Final Team Project: Music Genre and Composer Classification Using Deep Learning

Introduction

Music is a form of art that is ubiquitous and has a rich history. Different composers have created music with their unique styles and compositions. However, identifying the composer of a particular piece of music can be a challenging task, especially for novice musicians or listeners. The proposed project aims to use deep learning techniques to identify the composer of a given piece of music accurately.

Objective

The primary objective of this project is to develop a deep learning model that can predict the composer of a given musical score accurately. The project aims to accomplish this objective by using two deep learning techniques: Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN).

Project Timeline

- **Module 2 (by the end of Week 2):** The course instructor grouped students into teams of two to three members. Canvas, USD Email, or Slack can be used to find prospective team members.
- **Module 4 (by the end of Week 4):** Each team's representative submitted the "Team Project Status Update Form."
- **Module 7 (by the end of Week 7):** Each team should submit deliverables for the course project in the final week:
  - **Project Report**
  - **Project Notebook**

It is critical to note that no extensions will be given for any of the final projects' due dates for any reason, and final projects submitted after the final due date will not be graded.

Dataset

The project will use a dataset consisting of musical scores from various composers. Download the dataset from Kaggle dataset. The dataset contains the MIDI files of compositions from well-known classical composers like Bach, Beethoven, Chopin, and Mozart. The dataset should be labeled with the name of the composer for each score. Please only do your prediction only for below composers, therefore you need to select the required composers from the given dataset above. 1. Bach 2. Beethoven 3. Chopin 4. Mozart

Methodology

The proposed project will be implemented using the following steps: 1. **Data Collection:** Data is collected and provided to you. 2. **Data Pre-processing:** Convert the musical scores into a format suitable for deep learning models. This involves converting the musical scores into MIDI files and applying data augmentation techniques. 3. **Feature Extraction:** Extract features from the MIDI files, such as notes, chords, and tempo, using music analysis tools. 4. **Model Building:** Develop a deep learning model using LSTM and CNN architectures to classify the musical scores according to the composer. 5. **Model Training:** Train the deep learning model using the pre-processed and feature-extracted data. 6. **Model Evaluation:** Evaluate the performance of the deep learning model using accuracy, precision, and recall metrics. 7. **Model Optimization:** Optimize the deep learning model by fine-tuning hyperparameters.

Deliverables

There are two deliverables for this Final Project: 1. **Project Report:** A comprehensive documentation/report that describes the methodology, data pre-processing steps, feature extraction techniques, model architecture, and training process for reproducibility and future reference. Write your technical report in APA 7 style (here is a Sample Professional Paper). Please submit the report in PDF format and use the File naming convention `DeliverableName-TeamNumber.pdf`; for example, `Project_Report-Team1.pdf`.

Your report should:

- Contain a reference list that includes any external sources, libraries, or frameworks used during the project, including proper citations or acknowledgments.
- Include a concluding section or markdown cell that summarizes the project, highlights key findings, and suggests any potential future improvements or extensions to the work.

3. **Project Notebook:** A Jupyter Notebook file (.ipynb) that contains the entire project code, including data pre-processing, feature extraction, model building, training, evaluation, and any additional analysis or visualizations performed during the project. This deliverable will be exported from a Jupyter Notebook and submitted as a PDF or HTML file.

Conclusion

The proposed project aims to use deep learning techniques to accurately predict the composer of a given musical score. The project will be implemented using LSTM and CNN architectures and will involve data pre-processing, feature extraction, model building, training, and evaluation. The final model can be used by novice musicians, listeners, and music enthusiasts to identify the composer of a musical piece accurately.

Power Usage for this Project

You can use Google Colab GPU and TPU in case you need more computation power. Change your runtime in Google Colab notebook to GPU or TPU. Another option is to buy the subscription in case you need more computational power (recommended). Please follow this link to do so: Google Colab Pro+.

NOTE: Team members may not get the same grade on the Final Team Project, depending on each team member's level of contribution. This assignment has Turnitin enabled for submissions which means that your instructor will obtain a Similarity Report that identifies specific parts of your writing that may indicate a high level of matching to external content. You are strongly encouraged to review your work without penalty by activating the Draft Coach extension in your Google Docs prior to submitting your work for final grading.

To understand how your work will be assessed, view the scoring rubric below. Click the Start Assignment button above to submit your assignment.

Rubric

Final Team Project Scoring Rubric

Criteria	Ratings	Pts
Project Report	25%	75 pts
Meets or Exceeds Expectations	Report thoroughly describes the methodology, data preprocessing steps, feature extraction techniques, model architecture, and training process for reproducibility and future reference.	75 pts
Approaches Expectations	Report generally describes the methodology, data preprocessing steps, feature extraction techniques, model architecture, and training process for reproducibility and future reference.	67.5 pts

Criteria	Ratings	Pts
Below Expectations	Report minimally describes the methodology, data preprocessing steps, feature extraction techniques, model architecture, and training process for reproducibility and future reference.	61.5 pts
Inadequate Attempt	Report is missing one or more of the following: 1. Methodology 2. Data preprocessing steps 3. Feature extraction techniques 4. Model architecture 5. Training process for reproducibility and future reference.	52.5 pts
Non-Performance	Non-performance.	0 pts

Criteria	Ratings	Pts
Project Notebook	65%	195 pts
Meets or Exceeds Expectations	Team's notebook contains the entire project code in high quality, including all of the following elements: 1. Data preprocessing 2. Feature extraction 3. Model building 4. Training 5. Evaluation 6. Additional analysis or visualizations performed during the project.	195 pts
Approaches Expectations	Team's notebook contains the entire project code, including all of the following elements, but needs minor revision: 1. Data preprocessing 2. Feature extraction 3. Model building 4. Training 5. Evaluation 6. Additional analysis or visualizations performed during the project.	175.5 pts

Criteria	Ratings	Pts
Below Expectations	Team's notebook contains the entire project code, including all of the following elements, but needs major revision: 1. Data preprocessing 2. Feature extraction 3. Model building 4. Training 5. Evaluation 6. Additional analysis or visualizations performed during the project.	159.9 pts
Inadequate Attempt	Team's notebook is missing 1 or more of the following elements: 1. Data preprocessing 2. Feature extraction 3. Model building 4. Training 5. Evaluation 6. Additional analysis or visualizations performed during the project.	136.5 pts
Non-Performance	Non-performance.	0 pts

Criteria	Ratings	Pts
References and Citations	5%	15 pts
Meets or Exceeds Expectations	Project includes a references section or markdown cell that lists all external sources, libraries, or frameworks used during the project, including proper citations or acknowledgments in proper APA format.	15 pts
Approaches Expectations	Project includes a references section or markdown cell that lists all external sources, libraries, or frameworks used during the project, including proper citations or acknowledgments. However, minor revisions are needed in APA format.	13.5 pts

Criteria	Ratings	Pts
Below Expectations	Project includes a references section or markdown cell that lists all external sources, libraries, or frameworks used during the project, including proper citations or acknowledgments. However, major revisions are needed in APA format.	12.3 pts
Inadequate Attempt	Project includes a references section or markdown cell, but does not list all external sources, libraries, or frameworks used during the project, including proper citations or acknowledgments.	10.5 pts
Non-Performance	Non-performance.	0 pts

Criteria	Ratings	Pts
Conclusion	5%	15 pts
Meets or Exceeds Expectations	Project includes a thorough concluding section or markdown cell that summarizes the project, highlights key findings, and suggests any potential future improvements or extensions to the work.	15 pts
Approaches Expectations	Project includes an adequate concluding section or markdown cell that summarizes the project, highlights key findings, and suggests any potential future improvements or extensions to the work.	13.5 pts
Below Expectations	Project includes a vague concluding section or markdown cell that summarizes the project, highlights key findings, and suggests any potential future improvements or extensions to the work.	12.3 pts

Criteria	Ratings	Pts
Inadequate Attempt	Project includes a concluding section or markdown cell that is missing one or more of the following elements: 1. Summarizes the project. 2. Highlights key findings. 3. Suggests any potential future improvements or extensions to the work.	10.5 pts
Non-Performance	Non-performance.	0 pts

Total Points: 300

```
%matplotlib inline
# @title Bootstrap Google Colab {display-mode:"form"}

# @markdown # Use Google Drive
GOOGLE_DRIVE_FOLDER = "aai-511" # @param {type:"string"}

# @markdown # Technologies
GitHub = False  # @param {type:"boolean"}
OpenAI = False  # @param {type:"boolean"}
HuggingFace = True  # @param {type:"boolean"}
Kaggle = False  # @param {type:"boolean"}

# @markdown <br/>

# @markdown # Configure Repository
Repository = True # @param {type:"boolean"}
URL = "https://github.com/aai511-groupX/project.git" # @param {type:"string"}
REPO_PATH = "/content/aai-511/repos/project" # @param {type:"string"}

# @markdown <br/>

# @markdown # Install Packages
APT_PACKAGES = "fluidsynth fluid-soundfont-gm graphviz libfluidsynth3 tree"  #
@param {type:"string"}
PIP_PACKAGES = "autokeras black[jupyter] gensim isort midi2audio mido music21
pretty_midi py_midicsv scikit-learn scipy shap keras-tuner featuretools"  #
@param {type:"string"}

# REMOVE: Sample Folder
!rm -rf /content/sample_data

# SYMLINK: Google Drive folder to Files Pane (Top Level)
```

```
import contextlib, os, pathlib
from google.colab import drive
drive.mount('/content/drive')
drive_path = "/content/drive/MyDrive"
colab_notebooks_path = f"{drive_path}/Colab Notebooks"
project_path = f"{colab_notebooks_path}/{GOOGLE_DRIVE_FOLDER}"
!mkdir -p "{project_path}"
SHORTCUT = f"/content/{GOOGLE_DRIVE_FOLDER}"
if not os.path.exists(SHORTCUT):
    !ln -s "{project_path}" "{SHORTCUT}"
print(f"SHORTCUT --> {SHORTCUT}")

# ENSURE: Secrets
from google.colab import userdata
SECRETS = [
    ("GH_TOKEN", GitHub), #
https://github.com/settings/personal-access-tokens/new
    ("GITHUB_USERNAME", GitHub), # git config --global user.name
    ("GITHUB_EMAIL", GitHub), # git config --global user.email
    ("OPENAI_API_KEY", OpenAI), # https://platform.openai.com/api-keys
    ("HF_TOKEN", HuggingFace), #
https://huggingface.co/settings/tokens?new_token=true
    ("KAGGLE_USERNAME", Kaggle),
    ("KAGGLE_KEY", Kaggle), #
https://www.kaggle.com/settings#:~:text=Create%20New%20Token
]
for secret, enabled in SECRETS:
    if enabled:
        try:
            userdata.get(secret)
        except userdata.SecretNotFoundError:
            raise ValueError(f"Must set Google Colab secret: {secret}.")

# CONFIGURE: Environment
import os
os.environ['PIP_QUIET'] = '3'
os.environ['PIP_PROGRESS_BAR'] = 'off'
os.environ['PIP_ROOT_USER_ACTION'] = 'ignore'
os.environ['DEBIAN_FRONTEND'] = 'noninteractive'

# CONFIGURE: matplotlib
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300
plt.rcParams['savefig.dpi'] = 300

# DISABLE: Telemetry
os.environ['HF_HUB_DISABLE_TELEMETRY'] = '1'
```

```
os.environ['GRADIO_ANALYTICS_ENABLED'] = 'False'

# CONFIGURE: apt
# https://manpages.ubuntu.com/manpages/bionic/man5/apt.conf.5.html
APT_CONFIG = [
    'APT::Acquire::Retries "20";',
    'APT::Clean-Installed "true";',
    'APT::Get::Assume-Yes "true";',
    'APT::Get::Clean "always";',
    'APT::Get::Fix-Broken "true";',
    'APT::Install-Recommends "0";',
    'APT::Install-Suggests "0";',
    'APT::Sources::List::Disable-Auto-Refresh "true";',
    'Dpkg::Options "--force-confnew";',
    'Dpkg::Use-Pty "0";',
    'Quiet "2";',
]
with open('/etc/apt/apt.conf.d/01apt.conf', 'w') as file:
    for setting in APT_CONFIG:
        file.write(setting + '\n')

# INSTALL: uv
# https://github.com/astral-sh/uv
%pip install uv

# AUTHENTICATE: GitHub
# https://github.com/cli/cli/blob/trunk/docs/install_linux.md
if GitHub:
    !apt-get remove --purge gh > /dev/null
    !mkdir -p -m 755 /etc/apt/keyrings
    !wget -qO- https://cli.github.com/packages/githubcli-archive-keyring.gpg |
    tee /etc/apt/keyrings/githubcli-archive-keyring.gpg > /dev/null
    !chmod go+r /etc/apt/keyrings/githubcli-archive-keyring.gpg
    !echo "deb [arch=$(dpkg --print-architecture)
    signed-by=/etc/apt/keyrings/githubcli-archive-keyring.gpg]
    https://cli.github.com/packages stable main" | tee
    /etc/apt/sources.list.d/github-cli.list > /dev/null
    !apt-get update > /dev/null
    !apt-get install gh > /dev/null
    !gh auth login --hostname "github.com" --git-protocol https --with-token <<<
    {userdata.get("GH_TOKEN")}
    !git config --global pull.rebase false
    !git config --global credential.helper store
    !git config --global user.name {userdata.get("GITHUB_USERNAME")}
    !git config --global user.email {userdata.get("GITHUB_EMAIL")}

# AUTHENTICATE: OpenAI
```

```
# https://www.kaggle.com/settings
if OpenAI:
    os.environ["OPENAI_API_KEY"] = userdata.get("OPENAI_API_KEY")
    !uv pip install --system --quiet openai
    from openai import OpenAI
    client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

# AUTHENTICATE: Hugging Face
# https://huggingface.co/docs/huggingface_hub/en/quick-start#authentication
if HuggingFace:
    !uv pip install --system --quiet huggingface_hub[cli]
    !huggingface-cli login --add-to-git-credential --token
    {userdata.get("HF_TOKEN")} > /dev/null

# AUTHENTICATE: Kaggle
# https://www.kaggle.com/settings
if Kaggle:
    os.environ["KAGGLE_USERNAME"] = userdata.get("KAGGLE_USERNAME")
    os.environ["KAGGLE_KEY"] = userdata.get("KAGGLE_KEY")
    !uv pip install --system --quiet kaggle
    from kaggle.api.kaggle_api_extended import KaggleApi
    api = KaggleApi()
    api.authenticate()

# INSTALL: Dependencies
!apt-get install -qq --no-install-recommends {APT_PACKAGES}
!uv pip install --system --quiet {PIP_PACKAGES}

# HANDLE: Repository
if Repository:
    if REPO_PATH and os.path.exists(REPO_PATH):
        !cd {REPO_PATH} && git pull
    else:
        !git clone --recurse-submodules {GITHUB_URL} {REPO_PATH}

    !cd {REPO_PATH} && git submodule update --init --recursive
    !cd {REPO_PATH} && tree -L 2
    print(f"REPO --> {REPO_PATH}")

REPO = REPO_PATH
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
SHORTCUT --> /content/aai-511
Already up to date.
.
   checkpoints
```

```
   data
       external
       interim
       processed
       raw
   docs
       docs
       mkdocs.yml
       README.md
       Sample APA Professional Paper.pdf
   LICENSE
   Makefile
   models
   notebooks
       midi2csv.ipynb
       Project_Notebook_TeamX.ipynb
   pyproject.toml
   README.md
   references
   reports
       figures
   requirements.txt
   setup.cfg

13 directories, 11 files
REPO --> /content/aai-511/repos/project
```

```python
import os
import subprocess

# DEFINE: target composers
COMPOSERS = [
    "bach",
    "beethoven",
    "chopin",
    "mozart"
]

# DEFINE: raw midi data directory (organized by composer)
RAW_MIDI_DIR = f"{REPO}/data/raw/midi-classical-music/data/train"

# DEFINE: isolated directory for target composers
TARGET_COMPOSERS_DIR = f"{REPO}/data/raw/target-composers"
os.makedirs(TARGET_COMPOSERS_DIR, exist_ok=True)

# ISOLATE: target composers using rsync
for composer in COMPOSERS:
```

```
    source = f"{RAW_MIDI_DIR}/{composer}/"
    destination = f"{TARGET_COMPOSERS_DIR}/{composer}/"

    # Use rsync with --ignore-existing flag to skip existing files
    rsync_command = f"rsync -a --ignore-existing {source} {destination}"
    subprocess.run(rsync_command, shell=True, check=True)

    print(f"Copied new files for {composer}")

# COUNT: files per composer
total_files = 0
print(f"\n{'Composer':<15}{'Files':>10}")
print("-" * 25)
for composer in COMPOSERS:
    composer_dir = os.path.join(TARGET_COMPOSERS_DIR, composer)
    num_files = sum([len(files) for r, d, files in os.walk(composer_dir)])
    print(f"{composer:<15}{num_files:>10} files")
    total_files += num_files

print("-" * 25)
print(f"{'Total':<15}{total_files:>10} files")
print("-" * 25)
print(f"RAW_MIDI_DIR --> {RAW_MIDI_DIR}")
print(f"TARGET_COMPOSERS_DIR --> {TARGET_COMPOSERS_DIR}")
```

```
Composer            Files
-------------------------
bach                 1031 files
beethoven             220 files
chopin                136 files
mozart                258 files
-------------------------
Total                1645 files
-------------------------
RAW_MIDI_DIR -->
/content/aai-511/repos/project/data/raw/midi-classical-music/data/train
TARGET_COMPOSERS_DIR --> /content/aai-511/repos/project/data/raw/target-composers
```

```
import os
from tqdm.notebook import tqdm
import shutil

# FLATTEN: files into one directory for processing
INTERIM_MIDI_DIR = f"{REPO}/data/interim/midi"

# Delete the existing interim/midi folder if it exists
if os.path.exists(INTERIM_MIDI_DIR):
```

```
    shutil.rmtree(INTERIM_MIDI_DIR)

# Create a new interim/midi folder
!mkdir -p "{INTERIM_MIDI_DIR}"

for composer in tqdm(COMPOSERS, desc="Processing composers"):
    target_composer_dir = os.path.join(TARGET_COMPOSERS_DIR, composer)
    for root, dirs, files in os.walk(target_composer_dir):
        for filename in files:
            if filename.endswith(".mid"):
                old_filename = os.path.join(root, filename)
                new_filename = os.path.join(INTERIM_MIDI_DIR,
f"{composer}-{filename}")
                shutil.copy2(old_filename, new_filename)
            else:
                print(f"Skipping {root}/{filename} because it is not a MIDI
                file.")

interim_files = len([name for name in os.listdir(INTERIM_MIDI_DIR) if
name.endswith(".mid")])
print(f"{'Directory':<50}{'Files':>10}")
print("-" * 60)
print(f"{INTERIM_MIDI_DIR:<50}{interim_files:>10}")
print(f"{TARGET_COMPOSERS_DIR:<50}{total_files:>10}")
assert interim_files == total_files, f"FILE COUNT MISMATCH"
print("-" * 60)
print(f"INTERIM_MIDI_DIR --> {INTERIM_MIDI_DIR}")
```

```
Processing composers:   0%|          | 0/4 [00:00<?, ?it/s]

Directory                                                Files
------------------------------------------------------------
/content/aai-511/repos/project/data/interim/midi          1645
/content/aai-511/repos/project/data/raw/target-composers      1645
------------------------------------------------------------
INTERIM_MIDI_DIR --> /content/aai-511/repos/project/data/interim/midi
```

```
USE_WAVEFILES = False
if USE_WAVEFILES:
    import os
    import subprocess
    import wave
    from tqdm import tqdm
    import multiprocessing
    from functools import partial

    # DEFINE: Soundfont Path
```

```
    SOUNDFONT_PATH = "/usr/share/sounds/sf2/FluidR3_GM.sf2"

    # DEFINE: directories
    INTERIM_MIDI_DIR = f"{REPO}/data/interim/midi"
    INTERIM_WAV_DIR = f"{REPO}/data/interim/wav"
    os.makedirs(INTERIM_WAV_DIR, exist_ok=True)

    def convert_midi_to_wav(midi_file, midi_dir, wav_dir, soundfont_path):
        midi_path = os.path.join(midi_dir, midi_file)
        wav_path = os.path.join(wav_dir, midi_file.replace('.mid', '.wav'))

        # CONVERT: MIDI file to WAV (without pre-check for existence)
        subprocess.run(['fluidsynth', '-ni', soundfont_path, midi_path, '-F',
wav_path, '-r', '44100'])

        # CHECK: If conversion was successful (simpler check)
        if os.path.exists(wav_path) and os.path.getsize(wav_path) > 0:
            return f"CONVERTED: {midi_file}"
        else:
            return f"ERROR! Corrupted file: {wav_path}"


    # INITIALIZE: List of MIDI files
    midi_files = [f for f in os.listdir(INTERIM_MIDI_DIR) if f.endswith(".mid")]
    print(f"Total MIDI files found: {len(midi_files)}")

    # DETERMINE: Number of cores to use (all cores)
    num_cores = multiprocessing.cpu_count()
    print(f"Total number of CPU cores: {total_cores}")
    print(f"Number of cores being used: {num_cores}")

    # Create a partial function with fixed arguments
    convert_func = partial(convert_midi_to_wav,
                           midi_dir=INTERIM_MIDI_DIR,
                           wav_dir=INTERIM_WAV_DIR,
                           soundfont_path=SOUNDFONT_PATH)

    # CONVERT: MIDI to WAV using multiprocessing with tqdm
    with multiprocessing.Pool(processes=num_cores) as pool:
        results = list(tqdm(pool.imap(convert_func, midi_files, chunksize=10),
total=len(midi_files), desc="Converting MIDI to WAV"))

    # COUNT: Files in interim WAV directory
    wav_files = len([name for name in os.listdir(INTERIM_WAV_DIR) if
name.endswith(".wav")])
    print(f"\n{'Directory':<50}{'Files':>10}")
    print("-" * 60)
```

```
    print(f"{INTERIM_WAV_DIR:<50}{wav_files:>10}")
    assert wav_files == len(midi_files), f"FILE COUNT MISMATCH: Expected
    {len(midi_files)}, got {wav_files}"
    print("-" * 60)
    print(f"INTERIM_MIDI_DIR --> {INTERIM_MIDI_DIR}")
    print(f"INTERIM_WAV_DIR --> {INTERIM_WAV_DIR}")
    print(f"SOUNDFONT_PATH --> {SOUNDFONT_PATH}")

    # Print summary of results
    converted = sum(1 for r in results if r.startswith("CONVERTED"))
    existed = sum(1 for r in results if r.startswith("EXISTS"))
    errors = sum(1 for r in results if r.startswith("ERROR"))

    print("\nSummary:")
    print(f"Converted: {converted}")
    print(f"Already existed: {existed}")
    print(f"Errors: {errors}")
```

```
PREVIEW_AUDIO = False
if PREVIEW_AUDIO:
    import os
    import ipywidgets as widgets
    from IPython.display import display, Audio
    import subprocess

    # LIST: MIDI files in directory
    midi_files = sorted([f for f in os.listdir(INTERIM_MIDI_DIR) if
f.endswith('.mid')])

    # CREATE: dropdown widget to select MIDI file
    midi_dropdown = widgets.Dropdown(
        options=midi_files,
        description='MIDI File:',
        disabled=False,
    )

    # CREATE: button to load MIDI file
    load_button = widgets.Button(
        description='Load',
        disabled=False,
        button_style='primary',
        tooltip='Load MIDI file',
        icon='upload'
    )

    # CREATE: placeholder for audio widget and loading message
```

```
    audio_output = widgets.Output()
    loading_message = widgets.Label(value="", style={'font_weight': 'bold',
'color': 'red'})

    def load_midi(b):
        """Function to load and play MIDI file"""
        selected_file = midi_dropdown.value
        midi_path = os.path.join(INTERIM_MIDI_DIR, selected_file)

        # SHOW: loading message
        loading_message.value = "Loading..."

        # CONVERT: MIDI file to WAV
        wav_path = midi_path.replace('.mid', '.wav')
        subprocess.run(['fluidsynth', '-ni', SOUNDFONT_PATH, midi_path, '-F',
wav_path, '-r', '44100'])

        # UPDATE: audio widget
        with audio_output:
            audio_output.clear_output()
            display(Audio(wav_path))

        # HIDE: loading message
        loading_message.value = ""

    # ATTACH: load function to button
    load_button.on_click(load_midi)

    # DISPLAY: dropdown, button, loading message, and audio widget
    ui = widgets.VBox([midi_dropdown, load_button, loading_message,
audio_output])
    display(ui)
```

```
import os
import py_midicsv as pm
from tqdm.notebook import tqdm
import multiprocessing
from functools import partial
import mido

MIDI_FOLDER = f"{REPO}/data/interim/midi"
CSV_FOLDER = f"{REPO}/data/interim/csv"

if not os.path.exists(CSV_FOLDER):
    os.makedirs(CSV_FOLDER)
```

```
def is_midi_valid(midi_path):
    try:
        mido.MidiFile(midi_path)
        return True
    except Exception:
        return False

def midi_to_csv(filename, midi_folder=MIDI_FOLDER, csv_folder=CSV_FOLDER):
    midi_path = os.path.join(midi_folder, filename)
    csv_path = os.path.join(csv_folder, filename.replace(".mid", ".csv"))

    # Check if MIDI file is valid
    if not is_midi_valid(midi_path):
        return f"CORRUPTED MIDI: {filename}"

    # Check if CSV file already exists and is valid
    if os.path.exists(csv_path):
        try:
            with open(csv_path, 'r') as f:
                if f.read().strip():  # Check if file is not empty
                    return f"EXISTS: {filename}"
        except Exception:
            pass  # If there's any error reading the file, we'll recreate it

    try:
        # MIDI --> CSV
        csv_string_list = pm.midi_to_csv(midi_path)
        with open(csv_path, "w") as f:
            f.writelines(csv_string_list)
        return f"CONVERTED: {filename}"
    except Exception as e:
        return f"ERROR: {filename} - {str(e)}"

midi_files = [f for f in os.listdir(MIDI_FOLDER) if f.endswith(".mid")]

num_cores = max(1, multiprocessing.cpu_count() - 1)
print(f"Using {num_cores} CPU cores")

midi_to_csv_partial = partial(midi_to_csv, midi_folder=MIDI_FOLDER,
csv_folder=CSV_FOLDER)

with multiprocessing.Pool(num_cores) as pool:
    results = list(tqdm(pool.imap(midi_to_csv_partial, midi_files),
                        total=len(midi_files),
                        desc="Processing MIDI files"))

# Print summary
```

```
converted = sum(1 for r in results if r.startswith("CONVERTED"))
existed = sum(1 for r in results if r.startswith("EXISTS"))
errors = sum(1 for r in results if r.startswith("ERROR"))
corrupted = sum(1 for r in results if r.startswith("CORRUPTED MIDI"))

print("\nSummary:")
print(f"Converted: {converted}")
print(f"Already existed: {existed}")
print(f"Errors: {errors}")
print(f"Corrupted MIDI files: {corrupted}")

# Print errors and corrupted files if any
for result in results:
    if result.startswith("ERROR") or result.startswith("CORRUPTED MIDI"):
        print(result)
```

```
Using 95 CPU cores

Processing MIDI files:   0%|          | 0/1645 [00:00<?, ?it/s]


Summary:
Converted: 0
Already existed: 1645
Errors: 0
```

```
import os
import warnings
import pretty_midi
import numpy as np
import pandas as pd
from tqdm.notebook import tqdm
from scipy.stats import skew, kurtosis
from multiprocessing import Pool, cpu_count
import featuretools as ft

# Suppress warnings from pretty_midi
warnings.filterwarnings("ignore", category=RuntimeWarning, module="pretty_midi")

# Define paths
MIDI_FOLDER = os.path.join(REPO, "data", "interim", "midi")
CSV_FOLDER = os.path.join(REPO, "data", "processed")
PIANO_ROLL_FOLDER = os.path.join(REPO, "data", "interim", "piano_roll")

# Define the composers we're interested in
TARGET_COMPOSERS = ['beethoven', 'mozart', 'bach', 'chopin']
```

```
def extract_features(midi_file):
    try:
        midi_data = pretty_midi.PrettyMIDI(os.path.join(MIDI_FOLDER, midi_file))

        total_duration = midi_data.get_end_time()
        num_instruments = len(midi_data.instruments)
        notes = [note for instrument in midi_data.instruments for note in
instrument.notes]
        pitches = [note.pitch for note in notes]
        velocities = [note.velocity for note in notes]
        durations = [note.end - note.start for note in notes]

        composer = next((c for c in TARGET_COMPOSERS if c in midi_file.lower()),
"unknown")

        feature = {
            'file_name': midi_file,
            'composer': composer,
            'total_duration': total_duration,
            'num_instruments': num_instruments,
            'num_notes': len(notes),
            'avg_pitch': np.mean(pitches) if pitches else 0,
            'pitch_std': np.std(pitches) if pitches else 0,
            'pitch_range': np.ptp(pitches) if pitches else 0,
            'avg_velocity': np.mean(velocities) if velocities else 0,
            'velocity_std': np.std(velocities) if velocities else 0,
            'avg_duration': np.mean(durations) if durations else 0,
            'duration_std': np.std(durations) if durations else 0,
            'duration_range': np.ptp(durations) if durations else 0,
            'tempo': midi_data.estimate_tempo(),
            'note_density': len(notes) / total_duration if total_duration > 0
            else 0,
            'dynamic_range': np.ptp(velocities) if velocities else 0,
            'velocity_variance': np.var(velocities) if velocities else 0,
            'time_signature': midi_data.time_signature_changes[0].numerator if
            midi_data.time_signature_changes else 4,
            'key': midi_data.key_signature_changes[0].key_number if
            midi_data.key_signature_changes else 0,
            'mode': 'major' if midi_data.key_signature_changes and
            midi_data.key_signature_changes[0].key_number >= 0 else 'minor',
            'key_changes': len(midi_data.key_signature_changes),
            'tempo_changes': len(midi_data.get_tempo_changes()[1]),
            'pitch_entropy': -np.sum([(pitches.count(p)/len(pitches)) *
            np.log2(pitches.count(p)/len(pitches)) for p in set(pitches)]) if
            pitches else 0,
            'pitch_skewness': skew(pitches) if len(pitches) > 2 else 0,
            'pitch_kurtosis': kurtosis(pitches) if len(pitches) > 2 else 0,
```

```
        }

        # Generate piano roll
        piano_roll = midi_data.get_piano_roll(fs=100)  # 100 Hz sampling rate
        piano_roll = piano_roll[:, :1000]  # Truncate or pad to 10 seconds
        if piano_roll.shape[1] < 1000:
            piano_roll = np.pad(piano_roll, ((0, 0), (0, 1000 -
piano_roll.shape[1])))

        return feature, piano_roll

    except Exception as e:
        print(f"Error processing {midi_file}: {str(e)}")
        return None, None

# Get list of MIDI files for target composers
midi_files = [f for f in os.listdir(MIDI_FOLDER) if f.endswith('.mid') or
f.endswith('.midi')]
midi_files = [f for f in midi_files if any(composer in f.lower() for composer in
TARGET_COMPOSERS)]

print(f"Total MIDI files found: {len(midi_files)}")

print("Extracting features...")
num_cores = max(1, cpu_count() - 1)  # Leave one core free
with Pool(processes=num_cores) as pool:
    results = list(tqdm(pool.imap(extract_features, midi_files),
total=len(midi_files)))

# Separate results
features = []
piano_rolls = []

for feature, piano_roll in results:
    if feature is not None:
        features.append(feature)
        piano_rolls.append(piano_roll)

# Convert to DataFrame
features_df = pd.DataFrame(features)

print("Generating additional features with featuretools...")
es = ft.EntitySet(id="music")
es = es.add_dataframe(
    dataframe_name="features",
    dataframe=features_df,
    index="file_name"
```

```
)

# List available primitives
print("Available transform primitives:")
transform_primitives = ft.list_primitives().query("type ==
'transform'")['name'].tolist()
print(transform_primitives)

# Select a subset of available primitives
selected_primitives = ['absolute', 'divide_by_feature', 'multiply_numeric',
'add_numeric']

# Generate new features
feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_dataframe_name="features",
                                      trans_primitives=selected_primitives,
                                      max_depth=2,
                                      features_only=False,
                                      verbose=True)

# Merge new features with original features
features_df = pd.concat([features_df, feature_matrix], axis=1)

# Remove any duplicate columns
features_df = features_df.loc[:,~features_df.columns.duplicated()]

# Convert to numpy arrays
columns_to_drop = ['file_name', 'composer']
columns_to_drop = [col for col in columns_to_drop if col in features_df.columns]

X = features_df.drop(columns_to_drop, axis=1, errors='ignore').values
y = features_df['composer'].values
piano_rolls = np.array(piano_rolls)

print("Saving processed data...")
os.makedirs(CSV_FOLDER, exist_ok=True)
os.makedirs(PIANO_ROLL_FOLDER, exist_ok=True)
features_df.to_csv(os.path.join(CSV_FOLDER, 'features.csv'), index=False)
np.save(os.path.join(CSV_FOLDER, 'labels.npy'), y)
np.save(os.path.join(PIANO_ROLL_FOLDER, 'piano_rolls.npy'), piano_rolls)

print("Feature extraction complete.")
print(f"Extracted features for {len(features)} files.")
print(f"Features shape: {X.shape}")
print(f"Labels shape: {y.shape}")
print(f"Piano rolls shape: {piano_rolls.shape}")
```

```
Total MIDI files found: 1645
Extracting features...

  0%|          | 0/1645 [00:00<?, ?it/s]

Error processing beethoven-anhang_14_3.mid: Could not decode key with 3 flats and
mode 255
Error processing
mozart-piano_sonatas-nueva_carpeta-k281_piano_sonata_n03_3mov.mid: Could not
decode key with 2 flats and mode 2
Generating additional features with featuretools...

/usr/local/lib/python3.10/dist-packages/woodwork/type_sys/utils.py:33:
UserWarning: Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-expected,
please specify a format.
  pd.to_datetime(
/usr/local/lib/python3.10/dist-packages/woodwork/type_sys/utils.py:33:
UserWarning: Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-expected,
please specify a format.
  pd.to_datetime(
/usr/local/lib/python3.10/dist-packages/woodwork/type_sys/utils.py:33:
UserWarning: Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-expected,
please specify a format.
  pd.to_datetime(
/usr/local/lib/python3.10/dist-packages/woodwork/type_sys/utils.py:33:
UserWarning: Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-expected,
please specify a format.
  pd.to_datetime(
/usr/local/lib/python3.10/dist-packages/woodwork/type_sys/utils.py:33:
UserWarning: Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-expected,
please specify a format.
  pd.to_datetime(
/usr/local/lib/python3.10/dist-packages/woodwork/type_sys/utils.py:33:
UserWarning: Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-expected,
please specify a format.
  pd.to_datetime(
/usr/local/lib/python3.10/dist-packages/featuretools/synthesis/deep_feature_synthesis.py:169:
UserWarning: Only one dataframe in entityset, changing max_depth to 1 since
deeper features cannot be created
  warnings.warn(

Available transform primitives:
```

```
['one_digit_postal_code', 'is_first_week_of_month', 'season', 'time_since',
'time_since_previous', 'is_free_email_domain', 'exponential_weighted_average',
'cumulative_time_since_last_true', 'cumulative_time_since_last_false',
'file_extension', 'negate', 'nth_week_of_month', 'second', 'title_word_count',
'day_of_year', 'date_to_holiday', 'divide_by_feature', 'multiply_numeric',
'is_leap_year', 'expanding_count', 'less_than_scalar', 'greater_than_equal_to',
'week', 'isin', 'rolling_trend', 'two_digit_postal_code', 'number_of_mentions',
'absolute_diff', 'diff_datetime', 'url_to_tld', 'is_federal_holiday',
'number_of_words_in_quotes', 'days_in_month', 'sine', 'cum_min',
'full_name_to_title', 'is_weekend', 'is_lunch_time', 'is_working_hours',
'mean_characters_per_word', 'modulo_by_feature', 'greater_than_equal_to_scalar',
'date_to_time_zone', 'full_name_to_last_name', 'not', 'modulo_numeric_scalar',
'expanding_std', 'number_of_unique_words', 'not_equal_scalar',
'cityblock_distance', 'is_month_end', 'exponential_weighted_std', 'square_root',
'median_word_length', 'multiply_boolean', 'greater_than_scalar', 'percentile',
'subtract_numeric', 'day', 'less_than_equal_to', 'cum_sum', 'equal',
'rolling_mean', 'is_month_start', 'less_than_equal_to_scalar',
'num_unique_separators', 'punctuation_count', 'month', 'tangent', 'add_numeric',
'year', 'savgol_filter', 'greater_than', 'absolute', 'num_characters',
'natural_logarithm', 'scalar_subtract_numeric_feature', 'quarter',
'full_name_to_first_name', 'is_quarter_end', 'multiply_numeric_boolean',
'distance_to_holiday', 'haversine', 'num_words', 'rate_of_change', 'cum_max',
'upper_case_count', 'rolling_min', 'divide_numeric_scalar', 'add_numeric_scalar',
'not_equal', 'rolling_count', 'lag', 'less_than',
'exponential_weighted_variance', 'expanding_mean', 'age', 'same_as_previous',
'expanding_trend', 'number_of_hashtags', 'multiply_numeric_scalar',
'is_quarter_start', 'email_address_to_domain', 'cum_mean', 'rolling_std',
'equal_scalar', 'and', 'rolling_max', 'is_year_end', 'upper_case_word_count',
'geomidpoint', 'cum_count', 'weekday', 'or', 'latitude', 'expanding_min',
'url_to_protocol', 'cosine', 'expanding_max', 'numeric_lag', 'is_in_geobox',
'is_null', 'minute', 'whitespace_count', 'percent_change', 'hour',
'rolling_outlier_count', 'total_word_length', 'url_to_domain', 'modulo_numeric',
'subtract_numeric_scalar', 'longitude', 'number_of_common_words', 'diff',
'count_string', 'divide_numeric', 'is_year_start', 'part_of_day']
Built 530 features
Elapsed: 00:00 | Progress: 100%|
Saving processed data...
Feature extraction complete.
Extracted features for 1643 files.
Features shape: (1643, 529)
Labels shape: (1643,)
Piano rolls shape: (1643, 128, 1000)
```

```python
import os
import numpy as np
import pandas as pd
import tensorflow as tf
```

```
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Define paths
CSV_FOLDER = os.path.join(REPO, "data", "processed")
PIANO_ROLL_FOLDER = os.path.join(REPO, "data", "interim", "piano_roll")
CHECKPOINT_DIR = os.path.join(REPO, 'checkpoints')
os.makedirs(CHECKPOINT_DIR, exist_ok=True)
ENSEMBLE_MODEL_PATH = os.path.join(CHECKPOINT_DIR, 'ensemble_model.keras')

print("Loading data...")
features_df = pd.read_csv(os.path.join(CSV_FOLDER, 'features.csv'))

# Load labels with error handling
try:
    labels = np.load(os.path.join(CSV_FOLDER, 'labels.npy'), allow_pickle=True)
except ValueError:
    print("Error loading labels.npy. Falling back to 'composer' column from
    features_df.")
    labels = features_df['composer'].values

piano_rolls = np.load(os.path.join(PIANO_ROLL_FOLDER, 'piano_rolls.npy'))
print("Data loaded successfully!")

# Print data information
print(f"Number of samples: {len(features_df)}")
print(f"Number of features: {features_df.shape[1] - 2}")
print(f"Number of unique labels: {len(np.unique(labels))}")
print(f"Piano roll shape: {piano_rolls.shape}")

# Check for missing values
print("\nMissing values in features_df:")
print(features_df.isnull().sum())

# Display first few rows of features
print("\nFirst few rows of features_df:")
print(features_df.head())

# Display unique labels
print("\nUnique labels:")
print(np.unique(labels))
```

```
print("Preparing features...")
X = features_df.drop(['file_name', 'composer'], axis=1)
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
X_numeric = X[numeric_features]

# Check for and handle infinite values
inf_columns = X_numeric.columns[np.isinf(X_numeric).any()].tolist()
if inf_columns:
    print(f"Columns with infinite values: {inf_columns}")
    X_numeric = X_numeric.replace([np.inf, -np.inf], np.nan)

# Check for and handle very large values
max_value = np.finfo(np.float64).max
large_value_columns = X_numeric.columns[(X_numeric > max_value).any()].tolist()
if large_value_columns:
    print(f"Columns with very large values: {large_value_columns}")
    X_numeric = X_numeric.clip(upper=max_value)

# Handle NaN values
nan_columns = X_numeric.columns[X_numeric.isna().any()].tolist()
if nan_columns:
    print(f"Columns with NaN values: {nan_columns}")
    X_numeric = X_numeric.fillna(X_numeric.mean())

scaler = StandardScaler()
X_preprocessed = scaler.fit_transform(X_numeric)
print("Features prepared successfully!")

print("Encoding labels...")
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(labels)
print("Labels encoded successfully!")

print("Splitting data...")
X_train, X_test, X_train_piano, X_test_piano, y_train, y_test = train_test_split(
    X_preprocessed, piano_rolls, y, test_size=0.2, random_state=42
)
print("Data split successfully!")

def build_lstm_model(input_shape, num_classes):
    model = keras.Sequential([
        keras.layers.Input(shape=(input_shape,)),
        keras.layers.Reshape((-1, 1)),
        keras.layers.LSTM(128, return_sequences=True),
        keras.layers.LSTM(64),
        keras.layers.Dense(num_classes, activation='softmax')
    ])
```

```
    return model

def build_cnn_model(input_shape, num_classes):
    model = keras.Sequential([
        keras.layers.Input(shape=input_shape),
        keras.layers.Conv2D(32, (3, 3), activation='relu'),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Conv2D(64, (3, 3), activation='relu'),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Flatten(),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(num_classes, activation='softmax')
    ])
    return model

class EnsembleModel(keras.Model):
    def __init__(self, lstm_model, cnn_model, num_classes):
        super(EnsembleModel, self).__init__()
        self.lstm_model = lstm_model
        self.cnn_model = cnn_model
        self.dense = keras.layers.Dense(num_classes, activation='softmax')

    def call(self, inputs):
        lstm_input, cnn_input = inputs
        lstm_output = self.lstm_model(lstm_input)
        cnn_output = self.cnn_model(cnn_input)
        combined = tf.concat([lstm_output, cnn_output], axis=1)
        return self.dense(combined)

print("Building models...")
num_classes = len(np.unique(y))
lstm_model = build_lstm_model(X_train.shape[1], num_classes)
cnn_model = build_cnn_model((128, 1000, 1), num_classes)

print("Creating and compiling ensemble model...")
ensemble_model = EnsembleModel(lstm_model, cnn_model, num_classes)
ensemble_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

print("\nTraining ensemble model...")
history = ensemble_model.fit(
    [X_train, X_train_piano.reshape(-1, 128, 1000, 1)],
    y_train,
    epochs=50,
    batch_size=64,
    validation_split=0.2,
    callbacks=[
```

```
        keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
        keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=3),
        keras.callbacks.ModelCheckpoint(ENSEMBLE_MODEL_PATH, save_best_only=True)
    ],
    verbose=1
)

print("\nEvaluating ensemble model...")
y_pred = ensemble_model.predict([X_test, X_test_piano.reshape(-1, 128, 1000, 1)])
y_pred_classes = np.argmax(y_pred, axis=1)

print("\nEnsemble Model Results:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_classes):.4f}')
print(f'Precision: {precision_score(y_test, y_pred_classes,
average="weighted"):.4f}')
print(f'Recall: {recall_score(y_test, y_pred_classes, average="weighted"):.4f}')
print(f'F1 Score: {f1_score(y_test, y_pred_classes, average="weighted"):.4f}')

print("Generating confusion matrix...")
cm = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Ensemble Model')
plt.savefig(os.path.join(REPO, 'confusion_matrix_ensemble.png'))
plt.close()

print("Generating training history plot...")
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

```
plt.tight_layout()
plt.savefig(os.path.join(REPO, 'training_history.png'))
plt.close()

print("\nTraining completed. Model saved and plots generated.")
```

```
Loading data...
Data loaded successfully!
Number of samples: 1643
Number of features: 529
Number of unique labels: 4
Piano roll shape: (1643, 128, 1000)

Missing values in features_df:
file_name                            0
composer                             0
total_duration                       0
num_instruments                      0
num_notes                            0
                                    ..
time_signature * velocity_std        0
time_signature * velocity_variance   0
total_duration * velocity_std        0
total_duration * velocity_variance   0
velocity_std * velocity_variance     0
Length: 531, dtype: int64

First few rows of features_df:
                             file_name composer  total_duration  \
0  bach-bwv001-_400_chorales-015105b.mid     bach           27.50
1  bach-bwv001-_400_chorales-010107b.mid     bach           32.50
2  bach-bwv001-_400_chorales-005206b.mid     bach           35.40
3  bach-bwv001-_400_chorales-008906b.mid     bach           33.75
4  bach-bwv001-_400_chorales-007706b.mid     bach           45.00

   num_instruments  num_notes  avg_pitch  pitch_std  pitch_range  \
0                4        168  62.779762   7.627040           33
1                6        314  60.936306   6.845131           36
2                6        323  61.835913   7.336305           33
3                4        193  61.611399   8.718943           35
4                4        255  61.576471   7.523104           33

   avg_velocity  velocity_std  ...  tempo_changes * time_signature  \
0          96.0           0.0  ...                             4.0
1          96.0           0.0  ...                             4.0
2          96.0           0.0  ...                             4.0
3          96.0           0.0  ...                             4.0
```

```
4          96.0           0.0  ...                              4.0

   tempo_changes * total_duration  tempo_changes * velocity_std  \
0                           27.50                           0.0
1                           32.50                           0.0
2                           35.40                           0.0
3                           33.75                           0.0
4                           45.00                           0.0

   tempo_changes * velocity_variance  time_signature * total_duration  \
0                                0.0                            110.0
1                                0.0                            130.0
2                                0.0                            141.6
3                                0.0                            135.0
4                                0.0                            180.0

   time_signature * velocity_std  time_signature * velocity_variance  \
0                            0.0                                 0.0
1                            0.0                                 0.0
2                            0.0                                 0.0
3                            0.0                                 0.0
4                            0.0                                 0.0

   total_duration * velocity_std  total_duration * velocity_variance  \
0                            0.0                                 0.0
1                            0.0                                 0.0
2                            0.0                                 0.0
3                            0.0                                 0.0
4                            0.0                                 0.0

  velocity_std * velocity_variance
0                              0.0
1                              0.0
2                              0.0
3                              0.0
4                              0.0

[5 rows x 531 columns]

Unique labels:
['bach' 'beethoven' 'chopin' 'mozart']
Preparing features...
Columns with infinite values: ['1 / dynamic_range', '1 / key', '1 / key_changes',
'1 / velocity_std', '1 / velocity_variance']
Columns with NaN values: ['1 / dynamic_range', '1 / key', '1 / key_changes', '1 /
velocity_std', '1 / velocity_variance']
Features prepared successfully!
Encoding labels...
```

```
Labels encoded successfully!
Splitting data...
Data split successfully!
Building models...
Creating and compiling ensemble model...

Training ensemble model...
Epoch 1/50
17/17             42s 2s/step - accuracy: 0.5654 - loss: 1.2175 - val_accuracy:
0.6350 - val_loss: 1.0944 - learning_rate: 0.0010
Epoch 2/50
17/17             36s 2s/step - accuracy: 0.7036 - loss: 1.0523 - val_accuracy:
0.6502 - val_loss: 1.0660 - learning_rate: 0.0010
Epoch 3/50
17/17             36s 2s/step - accuracy: 0.6879 - loss: 1.0412 - val_accuracy:
0.6464 - val_loss: 1.0461 - learning_rate: 0.0010
Epoch 4/50
17/17             36s 2s/step - accuracy: 0.7083 - loss: 1.0085 - val_accuracy:
0.6578 - val_loss: 1.0257 - learning_rate: 0.0010
Epoch 5/50
17/17             36s 2s/step - accuracy: 0.7031 - loss: 0.9988 - val_accuracy:
0.6616 - val_loss: 1.0118 - learning_rate: 0.0010
Epoch 6/50
17/17             36s 2s/step - accuracy: 0.7142 - loss: 0.9675 - val_accuracy:
0.6578 - val_loss: 0.9995 - learning_rate: 0.0010
Epoch 7/50
17/17             36s 2s/step - accuracy: 0.6832 - loss: 0.9745 - val_accuracy:
0.6540 - val_loss: 0.9920 - learning_rate: 0.0010
Epoch 8/50
17/17             36s 2s/step - accuracy: 0.6986 - loss: 0.9525 - val_accuracy:
0.6540 - val_loss: 0.9735 - learning_rate: 0.0010
Epoch 9/50
17/17             34s 2s/step - accuracy: 0.6973 - loss: 0.9444 - val_accuracy:
0.6616 - val_loss: 0.9735 - learning_rate: 0.0010
Epoch 10/50
17/17             36s 2s/step - accuracy: 0.7051 - loss: 0.9330 - val_accuracy:
0.6616 - val_loss: 0.9606 - learning_rate: 0.0010
Epoch 11/50
17/17             36s 2s/step - accuracy: 0.7239 - loss: 0.8970 - val_accuracy:
0.6616 - val_loss: 0.9515 - learning_rate: 0.0010
Epoch 12/50
17/17             36s 2s/step - accuracy: 0.7170 - loss: 0.8941 - val_accuracy:
0.6616 - val_loss: 0.9495 - learning_rate: 0.0010
Epoch 13/50
17/17             34s 2s/step - accuracy: 0.6914 - loss: 0.9309 - val_accuracy:
0.6578 - val_loss: 0.9517 - learning_rate: 0.0010
Epoch 14/50
```

```
17/17            36s 2s/step - accuracy: 0.7080 - loss: 0.8804 - val_accuracy:
0.6578 - val_loss: 0.9237 - learning_rate: 0.0010
Epoch 15/50
17/17            36s 2s/step - accuracy: 0.6998 - loss: 0.8846 - val_accuracy:
0.6426 - val_loss: 0.9033 - learning_rate: 0.0010
Epoch 16/50
17/17            34s 2s/step - accuracy: 0.6900 - loss: 0.8771 - val_accuracy:
0.6426 - val_loss: 0.9058 - learning_rate: 0.0010
Epoch 17/50
17/17            36s 2s/step - accuracy: 0.6861 - loss: 0.8623 - val_accuracy:
0.6616 - val_loss: 0.8818 - learning_rate: 0.0010
Epoch 18/50
17/17            36s 2s/step - accuracy: 0.6789 - loss: 0.8603 - val_accuracy:
0.6540 - val_loss: 0.8733 - learning_rate: 0.0010
Epoch 19/50
17/17            36s 2s/step - accuracy: 0.6633 - loss: 0.8607 - val_accuracy:
0.6654 - val_loss: 0.8653 - learning_rate: 0.0010
Epoch 20/50
17/17            34s 2s/step - accuracy: 0.7077 - loss: 0.8274 - val_accuracy:
0.6540 - val_loss: 0.8692 - learning_rate: 0.0010
Epoch 21/50
17/17            36s 2s/step - accuracy: 0.6975 - loss: 0.8131 - val_accuracy:
0.6578 - val_loss: 0.8498 - learning_rate: 0.0010
Epoch 22/50
17/17            34s 2s/step - accuracy: 0.7108 - loss: 0.8101 - val_accuracy:
0.6464 - val_loss: 0.8595 - learning_rate: 0.0010
Epoch 23/50
17/17            34s 2s/step - accuracy: 0.6833 - loss: 0.8107 - val_accuracy:
0.6578 - val_loss: 0.8695 - learning_rate: 0.0010
Epoch 24/50
17/17            34s 2s/step - accuracy: 0.7045 - loss: 0.8208 - val_accuracy:
0.6578 - val_loss: 0.8578 - learning_rate: 0.0010
Epoch 25/50
17/17            34s 2s/step - accuracy: 0.6863 - loss: 0.8285 - val_accuracy:
0.6502 - val_loss: 0.8502 - learning_rate: 5.0000e-04
Epoch 26/50
17/17            36s 2s/step - accuracy: 0.7164 - loss: 0.7982 - val_accuracy:
0.6502 - val_loss: 0.8456 - learning_rate: 5.0000e-04
Epoch 27/50
17/17            36s 2s/step - accuracy: 0.7071 - loss: 0.8039 - val_accuracy:
0.6616 - val_loss: 0.8377 - learning_rate: 5.0000e-04
Epoch 28/50
17/17            36s 2s/step - accuracy: 0.7043 - loss: 0.8025 - val_accuracy:
0.6540 - val_loss: 0.8294 - learning_rate: 5.0000e-04
Epoch 29/50
17/17            34s 2s/step - accuracy: 0.7174 - loss: 0.7601 - val_accuracy:
0.6540 - val_loss: 0.8416 - learning_rate: 5.0000e-04
Epoch 30/50
```

```
17/17            34s 2s/step - accuracy: 0.6986 - loss: 0.7817 - val_accuracy:
0.6540 - val_loss: 0.8374 - learning_rate: 5.0000e-04
Epoch 31/50
17/17            34s 2s/step - accuracy: 0.6832 - loss: 0.8092 - val_accuracy:
0.6540 - val_loss: 0.8415 - learning_rate: 5.0000e-04
Epoch 32/50
17/17            34s 2s/step - accuracy: 0.7264 - loss: 0.7552 - val_accuracy:
0.6464 - val_loss: 0.8376 - learning_rate: 2.5000e-04
Epoch 33/50
17/17            34s 2s/step - accuracy: 0.7113 - loss: 0.7734 - val_accuracy:
0.6654 - val_loss: 0.8325 - learning_rate: 2.5000e-04

Evaluating ensemble model...
11/11            4s 322ms/step

Ensemble Model Results:
Accuracy: 0.7629
Precision: 0.6671
Recall: 0.7629
F1 Score: 0.7021
Generating confusion matrix...

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Generating training history plot...

Training completed. Model saved and plots generated.
```