1 Introduction

Basically, the task of this competition is to classify a song snippet into 4 categories according to the gender of the singers. After implementing several methods of preprocessing, I use **ResNet** as the model to train the data, and finally achieve the accuracy of 0.8042 as the highest score.

2 Preprocessing

For this competition, we have about 12000 snippets of songs, with length of each about 3s as our raw data. The mp3 files are not originally suitable for deep learning, so preprocessing step is the starting point of the whole competition and is quite important.

With the hint on audio processing, I tried **Librosa**, **torchaudio**, and implement some techiniques like fourier tranform, emphasizing on some frequency, and most importantly, mel-spectrogram to extract the features. I save the mel-spectrogram in graph, but this would lead to some information loss, so finally I use the raw feature of the mel-spectrogram as the dataset input of my model.

Moreover, I shared some idea with my classmates and they gave me advice on using **spleeter** to separate the vocal part and the background music part. Thus, the feature of audio with only vocal sound could be easier to classify. Afterwards, I found **spleeter** quite useful, and could bring my model up to 5% increase in accuracy.

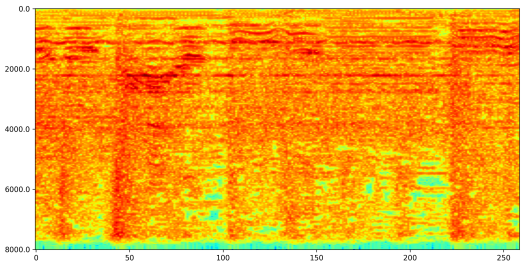

Figure 1: Mel-spectrogram

3 Model

During the competition, I tried several models, including **CNN**, **ResNet** and some of its variations. At first, I construct a **CNN** model with 3 convolutional layers and 2 fully connected layers, but the accuracy is not good enough (About 65%). Then I tried **ResNet**,

and the accuracy improved a lot. I tried **ResNet** with different layers, for example, **ResNet** 18, 34, 50, 101... And also other variations of **ResNet** like **DenseNet**. Running a single **ResNet** model could achieve an accuracy of 0.74 in optimal condition according to my experiment. My final submission implements **ResNet34** model, and I modify some layers to make the model be able to run on the data and work properly on the task. The input layer is changed to accept one dimensional 299*40 data, and the output layer is changed to 4 classes.

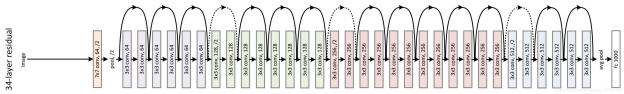

Figure 2: ResNet34

```
def __init__(self, num_classes=4, dropout_probability=0.2, input_channels=1):
    super(ResNetWithDropout, self).__init__()
    self.base_model = models.resnet34(pretrained=False)
    self.base_model.conv1 = nn.Conv2d(input_channels, 64, kernel_size=7, stride=2, padding=3, bias=False)
    num_features = self.base_model.fc.in_features
    self.base_model.fc = nn.Sequential(
        nn.Dropout(dropout_probability),
        nn.Linear(num_features, num_classes)
    )
```

Figure 3: Model modification

4 Experiment

In the training procedure, I use **CrossEntropyLoss** as the loss function which is widely used in classification task, and **AdamW** as the optimizer. The learning rate is set to 3e-4, and the batch size is set to 32. I train the model for about 20 epochs. I also add a dropout layer to avoid overfitting, and the dropout rate is set to 0.2.

Moreover, I also use the method of ensemble to improve the performance. I train several models with different random seeds, and finally average the output of the models to get the final result. This could improve the accuracy by 2% to 3%. I tried to train 4 or 5 models simultaneously at the beginning, and I found interesting that once I tried to run a ensemble model with 20 sub models, and the performance increased a lot, though quite time consuming. The ensemble with 20 models finally achieved the highest accuracy of 0.8042 at a try.

```
class EnsembleModel(nn.Module):
    def __init__(self, n_models, num_classes, dropout_probability, input_channels):
        super(EnsembleModel, self).__init__()
        self.models = nn.ModuleList([
            ResNetWithDropout(num_classes=num_classes, dropout_probability=dropout_probability, input_channels=input_channels)
            for _ in range(n_models)
        ])
```

Figure 4: ensemble implementation

I also get some of the tricks from my friends. For example I tried **Mixup**, weight computation, learning rate scheduler. Some of the tricks are quite useful.

```
class_weights = compute_class_weight('balanced', classes=np.unique(y_train),y=y_train)
class_weights = torch.tensor(class_weights, dtype=torch.float32).to(torch.device("cuda"))
```

Figure 5: weight computation

```
optimizers = [optim.AdamW(model.parameters(), lr=3e-4, weight_decay=1e-5) for model in ensemble_model.models]
for i in range(len(optimizers)):
    lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizers[i], T_0=num_epochs*len(train_loaders[i])//2)
```

Figure 6: lr scheduler

5 Discussion

Though I have tried different kinds of **ResNet** models, I am not sure if I find the best, or a rather better size of model for this task. I only tried several learning rate and training epochs, and I am not sure if the model could be improved by tuning the hyperparameters. I would tune the hyperparameters more carefully if there is more time.

Moreover, as I don't have a powerful GPU supporting cuda, I ran all my models on Kaggle platform, and it randomly disconnects, which is quite annoying and time consuming. I would like to have a powerful gaming laptop or rent a personal server next time.