Speech Understanding Programming Assignment 3

Name : Kushal Agrawal Roll No : M23CSA011

Gradio Link : Audio-deep-fake
Github Link : Github

Task 1 and 2 :

Use the SSL W2V model trained for LA and DF tracks of the ASVSpoof dataset . Report the AUC and EER on this dataset.

- We downloaded the XLS-R 300 M model from here and the Pre-trained SSL anti spoofing models from here
- Then I downloaded the Custom dataset from here .
- Pip install the requirements.txt file
- Log in wandb from terminal using wandb login “api key” command

Now that we have downloaded the custom dataset we unzip it and save it in the SSl_Anti-spoofing folder.
The code and the dataset would be in the same folder.

We can run the task 1&2 by entering the python evaluate_custom.py command in the terminal.

We got the **AUC : 0.611**
EER : 0.357

Code Overview :-

Importing Libraries:

- The code begins with importing necessary libraries such as os, torch, librosa, numpy, tqdm, sklearn.metrics, and torch.nn.

Utility Functions:

Two utility functions are defined:

- pad(x, max_len): This function pads the input audio x to a specified maximum length (max_len) by repeating the audio if its length is less than max_len.
- preprocess_audio(audio_path): This function loads and preprocesses audio files by padding them to a fixed length and returns the processed audio data.

Data Preparation:

- Paths to directories containing real and fake audio files are defined.
- Lists of file paths for real and fake audio samples are created using os.listdir() and list comprehension.

Model Initialization:

- The device ('cuda' if available, else 'cpu') is determined for model execution.
- An instance of the model (Model) is initialized and moved to the specified device.
- The model's state dictionary is loaded from a saved checkpoint file (best_la_model.pth).

Model Evaluation:

- The model is set to evaluation mode using .eval().
- For each audio file (both real and fake), the following steps are performed:
- The audio file is preprocessed using preprocess_audio().
- If the audio file format is supported, the processed audio is converted to a PyTorch tensor.
- The model predicts the probability of the input being a fake audio using model() and stores the output probability in the predictions list.
- The ground truth label (0 for real, 1 for fake) is appended to the ground_truth list based on the file path.

Evaluation Metrics Calculation:

- The predictions and ground truth labels are used to calculate the Receiver Operating Characteristic (ROC) curve and metrics such as the Area Under the Curve (AUC) and Equal Error Rate (EER).

- The EER is determined as the point on the ROC curve where the false positive rate (FPR) equals the false negative rate (FNR).
- Results Display:
- The calculated EER and AUC scores are printed to the console.

Task 3:

Analyze the performance of the model.

The performance metrics provided, AUC (Area Under the Curve) and EER (Equal Error Rate), offer insights into how well the model is distinguishing between real and fake audio samples.

AUC (Area Under the Curve):

- An AUC of 0.611 indicates that the model performs better than random chance but still has room for improvement. It suggests that the model is moderately successful in distinguishing between real and fake audio samples.
- However, an AUC of 0.611 falls below a strong classifier's performance, which typically has an AUC closer to 1. This suggests that the model may not be highly reliable in its predictions.

EER (Equal Error Rate):

- An EER of 0.357 indicates that the model's performance is relatively balanced between false acceptance and false rejection. In other words, the model is making errors at approximately the same rate for both types of errors.
- While a lower EER is desirable, 0.357 suggests that the model is making errors at a moderately high rate.

Possible Actions:

- Fine-tuning the model architecture, optimizing hyperparameters, or exploring different training strategies may enhance its performance.
- Increasing the diversity and size of the training dataset, particularly by adding more challenging or representative samples, could help improve the model's generalization ability.

Task 4 and 5:

Finetune the model on FOR dataset.
Report the performance using AUC and EER on For dataset.

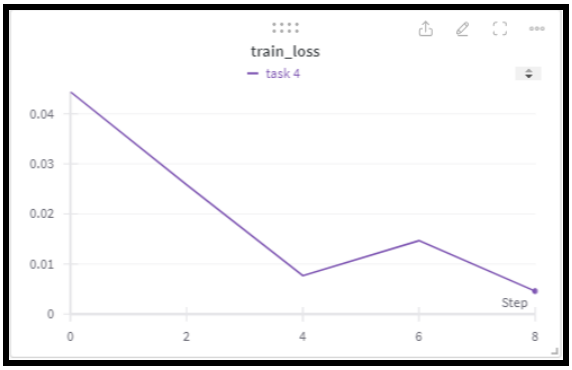
You can download the For dataset from here .
Unzip the file in the SSl_Anti-spoofing folder.

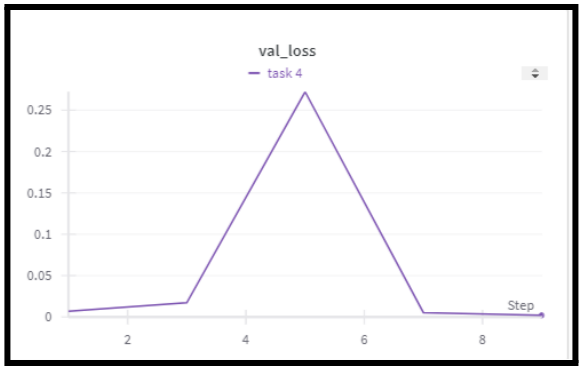
We can run the task 4&5 by entering the python fine_tune_for.py command in the terminal.

I fine-tuned for 5 epochs and for learning rate = 5e-5

We got the **AUC : 0.9738**
EER : 0.084

The below plots show the train loss and the val loss over 5 epochs , using Wandb.





Code Overview :-

Importing Libraries:

- The code begins by importing necessary libraries for audio processing, deep learning, optimization, and evaluation.
- These include os, torch, librosa, numpy, torch.nn, torch.utils.data, torchaudio, torch.optim, tqdm, scipy.optimize, scipy.interpolate, sklearn.metrics, model (presumably a custom model), and wandb (Weights & Biases for experiment tracking).

Initializing Weights & Biases (wandb):

- Weights & Biases is initialized with a project name and an experiment name.

Defining Dataset Class:

- AudioDataset class is defined to load audio files from a specified root directory, preprocess them, and provide them for training and validation. It subclasses torch.utils.data.Dataset.

Data Preparation:

- Training, testing, and validation datasets are created instances of AudioDataset class using respective directories.

Data Loaders:

- DataLoader objects are created for training, testing, and validation datasets to facilitate batch-wise data loading during training and validation.

Model Initialization and Loading:

- An instance of the model (Model) is initialized and loaded from a saved checkpoint file (best_la_model.pth). The model is moved to the appropriate device (GPU if available).

Training and Validation Functions:

- train_one_epoch() and validate_model() functions are defined to train and validate the model for one epoch, respectively. They use cross-entropy loss for training and validation.

Training Loop:

- The model is trained for a specified number of epochs (num_epochs). In each epoch, training and validation losses are computed and logged.

Model Saving:

- After training, the final trained model state dictionary is saved to a file (final_model.pth).

Evaluation Function:

- evaluate_model() function is defined to evaluate the trained model on the test dataset. It calculates the AUC (Area Under the Curve) and EER (Equal Error Rate) for the model's predictions.

Evaluation:

- The trained model is evaluated on the test dataset using the evaluate_model() function. AUC, EER, and the threshold at EER are printed to the console.

Task 6 :

Use the model trained on the FOR dataset to evaluate the custom dataset. Report the EER and AUC

Run python evaluate_custom_for.py command in terminal
We got the **AUC : 0.113**
EER : 0.833

Task 7 :

Comment on the change in performance

The significant decrease in AUC from 0.611 to 0.113 and increase in EER from 0.357 to 0.833 after fine-tuning on the custom dataset indicates a substantial deterioration in the model's performance. This change suggests that the fine-tuning process may not have been effective or may have introduced biases or overfitting to the custom dataset. Possible reasons for this decline in performance could include insufficient data for fine-tuning, a mismatch between the pre-trained model and the custom dataset, or inadequate hyperparameter tuning during fine-tuning.