Recognizing the Genre of Music

Group 10
111550002林宜頡　　111550042林筠蓁
111550057莊婷馨　　111550164廖涵玉

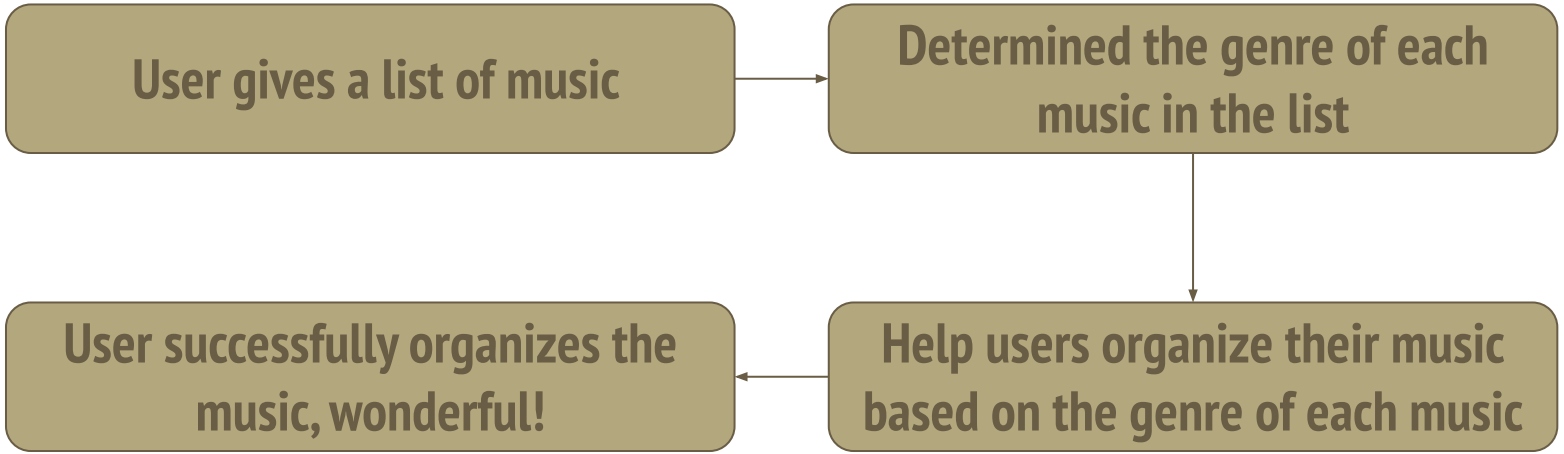
Introduction

- Abstract

  Recognizing music genres is crucial for organizing music, supporting the music industry, and enhancing user experiences on radio and streaming platforms. It helps listeners find music they like and avoid what they don't. Nowadays, with the large volume of music needing classification, manual classification is too time-consuming, so we aim to train AI to accomplish this task.

Introduction

- Goal – a music classifier



Related Work

- Music Genre Classification Project Using Machine Learning Techniques
  - Use Python speech feature to extract feature
  - Use KNN
- Our work
  - Use Librosa to extract feature
  - Use LSTM

Dataset

- Platform: Kaggle
  - GTZAN Dataset - Music Genre Classification
- GTZAN
  - A collection of 1000 audio tracks, each 30 seconds long
  - Including 10 music genres
    - blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and rock.

Dataset

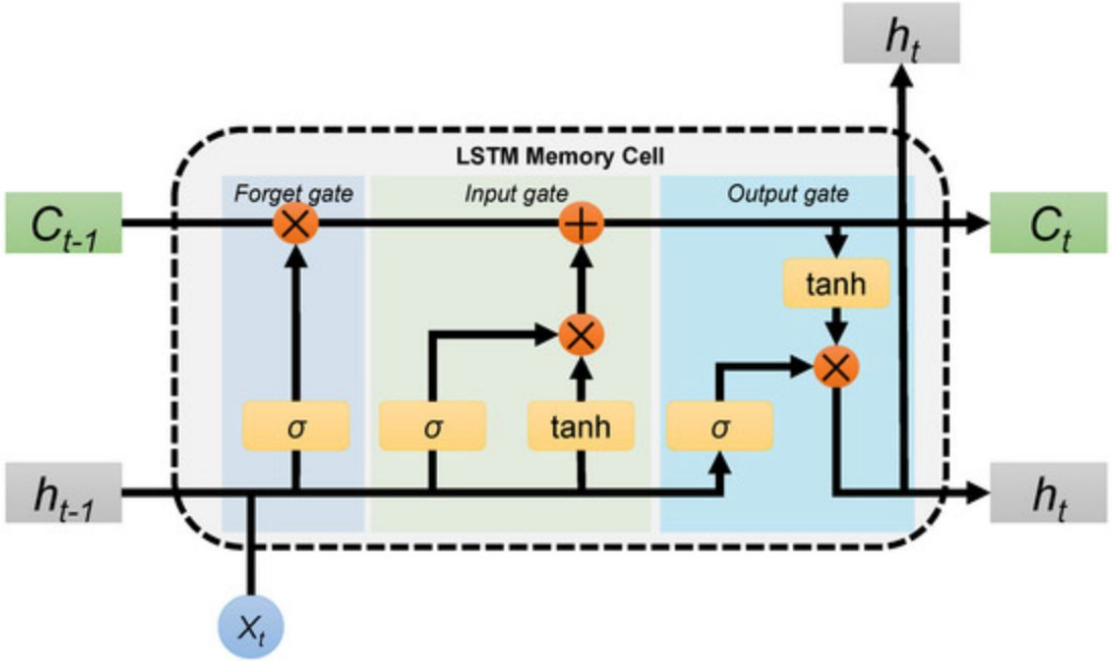
- Our adjustments
  - We have adjusted the genres to classical, country, hiphop, jazz, rock, blues, reggae, Kpop, and Cpop.
  - Removed genre
    - metal, disco (too confusing with the other genres)
    - pop (too old)
  - New data
    - blues and rock (replace the old data)
      - the origin dataset was too old
      - sound quality was too blurry
    - Kpop and Cpop

Dataset

- Preprocessing:
  - Find songs of the corresponding genre -> download their mp3 file -> convert them into .au files -> split each song into 30-second segments -> rename them according to their genre.

```
18      for audio_file in audio_files:
19          input_file = os.path.join(input_folder, audio_file)
20          audio = AudioSegment.from_file(input_file)
21          split_length_ms = 30 * 1000
22          num_segments = len(audio) // split_length_ms
23          for i in range(num_segments):
24              start_time = i * split_length_ms
25              end_time = (i + 1) * split_length_ms
26              segment = audio[start_time:end_time]
27              output_file = os.path.join(output_folder, f"blues.{str(count).zfill(6)}.au")
28              segment.export(output_file, format="au")
29              count += 1
```

Baseline - LSTM



Baseline - LSTM

- package: Pytorch
- epoch = 200
- batch size = 15
- dropout = 0.2
- learning rate = 0.001
- loss: CrossEntropyLoss
- optimizer: Adam

```python
class LSTM(nn.Module):
    model_path = "./model/lstm_model_all128_2.pth"

    def __init__(self, input_dim, hidden_dim, layer_num, batch_size, output_dim, dropout):
        super(LSTM, self).__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.layer_num = layer_num
        self.batch_size = batch_size
        self.dropout = nn.Dropout(dropout)

        # initialize LSTM
        self.lstm = nn.LSTM(input_dim, hidden_dim, layer_num)
        self.batch_norm = nn.BatchNorm1d(hidden_dim)
        # output
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, input, hidden=None):
        out, _ = self.lstm(input, hidden)
        out = out[:, -1, :]      # Get the last time step's output
        out = self.fc(out)
        return out
```

```python
model = LSTM(input_dim, hidden_dim, layer_num, batch_size, output_dim, dropout)
lossfunc = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

Baseline - LSTM

- Why LSTM?
  - Not common in music classification
  - Long-term dependency
  - Avoid Vanishing Gradient Problem

Main Approach

- Overall Algorithm
  - Use Librosa to extract audio features
  - Use Pytorch to implement LSTM model

Main Approach

- Code Structure
  - DataProcessing.py
    - Turn audio samples into model-processable data
  - LSTM.py
    - Construct LSTM model and model training
  - test.py
    - Process test data and predict their genre
    - Put the audio files with same genre into the same folder

Main Approach

1. Data Processing

   We use python package librosa for audio processing.

   Extract 4 features from the audio: mfcc, spectral centroid, chromagram and spectral contrast

```
64      # compute features
65      mfcc = librosa.feature.mfcc( y=y, sr=sr, hop_length=self.hop_length, n_mfcc=13 )
66      spectral_center = librosa.feature.spectral_centroid( y=y, sr=sr, hop_length=self.hop_length )
67      chroma = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=self.hop_length )
68      spectral_contrast = librosa.feature.spectral_contrast( y=y, sr=sr, hop_length=self.hop_length )
```

Main Approach

1. Data Processing

   In our dataset, there are only 100 samples for each genre, which is far from enough for a LSTM model. Therefore, we partition one 30-second audio sample into 5 audio samples of around 6 seconds.

   -> trade-off between the number and the length of samples.

```
70    # partition data sample into parts
71    for k in range(partition_num):
72        data[partition_num*i+k, 0:partition_len, 0:13] = mfcc.T[ k*partition_len:(k+1)*partition_len, :]
73        data[partition_num*i+k, 0:partition_len, 13:14] = spectral_center.T[ k*partition_len:(k+1)*partition_len, :]
74        data[partition_num*i+k, 0:partition_len, 14:26] = chroma.T[ k*partition_len:(k+1)*partition_len, :]
75        data[partition_num*i+k, 0:partition_len, 26:33] = spectral_contrast.T[ k*partition_len:(k+1)*partition_len, :]
```

Main Approach

2. Model Training

Use Pytorch build a LSTM model and initialize it

6 parameters:

dimension of input feature	dimension of the hidden layer
the number of LSTM layer	batch size
output dimension(number of genre)	dropout probability

```
75      model = LSTM(input_dim, hidden_dim, layer_num, batch_size, output_dim, dropout)
76      lossfunc = nn.CrossEntropyLoss()
77      optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

Main Approach

2. Model Training

Load feature data extracted from Data Proccessing step, and we divide the data into small batches when training

```
59      train_X = torch.from_numpy(music_data.train_X).type(torch.Tensor)
60      train_Y = torch.from_numpy(music_data.train_Y).type(torch.Tensor)
61      dev_X = torch.from_numpy(music_data.dev_X).type(torch.Tensor)
62      dev_Y = torch.from_numpy(music_data.dev_Y).type(torch.Tensor)
```

```
91          for i in range(num_batch):
92              model.zero_grad()
93              batch_X = train_X[ i*batch_size : (i+1)*batch_size , : , : ]
94              batch_Y = train_Y[ i*batch_size : (i+1)*batch_size , : ]
```

Main Approach

2. Model Training

Feed the data to the model to compute the output, and compute the loss. Use the loss to do backward propagation, and compute accuracy by the predictions.

```
# Forward pass
outputs = model(batch_X)

loss = lossfunc(outputs, batch_Y)

# Backward and optimize
loss.backward()
optimizer.step()
```

```
def accuracy(outputs, Y, batch_size):
    _, predicted = torch.max(outputs, 1)
    _, trueclass = torch.max(Y, 1)
    # print("pre ", predicted)
    # print("true ", trueclass)
    correct_num = (predicted == trueclass).sum().item()
    return (correct_num / batch_size) * 100
```

Main Approach

2. Model Training

   Do validation every 10 epochs and compute accuracy and loss

   Finally, we save the model to use it in the Data Testing step

```
182     # save the model state
183     torch.save(model.state_dict(), model.model_path)
```

Main Approach

3. Data Testing

   Extract features from the test audio like in data processing. Send the feature data into the trained model and get predicted genres.

```
50      features = extract_feature(song)
51      features_tensor = torch.from_numpy(features).type(torch.Tensor)
52      outputs = model(features_tensor)
53      _, prediction = torch.max(outputs, 1)
```

Main Approach

3. Data Testing

   Create folders for each genre and sort the audio samples. Then the user can get a catagorized folder of audio samples.

```
80                prediction = predict(model, song_path)
81                target_folder = os.path.join(input_folder, prediction)
82                shutil.move(song_path, target_folder)
83                genre_dict[prediction].append(song[:-4])
```

Main Approach - Experiment Methods

- Experiment 1
  - Dataset without Kpop and Cpop (7 genres)
  - LSTM models
    - 1-layered or 2-layered
    - hidden_dim = 128 or hidden_dim = 256
- Experiment 2
  - Dataset with 9 genres
  - LSTM models
    - 1-layered or 2-layered
    - hidden_dim = 128 or hidden_dim = 256

Evaluation Metrics

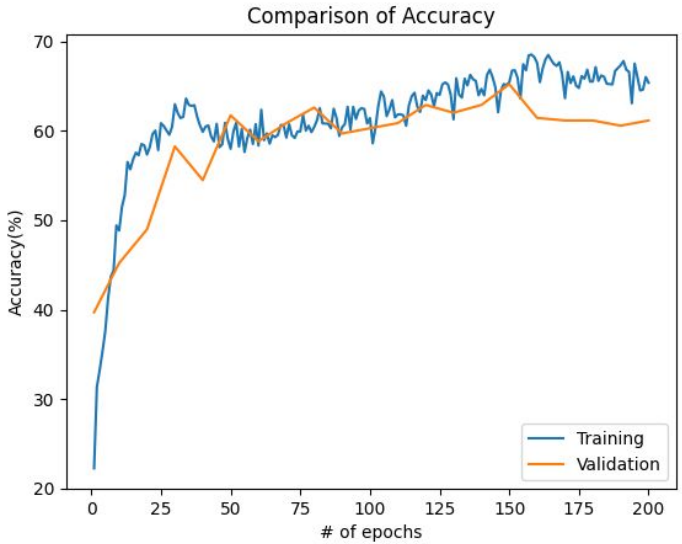
- Accuracy on test data (rounded off to 2nd decimal place)

$$\text{accuracy} = \frac{\text{number of correct samples}}{\text{total number of samples}}$$

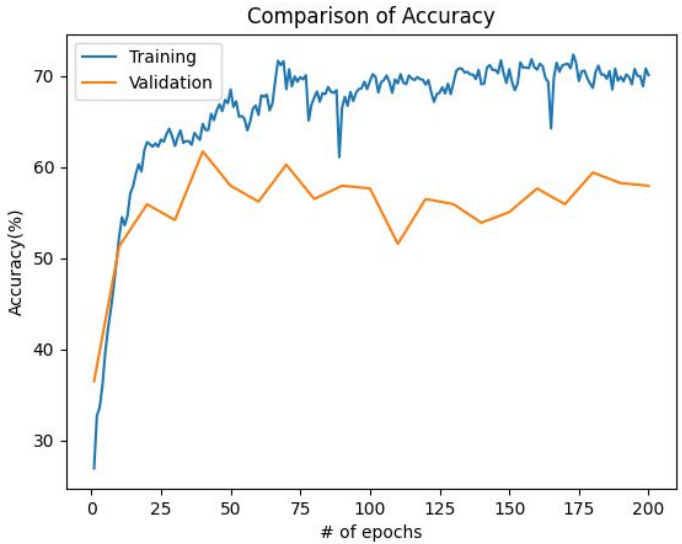
	1-layered hidden_num = 128	1-layered hidden_num = 256	2-layered hidden_num = 128	2-layered hidden_num = 256
7-genre dataset	54.93 %	61.97 %	60.56 %	59.15 %
9-genre dataset	64.44 %	51.11 %	60.00 %	61.11 %

Results & Analysis - Experiment 1

- Train and Validation Accuracy



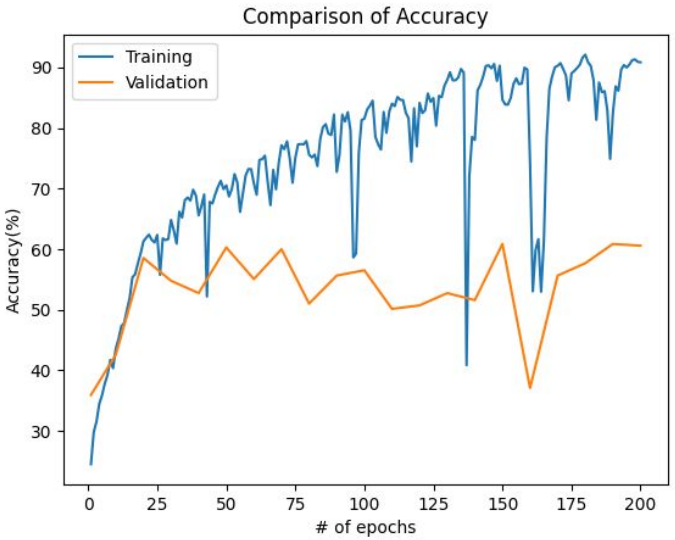
1-layered, hidden_dim=128



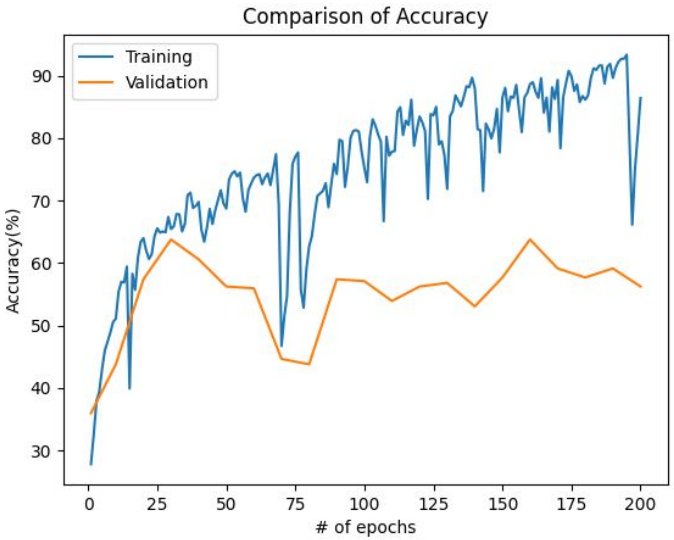
1-layered, hidden_dim=256

Results & Analysis - Experiment 1

- **Train and Validation Accuracy**



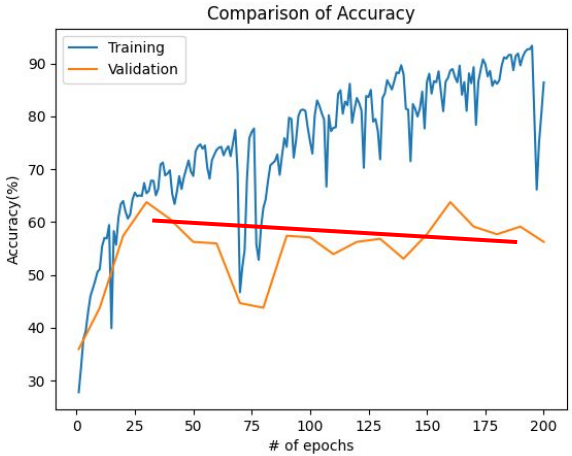
2-layered, hidden_dim=128



2-layered, hidden_dim=256

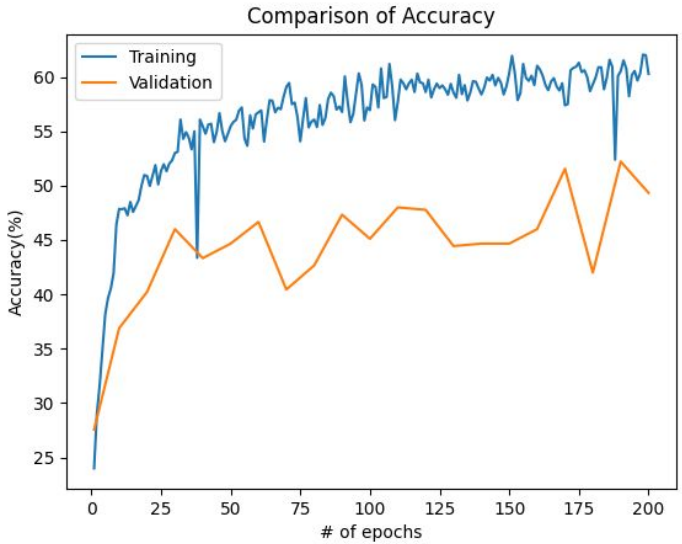
Results & Analysis - Experiment 1

- Observation 1
  - The 2-layered models show sign of overfitting
    - enlarge dataset
    - employ more generalization methods
  - Single-layer models seem to be better choice

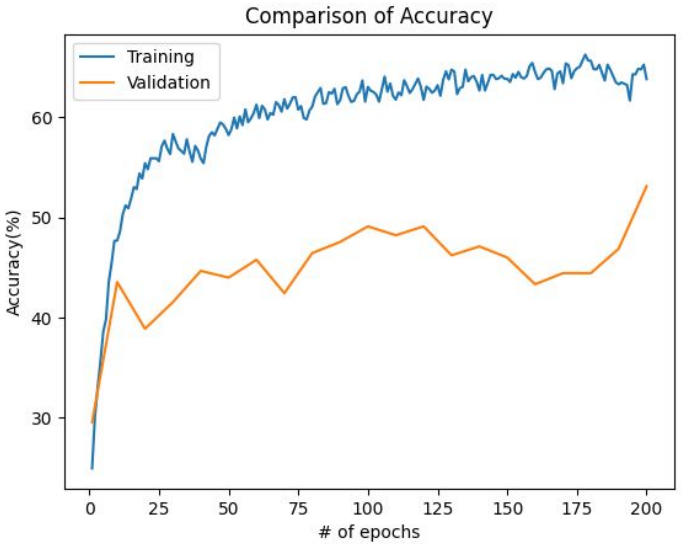


Results & Analysis - Experiment 2

- **Train and Validation Accuracy**



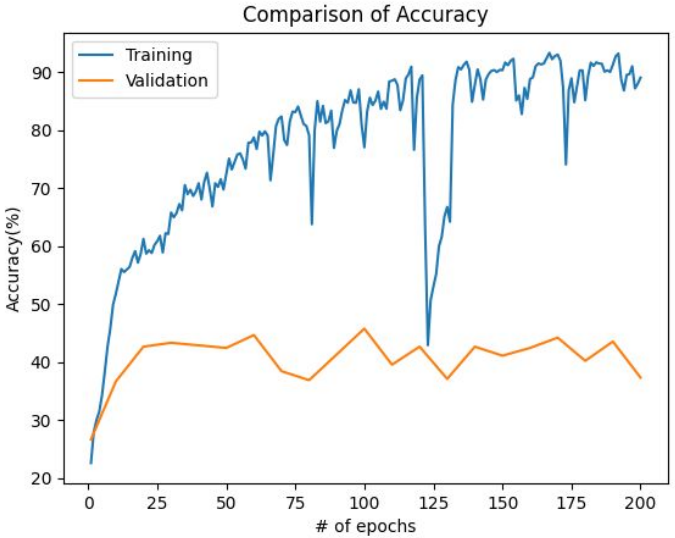
1-layered, hidden_dim=128



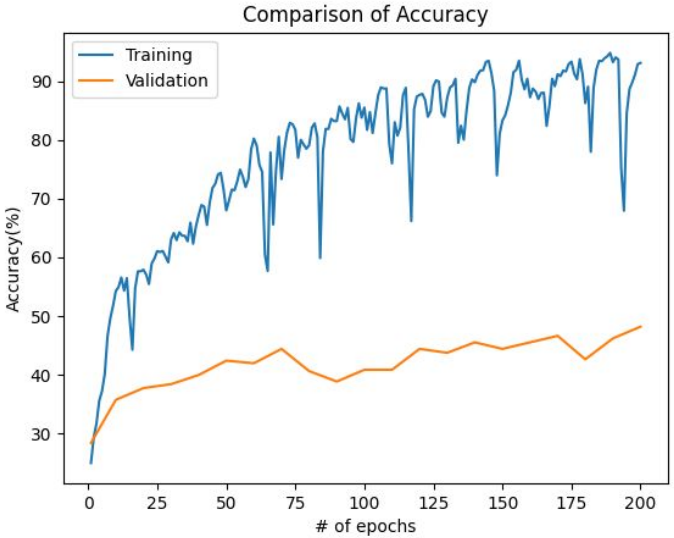
1-layered, hidden_dim=256

Results & Analysis - Experiment 2

- **Train and Validation Accuracy**



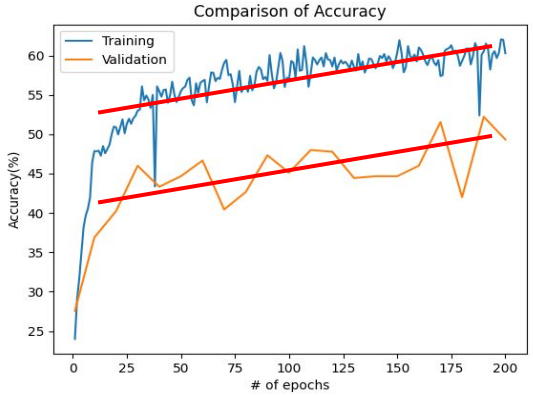
2-layered, hidden_dim=128



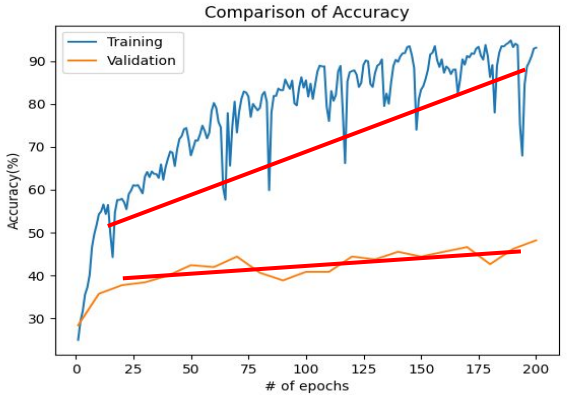
2-layered, hidden_dim=256

Results & Analysis - Experiment 2

- Observation 2
  - 2-layer model can fit train data very well -> up to 90 %
  - single-layer model has better performance on validation data
    - shows a similar trend on training and validation



1-layered, hidden_dim=128



2-layered, hidden_dim=256

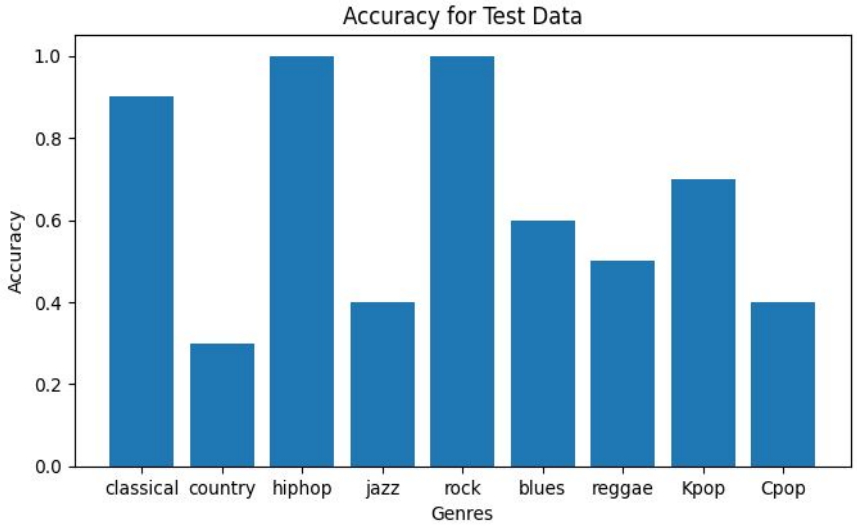
Results & Analysis

- Observation 3
  - accuracy does not decrease drastically after adding Kpop and Cpop

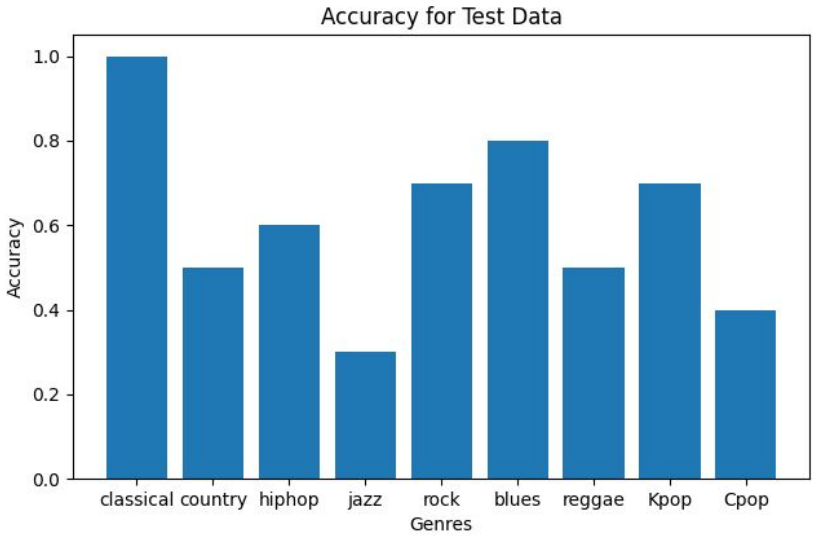
	1-layered hidden_num = 128	1-layered hidden_num = 256	2-layered hidden_num = 128	2-layered hidden_num = 256
7-genre dataset	54.93 %	61.97 %	60.56 %	59.15 %
9-genre dataset	64.44 %	51.11 %	60.00 %	61.11 %

Results & Analysis

- Accuracy on each genre of test data for 2 best model



1-layered, hidden_dim=128

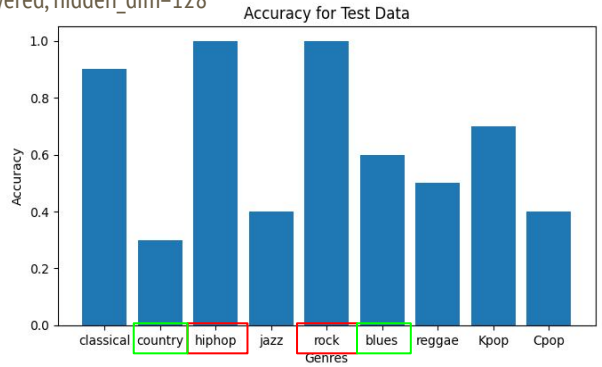


2-layered, hidden_dim=256

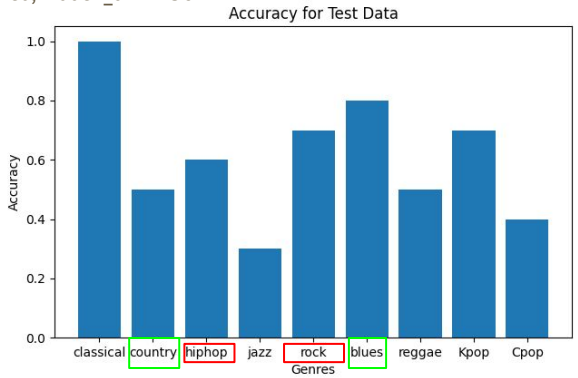
Results & Analysis

- Observation 4
  - Genres with heavy beats are recognized better by the first model
    - hiphop and rock
  - Genres with softer melodies are recognized better by the second model
    - country and blues
  - Model perform well on “classical”

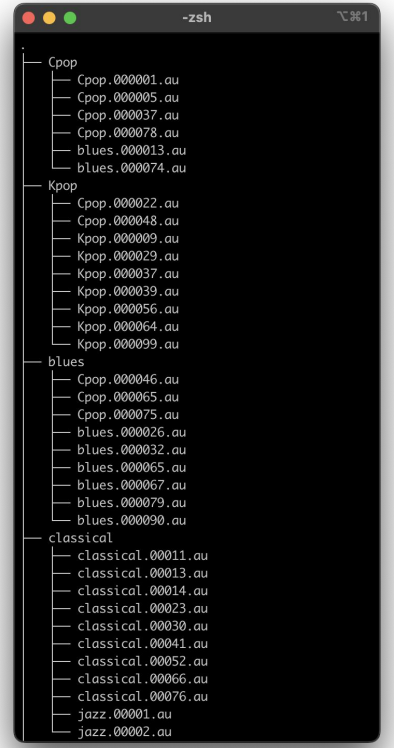
1-layered, hidden_dim=128

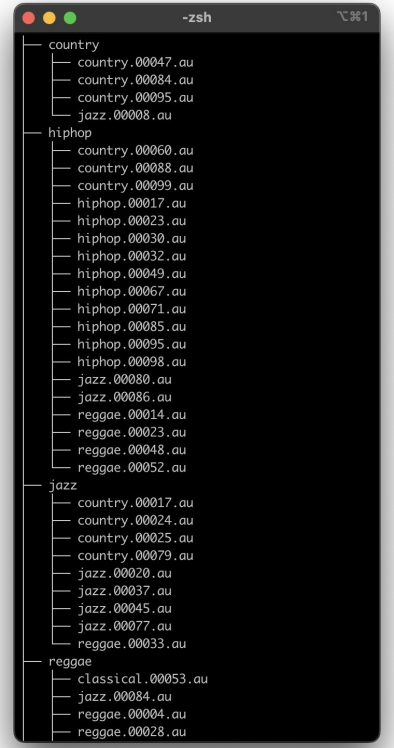


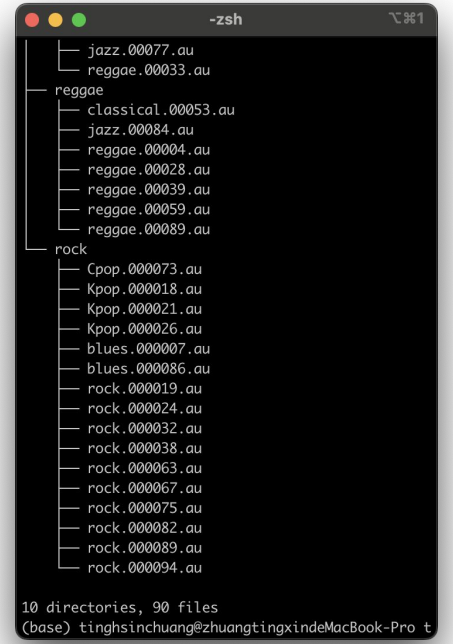
2-layered, hidden_dim=256



Results & Analysis - Final Result







Results & Analysis - Limitations

- Can only reach around 50% - 60% accuracy on validation and test data
- Model rely hugely on the variation and amount of training dataset
  - different patterns for same genre
  - genres evolve through time

- Improvements
  - change model architecture
  - enlarge and diversify dataset
  - auto-updating

Results & Analysis - Limitations

- Improvements
  - Model Architecture
    - Combine two LSTM with different hyper-parameters
    - Combine CNN and LSTM to improve accuracy
  - Dataset
    - random cropping on audio samples
    - incorporate web scraping technique to enlarge dataset more efficiently
    - auto-update dataset and re-train model

GitHub Link

AIFinal-Music_Genre_Recognizer

https://github.com/vch2128/AIFinal-Music_Genre_Recognizer

Reference

- https://zhenglungwu.medium.com/pytorch%E5%AF%A6%E4%BD%9Clstm%E5%9F%B7%E8%A1%8C%E8%A8%8A%E8%99%9F%E9%A0%90%E6%B8%AC-d1d3f17549e7
- https://machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/
- https://medium.com/@premtibadiya/music-genre-classification-using-rnn-lstm-1c212ba21e06

Contributions

- 111550002林宜頡（15%）:
  - Idea for the topic / Music collection / Test and classifier / Slides
- 111550042林筠蓁（15%）:
  - Music collection / Slides / Recording
- 111550057莊婷馨（55%）:
  - Data Processing / LSTM model / Experiment / Data Analysis / Slides
- 111550164廖涵玉（15%）:
  - Idea for the topic / Music collection / Dataset management and preprocessing

Thank you!