Name: Bikash Dutta
Roll No: D22CS051
Topic: Speech Understanding

Programming Assignment 3

Note: All the run logs from wandb can be found here
Demo Link: https://huggingface.co/spaces/d22cs051/Audio-Deepfake-Detection

Question 1. Audio Deepfake Detection

Task 1 + 2 + 3 + 4 + 5 + 6:

- Use the SSL W2V model.
- Compute AUC and EER on the provided dataset (pre-trained).
- Analyze the performance of the model.
- Finetune the model on the FOR dataset.
- Report the performance on the For dataset.
- Use the finetuned model to evaluate the custom dataset.

I. Custom Dataset Classes for provided Dataset and FOR Dataset

1. Imports:
   - Imports necessary modules and classes from PyTorch, pathlib, librosa, and custom modules like RawBoost.

2. CustomDataset Class:
   - Inherits from PyTorch's Dataset class.
   - __init__ method:
     - Takes a directory path and a configuration object as input.
     - Finds all .wav and .mp3 files in the "Real" and "Fake" subdirectories.
     - Creates a list of tuples (audio_file_path, label), where label is 0 for real and 1 for fake.
   - __len__ method: Returns the length of the dataset.
   - __getitem__ method:
     - Loads an audio file using librosa with a sampling rate of 16000 Hz.
     - Processes the audio data using the process_Rawboost_feature function with the provided configuration and algorithm.
     - Pads the processed audio data to a fixed length of 64600 samples (approximately 4 seconds) using the pad function.
     - Converts the padded audio data to a PyTorch tensor.
     - Returns the tensor and the corresponding label as a tuple.

3. FOR2SsecDataset Class:

- Inherits from CustomDataset.
- __init__ method:
    - Takes a directory path, a configuration object, and a split (training, testing, or validation) as input.
    - Finds all .wav and .mp3 files in the specified split subdirectories (e.g., "training/real", "training/fake").
    - Creates a list of tuples (audio_file_path, label), where label is 0 for real and 1 for fake.
    - Sets the fixed length for padding to 32000 samples (2 seconds).

4. pad Function:
- Pads or truncates an audio signal to a fixed length (max_len).
- If the signal is shorter than max_len, it is repeated and truncated to the desired length.
- If the signal is longer than max_len, it is truncated to the desired length.

5. process_Rawboost_feature Function:
- Applies various data augmentation techniques to the input audio feature.
- The augmentation techniques include convolutive noise, impulsive noise, and colored additive noise.
- The specific augmentation technique is determined by the algo parameter in the configuration object.
- Each augmentation technique has its own set of parameters controlled by the configuration object.
- The function can apply a single augmentation technique or a combination of techniques in series or parallel.
- If algo is set to a value other than the available options, the original audio feature is returned without any augmentation.

6. Main Function (testing purpose only):
- Defines command-line arguments for configuring the data augmentation parameters.
- Instantiates the CustomDataset and FOR2SsecDataset classes with the provided configuration.
- Iterates over the datasets using a progress bar from the tqdm library.

Note: Fake audio is represented by label 1 and Real is represented by 0 as mentioned in __init__ for the CustomDataset class.

II. Main_Assing file to do evaluation finetuning and re-evaluation

1. Imports and Configurations:
- Imports necessary modules and classes.

- Creates a configuration object (config) from the Config class, which holds various hyperparameters and settings.
- Initializes Weights & Biases (wandb) for logging and visualization, including setting the project name, run name, and logging the configuration dictionary.

2. Device Setup and Model Loading:
   - Determines the device (CPU or GPU) based on GPU availability.
   - Instantiates the Model class with the provided configuration and device.
   - If a model path is specified in the configuration, load the pre-trained model weights.

3. Dataset Loading and Evaluation (Custom Dataset):
   - Instantiates the CustomDataset with the specified directory and configuration.
   - Creates a DataLoader for the CustomDataset with specified batch size, number of workers, and shuffling.
   - Evaluates the pre-trained model on the CustomDataset by iterating over the dataloader.
   - Computes and logs batch-wise and average AUC, accuracy, and EER using wandb.

4. Dataset Loading (FOR2SsecDataset):
   - Instantiates the FOR2SsecDataset for training, validation, and testing splits using the specified directory and configuration.
   - Creates DataLoader objects for the combined training and validation dataset, as well as the testing dataset.

5. Optimizer, Loss Function, and Layer Freezing:
   - Defines the optimizer (Adam) with a specified learning rate and weight decay.
   - Defines the loss function (CrossEntropyLoss).
   - Freezes or unfreezes specific layers of the model for fine-tuning by setting the requires_grad attribute based on layer names.

6. Training Loop:
   - Iterates over epochs and batches from the training dataloader.
   - Performs forward pass, calculates loss, backpropagates, and updates model parameters using the optimizer.
   - Logs training metrics (loss, AUC, accuracy, EER) for each batch and epoch using wandb.
   - Evaluates the model on the test dataset after each epoch and logs the corresponding metrics.
   - Creates a wandb table with audio samples, predictions, and ground truth labels for the FOR dataset.

7. Post-training Evaluation (Custom Dataset):

- Evaluates the fine-tuned model on the CustomDataset by iterating over the dataloader.
- Computes and logs batch-wise and average AUC, accuracy, and EER using wandb.
- Creates a wandb table with audio samples, predictions, and ground truth labels for the CustomDataset.

8. Model Saving:
   - Creates a model checkpoint dictionary containing the model state dict, optimizer state dict, loss function state dict, and current epoch.
   - Saves the model checkpoint to a file (for_trained_model.ckpt) using PyTorch's torch.save.

9. Cleanup:
   - Finalizes the wandb run, which uploads the logged data and artifacts to the wandb server.

III. Results:

All the results can be found on the below link

WandB Logs

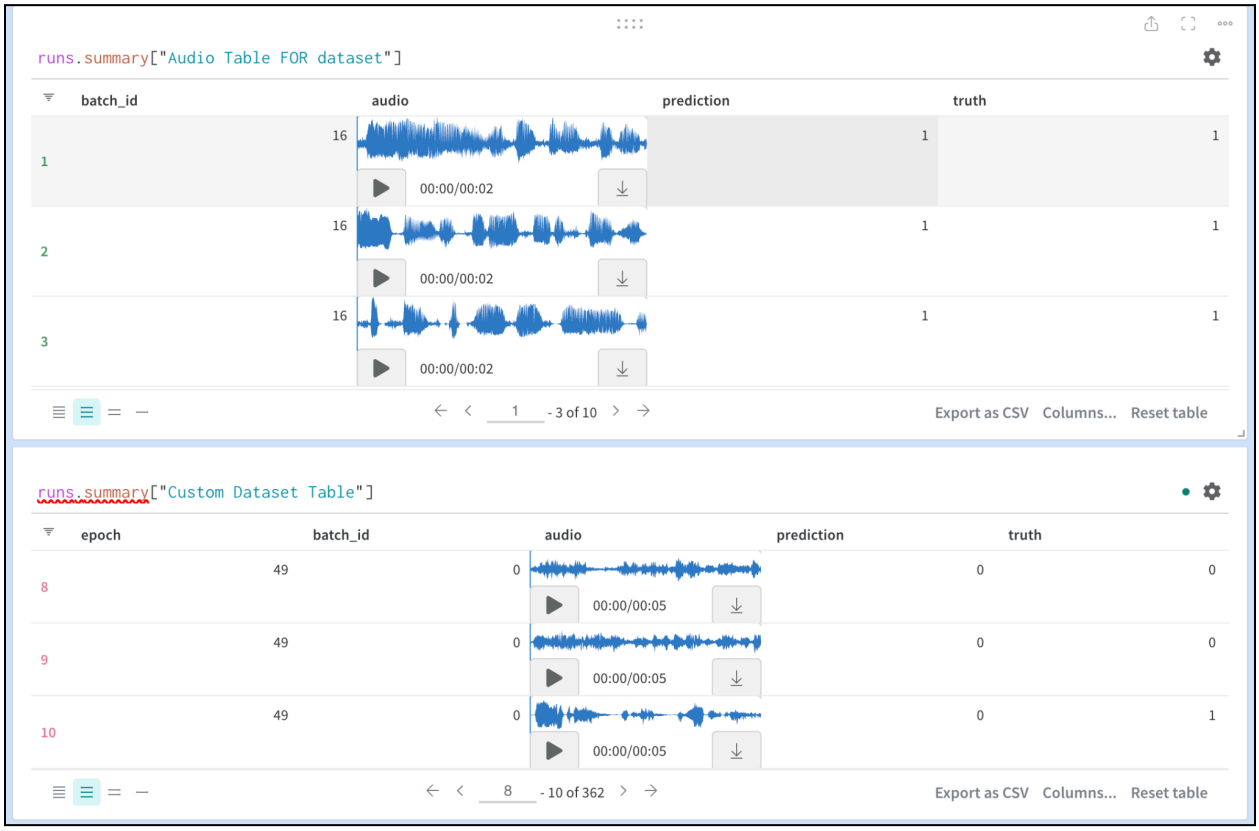

Fig 1: Prediction Tables results on the evaluation sets

Fig 2: Training-Test Curves and Plots Combined for all datasets

Fig 3: Training-Test Curves and Plots Combined for all datasets

Metrics	Pre-training Custom Dataset	FOR Dataset Evaluation	Post-training Custom Dataset
ACC (↑)	0.4823	0.9283	0.7832
EER (↓)	0.8104	0.5340	0.4784
AUC (↑)	0.4109	0.9864	0.8048

Table 1: Combined results for both datasets on the best-performing model using accuracy, equal error rate, and AUC.

IV. Analysis and Interpretation of Results:

- Unfreezing more layers in our scenario does not help because the number of samples present in the training set is not very large.

1. Pre-training Custom Dataset:
   - Accuracy (ACC ↑):
     - The model achieves an accuracy of 0.4823, meaning it correctly classifies only around 48% of the samples.
     - An accuracy of 0.5 would be achieved by random guessing, so an accuracy of 0.4823 is lower than random guessing, indicating poor performance.
   - Equal Error Rate (EER ↓):
     - The Equal Error Rate (EER) is a metric commonly used in system verification or spoofing detection systems. Here, we are treating deep fakes as spoofs.
     - An EER of 0.8104 means that when the false acceptance and rejection rates are equal, the error rate is approximately 81%.
     - Lower EER values are desirable, and an EER of 0.8104 indicates poor performance.
   - Area Under the Receiver Operating Characteristic Curve (AUC ↑):
     - The AUC measures the ability of the model to distinguish between positive and negative classes.
     - An AUC of 0.5 corresponds to a random classifier, while an AUC of 1.0 represents a perfect classifier.
     - The model achieves an AUC of 0.4109, which is lower than 0.5, indicating that the model performs worse than a random classifier.

2. FOR Dataset Evaluation:
   - Accuracy (ACC):
     - The model achieves an accuracy of 0.9283, meaning it correctly classifies approximately 92.83% of the samples.

    - An accuracy of 0.9283 is considered good and indicates that the model has learned to classify the samples effectively.
- Equal Error Rate (EER):
    - The Equal Error Rate (EER) is 0.5340, which means that when the false acceptance rate and false rejection rate are equal, the error rate is approximately 53.4%.
    - Lower EER values are desirable, and an EER of 0.5340 is better.
- Area Under the Receiver Operating Characteristic Curve (AUC):
    - The model achieves an AUC of 0.9864, which is close to the value of 1.0.
    - An AUC of 0.9864 indicates that the model can distinguish between positive and negative classes.

3. Post-training Custom Dataset:
- Accuracy (ACC):
    - The model achieves an accuracy of 0.7832, meaning it correctly classifies approximately 78.32% of the samples.
    - Compared to the pre-training accuracy of 0.4823, the post-training accuracy has improved substantially, indicating that the fine-tuning process has helped the model generalize better to the Custom Dataset.
- Equal Error Rate (EER):
    - The Equal Error Rate (EER) is 0.4784, which means that when the false acceptance rate and false rejection rate are equal, the error rate is approximately 47.84%.
    - Compared to the pre-training EER of 0.8104, the post-training EER has significantly improved.
- Area Under the Receiver Operating Characteristic Curve (AUC):
    - The model achieves an AUC of 0.8048, significantly higher than the pre-training AUC of 0.4109.
    - An AUC of 0.8048 indicates that the model can distinguish between positive and negative classes on the Custom Dataset.

V. Recommendations for Improvement:
- Taking longer sequences of the audio stream may help the model's performance.
- Need to find the trade-off between the number of trainable parameters with respect to the sample space needed.

VI. Conclusion:
- Model performance remains constant after 25 epochs because the backbone is completely frozen.
- Hyperparameter tuning may lead to better performance.

- The observations from the experiment reveal how fine-tuning impacts the model's generalizability for out-of-distribution samples.
- Improved metrics post-fine-tuning indicate that the model has learned task-specific features, enhancing model performance.
- Monitoring performance throughout the experiment provides insights for fine-tuning and further adjustments on further iterations in the training process.

References:

[1] https://github.com/TakHemlata/SSL_Anti-spoofing.git
[2] https://www.eecs.yorku.ca/~bil/Datasets/for-2sec.tar.gz
[3] https://pytorch.org/audio/stable/pipelines.html
[4] https://pytorch.org/audio/main/tutorials/
[5] https://github.com/facebookresearch/fairseq