bird_song_dataset

May 27, 2024

[33]:
```
import os
import pandas as pd
import torch
from torch.utils.data import Dataset
import librosa
import librosa.display
import numpy as np
from torchvision.transforms import ToTensor
from sklearn.preprocessing import LabelEncoder
import random
from collections import defaultdict
from IPython.display import Audio, display
import plotly.express as px
```

[34]:
```
# Class to manage paths related to data, models, and results storage
class DataPaths:
    # Establish base directories relative to the script location
    def __init__(self):
        self.script_dir = os.path.dirname(os.path.abspath(__file__))
        self.data_dir = os.path.join(self.script_dir, os.pardir, 'data')
        self.models_dir = os.path.join(self.script_dir, os.pardir, 'models')
        self.results_dir = os.path.join(self.script_dir, os.pardir, 'results')
        # Dictionary storing relevant file and directory paths
        self.paths = {
            'csv_file_path': os.path.join(self.data_dir, 'bird_songs_metadata.
 ↪csv'),
            'wav_files_dir': os.path.join(self.data_dir, 'wavfiles'),
            'models_dir': self.models_dir,
            'results_dir': self.results_dir,
            'runs_dir': os.path.join(self.results_dir, 'runs')
        }

    # Method to retrieve the paths dictionary
    def get_paths(self):
        return self.paths
```

[35]:
```python
# Class to manage device allocation (CPU/GPU)
class DeviceManager:
    # Automatically determine and set the computing device
    def __init__(self):
        self.device = self.determine_device()

    # Device determination based on availability of hardware acceleration
    def determine_device(self):
        if torch.backends.mps.is_available():
            print("Using MPS (Apple Silicon GPU)")
            return torch.device("mps")
        elif torch.cuda.is_available():
            print("CUDA is available. Using GPU.")
            return torch.device("cuda")
        else:
            print("Neither MPS nor CUDA is available. Using CPU.")
            return torch.device("cpu")
```

[36]:
```python
# Dataset class to handle bird song data
class BirdSongDataset(Dataset):
    def __init__(self, csv_file, root_dir, transform=None):
        self.bird_metadata = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform
        self.label_encoder = LabelEncoder()
        # Encode bird species labels numerically
        self.labels = self.label_encoder.fit_transform(self.bird_metadata.iloc[:
 ↪, 4])

    def __len__(self):
        # Return the total number of samples in the dataset
        return len(self.bird_metadata)

    def __getitem__(self, idx):
        # Retrieve single data point from the dataset
        if torch.is_tensor(idx):
            idx = idx.tolist()

        # Load and transform the audio file into a Mel-spectrogram
        audio_path = os.path.join(self.root_dir, self.bird_metadata.iloc[idx,␣
 ↪-1])
        audio, sr = librosa.load(audio_path, sr=None)
        mel_spec = librosa.feature.melspectrogram(y=audio, sr=sr)
        mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)

        if self.transform:
            mel_spec_db = self.transform(mel_spec_db)
```

```
        mel_spec_db_tensor = torch.from_numpy(mel_spec_db).float()
        mel_spec_db_tensor = mel_spec_db_tensor.unsqueeze(0)

        label = self.labels[idx]
        label_tensor = torch.tensor(label, dtype=torch.long)
        label_string = self.label_encoder.inverse_transform([label])[0]

        sample = {'spectrogram': mel_spec_db_tensor, 'label': label_tensor,␣
↪'label_string': label_string, 'audio_path': audio_path}
        return sample

    def get_label_encoder(self):
        # Return the label encoder instance
        return self.label_encoder

    def plot_class_distribution(self):
        # Plot the distribution of classes in the dataset using an interactive␣
↪bar chart
        class_counts = pd.Series(self.labels).value_counts().reset_index()
        class_counts.columns = ['class_label', 'count']
        class_counts['class_label'] = self.label_encoder.
↪inverse_transform(class_counts['class_label'])

        fig = px.bar(class_counts, x='class_label', y='count',
                     title="Class Distribution",
                     color="class_label",
                     labels={'count': 'Frequency', 'class_label': 'Class Label'})
        return fig


    def sample_n_examples_per_class(self, n):
        # Sample 'n' examples from each class
        indices_per_class = defaultdict(list)
        for idx, label in enumerate(self.labels):
            indices_per_class[label].append(idx)

        sampled_indices = []
        for label, indices in indices_per_class.items():
            if len(indices) >= n:
                sampled_indices.extend(random.sample(indices, n))
            else:
                print(f"Not enough samples in class {label} for sampling {n}␣
↪examples. Only using available {len(indices)} samples.")
                sampled_indices.extend(indices)

        return [self[i] for i in sampled_indices]
```

```
    def visualize_spectrogram_plotly(self, spectrogram, label_string, sr=22050):
        # Visualize a spectrogram using an interactive Plotly figure
        spectrogram_2d = spectrogram.squeeze(0)
        hop_length = 512
        y_axis = librosa.mel_frequencies(n_mels=spectrogram_2d.shape[0],␣
↪fmin=0, fmax=sr/2)
        x_axis = np.linspace(0, spectrogram_2d.shape[1] * hop_length / sr,␣
↪num=spectrogram_2d.shape[1])

        fig = px.imshow(spectrogram_2d,
                        labels={'x': "Time (s)", 'y': "Frequency (Hz)", 'color':
↪ "Amplitude (dB)"},
                        x=x_axis,
                        y=y_axis,
                        aspect='auto',
                        origin='lower',
                        color_continuous_scale='viridis')

        fig.update_layout(title=f'Spectrogram of {label_string}',
                        xaxis_title='Time (s)',
                        yaxis_title='Frequency (Hz)')
        return fig

    def display_samples(self, num_samples=1):
        """
        Display a specified number of bird song examples, with labels,
        spectrograms, and audio players. Intended for Jupyter notebooks.
        """
        from IPython.display import display, Audio
        samples = self.sample_n_examples_per_class(num_samples)
        for i, sample in enumerate(samples):
            print(f"Example {i+1}: Label - {sample['label_string']}␣
↪({sample['label']})")
            fig = self.visualize_spectrogram_plotly(sample['spectrogram'],␣
↪sample['label_string'], sr=22050)
            fig.show()
            display(Audio(sample['audio_path']))
            print(f"\n{'-'*80}\n")

    def display_samples_streamlit(self, num_samples=1):
        """
        Display a specified number of bird song examples, with labels,
        spectrograms, and audio players. Intended for Streamlit.
        """
        samples = self.sample_n_examples_per_class(num_samples)
        sample_data = []
```

```
        for i, sample in enumerate(samples):
            spectrogram_fig = self.
 ↪visualize_spectrogram_plotly(sample['spectrogram'], sample['label_string'],␣
 ↪sr=22050)
            audio_clip = sample['audio_path']
            label_info = f"Example {i+1}: Label - {sample['label_string']}␣
 ↪({sample['label']})"
            sample_data.append({'fig': spectrogram_fig, 'audio': audio_clip,␣
 ↪'label_info': label_info})
        return sample_data
```

[37]:
```
# Get dynamic paths
data_paths = DataPaths()
paths = data_paths.get_paths()
print(paths.keys())
```

```
dict_keys(['csv_file_path', 'wav_files_dir', 'models_dir', 'results_dir',
'runs_dir'])
```

[38]:
```
# Instantiate dataset class
dataset = BirdSongDataset(csv_file=csv_file_path, root_dir=wav_files_dir)
print(f"Dataset size: {len(dataset)}")
```

```
Dataset size: 5422
```

[39]:
```
# Determine accelerator device
device_manager = DeviceManager()
current_device = device_manager.device
current_device
```

```
Using MPS (Apple Silicon GPU)
```

[39]:
```
device(type='mps')
```

[40]:
```
# Display the class distribution chart
class_dist_fig = dataset.plot_class_distribution()
class_dist_fig.show()
```

[12]:
```
# Display samples with spectrograms and audio
samples = dataset.sample_n_examples_per_class(1)
for i, sample in enumerate(samples):
    print(f"Example {i+1}: Label - {sample['label_string']}␣
 ↪({sample['label']})")

    # Visualize the spectrogram
```

```
    spectrogram_fig = dataset.
↪visualize_spectrogram_plotly(sample['spectrogram'], sample['label_string'],␣
↪sr=22050)
    spectrogram_fig.show()

    # Display the audio player
    display(Audio(sample['audio_path']))

    print(f"\n{'-'*80}\n")
```

```
Example 1: Label - Bewick's Wren (1)
<IPython.lib.display.Audio object>

--------------------------------------------------------------------------------

Example 2: Label - Northern Mockingbird (3)
<IPython.lib.display.Audio object>

--------------------------------------------------------------------------------

Example 3: Label - American Robin (0)
<IPython.lib.display.Audio object>

--------------------------------------------------------------------------------

Example 4: Label - Song Sparrow (4)
<IPython.lib.display.Audio object>

--------------------------------------------------------------------------------

Example 5: Label - Northern Cardinal (2)
<IPython.lib.display.Audio object>

--------------------------------------------------------------------------------
```