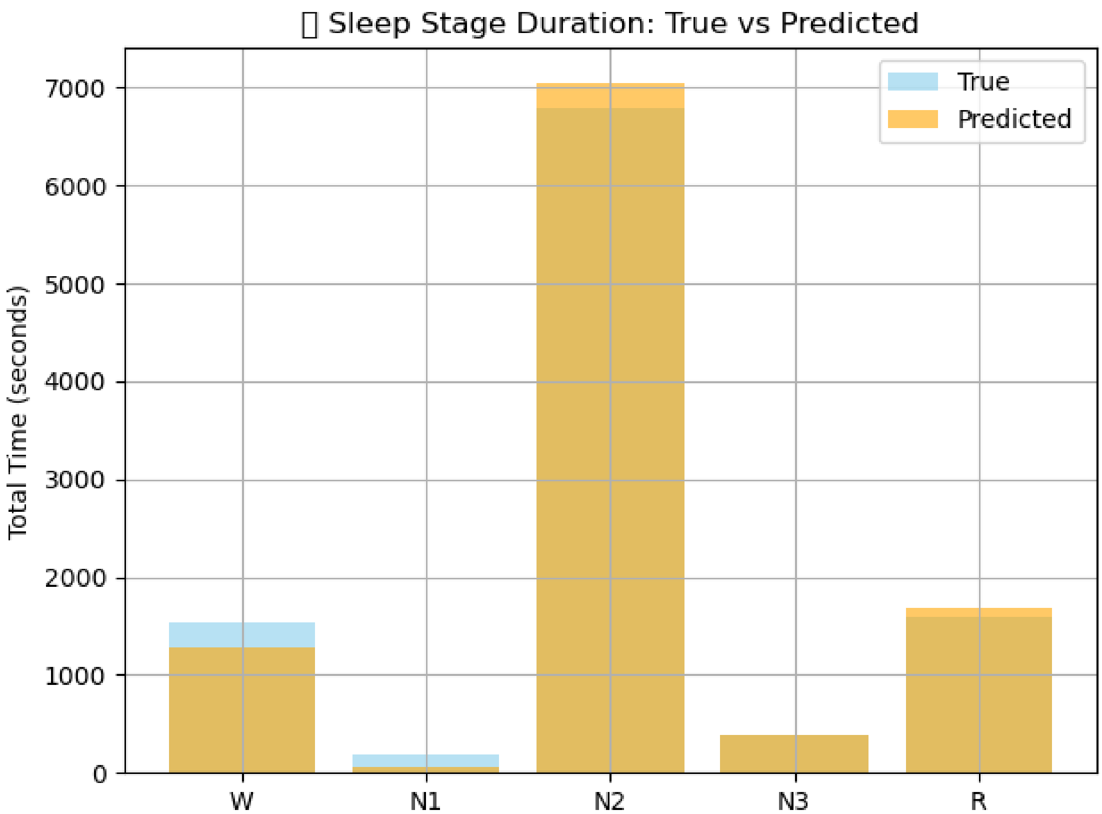
[ ]:

[80]:

```python
from collections import Counter
import matplotlib.pyplot as plt

true_counts = Counter(all_labels)
pred_counts = Counter(all_preds)

labels = ["W", "N1", "N2", "N3", "R"]
true_durations = [true_counts.get(i, 0) * 30 for i in range(5)]  # seconds
pred_durations = [pred_counts.get(i, 0) * 30 for i in range(5)]

plt.bar(labels, true_durations, alpha=0.6, label="True", color='skyblue')
plt.bar(labels, pred_durations, alpha=0.6, label="Predicted", color='orange')
plt.ylabel("Total Time (seconds)")
plt.title("  Sleep Stage Duration: True vs Predicted")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

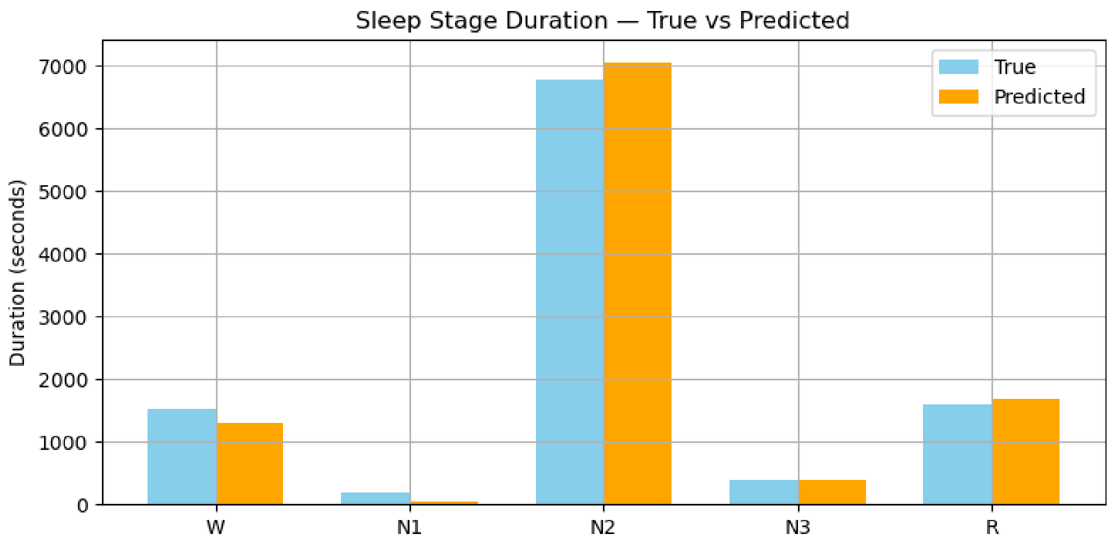
```
/var/folders/jd/lt8pv90x74741r8x6g2_d5lh0000gn/T/ipykernel_73376/1091388698.py:1
7: UserWarning: Glyph 128300 (\N{MICROSCOPE}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 128300
(\N{MICROSCOPE}) missing from font(s) DejaVu Sans.
```

```
fig.canvas.print_figure(bytes_io, **kw)
```



[81]:
```
from collections import Counter
import matplotlib.pyplot as plt

labels = ["W", "N1", "N2", "N3", "R"]
true_counts = Counter(all_labels)
pred_counts = Counter(all_preds)

true_durations = [true_counts.get(i, 0) * 30 for i in range(5)]  # in seconds
pred_durations = [pred_counts.get(i, 0) * 30 for i in range(5)]

bar_width = 0.35
x = np.arange(len(labels))

plt.figure(figsize=(8, 4))
plt.bar(x - bar_width/2, true_durations, bar_width, label='True',␣
 ↪color='skyblue')
plt.bar(x + bar_width/2, pred_durations, bar_width, label='Predicted',␣
 ↪color='orange')
plt.xticks(x, labels)
```

```
plt.ylabel("Duration (seconds)")
plt.title("Sleep Stage Duration — True vs Predicted")
plt.legend()
plt.tight_layout()
plt.grid(True)
plt.show()
```



[83]:
```
import pandas as pd
import numpy as np
from datetime import timedelta

# Load your Apple Watch sleep data
df = pd.read_csv('/Users/veeralpatel/ECE284FinalProject/Application/sleep_data.
 ↪csv')

# Parse timestamps
df['Start Time'] = pd.to_datetime(df['Start Time'])
df['End Time'] = pd.to_datetime(df['End Time'])

# Map sleep stages from Apple format to standard
stage_map = {
    'Core': 2,
    'Light': 2,
    'Deep': 3,
    'REM': 4,
    'Awake': 0
}
df = df[['Start Time', 'End Time', 'Category']]
```

```
df['Stage'] = df['Category'].map(stage_map)
df = df.dropna(subset=['Stage'])
df['Stage'] = df['Stage'].astype(int)

# Add Date column to group by sleep sessions
df['Date'] = df['Start Time'].dt.date

# Analyze last 7 nights of data
recent_dates = sorted(df['Date'].unique())[-7:]
recent_df = df[df['Date'].isin(recent_dates)]

# Function to find best wake-up time (lowest average stage in last 2 hours)
def find_best_wakeup_time(night_df):
    night_df = night_df.sort_values('Start Time')
    end_time = night_df['End Time'].max()
    two_hr_window = night_df[night_df['Start Time'] >= (end_time -␣
 ↪timedelta(hours=2))]

    # Rolling 3-segment mean
    rolling_score = two_hr_window['Stage'].rolling(window=3, center=True).mean()
    best_idx = rolling_score.idxmin()
    if pd.notna(best_idx):
        return two_hr_window.loc[best_idx]['Start Time']
    return np.nan

# Apply per night
wakeup_times = recent_df.groupby('Date').apply(find_best_wakeup_time).
 ↪reset_index()
wakeup_times.columns = ['Date', 'Recommended Wake-up Time']

# Show output
print(wakeup_times)

# Optionally: save to CSV
wakeup_times.to_csv("recommended_wakeup_times.csv", index=False)
```

```
         Date Recommended Wake-up Time
0  2023-05-08      2023-05-08 07:24:52
1  2023-05-09      2023-05-09 05:37:00
2  2023-05-10      2023-05-10 07:28:02
3  2023-05-11      2023-05-11 06:25:28
4  2023-05-12      2023-05-12 06:31:46
5  2023-05-13      2023-05-13 08:44:50
6  2023-05-14      2023-05-14 06:01:06
```

```
/var/folders/jd/lt8pv90x74741r8x6g2_d5lh0000gn/T/ipykernel_73376/3800324789.py:4
6: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
```

```
columns will be excluded from the operation. Either pass `include_groups=False`
to exclude the groupings or explicitly select the grouping columns after groupby
to silence this warning.
  wakeup_times =
recent_df.groupby('Date').apply(find_best_wakeup_time).reset_index()
```