DREAMTmultimodel

June 7, 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
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



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'
```

[27]:
```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
```

[ ]:
```
df = df.iloc[::100].reset_index(drop=True)  # if downsampling from 100Hz
WINDOW = 30  # 1 Hz sampling = 30 seconds
X_segments, y_segments = [], []
channels = ["ACC_X", "ACC_Y", "ACC_Z", "TEMP", "EDA", "HR"]

for i in range(0, len(df) - WINDOW, WINDOW):
    chunk = df.iloc[i:i+WINDOW]
    if chunk["Sleep_Stage"].nunique() == 1:  # single label per epoch
        X_segments.append(chunk[channels].values)
        y_segments.append(chunk["Sleep_Stage"].iloc[0])

X = np.stack(X_segments)
y = np.array(y_segments)
print(f"Total epochs: {len(X)} - Shape: {X.shape}")
```

```
Total epochs: 74694 - Shape: (74694, 30, 6)
```

[ ]:
```
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)
```

```python
    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)
```

[ ]:
```python
weights = compute_class_weight(class_weight='balanced', classes=np.unique(y),␣
 ↪y=y)
class_weights = torch.tensor(weights, dtype=torch.float32)
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)
```

```python
EPOCHS = 50
best_loss = float('inf')

for epoch in range(EPOCHS):
    model.train()
    total_loss, 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()

        total_loss += loss.item()
        correct += (out.argmax(1) == yb).sum().item()
        total += yb.size(0)

    acc = correct / total
    print(f"Epoch {epoch+1}: Loss = {total_loss:.4f}, Train Acc = {acc:.4f}")
```

```
Epoch 1: Loss = 1967.2208, Train Acc = 0.4695
Epoch 2: Loss = 1590.8708, Train Acc = 0.5465
Epoch 3: Loss = 1426.5228, Train Acc = 0.5830
Epoch 4: Loss = 1323.0141, Train Acc = 0.6263
Epoch 5: Loss = 1279.9580, Train Acc = 0.6416
Epoch 6: Loss = 1219.4843, Train Acc = 0.6562
Epoch 7: Loss = 1182.1724, Train Acc = 0.6670
Epoch 8: Loss = 1156.8686, Train Acc = 0.6756
Epoch 9: Loss = 1103.3902, Train Acc = 0.6920
Epoch 10: Loss = 1092.3170, Train Acc = 0.6975
Epoch 11: Loss = 1051.3358, Train Acc = 0.7096
Epoch 12: Loss = 1034.9219, Train Acc = 0.7149
Epoch 13: Loss = 1021.7000, Train Acc = 0.7199
Epoch 14: Loss = 994.8490, Train Acc = 0.7309
Epoch 15: Loss = 972.1631, Train Acc = 0.7353
Epoch 16: Loss = 963.5146, Train Acc = 0.7359
Epoch 17: Loss = 959.4543, Train Acc = 0.7426
Epoch 18: Loss = 951.3234, Train Acc = 0.7360
Epoch 19: Loss = 931.9283, Train Acc = 0.7439
Epoch 20: Loss = 915.5760, Train Acc = 0.7491
Epoch 21: Loss = 915.4083, Train Acc = 0.7507
Epoch 22: Loss = 903.2461, Train Acc = 0.7514
Epoch 23: Loss = 886.5014, Train Acc = 0.7573
Epoch 24: Loss = 889.9289, Train Acc = 0.7545
Epoch 25: Loss = 880.8766, Train Acc = 0.7586
Epoch 26: Loss = 878.8069, Train Acc = 0.7579
Epoch 27: Loss = 860.3005, Train Acc = 0.7630
```

```
Epoch 28: Loss = 864.4827, Train Acc = 0.7600
Epoch 29: Loss = 855.0585, Train Acc = 0.7616
Epoch 30: Loss = 847.2581, Train Acc = 0.7647
Epoch 31: Loss = 845.9448, Train Acc = 0.7668
Epoch 32: Loss = 849.1629, Train Acc = 0.7681
Epoch 33: Loss = 840.4363, Train Acc = 0.7636
Epoch 34: Loss = 830.7756, Train Acc = 0.7721
Epoch 35: Loss = 833.1993, Train Acc = 0.7730
Epoch 36: Loss = 846.6492, Train Acc = 0.7649
Epoch 37: Loss = 820.7876, Train Acc = 0.7740
Epoch 38: Loss = 824.3695, Train Acc = 0.7727
Epoch 39: Loss = 825.2940, Train Acc = 0.7722
Epoch 40: Loss = 806.6458, Train Acc = 0.7788
Epoch 41: Loss = 805.6579, Train Acc = 0.7768
Epoch 42: Loss = 805.7576, Train Acc = 0.7770
Epoch 43: Loss = 816.6465, Train Acc = 0.7755
Epoch 44: Loss = 798.3226, Train Acc = 0.7805
Epoch 45: Loss = 795.2177, Train Acc = 0.7780
Epoch 46: Loss = 794.1573, Train Acc = 0.7835
Epoch 47: Loss = 788.8274, Train Acc = 0.7819
Epoch 48: Loss = 786.7594, Train Acc = 0.7808
Epoch 49: Loss = 791.7947, Train Acc = 0.7823
Epoch 50: Loss = 793.1518, Train Acc = 0.7804
```

[31]:

```python
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}")

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: 0.7205
              precision    recall  f1-score   support

           0       0.64      0.73      0.68      1136
           1       0.37      0.83      0.51       848
```

```
           2       0.99      0.63      0.77      8496
           3       0.50      0.87      0.63      1777
           4       0.72      0.87      0.79      2682

    accuracy                           0.72     14939
   macro avg       0.64      0.79      0.68     14939
weighted avg       0.82      0.72      0.74     14939
```

[32]:

```
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)

# === 4. Load model ===
model.load_state_dict(torch.load("dreamt.pt"))
model.eval()

# === 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))
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()
```

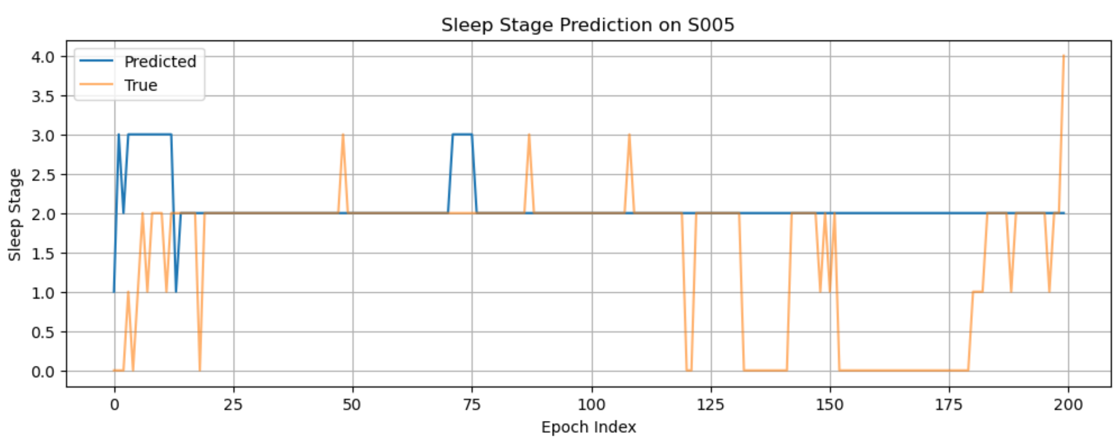
```
Unique labels in Sleep_Stage: [0 1 2 3 4]
DataFrame length: 2198997
```

```
/var/folders/jd/1t8pv90x74741r8x6g2_d5lh0000gn/T/ipykernel_73376/4026704073.py:4
3: FutureWarning: You are using `torch.load` with `weights_only=False` (the
current default value), which uses the default pickle module implicitly. It is
possible to construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  model.load_state_dict(torch.load("dreamt.pt"))
```

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

```
           W       0.00      0.00      0.00        93
          N1       0.00      0.00      0.00        39
          N2       0.66      0.98      0.78       479
          N3       0.00      0.00      0.00         5
           R       0.00      0.00      0.00       116

    accuracy                           0.64       732
   macro avg       0.13      0.20      0.16       732
weighted avg       0.43      0.64      0.51       732
```

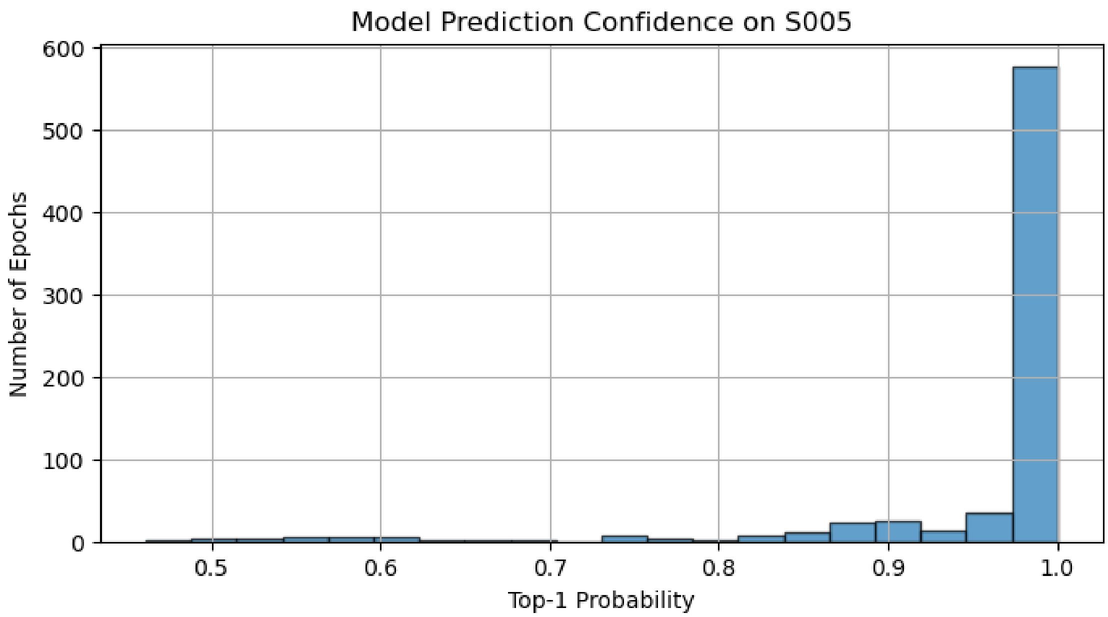
```
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()}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



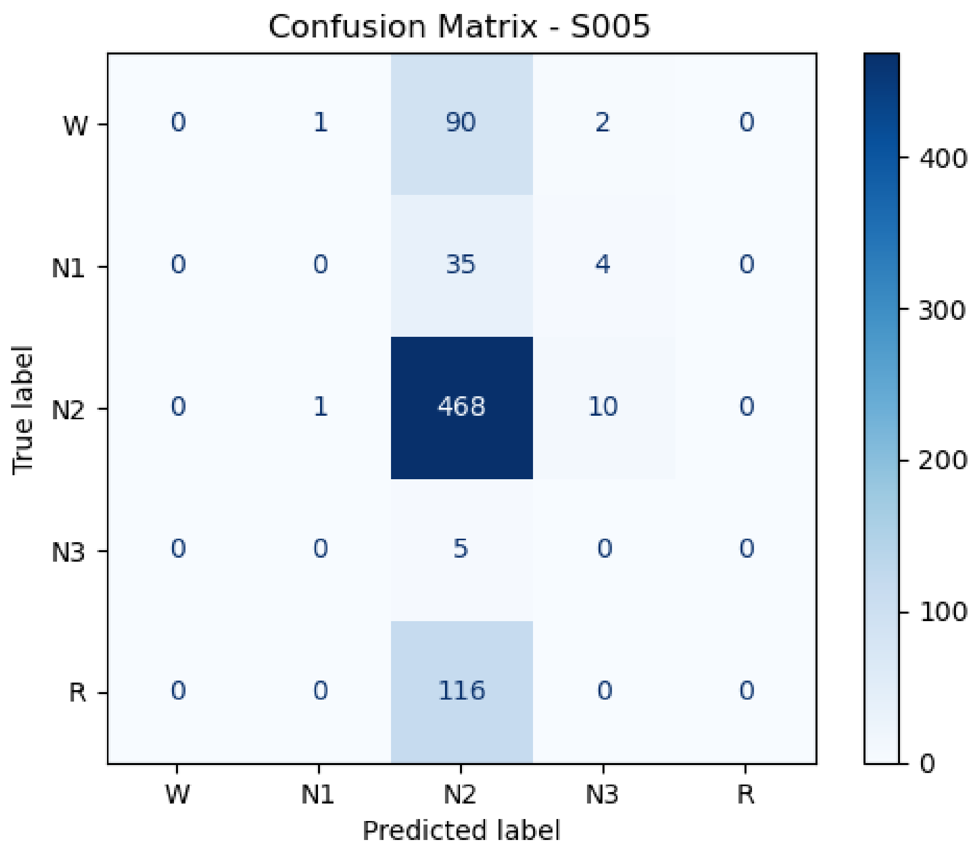
[33]:
```python
with torch.no_grad():
    logits = model(X_s005_tensor)
    probs = torch.softmax(logits, dim=1)
    preds = probs.argmax(dim=1)
```

```
# === 2. Show sample predictions and confidence ===
top_probs = probs.max(dim=1).values
print("  Sample predicted stage labels:", preds[:10].tolist())
print("  Sample confidences:", top_probs[:10].tolist())
print("  Avg confidence across all predictions:", round(top_probs.mean().
 ↪item(), 3))

# === 3. Plot histogram of predicted probabilities ===
plt.figure(figsize=(8, 4))
plt.hist(top_probs.numpy(), bins=20, alpha=0.7, edgecolor='black')
plt.title("Model Prediction Confidence on S005")
plt.xlabel("Top-1 Probability")
plt.ylabel("Number of Epochs")
plt.grid(True)
plt.show()

# === 4. Breakdown of predictions by class ===
unique, counts = np.unique(preds.numpy(), return_counts=True)
print("  Prediction distribution (stage: count):")
for u, c in zip(unique, counts):
    print(f"  Stage {u}: {c} predictions")

# === 5. Optional: Show confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_s005_tensor, preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["W", "N1",␣
 ↪"N2", "N3", "R"])
disp.plot(cmap="Blues")
plt.title("Confusion Matrix - S005")
plt.show()
```

```
 Sample predicted stage labels: [1, 3, 2, 3, 3, 3, 3, 3, 3, 3]
 Sample confidences: [0.5538374781608582, 0.6889937520027161,
0.7037439346313477, 0.6533932089805603, 0.5842940211296082, 0.5429820418357849,
0.5462482571601868, 0.5832931399345398, 0.5971276760101318, 0.6084995865821838]
 Avg confidence across all predictions: 0.96
```

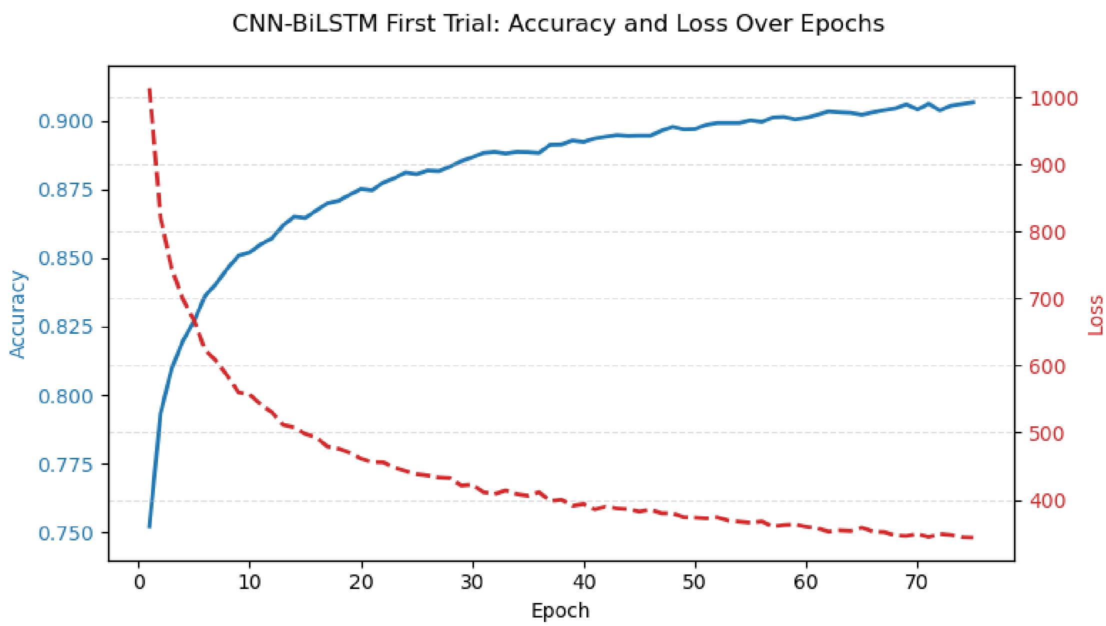



```
Prediction distribution (stage: count):
 Stage 1: 2 predictions
 Stage 2: 714 predictions
 Stage 3: 16 predictions
```



[34]:
```
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
0.900
0.875
0.850
0.825
0.800
0.775
0.750
Loss
1000
900
800
700
600
500
400
Epoch
0
10
20
30
40
50
60
70