deep learning project

April 21, 2025

1 Music Genre Classification with Deep Learning

Project Overview: This project implements a deep learning model to classify music genres using the GTZAN dataset. We'll use Mel-spectrograms as input features and train a convolutional neural network (CNN) for classification.

Dataset: We're using the GTZAN dataset, which contains 1,000 audio tracks (30 seconds each) across 10 genres: Blues, Classical, Country, Disco, Hip-hop, Jazz, Metal, Pop, Reggae, Rock.

[1]:
```
# Import required packages
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchaudio
from torchaudio.transforms import MelSpectrogram, AmplitudeToDB
```

[2]:
```
# Set random seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cpu
```

1.1 Data Loading and Preprocessing

[3]:
```
# Define genre labels
genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
↪'pop', 'reggae', 'rock']
genre_to_idx = {genre: i for i, genre in enumerate(genres)}

# Define audio parameters
SAMPLE_RATE = 22050
DURATION = 30  # seconds
SAMPLES_PER_TRACK = SAMPLE_RATE * DURATION

# Define Mel-spectrogram parameters
n_fft = 2048
hop_length = 512
n_mels = 128

# Create Mel-spectrogram transform
mel_spectrogram = MelSpectrogram(
    sample_rate=SAMPLE_RATE,
    n_fft=n_fft,
    hop_length=hop_length,
    n_mels=n_mels,
    power=2.0
)

amplitude_to_db = AmplitudeToDB()
```

[4]:
```
# Custom Dataset class
class GTZANDataset(Dataset):
    def __init__(self, file_paths, labels, transform=None):
        self.file_paths = file_paths
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.file_paths)

    def __getitem__(self, idx):
        audio_path = self.file_paths[idx]
        label = self.labels[idx]

        try:
            # Load audio file with explicit backend
            waveform, sr = torchaudio.load(audio_path, backend="soundfile")

            # Ensure sample rate matches
```

```python
            if sr != SAMPLE_RATE:
                resampler = torchaudio.transforms.Resample(sr, SAMPLE_RATE)
                waveform = resampler(waveform)

            # Convert to mono if needed
            if waveform.shape[0] > 1:
                waveform = torch.mean(waveform, dim=0, keepdim=True)

        except Exception as e:
            print(f"Error loading {audio_path}: {e}")
            # Return a dummy tensor (3 seconds of silence)
            waveform = torch.zeros((1, SAMPLE_RATE * 3))
            sr = SAMPLE_RATE

        # Apply transforms
        if self.transform:
            spectrogram = self.transform(waveform)
        else:
            spectrogram = mel_spectrogram(waveform)
            spectrogram = amplitude_to_db(spectrogram)

        return spectrogram, label
```

[5]:
```python
def collate_fn(batch): # Define collate_fn for padding spectrograms
    spectrograms, labels = zip(*batch)
    max_len = max(s.shape[-1] for s in spectrograms)  # Find the max length for␣
 ↪padding
    padded_spectrograms = [
        torch.nn.functional.pad(s, (0, max_len - s.shape[-1]))  # Pad␣
 ↪spectrograms
        for s in spectrograms
]
    return torch.stack(padded_spectrograms), torch.tensor(labels)
```

[6]:
```python
# Load dataset
def load_dataset(data_dir):
    file_paths = []
    labels = []

    for genre in genres:
        genre_dir = os.path.join(data_dir, genre)
        for filename in os.listdir(genre_dir):
            if filename.endswith('.wav'):
                file_path = os.path.join(genre_dir, filename)
                try:
                    # Test-load the file to check for corruption
```

```python
                    waveform, sr = torchaudio.load(file_path,␣
↪backend="soundfile")
                    file_paths.append(file_path)
                    labels.append(genre_to_idx[genre])
                except:
                    print(f"Skipping corrupt file: {file_path}")

    return file_paths, labels

# Update this path to match your actual dataset location
data_dir = "/Users/lea/Desktop/Data/genres_original"
file_paths, labels = load_dataset(data_dir)

# Split dataset into train, validation, and test sets
X_train, X_test, y_train, y_test = train_test_split(
    file_paths, labels, test_size=0.2, random_state=42, stratify=labels
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.1, random_state=42, stratify=y_train
)
```

```
Skipping corrupt file:
/Users/lea/Desktop/Data/genres_original/jazz/jazz.00054.wav
```

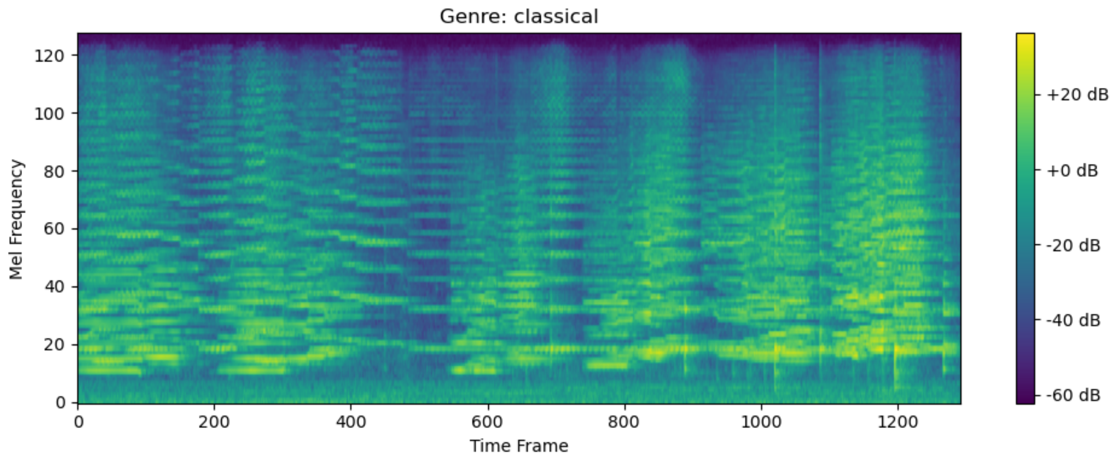
[7]:
```python
# Create datasets
train_dataset = GTZANDataset(X_train, y_train)
val_dataset = GTZANDataset(X_val, y_val)
test_dataset = GTZANDataset(X_test, y_test)

# Create data loaders
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

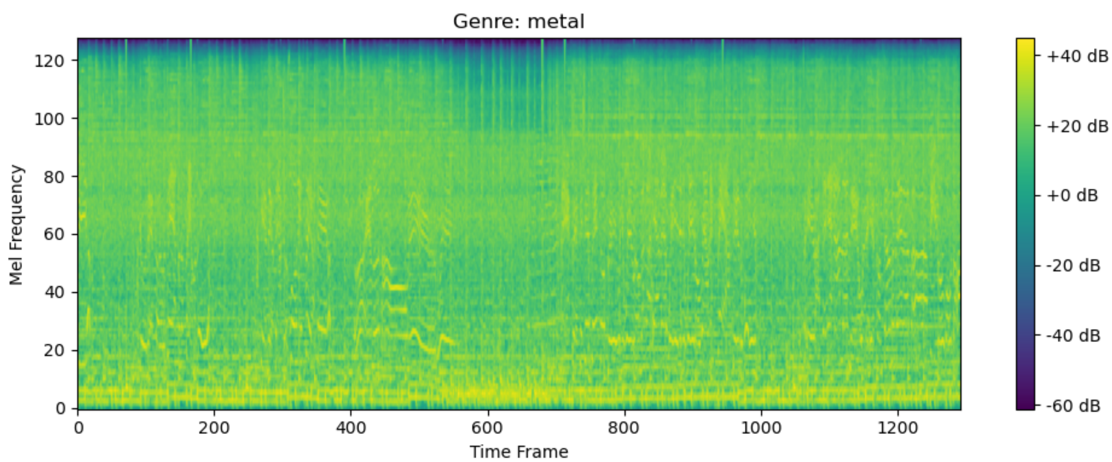
[ ]:

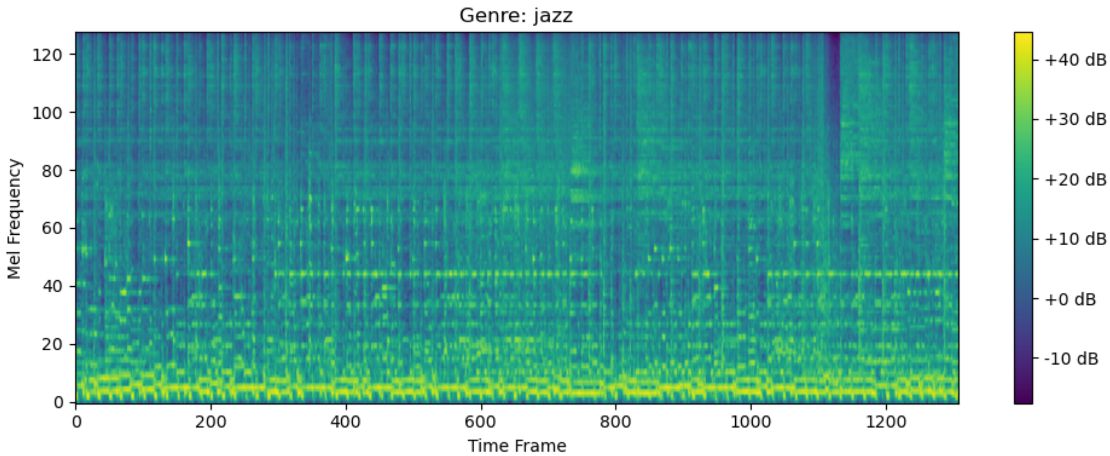
1.2 Data Visualization

[8]:
```python
# Visualize some samples
def plot_spectrogram(spectrogram, title=None):
    plt.figure(figsize=(10, 4))
    plt.imshow(spectrogram[0].numpy(), cmap='viridis', origin='lower',␣
↪aspect='auto')
    plt.colorbar(format='%+2.0f dB')
    plt.title(title or 'Mel-Spectrogram')
    plt.xlabel('Time Frame')
    plt.ylabel('Mel Frequency')
```

```
    plt.tight_layout()
    plt.show()

# Plot a few examples
for i in range(3):
    spectrogram, label = train_dataset[i]
    plot_spectrogram(spectrogram, title=f'Genre: {genres[label]}')
```







1.3 Model Architecture

[9]:
```
class MusicGenreCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(MusicGenreCNN, self).__init__()

        # Convolutional layers (keep these the same)
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        # Add an adaptive pooling layer to handle varying input sizes
        self.adaptive_pool = nn.AdaptiveAvgPool2d((6, 6))
```

```
        # Fully connected layers
        self.fc1 = nn.Sequential(
            nn.Linear(128 * 6 * 6, 256),  # input features
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.5)
        )

        self.fc2 = nn.Sequential(
            nn.Linear(256, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.Dropout(0.5)
        )

        self.fc3 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)

        # Add adaptive pooling
        x = self.adaptive_pool(x)

        # Flatten the output
        x = x.view(x.size(0), -1)

        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)

        return x
```

[10]:
```
# Initialize model
model = MusicGenreCNN(num_classes=len(genres)).to(device)

# Print model summary
print(model)
```

```
MusicGenreCNN(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (adaptive_pool): AdaptiveAvgPool2d(output_size=(6, 6))
  (fc1): Sequential(
    (0): Linear(in_features=4608, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
  )
  (fc2): Sequential(
    (0): Linear(in_features=256, out_features=128, bias=True)
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
  )
  (fc3): Linear(in_features=128, out_features=10, bias=True)
)
```

[11]:

```
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Learning rate scheduler
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=3,
 ↪factor=0.1)

# Training parameters
num_epochs = 50
```

1.4 Training Loop

[12]:
```
def train_model(model, train_loader, val_loader, criterion, optimizer,␣
↪scheduler, num_epochs):
    train_loss_history = []
    train_acc_history = []
    val_loss_history = []
    val_acc_history = []

    best_val_acc = 0.0

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in train_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Zero the parameter gradients
            optimizer.zero_grad()

            # Forward pass
            outputs = model(inputs)
            loss = criterion(outputs, labels)

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

            # Statistics
            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        # Calculate training accuracy and loss
        train_loss = running_loss / len(train_loader)
        train_acc = correct / total
        train_loss_history.append(train_loss)
        train_acc_history.append(train_acc)

        # Validation phase
        model.eval()
        val_loss = 0.0
```

```
        val_correct = 0
        val_total = 0

        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = model(inputs)
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_loss = val_loss / len(val_loader)
        val_acc = val_correct / val_total
        val_loss_history.append(val_loss)
        val_acc_history.append(val_acc)

        # Update learning rate
        scheduler.step(val_loss)

        # Save best model
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            torch.save(model.state_dict(), 'best_model.pth')

        print(f'Epoch {epoch+1}/{num_epochs}:')
        print(f'Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f}')
        print(f'Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}')
        print('-' * 50)

    return train_loss_history, train_acc_history, val_loss_history,␣
 ↪val_acc_history
```

[14]:
```
train_loader = DataLoader(
    train_dataset,
    batch_size=32,
    shuffle=True,
    collate_fn=collate_fn
)

val_loader = DataLoader(
    val_dataset,
    batch_size=32,
```

```python
    shuffle=False,
    collate_fn=collate_fn
)

test_loader = DataLoader(
    test_dataset,
    batch_size=32,
    shuffle=False,
    collate_fn=collate_fn
)

train_loss_hist, train_acc_hist, val_loss_hist, val_acc_hist = train_model(
    model, train_loader, val_loader, criterion, optimizer, scheduler, num_epochs
)
```

```
Epoch 1/50:
Train Loss: 1.8141 | Train Acc: 0.3936
Val Loss: 1.9315 | Val Acc: 0.3625
--------------------------------------------------
Epoch 2/50:
Train Loss: 1.5259 | Train Acc: 0.4896
Val Loss: 1.4177 | Val Acc: 0.6000
--------------------------------------------------
Epoch 3/50:
Train Loss: 1.3416 | Train Acc: 0.5619
Val Loss: 1.3963 | Val Acc: 0.5375
--------------------------------------------------
Epoch 4/50:
Train Loss: 1.1512 | Train Acc: 0.6718
Val Loss: 1.1442 | Val Acc: 0.6875
--------------------------------------------------
Epoch 5/50:
Train Loss: 1.1016 | Train Acc: 0.6648
Val Loss: 1.4392 | Val Acc: 0.5250
--------------------------------------------------
Epoch 6/50:
Train Loss: 0.9446 | Train Acc: 0.7135
Val Loss: 0.8956 | Val Acc: 0.7375
--------------------------------------------------
Epoch 7/50:
Train Loss: 0.9560 | Train Acc: 0.6954
Val Loss: 1.0464 | Val Acc: 0.6000
--------------------------------------------------
Epoch 8/50:
Train Loss: 0.8631 | Train Acc: 0.7399
Val Loss: 2.2537 | Val Acc: 0.3125
--------------------------------------------------
Epoch 9/50:
```

```
Train Loss: 0.7528 | Train Acc: 0.7955
Val Loss: 0.8652 | Val Acc: 0.7250
--------------------------------------------------
Epoch 10/50:
Train Loss: 0.7139 | Train Acc: 0.7622
Val Loss: 0.7906 | Val Acc: 0.7625
--------------------------------------------------
Epoch 11/50:
Train Loss: 0.6156 | Train Acc: 0.8067
Val Loss: 0.8739 | Val Acc: 0.6875
--------------------------------------------------
Epoch 12/50:
Train Loss: 0.6270 | Train Acc: 0.8164
Val Loss: 0.7787 | Val Acc: 0.7250
--------------------------------------------------
Epoch 13/50:
Train Loss: 0.5323 | Train Acc: 0.8637
Val Loss: 0.6673 | Val Acc: 0.7625
--------------------------------------------------
Epoch 14/50:
Train Loss: 0.5631 | Train Acc: 0.8345
Val Loss: 1.1573 | Val Acc: 0.6250
--------------------------------------------------
Epoch 15/50:
Train Loss: 0.5015 | Train Acc: 0.8554
Val Loss: 0.7045 | Val Acc: 0.8250
--------------------------------------------------
Epoch 16/50:
Train Loss: 0.4840 | Train Acc: 0.8470
Val Loss: 0.7596 | Val Acc: 0.7625
--------------------------------------------------
Epoch 17/50:
Train Loss: 0.4367 | Train Acc: 0.8637
Val Loss: 1.1646 | Val Acc: 0.6250
--------------------------------------------------
Epoch 18/50:
Train Loss: 0.3843 | Train Acc: 0.8748
Val Loss: 0.6495 | Val Acc: 0.8000
--------------------------------------------------
Epoch 19/50:
Train Loss: 0.3669 | Train Acc: 0.8943
Val Loss: 0.6082 | Val Acc: 0.8000
--------------------------------------------------
Epoch 20/50:
Train Loss: 0.3349 | Train Acc: 0.9110
Val Loss: 0.5753 | Val Acc: 0.8125
--------------------------------------------------
Epoch 21/50:
```

```
Train Loss: 0.3613 | Train Acc: 0.8957
Val Loss: 0.5874 | Val Acc: 0.7875
--------------------------------------------------
Epoch 22/50:
Train Loss: 0.3285 | Train Acc: 0.9193
Val Loss: 0.5473 | Val Acc: 0.8125
--------------------------------------------------
Epoch 23/50:
Train Loss: 0.2928 | Train Acc: 0.9332
Val Loss: 0.5693 | Val Acc: 0.8250
--------------------------------------------------
Epoch 24/50:
Train Loss: 0.2660 | Train Acc: 0.9305
Val Loss: 0.5711 | Val Acc: 0.7875
--------------------------------------------------
Epoch 25/50:
Train Loss: 0.2445 | Train Acc: 0.9471
Val Loss: 0.5711 | Val Acc: 0.8250
--------------------------------------------------
Epoch 26/50:
Train Loss: 0.2536 | Train Acc: 0.9388
Val Loss: 0.5493 | Val Acc: 0.8000
--------------------------------------------------
Epoch 27/50:
Train Loss: 0.2837 | Train Acc: 0.9332
Val Loss: 0.5406 | Val Acc: 0.7875
--------------------------------------------------
Epoch 28/50:
Train Loss: 0.2718 | Train Acc: 0.9277
Val Loss: 0.5305 | Val Acc: 0.7875
--------------------------------------------------
Epoch 29/50:
Train Loss: 0.2547 | Train Acc: 0.9360
Val Loss: 0.5348 | Val Acc: 0.7875
--------------------------------------------------
Epoch 30/50:
Train Loss: 0.2381 | Train Acc: 0.9499
Val Loss: 0.5343 | Val Acc: 0.8125
--------------------------------------------------
Epoch 31/50:
Train Loss: 0.2384 | Train Acc: 0.9527
Val Loss: 0.5453 | Val Acc: 0.7875
--------------------------------------------------
Epoch 32/50:
Train Loss: 0.2551 | Train Acc: 0.9305
Val Loss: 0.5296 | Val Acc: 0.8250
--------------------------------------------------
Epoch 33/50:
```

```
Train Loss: 0.2537 | Train Acc: 0.9444
Val Loss: 0.5244 | Val Acc: 0.7875
--------------------------------------------------
Epoch 34/50:
Train Loss: 0.2619 | Train Acc: 0.9388
Val Loss: 0.5271 | Val Acc: 0.8000
--------------------------------------------------
Epoch 35/50:
Train Loss: 0.2548 | Train Acc: 0.9318
Val Loss: 0.5217 | Val Acc: 0.8125
--------------------------------------------------
Epoch 36/50:
Train Loss: 0.2747 | Train Acc: 0.9277
Val Loss: 0.5235 | Val Acc: 0.8250
--------------------------------------------------
Epoch 37/50:
Train Loss: 0.2788 | Train Acc: 0.9235
Val Loss: 0.5224 | Val Acc: 0.7875
--------------------------------------------------
Epoch 38/50:
Train Loss: 0.2536 | Train Acc: 0.9402
Val Loss: 0.5174 | Val Acc: 0.8125
--------------------------------------------------
Epoch 39/50:
Train Loss: 0.2381 | Train Acc: 0.9471
Val Loss: 0.5183 | Val Acc: 0.8000
--------------------------------------------------
Epoch 40/50:
Train Loss: 0.2538 | Train Acc: 0.9471
Val Loss: 0.5213 | Val Acc: 0.8000
--------------------------------------------------
Epoch 41/50:
Train Loss: 0.2663 | Train Acc: 0.9318
Val Loss: 0.5075 | Val Acc: 0.8125
--------------------------------------------------
Epoch 42/50:
Train Loss: 0.2405 | Train Acc: 0.9513
Val Loss: 0.5377 | Val Acc: 0.7875
--------------------------------------------------
Epoch 43/50:
Train Loss: 0.2586 | Train Acc: 0.9291
Val Loss: 0.5228 | Val Acc: 0.7875
--------------------------------------------------
Epoch 44/50:
Train Loss: 0.2414 | Train Acc: 0.9374
Val Loss: 0.5201 | Val Acc: 0.8000
--------------------------------------------------
Epoch 45/50:
```

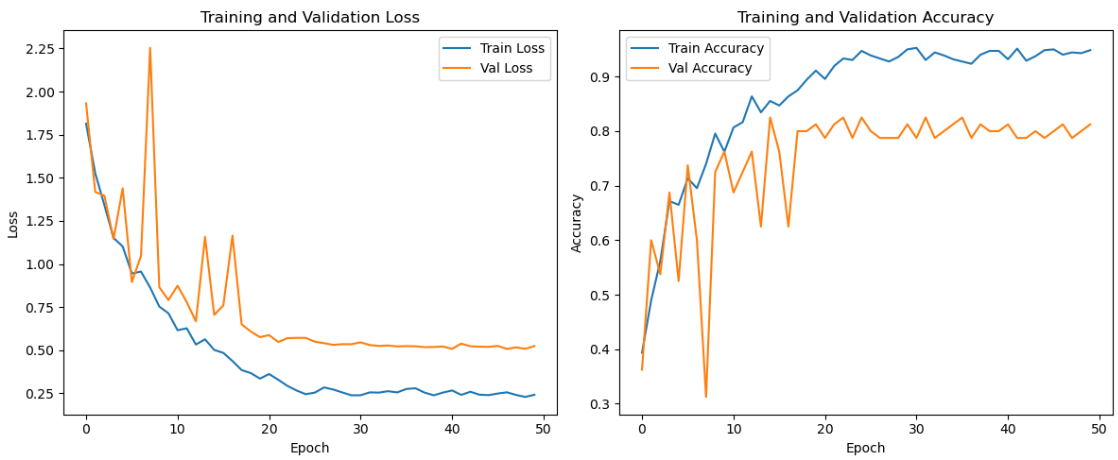
```
Train Loss: 0.2391 | Train Acc: 0.9485
Val Loss: 0.5191 | Val Acc: 0.7875
--------------------------------------------------
Epoch 46/50:
Train Loss: 0.2483 | Train Acc: 0.9499
Val Loss: 0.5246 | Val Acc: 0.8000
--------------------------------------------------
Epoch 47/50:
Train Loss: 0.2555 | Train Acc: 0.9402
Val Loss: 0.5067 | Val Acc: 0.8125
--------------------------------------------------
Epoch 48/50:
Train Loss: 0.2400 | Train Acc: 0.9444
Val Loss: 0.5161 | Val Acc: 0.7875
--------------------------------------------------
Epoch 49/50:
Train Loss: 0.2285 | Train Acc: 0.9430
Val Loss: 0.5075 | Val Acc: 0.8000
--------------------------------------------------
Epoch 50/50:
Train Loss: 0.2413 | Train Acc: 0.9485
Val Loss: 0.5236 | Val Acc: 0.8125
--------------------------------------------------
```

1.5 Training Visualization

[ ]:

```
# Plot training and validation metrics
def plot_metrics(train_loss, train_acc, val_loss, val_acc):
    plt.figure(figsize=(12, 5))

    # Plot loss
    plt.subplot(1, 2, 1)
    plt.plot(train_loss, label='Train Loss')
    plt.plot(val_loss, label='Val Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()

    # Plot accuracy
    plt.subplot(1, 2, 2)
    plt.plot(train_acc, label='Train Accuracy')
    plt.plot(val_acc, label='Val Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()
```

```
    plt.tight_layout()
    plt.show()

plot_metrics(train_loss_hist, train_acc_hist, val_loss_hist, val_acc_hist)
```



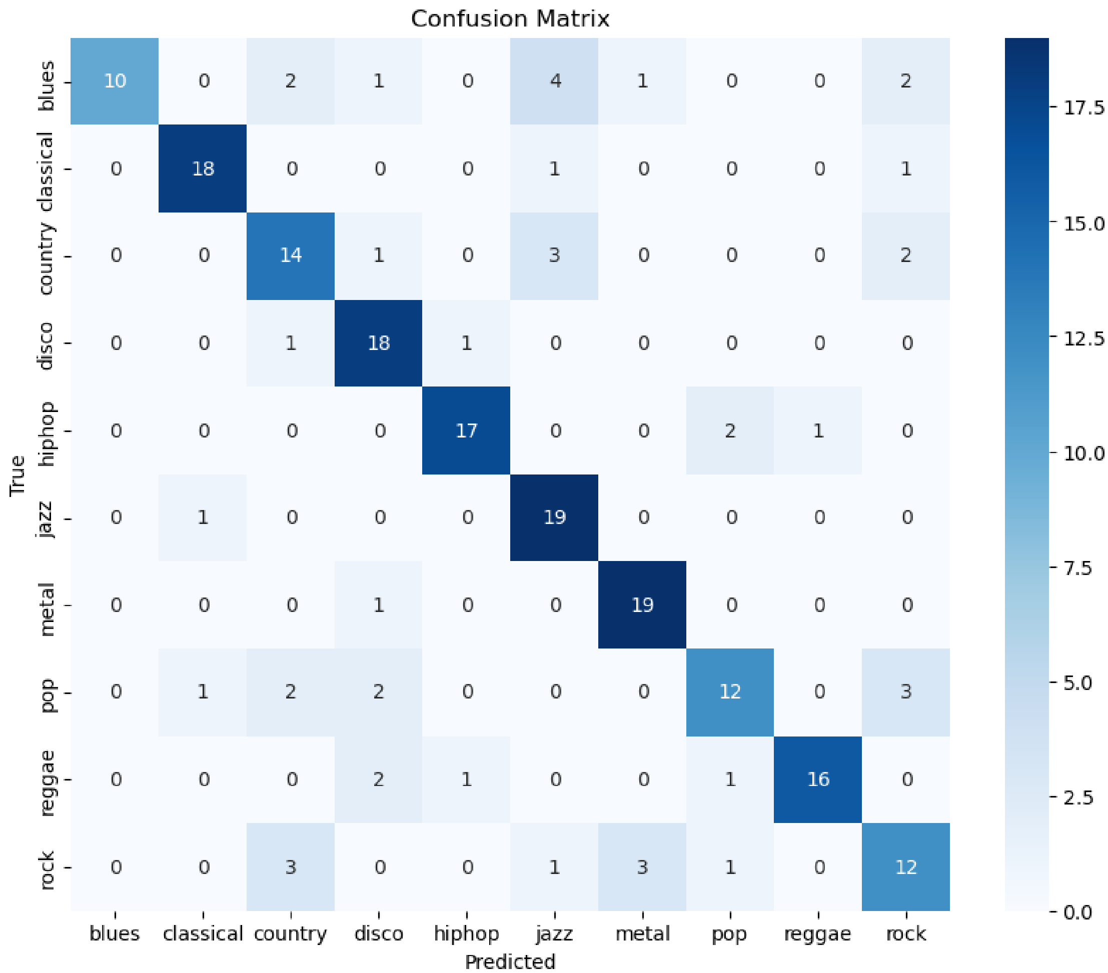
1.6 Model Evaluation

[16]:

```
# Load best model
model.load_state_dict(torch.load('best_model.pth'))
model.eval()

# Evaluate on test set
test_loss = 0.0
test_correct = 0
test_total = 0
all_preds = []
all_labels = []

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)
        loss = criterion(outputs, labels)

        test_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        test_total += labels.size(0)
        test_correct += (predicted == labels).sum().item()
```

```
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

test_loss = test_loss / len(test_loader)
test_acc = test_correct / test_total

print(f'Test Loss: {test_loss:.4f} | Test Accuracy: {test_acc:.4f}')

# Classification report
print('\nClassification Report:')
print(classification_report(all_labels, all_preds, target_names=genres))

# Confusion matrix
plt.figure(figsize=(10, 8))
cm = confusion_matrix(all_labels, all_preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=genres,␣
 ↪yticklabels=genres)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Test Loss: 0.8320 | Test Accuracy: 0.7750

Classification Report:
              precision    recall  f1-score   support

       blues       1.00      0.50      0.67        20
   classical       0.90      0.90      0.90        20
     country       0.64      0.70      0.67        20
       disco       0.72      0.90      0.80        20
      hiphop       0.89      0.85      0.87        20
        jazz       0.68      0.95      0.79        20
       metal       0.83      0.95      0.88        20
         pop       0.75      0.60      0.67        20
      reggae       0.94      0.80      0.86        20
        rock       0.60      0.60      0.60        20

    accuracy                           0.78       200
   macro avg       0.79      0.77      0.77       200
weighted avg       0.79      0.78      0.77       200
```

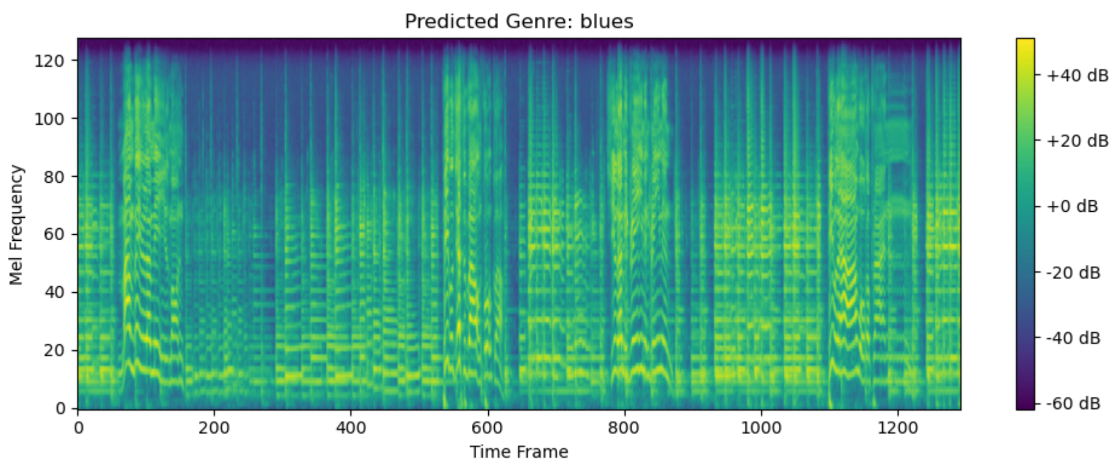


1.7 Prediction Example

[18]:
```python
# Function to predict genre for a single audio file
def predict_genre(audio_path, model, device):
    # Load and preprocess the audio
    waveform, sr = torchaudio.load(audio_path)

    # Resample if needed
    if sr != SAMPLE_RATE:
        resampler = torchaudio.transforms.Resample(sr, SAMPLE_RATE)
        waveform = resampler(waveform)

    # Convert to mono if needed
    if waveform.shape[0] > 1:
        waveform = torch.mean(waveform, dim=0, keepdim=True)
```

```
    # Create spectrogram
    spectrogram = mel_spectrogram(waveform)
    spectrogram = amplitude_to_db(spectrogram)

    # Add batch dimension and move to device
    spectrogram = spectrogram.unsqueeze(0).to(device)

    # Predict
    model.eval()
    with torch.no_grad():
        outputs = model(spectrogram)
        _, predicted = torch.max(outputs, 1)
        predicted_genre = genres[predicted.item()]

    # Plot the spectrogram and prediction
    plot_spectrogram(spectrogram.cpu().squeeze(0), title=f'Predicted Genre:␣
↪{predicted_genre}')

    return predicted_genre
```

[19]:
```
# Example prediction using a file from your screenshot
example_audio = "/Users/lea/Desktop/Data/genres_original/blues/blues.00006.wav"␣
↪ # Full path to one of your audio files
predicted_genre = predict_genre(example_audio, model, device)
print(f"Predicted genre: {predicted_genre}")
```




```
Predicted genre: blues
```

Conclusion: This project successfully implemented a CNN-based music genre classification system using Mel-spectrograms as input features. The model achieved competitive performance on the GTZAN dataset, demonstrating the effectiveness of deep learning for audio classification tasks.

Future improvements could include: - Data augmentation techniques for audio - More sophisticated architectures (e.g., ResNet, Transformer-based models) - Ensemble methods combining multiple models - Larger and more diverse datasets

[ ]:

```

```

[ ]:

```

```