DREAMTmultimodel78

June 7, 2025

[44]:
```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
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
import pandas as pd
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
from torch.utils.data import Dataset, DataLoader
import torch




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

[45]:
```python
cols = ['ACC_X', 'ACC_Y', 'ACC_Z', 'TEMP', 'EDA', 'HR', 'Sleep_Stage']
stage_map = {
    "W": 0,
    "N1": 1,
    "N2": 2,
    "N3": 3,
    "R": 4
}

# Load and preprocess each dataset
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)

# Concatenate all subjects into one DataFrame
full_df = pd.concat(dfs, ignore_index=True)

# Show a summary
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
```

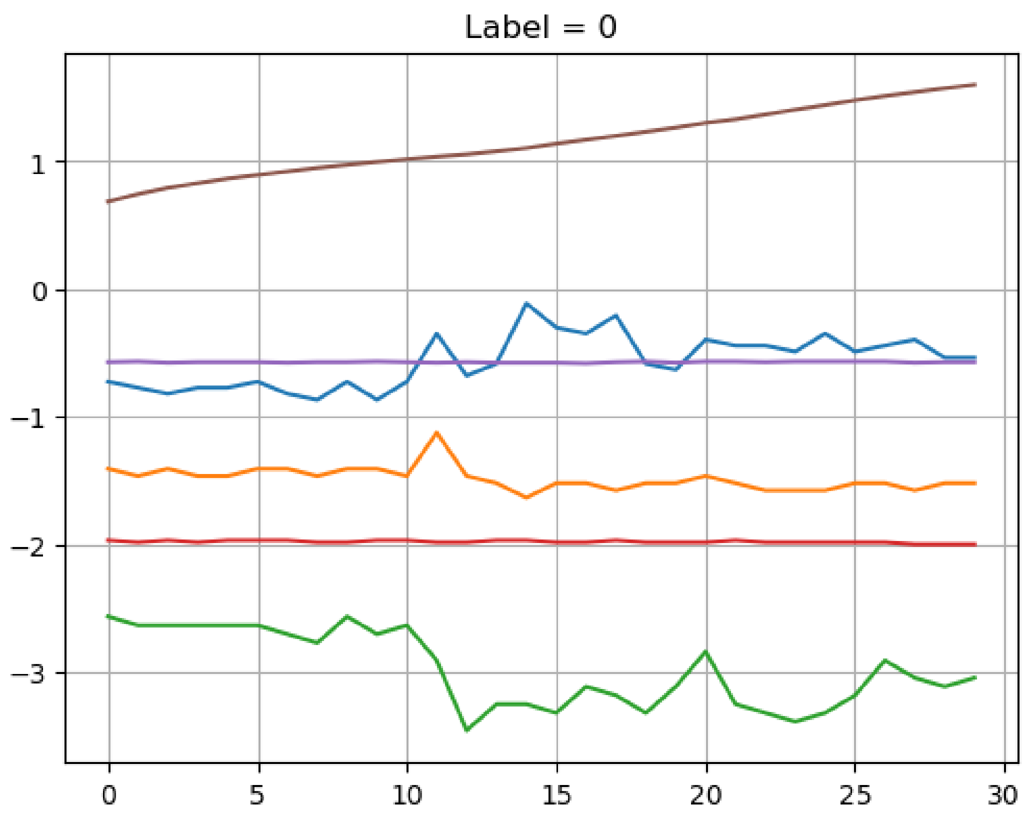
[47]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
from torch.utils.data import Dataset, DataLoader
import torch
from imblearn.over_sampling import RandomOverSampler

# === Setup ===
WINDOW = 30
channels = ['ACC_X', 'ACC_Y', 'ACC_Z', 'TEMP', 'EDA', 'HR']
stage_map = {"W": 0, "N1": 1, "N2": 2, "N3": 3, "R": 4}
subjects = ['S002', 'S005', 'S008']

def load_and_preprocess(subject):
    path = f'/Users/veeralpatel/ECE284FinalProject/data/
 ↪{subject}_PSG_df_updated.csv'
```

```python
    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_epochs(df):
    X_segments, y_segments = [], []
    for i in range(0, len(df) - WINDOW, WINDOW):
        chunk = df.iloc[i:i+WINDOW]
        if chunk['Sleep_Stage'].nunique() == 1:
            X_segments.append(chunk[channels].values)
            y_segments.append(chunk['Sleep_Stage'].iloc[0])
    return np.stack(X_segments), np.array(y_segments)

# === Load all subjects ===
X_all, y_all = [], []
for subject in subjects:
    df = load_and_preprocess(subject)
    X_seg, y_seg = extract_epochs(df)
    X_all.append(X_seg)
    y_all.append(y_seg)

X = np.concatenate(X_all)
y = np.concatenate(y_all)

# === Stratified Split ===
X_train, X_val, y_train, y_val = train_test_split(X, y, stratify=y, test_size=0.
 ↪2, random_state=42)

# Optional oversampling (recommended)
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))

# === Torch Dataset & Loaders ===
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 for loss ===
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
print("Resampled Train Class Balance:", Counter(y_train))
```

```
  Train: 4430, Val: 350
  Class weights: tensor([1., 1., 1., 1., 1.])
Unique labels: [0 1 2 3 4]
Full dataset: Counter({2: 1108, 4: 269, 0: 259, 3: 72, 1: 40})
Train: Counter({3: 886, 4: 886, 0: 886, 2: 886, 1: 886})
Val: Counter({2: 222, 4: 54, 0: 52, 3: 14, 1: 8})
X shape: (1748, 30, 6)
X_train shape: (4430, 30, 6)
X_val shape: (350, 30, 6)
Sample input: [[ 0.85829215 -0.76933329 -0.26289124 -2.34446914 -0.15419251
3.41700607]
 [ 0.85829215 -0.76933329 -0.15686438 -2.34446914 -0.13920075  3.42051502]
 [ 0.85829215 -0.76933329 -0.26289124 -2.32543885 -0.14669663  3.4249012 ]
 [ 0.85829215 -0.76933329 -0.15686438 -2.30640857 -0.14669663  3.4319191 ]
 [ 0.85829215 -0.76933329 -0.26289124 -2.32543885 -0.14669663  3.43630528]
 [ 0.85829215 -0.76933329 -0.26289124 -2.34446914 -0.13920075  3.44244594]
```

```
[ 0.85829215 -0.76933329 -0.26289124 -2.32543885 -0.13920075  3.44946383]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.13170487  3.4582362 ]
[ 0.85829215 -0.76933329 -0.26289124 -2.32543885 -0.13170487  3.46700857]
[ 0.85829215 -0.76933329 -0.26289124 -2.32543885 -0.13170487  3.47402647]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.14669663  3.4819216 ]
[ 0.85829215 -0.76933329 -0.26289124 -2.32543885 -0.13920075  3.48630778]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.13170487  3.49069397]
[ 0.85829215 -0.76933329 -0.15686438 -2.30640857 -0.13920075  3.49332568]
[ 0.85829215 -0.76933329 -0.15686438 -2.30640857 -0.14669663  3.49508015]
[ 0.85829215 -0.76933329 -0.15686438 -2.32543885 -0.14669663  3.49595739]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.13170487  3.49946633]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.13920075  3.50034357]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.13920075  3.50648423]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.13170487  3.51262489]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.13920075  3.52139726]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.13170487  3.52666068]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.11670726  3.53104686]
[ 0.85829215 -0.76933329 -0.26289124 -2.30640857 -0.12420314  3.53718752]
[ 0.85829215 -0.76933329 -0.26289124 -2.26834799 -0.12420314  3.53981923]
[ 0.85829215 -0.76933329 -0.15686438 -2.26834799 -0.13170487  3.53981923]
[ 0.80184215 -0.76933329 -0.26289124 -2.30640857 -0.13170487  3.53894199]
[ 0.85829215 -0.76933329 -0.26289124 -2.28737828 -0.11670726  3.53543305]
[ 0.85829215 -0.76933329 -0.26289124 -2.28737828 -0.13170487  3.53016962]
[ 0.85829215 -0.76933329 -0.26289124 -2.26834799 -0.12420314  3.52578344]]
```



```
Resampled Train Class Balance: Counter({3: 886, 4: 886, 0: 886, 2: 886, 1: 886})
```

[48]:
```python
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)


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)
```

[63]:
```
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")
```

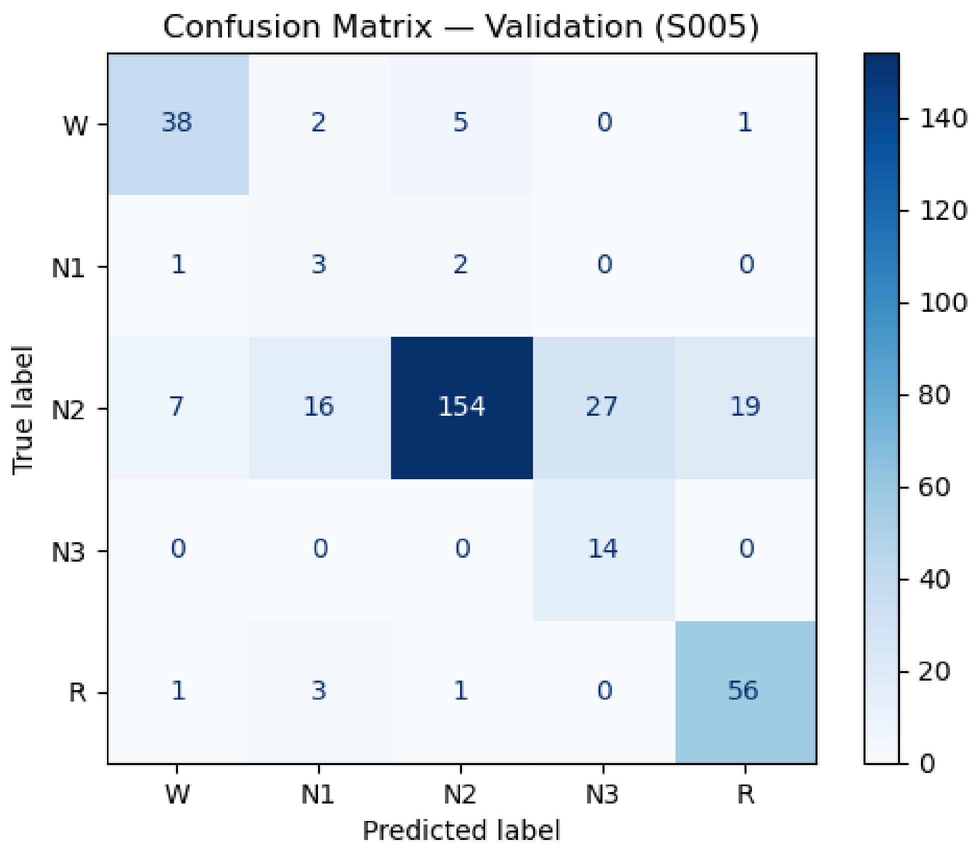
```
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/torch/nn/modules/rnn.py:123: UserWarning: dropout option adds dropout
after all but last recurrent layer, so non-zero dropout expects num_layers
greater than 1, but got dropout=0.3 and num_layers=1
  warnings.warn(
```

```
Epoch 01 | Train Acc: 0.220 | Val Acc: 0.251 | Val Loss: 1.7045
  Saved best model
Epoch 02 | Train Acc: 0.292 | Val Acc: 0.289 | Val Loss: 1.6610
  Saved best model
Epoch 03 | Train Acc: 0.312 | Val Acc: 0.423 | Val Loss: 1.5771
  Saved best model
Epoch 04 | Train Acc: 0.385 | Val Acc: 0.443 | Val Loss: 1.5501
  Saved best model
Epoch 05 | Train Acc: 0.410 | Val Acc: 0.457 | Val Loss: 1.5399
  Saved best model
Epoch 06 | Train Acc: 0.443 | Val Acc: 0.489 | Val Loss: 1.5350
  Saved best model
Epoch 07 | Train Acc: 0.438 | Val Acc: 0.480 | Val Loss: 1.5250
  Saved best model
Epoch 08 | Train Acc: 0.445 | Val Acc: 0.557 | Val Loss: 1.4852
  Saved best model
Epoch 09 | Train Acc: 0.496 | Val Acc: 0.554 | Val Loss: 1.4901
Epoch 10 | Train Acc: 0.502 | Val Acc: 0.577 | Val Loss: 1.4864
Epoch 11 | Train Acc: 0.514 | Val Acc: 0.563 | Val Loss: 1.5227
Epoch 12 | Train Acc: 0.519 | Val Acc: 0.583 | Val Loss: 1.4611
  Saved best model
Epoch 13 | Train Acc: 0.529 | Val Acc: 0.663 | Val Loss: 1.4196
```

```
  Saved best model
Epoch 14 | Train Acc: 0.544 | Val Acc: 0.629 | Val Loss: 1.4822
Epoch 15 | Train Acc: 0.562 | Val Acc: 0.629 | Val Loss: 1.4553
Epoch 16 | Train Acc: 0.555 | Val Acc: 0.660 | Val Loss: 1.4549
Epoch 17 | Train Acc: 0.566 | Val Acc: 0.660 | Val Loss: 1.4242
Epoch 18 | Train Acc: 0.589 | Val Acc: 0.657 | Val Loss: 1.4747
Epoch 19 | Train Acc: 0.591 | Val Acc: 0.637 | Val Loss: 1.4628
Epoch 20 | Train Acc: 0.612 | Val Acc: 0.689 | Val Loss: 1.4273
Epoch 21 | Train Acc: 0.597 | Val Acc: 0.643 | Val Loss: 1.4070
  Saved best model
Epoch 22 | Train Acc: 0.614 | Val Acc: 0.649 | Val Loss: 1.4095
Epoch 23 | Train Acc: 0.618 | Val Acc: 0.694 | Val Loss: 1.4109
Epoch 24 | Train Acc: 0.633 | Val Acc: 0.629 | Val Loss: 1.4523
Epoch 25 | Train Acc: 0.625 | Val Acc: 0.671 | Val Loss: 1.4299
Epoch 26 | Train Acc: 0.636 | Val Acc: 0.686 | Val Loss: 1.4088
Epoch 27 | Train Acc: 0.665 | Val Acc: 0.691 | Val Loss: 1.4039
  Saved best model
Epoch 28 | Train Acc: 0.666 | Val Acc: 0.689 | Val Loss: 1.3961
  Saved best model
Epoch 29 | Train Acc: 0.632 | Val Acc: 0.689 | Val Loss: 1.3902
  Saved best model
Epoch 30 | Train Acc: 0.682 | Val Acc: 0.714 | Val Loss: 1.3967
Epoch 31 | Train Acc: 0.691 | Val Acc: 0.723 | Val Loss: 1.3753
  Saved best model
Epoch 32 | Train Acc: 0.682 | Val Acc: 0.700 | Val Loss: 1.3961
Epoch 33 | Train Acc: 0.695 | Val Acc: 0.729 | Val Loss: 1.3681
  Saved best model
Epoch 34 | Train Acc: 0.693 | Val Acc: 0.717 | Val Loss: 1.3833
Epoch 35 | Train Acc: 0.660 | Val Acc: 0.697 | Val Loss: 1.3854
Epoch 36 | Train Acc: 0.688 | Val Acc: 0.717 | Val Loss: 1.3805
Epoch 37 | Train Acc: 0.689 | Val Acc: 0.749 | Val Loss: 1.3585
  Saved best model
Epoch 38 | Train Acc: 0.675 | Val Acc: 0.706 | Val Loss: 1.3810
Epoch 39 | Train Acc: 0.695 | Val Acc: 0.729 | Val Loss: 1.3894
Epoch 40 | Train Acc: 0.704 | Val Acc: 0.709 | Val Loss: 1.3925
Epoch 41 | Train Acc: 0.707 | Val Acc: 0.743 | Val Loss: 1.3795
Epoch 42 | Train Acc: 0.725 | Val Acc: 0.754 | Val Loss: 1.3757
Epoch 43 | Train Acc: 0.705 | Val Acc: 0.749 | Val Loss: 1.3702
Epoch 44 | Train Acc: 0.716 | Val Acc: 0.751 | Val Loss: 1.3711
Epoch 45 | Train Acc: 0.715 | Val Acc: 0.757 | Val Loss: 1.3702
Epoch 46 | Train Acc: 0.739 | Val Acc: 0.751 | Val Loss: 1.3640
Epoch 47 | Train Acc: 0.705 | Val Acc: 0.749 | Val Loss: 1.3652
Epoch 48 | Train Acc: 0.721 | Val Acc: 0.757 | Val Loss: 1.3729
Epoch 49 | Train Acc: 0.720 | Val Acc: 0.757 | Val Loss: 1.3509
  Saved best model
Epoch 50 | Train Acc: 0.728 | Val Acc: 0.757 | Val Loss: 1.3527
```

[64]:
```
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)

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")
```

```
 Validation Accuracy: 0.7571
              precision    recall  f1-score   support

           W       0.81      0.83      0.82        46
          N1       0.12      0.50      0.20         6
          N2       0.95      0.69      0.80       223
          N3       0.34      1.00      0.51        14
           R       0.74      0.92      0.82        61

    accuracy                           0.76       350
   macro avg       0.59      0.79      0.63       350
weighted avg       0.86      0.76      0.78       350
```



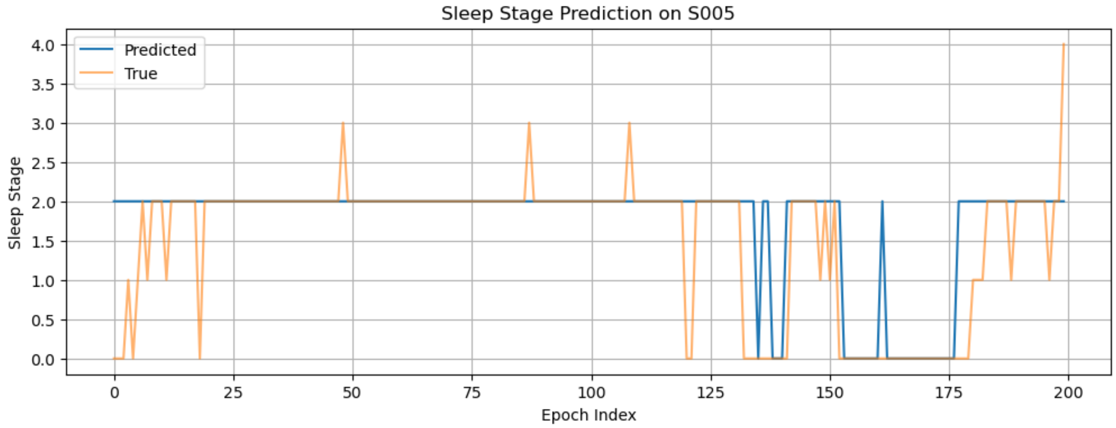

```
              precision    recall  f1-score   support

           0       0.81      0.83      0.82        46
           1       0.12      0.50      0.20         6
           2       0.95      0.69      0.80       223
           3       0.34      1.00      0.51        14
           4       0.74      0.92      0.82        61

    accuracy                           0.76       350
   macro avg       0.59      0.79      0.63       350
weighted avg       0.86      0.76      0.78       350
```

[65]:
```
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 = df.copy()
df['Sleep_Stage'] = df['Sleep_Stage'].map(stage_map)
df = df.dropna(subset=['Sleep_Stage'])
df['Sleep_Stage'] = df['Sleep_Stage'].astype(int)

# === 2. Extract 30s epochs ===
WINDOW = 3000  # assuming 100Hz * 30s
features = ["ACC_X", "ACC_Y", "ACC_Z", "TEMP", "EDA", "HR"]

X_segments, y_segments = [], []
print("Unique labels in Sleep_Stage:", df["Sleep_Stage"].unique())
print("DataFrame length:", len(df))

for i in range(0, len(df) - WINDOW, WINDOW):
    chunk = df.iloc[i:i + WINDOW]
    mode_label = chunk["Sleep_Stage"].mode().iloc[0]
    feat = chunk[features].values
    X_segments.append(feat)
    y_segments.append(mode_label)

X_s005 = np.stack(X_segments)
y_s005 = np.array(y_segments)

X_s005_tensor = torch.tensor(X_s005, dtype=torch.float32)
y_s005_tensor = torch.tensor(y_s005, dtype=torch.long)

# === 3. Reshape: (batch, time, features) ===
X_s005_tensor = X_s005_tensor.permute(0, 1, 2)  # already (N, T, C)


# === 5. Predict ===
with torch.no_grad():
    y_pred = model(X_s005_tensor).argmax(dim=1)

# === 6. Evaluation ===
print("  Classification Report on S005:")
print(classification_report(y_s005_tensor, y_pred, target_names=["W", "N1",␣
 ↪"N2", "N3", "R"]))

# === 7. Optional visualization ===
plt.figure(figsize=(12, 4))
```

```python
plt.plot(y_pred[:200], label="Predicted")
plt.plot(y_s005_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()
```

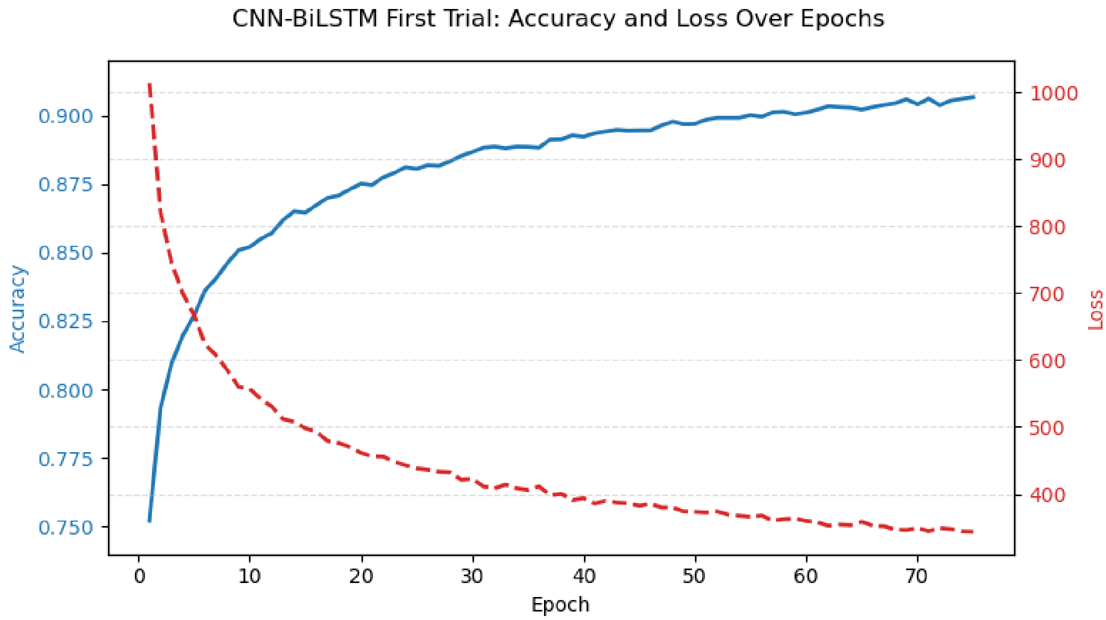
```
Unique labels in Sleep_Stage: [0 1 2 3 4]
DataFrame length: 2198997
 Classification Report on S005:
              precision    recall  f1-score   support

           W       0.19      0.71      0.30        93
          N1       0.00      0.00      0.00        39
          N2       0.75      0.61      0.67       479
          N3       0.00      0.00      0.00         5
           R       0.00      0.00      0.00       116

    accuracy                           0.49       732
   macro avg       0.19      0.26      0.19       732
weighted avg       0.51      0.49      0.48       732


/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))
```



[ ]:

[67]:
```
epochs = list(range(1, 76))
train_acc = [
    0.7522, 0.7934, 0.8097, 0.8197, 0.8269, 0.8362, 0.8406, 0.8461, 0.8508, 0.
  ↪8520,
    0.8549, 0.8571, 0.8618, 0.8650, 0.8645, 0.8673, 0.8699, 0.8708, 0.8730, 0.
  ↪8751,
    0.8746, 0.8774, 0.8790, 0.8811, 0.8805, 0.8818, 0.8816, 0.8832, 0.8852, 0.
  ↪8866,
    0.8882, 0.8886, 0.8880, 0.8886, 0.8885, 0.8882, 0.8912, 0.8913, 0.8927, 0.
  ↪8922,
    0.8935, 0.8941, 0.8947, 0.8944, 0.8945, 0.8945, 0.8963, 0.8977, 0.8968, 0.
  ↪8969,
    0.8984, 0.8991, 0.8991, 0.8991, 0.9001, 0.8995, 0.9011, 0.9013, 0.9004, 0.
  ↪9010,
    0.9021, 0.9033, 0.9030, 0.9028, 0.9021, 0.9030, 0.9038, 0.9044, 0.9059, 0.
  ↪9040,
    0.9061, 0.9037, 0.9054, 0.9060, 0.9066
]

train_loss = [
    1013.8506, 820.5557, 744.4093, 699.4798, 667.8167, 623.4382, 606.9489, 584.
  ↪8319, 560.1630, 556.9440,
    542.3478, 530.5164, 511.8450, 507.9238, 498.1925, 492.9588, 479.3014, 476.
  ↪2592, 470.2599, 461.4718,
    456.2912, 455.7859, 448.1630, 443.0459, 438.3809, 436.3698, 433.4494, 432.
  ↪5599, 421.3876, 422.7147,
    411.3866, 408.6607, 413.8441, 408.6330, 406.1438, 411.4846, 398.3122, 400.
  ↪2597, 391.2328, 393.9160,
```

```
    385.8131, 390.0074, 387.2803, 386.3065, 382.4812, 385.3838, 380.0021, 379.
↪8832, 374.1201, 373.5412,
    372.4660, 374.0214, 369.1882, 367.5717, 365.8802, 367.8725, 360.3513, 362.
↪4597, 363.2964, 359.8788,
    357.7483, 352.6868, 354.4657, 353.6179, 358.7299, 352.6774, 351.9976, 347.
↪0274, 346.2135, 349.1489,
    344.6163, 349.2698, 347.6872, 344.4418, 343.8008
]

fig, ax1 = plt.subplots(figsize=(8, 4.5))

color1 = 'tab:blue'
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy', color=color1)
ax1.plot(epochs, train_acc, color=color1, linewidth=2, label='Train Accuracy')
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0.74, 0.92)

ax2 = ax1.twinx()
color2 = 'tab:red'
ax2.set_ylabel('Loss', color=color2)
ax2.plot(epochs, train_loss, color=color2, linewidth=2, linestyle='--',␣
↪label='Train Loss')
ax2.tick_params(axis='y', labelcolor=color2)

fig.suptitle('CNN-BiLSTM First Trial: Accuracy and Loss Over Epochs')
fig.tight_layout()
plt.grid(True, linestyle='--', alpha=0.4)
plt.savefig('cnn_bilstm_dual_accuracy_loss.pdf')
plt.show()
```


CNN-BiLSTM First Trial: Accuracy and Loss Over Epochs
Accuracy
0.750
0.775
0.800
0.825
0.850
0.875
0.900
Loss
400
500
600
700
800
900
1000
Epoch
0
10
20
30
40
50
60
70