Machine Learning
Instructor: Prof. Tam
Official name: Zhuoran Chen
Kaggle account/name on leader board: Jolia Chen
NetID: zc2745
(Since the explanation is more than 2 pages, you may choose to skip the detailed descriptions of the unsuccessful attempts, which are indicated in blue.)

Machine Learning Report of Final Project

description of the Zip file

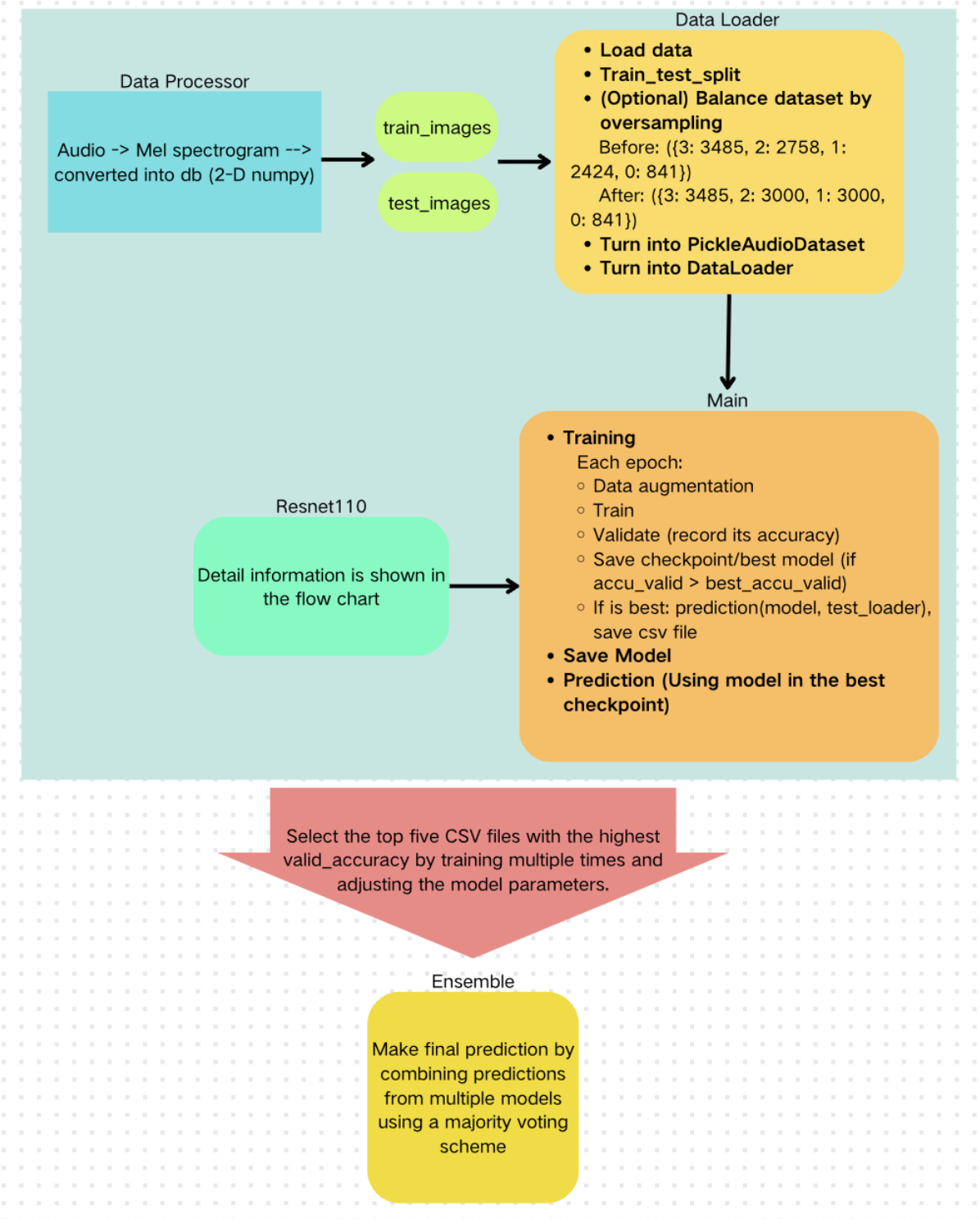
There are three files of datasets: directory "data_v1", "data_v2", and "data_v3". Each of them uses different methods to process the dataset (each data directory contains its corresponding data processor code "data_processor_v{index}").

I tried four types of models: directory "Densenet", "Inception", "CNN", and "resnet_v{index}". There are four variants of resnets, and each of them is trained on a different dataset or may balance the dataset by sampling.

Model	CNN, Inception, Densenet	resnet_v1	resnet_v2	resnet_v3	resnet_v4
Dataset	data_v1	data_v1	data_v2	data_v3	data_v1
Dataloader: whether using a balanced dataset	No	Yes {3: 3485, 2: 2758, 1: 2424, 0: 841} → ({3: 3485, 2: 3000, 1: 3000, 0: 841})	No	No	Yes {3: 3485, 2: 2758, 1: 2424, 0: 841} → ({3: 3485, 2: 3485, 1: 3485, 0: 3485})
Final decision of model and dataset	No	Yes	No	No	No

Disclaimer: For data_v2, due to the reason that each audio numpy contains too many features (features.shape: [194, 259]), the size of data_v2 is 2.3 GB. The preprocessed dataset doesn't help the model perform better on the test set (details are included in resnet_v2: "training_log"), data_v3 neither. Therefore, to save file size, I do not include processed data under these two directories in my zip file.

Training procedure, time and GPUs



The training process becomes overfitting after approximately 35 epochs, with the validation accuracy plateauing at around 85% (details are recorded under ./ resnet_v1 / training_log).

The training time is based on the dataset. For data_v1, each audio feature is a [64, 259] numpy (before converted into dataloader), and it takes an hour to finish 50 epochs; for data_v2, each audio feature is converted into a [149, 259] numpy, and it takes two hours to process 9 epochs; For data_v3, each audio feature is a [149, 1] numpy, it takes 30 mins to process 50 epochs. For my final decision (resnet_1), I trained the model several times to get multiple CSV for my final ensemble process. It took me 40 mins to train one model and 3 CSV were selected from it; I trained models 5 times by using different parameters and came up with 20 CSV. Among them, I choose 5 CSV with top 5 validation accuracy as my final ensemble resources. Therefore, it took me 4 hours to finish these processes. But you can also select CSV of the top 5 validation accuracy from one training procedure, and it only takes an hour to get the ideal ensembled output.

I used NYU Greene HPC to train my model. I first tried to use Apple MPS, but it caused my computer to lag. Then I turned to HPC, it works so much better than running locally. To assess various parameter settings, I typically run tests in parallel on two 32 GB GPUs (For data_v2, due to its large input size, it needs 64 GB for up to 8 hours to run through 40 epochs).

Data Preprocessing: whether to extract multiple features (data_processor.py)

Dataset:

data_v1: The dataset is generated by data_processor_v1.py.

It converts audio files (in MP3 format) into Mel Spectrograms and saves them as image files. It loads the audio file using librosa.load function, extracts the Mel Spectrogram using librosa.feature.melspectrogram function, and converts the power spectrogram to decibel units using librosa.power_to_db. Finally, it saves the log-scaled Mel Spectrogram to a file using pickle serialization.

data_v2: The dataset is generated by data_processor_v2.py.

It first loads the audio file using librosa.load function. Then, it extracts several audio features including Mel Frequency Cepstral Coefficients (MFCCs), Mel Spectrogram, Chroma STFT, Spectral Centroid, Spectral Contrast, and Tonnetz. Finally, these features are stacked vertically into a single feature matrix and saved to a file using pickle serialization.

data_v3: The dataset is generated by data_processor_v3.py.

Compared to data_v2, it processes the mean value of extracted features along certain dimensions to get single-dimensional feature vectors.

To save time, I converted all the audio files into tensors and stored them in f“data_{version}” directory in advance, so that I do not need to repetitively convert audio into tensors whenever I start training my model, which costs time.

The reason why I chose data_v1: I tried three types of preprocessing methods (as I mentioned above), and only data_v1 helps the model (resnet_v1) achieve better performance.

For data_v2 (where its associated trained model is resnet_v2), after each epoch, there is less discrepancy between training accuracy and validation accuracy (2~3%, and its valid accuracy up to 74% at epoch 10, which is better than resnet_v1) and its test set accuracy is 10% lower than its valid accuracy, which seems normal, but it takes a long time to train for one epoch (20 mins for one epoch), therefore I turn to data_v1 and data_v3. For resnet_v3 (where its associated processed dataset is data_v3), its training speed is much faster than resnet_v2 but has a low test accuracy (I set 150 epochs and save model parameter when its validation accuracy is higher than the previous model. Its best model achieves 84.020% validation accuracy at epoch 119, but only achieves 66% test accuracy), therefore I gave up data_v3.

Data Loader: whether to balance the dataset by sampling (data_loader.py)

data_loader.py under resnet_v1 and resnet_v4 is different.
Similarity:

In data_loader, after loading the Mel spectrogram and balancing the dataset, the code normalizes a Mel spectrogram by subtracting its mean and dividing it by its standard deviation. If the standard deviation is zero, a small value (epsilon) is added to it to prevent division by zero.

Difference:
data_loader.py under resnet_v1:

Balanced sampling dataset: The dataset was balanced by sampling the number of classes that were lower than 3000 and increasing them to 3000. {3: 3485, 2: 2758, 1: 2424, 0: 841} → ({3: 3485, 2: 3000, 1: 3000, 0: 841})

data_loader.py under resnet_v4:

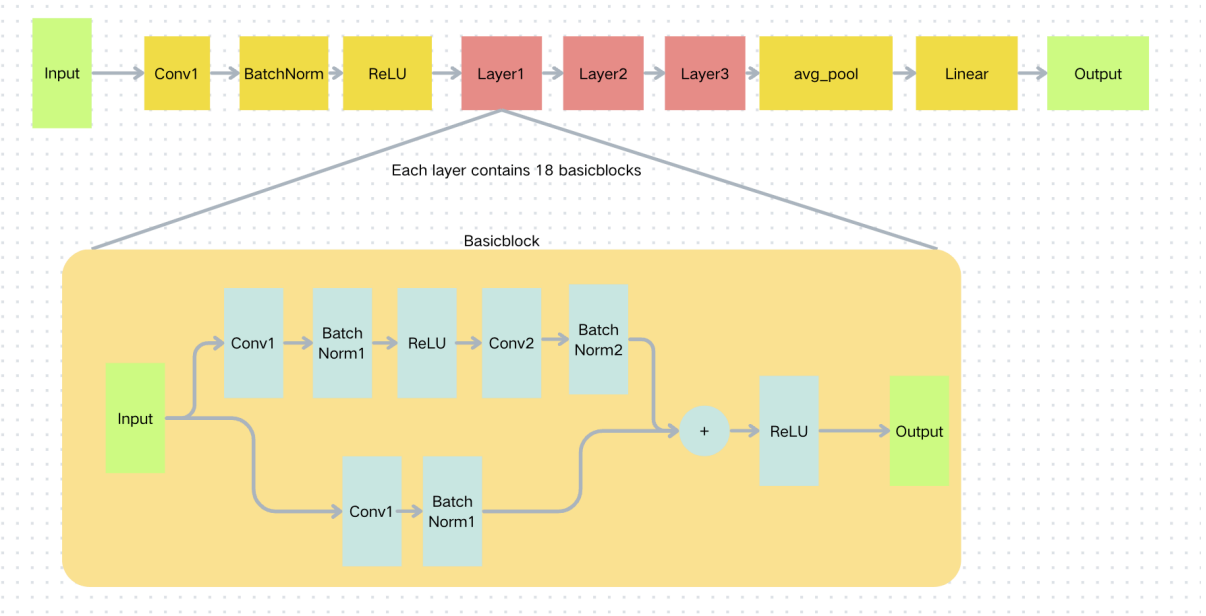
Balanced sampling dataset: Balanced the dataset into all classes that have the same number of labels. {3: 3485, 2: 2758, 1: 2424, 0: 841} → ({3: 3485, 2: 3485, 1: 3485, 0: 3485})

The reason why I chose data_loader.py under resnet_v1: I printed out the prediction of the validation set and compared them with labels. I found out that most errors are caused by distinguishing class 2 and 3 or class 1 and 3, which means class 0 is classified perfectly. Besides, if balancing the dataset into all classes that have the same number of labels, the training dataset reaches overfitting (When epoch reaches around 10, most batches' accuracies reach 100%) soon and fails to finetune parameters within the model, which causes a bad prediction on the test set. Therefore, I only increased the number of class 2 and 1 to a certain number (3000).

Training Model (resnet.py)

The reason why I chose resnet: For CNN, the highest test accuracy is 66.74; For Inspection, I only tried to use TensorFlow but not PyTorch, which failed; For densest, I failed to solve the import error. For resnet, resnet_v1 is first trained on data_v1without balanced sampling and later processes a balanced sampling dataset; resnet_v2 is based on data_v2; resnet_v3 is based on data_v3; resnet_v4 is based on data_v1 and uses a balanced sampling dataset (slightly different from the balanced methods used in resnet_v1).

Resnet architecture (resnet 110):



It is modified from GitHub https://github.com/akamaster/pytorch_resnet_cifar10, the only change is turning the input channel from 3 to 1 and class_num to 4.

Note: model parameters are stored at ./resnet_v1/resnet_model/. Both model_best1.th (without using balanced dataset sampling) and model_best.th (using balanced dataset sampling) can achieve high accuracy on the test set.

Hyper-parameters (. / resnet_v1/ main.py + training_log)

The best parameter I have tried is self.batch_size = 128, self.lr = 1e-2, self.momentum = 0.9, self.weight_decay = 1e-2.

I first tried to finetune the learning rate and batch size. I tested from lr = 1e-6 / bs = 32, lr = 1e-4 / bs = 32, lr = 1e-4 / bs = 64, and increasing gradually. When lr = 1e-2 / bs = 128, it achieves the best performance. Similarly, I found the best value for momentum and weight decay by using the same methods.

Optimization methods

1. Data augmentation: to increase the diversity of the dataset, I add data augmentation before each epoch, including Random Time Shift, Adding Random Noise, Random Pitch Shift, Random Speed Change, and SpecAugment.They are randomly added to the input data, and this decreases the risk of overfitting brought by a balanced dataset by sampling.
2. lr_scheduler: I choose lr_scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=10, verbose=True), which enables fine-tuning of the learning rate, ensuring that it decreases when the model performance plateaus;
3. Optimizer: optimizer = torch.optim.SGD(model.parameters(), args.lr,momentum=args.momentum,weight_decay=args.weight_decay);
4. Loss function: criterion = nn.CrossEntropyLoss();
5. Min Loss Threshold: another way to prevent overfitting is if the loss of the train set is lower than a certain threshold (eg. Loss < 0.1), then stop training;
6. Warm-up learning rate: in the first few epochs use low lr to stabilize the model;
7. Early stopping: I printed out validation accuracy after each training epoch so that I could check whether the model was overfitting or bad performance;
8. Ensemble: Save CSV files that correspond to the top 4~5 high validation accuracy in each training, then aggregate them through voting to determine final class predictions for each audio sample.

Reference and Acknowledgement

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Deep Residual Learning for Image Recognition. arXiv:1512.03385
[2] https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py
Thanks to Yerlan Idelbayev who provides the code of resnet 101.