Affective Computing for Emotion Recognition Using EEG Signals

Muthiah Sivavelan (2021IMG-034)

Supervisor: Dr. Anjali

ABV-Indian Institute of Information Technology and Management Gwalior

विश्वजीवनामृतं ज्ञानम्

Outline

Problem Statement

- Accurate emotion recognition is crucial for enhancing human-computer interaction, mental health monitoring, and personalized user experiences.
- EEG signals provide detailed insights into brain activity, offering a non-invasive and effective way to capture emotional states.
- My approach compares manual feature extraction through various decompositions (DWT and EMD) with deep learning models that automatically detect important features, analyzing multiple methods for feature extraction and classification to enhance emotion recognition accuracy. Main Reference Paper: [1]

Objective

- Analyze and compare the effectiveness of Discrete Wavelet Transform (DWT) and Empirical Mode Decomposition (EMD) for extracting features from EEG signals.
- Assess the performance of statistical features extracted from decomposition techniques using traditional machine learning models like XGBoost and Random Forest.
- Utilize Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to classify emotions from EEG signals and evaluate their performance.
- Aim to improve the accuracy of emotion classification by integrating and comparing various feature extraction and deep learning methodologies.

Literature Review

- Cimtay *et al.* used a convolutional neural network (CNN) to extract the spatial features of emotional activities, and accuracies of 86.56% and 72.81% were achieved on the SEED and DEAP datasets, respectively [2].
- Wang et al. (2021) utilized Empirical Mode Decomposition (EMD) to decompose EEG signals into Intrinsic Mode Functions (IMFs) for emotion classification [3].
- Yang et al. (2022) introduced a hybrid neural network combining CNN and RNN to learn both spatial and temporal features from EEG signals, achieving 90.80% and 91.03% for valence and arousal emotion classification. [4].

Solution Approach

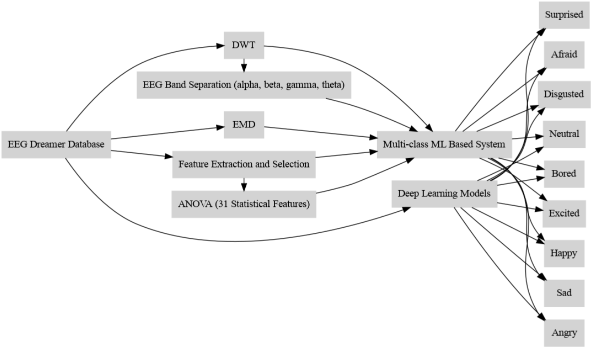

Figure 1: Project Workflow

Solution Approach

- Data Collection and Preparation: Utilize the DREAMER dataset, which includes EEG and ECG signals along with emotion labels. Preprocess the EEG signals by cleaning, normalizing, and splitting the data into training, validation, and test sets.
- Apply Discrete Wavelet Transform (DWT) to decompose EEG signals into 4 frequency bands.
- Use Empirical Mode Decomposition (EMD) to extract Intrinsic Mode Functions (7 IMFs for each of the 14 electrodes signals).
- Compute 31 statistical features from the each of the EEG electrode data.
- Implement machine learning models such as XGBoost and Random Forest, using features extracted from DWT, EMD, and statistical analysis combined and separately and fine-tune each models.

Solution Approach

- Employ deep learning models, specifically Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to classify emotions directly from raw EEG signals and fine tune them.
- Evaluating the accuracy for classification of the 9 emotions for each of the models.
- Analyze the results to determine the strengths and weaknesses of each method.
- Document the findings, methodologies, and performance results. Suggest potential improvements and explore future directions, such as integrating advanced techniques or other deep learning models.

Results and Inferences

```
    for fold, (train_idx, val_idx) in enumerate(kf.split(dataset)):
        print(f'Fold {fold+1}/{k_folds}')

        val_subset = Subset(dataset, val_idx)
        val_loader = DataLoader(val_subset, batch_size=64, shuffle=False)

        model.eval()

        correct_predictions = 0
        total_predictions = 0
        with torch.no_grad():
            for batch_x, batch_y in val_loader:
                batch_x, batch_y = batch_x.to(device), batch_y.to(device)
                outputs = model(batch_x)
                predicted_classes = torch.argmax(outputs, dim=1)
                correct_predictions += (predicted_classes == batch_y).sum().item()
                total_predictions += batch_x.size(0)

        accuracy = correct_predictions / total_predictions * 100
        val_accuracies.append(accuracy)

    mean_accuracy = np.mean(val_accuracies)
    print(f'Final Model Mean Validation Accuracy: {mean_accuracy:.2f}%')

evaluate_final_model()
```

```
Fold 1/5
Fold 2/5
Fold 3/5
Fold 4/5
Fold 5/5
Final Model Mean Validation Accuracy: 90.58%
```

Figure 2: CNN Accuracy with 5-fold cross validation

Results and Inferences

```
Fold 1/5
Fold 2/5
Fold 3/5
Fold 4/5
Fold 5/5
Final Model Mean Validation Accuracy for Class amusement: 96.79%
Final Model Mean Validation Accuracy for Class anger: 92.14%
Final Model Mean Validation Accuracy for Class calmness: 88.33%
Final Model Mean Validation Accuracy for Class disgust: 90.91%
Final Model Mean Validation Accuracy for Class excitement: 95.00%
Final Model Mean Validation Accuracy for Class fear: 89.17%
Final Model Mean Validation Accuracy for Class happiness: 90.73%
Final Model Mean Validation Accuracy for Class sadness: 90.95%
Final Model Mean Validation Accuracy for Class surprise: 91.46%
```

Figure 3: Class-wise accuracy in detecting emotions using CNN model

Results and Inferences

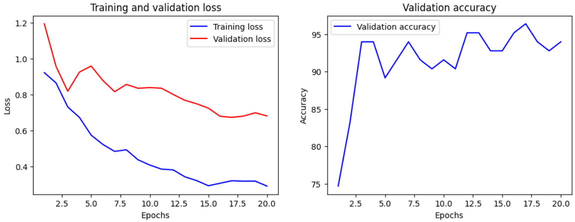

Figure 4: CNN model training curve

Results and Inferences

```
# Evaluating on random part of dataset just to check
```

```
import torch
from torch.utils.data import DataLoader, random_split
import numpy as np

model.eval()  # Set the model to evaluation mode

# Assuming standardized_data and y_one_hot are already defined and converted to tensors
standardized_data = torch.tensor(standardized_data, dtype=torch.float32).view(414, 14, 7680)
# y_one_hot = torch.tensor(y_one_hot, dtype=torch.float32)
# y_indices = torch.argmax(y_one_hot, dim=1)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Create a dataset
dataset = TensorDataset(standardized_data, y_indices)

# Split the dataset into training and test sets (e.g., 80% train, 20% test)
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Create a DataLoader for the test dataset
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=True)

# Evaluate the model on the test dataset
correct_predictions = 0
total_predictions = 0
with torch.no_grad():
    for batch_x, batch_y in test_loader:
        batch_x, batch_y = batch_x.to(device), batch_y.to(device)
        outputs = model(batch_x)
        predicted_classes = torch.argmax(outputs, dim=1)
        correct_predictions += (predicted_classes == batch_y).sum().item()
        total_predictions += batch_x.size(0)

test_accuracy = correct_predictions / total_predictions * 100
print(f"Accuracy on test dataset: {test_accuracy:.2f}%")
```

```
Accuracy on test dataset: 93.98%
```

Results and Inferences

```
Epoch [7/10], Training Loss: 1.4825
Accuracy on training set: 64.16%
Validation Loss: 2.1728
Accuracy on validation set: 18.29%
Epoch [8/10], Training Loss: 1.4647
Accuracy on training set: 68.07%
Validation Loss: 2.1655
Accuracy on validation set: 15.85%
Epoch [9/10], Training Loss: 1.3514
Accuracy on training set: 71.39%
Validation Loss: 2.1590
Accuracy on validation set: 14.63%
Epoch [10/10], Training Loss: 1.3083
Accuracy on training set: 73.49%
Validation Loss: 2.1520
Accuracy on validation set: 15.85%
Loaded the best model parameters for final evaluation or further training.
```

Figure 6: RNN model with Attention(Overfitting)

Results and Inferences

```
Epoch [15/20], Loss: 0.8493
Accuracy on test set: 71.08%
Epoch [16/20], Loss: 0.8100
Accuracy on test set: 67.47%
Epoch [17/20], Loss: 0.7775
Accuracy on test set: 66.27%
Epoch [18/20], Loss: 0.7754
Accuracy on test set: 68.67%
Epoch [19/20], Loss: 0.7561
Accuracy on test set: 60.24%
Epoch [20/20], Loss: 0.7318
Accuracy on test set: 77.11%
Saved the best model parameters.
Loaded the best model parameters for final evaluation or further training.
```

Figure 7: Accuracy using different architecture of CNN

Results and Inferences

Fitting 3 folds for each of 216 candidates, totalling 648 fits
Best parameters found: {'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
Accuracy: 0.13
Classification Report:

	precision	recall	f1-score	support
amusement	0.27	0.38	0.32	8
anger	0.11	0.11	0.11	9
calmness	0.20	0.18	0.19	11
disgust	0.17	0.08	0.11	12
excitement	0.11	0.09	0.10	11
fear	0.27	0.27	0.27	11
happiness	0.00	0.00	0.00	8
sadness	0.00	0.00	0.00	8
surprise	0.00	0.00	0.00	5
accuracy			0.13	83
macro avg	0.13	0.12	0.12	83
weighted avg	0.14	0.13	0.13	83

Figure 8: Random Forest Classification Matrix

References I

[1] K. S. Kamble and J. Sengupta, "Ensemble machine learning-based affective computing for emotion recognition using dual-decomposed eeg signals," *IEEE Sensors Journal*, vol. 22, no. 3, pp. 2670–2678, 2022.

[2] E. Cimtay *et al.*, "Emotion recognition using cnn and eeg signals," *Journal of Neural Engineering*, vol. 17, no. 6, p. 066013, 2020.

[3] X. Wang, L. Zhang, and W. Zhou, "Empirical mode decomposition based emotion classification from eeg signals," *IEEE Transactions on Biomedical Engineering*, vol. 68, no. 8, pp. 2241–2250, 2021.

[4] W. Yang, M. Liu, and H. Zhang, "Hybrid neural network for eeg-based emotion classification," *Neurocomputing*, vol. 464, pp. 105–114, 2022.