inference

May 27, 2024

[1]:
```python
import torch
from torchvision.transforms import ToTensor
from bird_song_dataset import BirdSongDataset, DataPaths, DeviceManager
from train import SimpleCNN
import numpy as np
from sklearn.metrics import confusion_matrix
import plotly.figure_factory as ff
import pickle
import json
```

[2]:
```python
def evaluate_model(model_directory):
    # Load the best model for each type of directory
    best_model = SimpleCNN(num_classes=5).to(device)
    best_model_path = f"{paths['models_dir']}/{model_directory}/model_best.pth"
    best_model.load_state_dict(torch.load(best_model_path))
    best_model.eval()

    test_loss = 0.0
    correct = 0
    total = 0
    true_labels = []
    pred_labels = []
    inference_results = []

    with torch.no_grad():
        # Loop over batches in the test dataset
        for batch in test_loader:
            # Extract inputs and labels from the batch
            inputs = batch['spectrogram'].to(device)
            labels = batch['label'].to(device)
            # Forward pass: compute model output
            outputs = best_model(inputs)
            audio_paths = batch['audio_path']

            # Calculate loss
            loss = criterion(outputs, labels)
            test_loss += loss.item()
```

```
            # Calculate accuracy
            _, preds = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (preds == labels).sum().item()

            # Store labels for confusion matrix as lists
            true_labels.extend(labels.cpu().tolist())
            pred_labels.extend(preds.cpu().tolist())

            # Collect inference results for each batch
            for audio_path, true_label_numeric, pred_label_numeric in␣
↪zip(audio_paths, labels.cpu().tolist(), preds.cpu().tolist()):
                true_label_name = label_encoder.
↪inverse_transform([true_label_numeric])[0]
                pred_label_name = label_encoder.
↪inverse_transform([pred_label_numeric])[0]

                inference_results.append({
                    'true_label': true_label_name,
                    'pred_label': pred_label_name
                })

    # Save the collected inference results
    with open(f"{paths['results_dir']}/inference_results_{model_directory}.
↪pkl", 'wb') as f:
        pickle.dump(inference_results, f)

    # Print metrics
    print(f'Test Loss: {test_loss / len(test_loader):.4f}')
    print(f'Test Accuracy: {100 * correct / total:.2f}%')

    # Save metrics to JSON
    metrics = {
        'test_loss': f'{test_loss / len(test_loader):.4f}',
        'test_accuracy': f'{100 * correct / total:.2f}'
    }

    with open(f"{paths['results_dir']}/metrics/metrics_{model_directory}.json",␣
↪'w') as f:
        json.dump(metrics, f, indent=4)

    # Confusion matrix
    true_labels_np = np.array(true_labels)
    pred_labels_np = np.array(pred_labels)
    true_label_strings = label_encoder.inverse_transform(true_labels_np)
    predicted_label_strings = label_encoder.inverse_transform(pred_labels_np)
```

```
    cm = confusion_matrix(true_label_strings, predicted_label_strings)

    # Annotations for the heatmap
    annotations = [[str(value) for value in row] for row in cm.tolist()]
    class_labels = label_encoder.classes_.tolist()

    # Create heatmap
    fig = ff.create_annotated_heatmap(
        z=cm,
        x=class_labels,
        y=class_labels,
        annotation_text=annotations,
        colorscale='Viridis'
    )

    # Figure layout
    fig.update_layout(
        title=f'Confusion Matrix: {model_directory}',
        xaxis=dict(title='Predicted Labels', tickangle=-45,␣
↪tickvals=list(range(len(class_labels))), ticktext=class_labels),
        yaxis=dict(title='True Labels',␣
↪tickvals=list(range(len(class_labels))), ticktext=class_labels)
    )

    fig.show()

    # Save the confusion matrix
    with open(f"{paths['results_dir']}//confusion_matrices/
↪confusion_matrix_{model_directory}.fig", "wb") as f:
        pickle.dump(fig, f)
```

[3]:
```
# 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'])
```

[4]:
```
# Determine accelerator device
device_manager = DeviceManager()
device = device_manager.device
device
```

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

[4]:
```
device(type='mps')
```

[5]:

```
# Instantiate dataset class
bird_dataset = BirdSongDataset(csv_file=paths['csv_file_path'],␣
 ↪root_dir=paths['wav_files_dir'])
label_encoder = bird_dataset.get_label_encoder()
test_loader = torch.utils.data.DataLoader(bird_dataset, batch_size=64,␣
 ↪shuffle=False)
criterion = torch.nn.CrossEntropyLoss()

# List of model directories to evaluate
model_directories = ['model_sch_lr_es_strat', 'model_sch_lr_es',␣
 ↪'model_fixed_lr']

for model_dir in model_directories:
    evaluate_model(model_dir)
```

```
Test Loss: 0.2573
Test Accuracy: 91.57%

Test Loss: 0.3064
Test Accuracy: 89.97%

Test Loss: 0.5114
Test Accuracy: 81.34%
```