Wildlife Detection System - Model Performance Reporting Implementation Guide

This guide provides step-by-step instructions for implementing a Model Performance Reporting feature in the Wildlife Detection System. This new feature will allow tracking and visualization of machine learning model performance metrics, helping researchers understand the current state of the model and identify improvement opportunities.

1. Overview

The Model Performance Reporting feature will:

- Display current active model details (name, version, training date)
- Show performance metrics (mAP, precision, recall)
- Provide per-species performance breakdown
- Display a confusion matrix
- Show recent detection statistics
- Analyze potential improvement areas

2. Backend Implementation

2.1. Create the Model Performance Service

Create a new file: `/api/app/services/model_performance_service.py`

python

```python
import os
import json
import pandas as pd
import numpy as np
from datetime import datetime
from flask import current_app
from app import db
from app.models.models import Species, Annotation, Image

class ModelPerformanceService:
    """Service for tracking and analyzing model performance."""

    @staticmethod
    def get_current_model_details():
        """Get the current active model details."""
        # Path to trained models
        models_dir = current_app.config.get('MODEL_FOLDER', os.path.join(os.path.dirna
            os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), 'models', 't

        # Find most recent model folder by creation date
        model_folders = []
        for folder in os.listdir(models_dir):
            folder_path = os.path.join(models_dir, folder)
            if os.path.isdir(folder_path) and 'wildlife_detector' in folder:
                creation_time = os.path.getctime(folder_path)
                model_folders.append((folder, creation_time))

        if not model_folders:
            return None

        # Sort by creation time (newest first)
        model_folders.sort(key=lambda x: x[1], reverse=True)
        latest_model_folder = model_folders[0][0]
        latest_model_path = os.path.join(models_dir, latest_model_folder)

        # Get model details from args.yaml or similar file
        args_path = os.path.join(latest_model_path, 'args.yaml')
        if os.path.exists(args_path):
            with open(args_path, 'r') as f:
                import yaml
                args = yaml.safe_load(f)
        else:
            args = {}

        # Get creation date
        creation_date = datetime.fromtimestamp(model_folders[0][1]).strftime('%Y-%m-%d
```

```
    # Check if weights exist
    best_weights_path = os.path.join(latest_model_path, 'weights', 'best.pt')
    weights_exist = os.path.exists(best_weights_path)

    return {
        'model_name': latest_model_folder,
        'created_at': creation_date,
        'weights_file': 'best.pt' if weights_exist else 'N/A',
        'config': args
    }

@staticmethod
def get_performance_metrics():
    """Get performance metrics for the current model."""
    # Path to trained models
    models_dir = current_app.config.get('MODEL_FOLDER', os.path.join(os.path.dirna
        os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), 'models', 't

    # Find most recent model folder by creation date
    model_folders = []
    for folder in os.listdir(models_dir):
        folder_path = os.path.join(models_dir, folder)
        if os.path.isdir(folder_path) and 'wildlife_detector' in folder:
            creation_time = os.path.getctime(folder_path)
            model_folders.append((folder, creation_time))

    if not model_folders:
        return None

    # Sort by creation time (newest first)
    model_folders.sort(key=lambda x: x[1], reverse=True)
    latest_model_folder = model_folders[0][0]
    latest_model_path = os.path.join(models_dir, latest_model_folder)

    # Get results from results.csv
    results_path = os.path.join(latest_model_path, 'results.csv')
    if not os.path.exists(results_path):
        return {
            'precision': 0,
            'recall': 0,
            'mAP50': 0,
            'mAP50-95': 0,
            'per_class': {}
        }

    # Parse results.csv
```

```
        results = pd.read_csv(results_path)

        # Get the last row (final epoch)
        last_row = results.iloc[-1]

        # Format performance metrics
        performance = {
            'precision': last_row.get('precision', 0),
            'recall': last_row.get('recall', 0),
            'mAP50': last_row.get('mAP_0.5', 0),
            'mAP50-95': last_row.get('mAP_0.5:0.95', 0),
            'per_class': {}
        }

        # Try to get per-class metrics from a class breakdown file if it exists
        class_metrics_path = os.path.join(latest_model_path, 'class_metrics.json')
        if os.path.exists(class_metrics_path):
            try:
                with open(class_metrics_path, 'r') as f:
                    performance['per_class'] = json.load(f)
            except Exception as e:
                print(f"Error loading class metrics: {e}")

        return performance

    @staticmethod
    def get_confusion_matrix():
        """Get confusion matrix for species predictions."""
        # Path to trained models
        models_dir = current_app.config.get('MODEL_FOLDER', os.path.join(os.path.dirna
            os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), 'models', 't

        # Find most recent model folder by creation date
        model_folders = []
        for folder in os.listdir(models_dir):
            folder_path = os.path.join(models_dir, folder)
            if os.path.isdir(folder_path) and 'wildlife_detector' in folder:
                creation_time = os.path.getctime(folder_path)
                model_folders.append((folder, creation_time))

        if not model_folders:
            return None

        # Sort by creation time (newest first)
        model_folders.sort(key=lambda x: x[1], reverse=True)
        latest_model_folder = model_folders[0][0]
        latest_model_path = os.path.join(models_dir, latest_model_folder)
```

```
    # Get confusion matrix if it exists
    confusion_path = os.path.join(latest_model_path, 'confusion_matrix.json')
    if os.path.exists(confusion_path):
        try:
            with open(confusion_path, 'r') as f:
                return json.load(f)
        except Exception as e:
            print(f"Error loading confusion matrix: {e}")

    # If no confusion matrix file exists, generate a placeholder
    all_species = Species.query.all()
    num_species = len(all_species)

    # Create empty confusion matrix
    matrix = np.zeros((num_species, num_species))
    class_names = [s.name for s in all_species]

    return {
        'matrix': matrix.tolist(),
        'class_names': class_names
    }

@staticmethod
def get_recent_detection_stats():
    """Get statistics about recent model detections."""
    # Get recently verified vs. unverified annotations
    recent_annotations = Annotation.query.order_by(Annotation.created_at.desc()).l

    total = len(recent_annotations)
    verified = sum(1 for a in recent_annotations if a.is_verified)
    unverified = total - verified

    # Calculate correction rate (how often humans correct model predictions)
    correction_rate = 0
    corrected_count = 0

    # Identify species that often need correction
    species_corrections = {}
    all_species = Species.query.all()
    for s in all_species:
        species_corrections[s.name] = {'total': 0, 'corrected': 0}

    # Count corrections
    for a in recent_annotations:
        if a.confidence is not None:  # This was a model prediction
            species_name = Species.query.get(a.species_id).name
```

```
                species_corrections[species_name]['total'] += 1

                if a.updated_at > a.created_at:  # Annotation was updated after creati
                    corrected_count += 1
                    species_corrections[species_name]['corrected'] += 1

        if total > 0:
            correction_rate = (corrected_count / total) * 100

        # Calculate per-species correction rates
        for species, counts in species_corrections.items():
            if counts['total'] > 0:
                counts['correction_rate'] = (counts['corrected'] / counts['total']) *
            else:
                counts['correction_rate'] = 0

        # Filter to species with at least one detection
        filtered_species = {s: c for s, c in species_corrections.items() if c['total']

        return {
            'total_recent': total,
            'verified_count': verified,
            'unverified_count': unverified,
            'correction_rate': correction_rate,
            'species_corrections': filtered_species
        }

    @staticmethod
    def analyze_improvement_opportunities():
        """Analyze areas where the model could be improved."""
        # Get performance metrics
        performance = ModelPerformanceService.get_performance_metrics()
        detection_stats = ModelPerformanceService.get_recent_detection_stats()

        # Count species representation in the dataset
        species_counts = {}
        for species in Species.query.all():
            count = Annotation.query.filter_by(species_id=species.id).count()
            species_counts[species.name] = count

        # Find underrepresented species (fewer than 50 examples)
        underrepresented = {}
        for species, count in species_counts.items():
            if count < 50 and count > 0:  # Only include species that exist but are un
                underrepresented[species] = count

        # Find species with high correction rates
```

```
    problem_species = {}
    for species, stats in detection_stats['species_corrections'].items():
        if stats['correction_rate'] > 25 and stats['total'] > 10:  # At least 25%
            problem_species[species] = stats['correction_rate']

    # Get suggestions based on findings
    suggestions = []

    if performance.get('mAP50', 0) < 0.7:
        suggestions.append("The model's overall accuracy (mAP50) is below 70%. Con

    if underrepresented:
        top_underrepresented = sorted(underrepresented.items(), key=lambda x: x[1]
        species_list = ', '.join([f"{s} ({c} examples)" for s, c in top_underrepre
        suggestions.append(f"Collect more training data for underrepresented speci

    if problem_species:
        top_problems = sorted(problem_species.items(), key=lambda x: x[1], reverse
        species_list = ', '.join([f"{s} ({c:.1f}% correction rate)" for s, c in to
        suggestions.append(f"Review and improve annotations for frequently correct

    # Additional generic suggestions
    suggestions.append("Consider data augmentation techniques to improve model rob
    suggestions.append("Evaluate model performance in different lighting condition

    return {
        'underrepresented_species': underrepresented,
        'problem_species': problem_species,
        'improvement_suggestions': suggestions
    }
```

2.2. Create System Routes for Model Performance

Modify the file: `/api/app/routes/system.py`

Add this new route at the end of the file:

```python
from app.services.model_performance_service import ModelPerformanceService

# Add this route after existing routes
@system.route('/api/system/model-performance')
def get_model_performance():
    """Get comprehensive model performance metrics."""
    try:
        model_details = ModelPerformanceService.get_current_model_details()
        performance_metrics = ModelPerformanceService.get_performance_metrics()
        confusion_matrix = ModelPerformanceService.get_confusion_matrix()
        detection_stats = ModelPerformanceService.get_recent_detection_stats()
        improvement_opportunities = ModelPerformanceService.analyze_improvement_opportu

        return jsonify({
            'success': True,
            'model_details': model_details,
            'performance_metrics': performance_metrics,
            'confusion_matrix': confusion_matrix,
            'detection_stats': detection_stats,
            'improvement_opportunities': improvement_opportunities
        })
    except Exception as e:
        return jsonify({
            'success': False,
            'message': f'Error retrieving model performance: {str(e)}'
        }), 500
```

3. Frontend Implementation

3.1. Create Model Performance Template

Create a new file: `/api/app/templates/model_performance.html`

html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Model Performance - Wildlife Detection System</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f5f7fa;
            color: #333;
        }

        .container {
            max-width: 1200px;
            margin: 0 auto;
            padding: 20px;
        }

        header {
            background-color: #2c3e50;
            color: white;
            padding: 20px;
            text-align: center;
        }

        h1 {
            margin: 0;
            font-size: 2em;
        }

        .subtitle {
            font-size: 1.1em;
            margin-top: 10px;
            color: #ecf0f1;
        }

        .card {
            background-color: white;
            border-radius: 10px;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            margin-bottom: 20px;
            overflow: hidden;
            transition: transform 0.3s ease, box-shadow 0.3s ease;
```

```
}

.card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 20px rgba(0, 0, 0, 0.15);
}

.card-header {
    background-color: #3498db;
    color: white;
    padding: 15px;
    font-weight: bold;
    font-size: 1.2em;
}

.card-content {
    padding: 15px;
}

.grid-2 {
    display: grid;
    grid-template-columns: repeat(2, 1fr);
    gap: 20px;
}

.grid-3 {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 20px;
}

.info-box {
    background-color: #f8f9fa;
    border-radius: 5px;
    padding: 15px;
    margin-bottom: 15px;
    border-left: 4px solid #3498db;
}

.metric-value {
    font-size: 2em;
    font-weight: bold;
    color: #2c3e50;
    margin: 10px 0;
}

.metric-label {
```

```
    font-size: 0.9em;
    color: #7f8c8d;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin-bottom: 15px;
}

table th, table td {
    padding: 10px;
    text-align: left;
    border-bottom: 1px solid #eee;
}

table th {
    background-color: #f1f2f6;
    font-weight: bold;
}

.warning {
    background-color: #fcf8e3;
    border-left: 4px solid #f39c12;
}

.success {
    background-color: #dff0d8;
    border-left: 4px solid #2ecc71;
}

.danger {
    background-color: #f2dede;
    border-left: 4px solid #e74c3c;
}

.chart-container {
    height: 300px;
    margin-bottom: 20px;
}

.button {
    display: inline-block;
    background-color: #3498db;
    color: white;
    padding: 10px 15px;
    border-radius: 5px;
```

```
            text-decoration: none;
            transition: background-color 0.3s ease;
            cursor: pointer;
            border: none;
            font-family: inherit;
            font-size: 1em;
        }

        .button:hover {
            background-color: #2980b9;
        }

        .back-link {
            margin-bottom: 20px;
            display: inline-block;
        }

        footer {
            margin-top: 50px;
            text-align: center;
            padding: 20px;
            color: #7f8c8d;
            font-size: 0.9em;
        }
    </style>
    <!-- Include Chart.js for visualizations -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js@3.7.1/dist/chart.min.js"></scri
</head>
<body>
    <header>
        <h1>Model Performance Dashboard</h1>
        <div class="subtitle">Wildlife Detection System Model Evaluation</div>
    </header>

    <div class="container">
        <a href="/" class="button back-link">← Back to Dashboard</a>

        <div id="loading-message">Loading model performance data...</div>

        <div id="model-details-section" style="display:none;" class="card">
            <div class="card-header">Model Details</div>
            <div class="card-content">
                <div class="grid-3">
                    <div class="info-box">
                        <div class="metric-label">Model Name</div>
                        <div class="metric-value" id="model-name"></div>
                    </div>
```

```
                <div class="info-box">
                    <div class="metric-label">Creation Date</div>
                    <div class="metric-value" id="creation-date"></div>
                </div>
                <div class="info-box">
                    <div class="metric-label">Weights File</div>
                    <div class="metric-value" id="weights-file"></div>
                </div>
            </div>
            <div id="model-config"></div>
        </div>
    </div>

    <div class="grid-2">
        <div id="performance-metrics-section" style="display:none;" class="card">
            <div class="card-header">Performance Metrics</div>
            <div class="card-content">
                <div class="grid-2">
                    <div class="info-box">
                        <div class="metric-label">Precision</div>
                        <div class="metric-value" id="precision-value"></div>
                    </div>
                    <div class="info-box">
                        <div class="metric-label">Recall</div>
                        <div class="metric-value" id="recall-value"></div>
                    </div>
                    <div class="info-box">
                        <div class="metric-label">mAP@0.5</div>
                        <div class="metric-value" id="map50-value"></div>
                    </div>
                    <div class="info-box">
                        <div class="metric-label">mAP@0.5:0.95</div>
                        <div class="metric-value" id="map50-95-value"></div>
                    </div>
                </div>

                <div class="chart-container">
                    <canvas id="per-class-chart"></canvas>
                </div>
            </div>
        </div>

        <div id="detection-stats-section" style="display:none;" class="card">
            <div class="card-header">Detection Statistics</div>
            <div class="card-content">
                <div class="grid-3">
                    <div class="info-box">
```

```
                    <div class="metric-label">Recent Detections</div>
                    <div class="metric-value" id="total-recent"></div>
                </div>
                <div class="info-box">
                    <div class="metric-label">Verified</div>
                    <div class="metric-value" id="verified-count"></div>
                </div>
                <div class="info-box">
                    <div class="metric-label">Correction Rate</div>
                    <div class="metric-value" id="correction-rate"></div>
                </div>
            </div>

            <h3>Species Requiring Corrections</h3>
            <div class="chart-container">
                <canvas id="correction-chart"></canvas>
            </div>
        </div>
    </div>
</div>

<div id="confusion-matrix-section" style="display:none;" class="card">
    <div class="card-header">Confusion Matrix</div>
    <div class="card-content">
        <div class="chart-container" style="height: 500px;">
            <canvas id="confusion-matrix-chart"></canvas>
        </div>
    </div>
</div>

<div id="improvement-section" style="display:none;" class="card">
    <div class="card-header">Improvement Opportunities</div>
    <div class="card-content">
        <h3>Suggested Improvements</h3>
        <ul id="suggestions-list"></ul>

        <div class="grid-2">
            <div>
                <h3>Underrepresented Species</h3>
                <table id="underrepresented-table">
                    <thead>
                        <tr>
                            <th>Species</th>
                            <th>Example Count</th>
                        </tr>
                    </thead>
                    <tbody></tbody>
```

```
                    </table>
                </div>
                <div>
                    <h3>Problem Species</h3>
                    <table id="problem-table">
                        <thead>
                            <tr>
                                <th>Species</th>
                                <th>Correction Rate</th>
                            </tr>
                        </thead>
                        <tbody></tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
</div>

<footer>
    Wildlife Detection System - Based on Prof. Peeva's Requirements - © 2025
</footer>

<script>
    // Fetch model performance data when the page loads
    document.addEventListener('DOMContentLoaded', function() {
        fetchModelPerformance();
    });

    async function fetchModelPerformance() {
        try {
            const response = await fetch('/api/system/model-performance');
            const data = await response.json();

            if (data.success) {
                // Hide loading message
                document.getElementById('loading-message').style.display = 'none';

                // Display all sections
                document.getElementById('model-details-section').style.display = 'b
                document.getElementById('performance-metrics-section').style.displ
                document.getElementById('detection-stats-section').style.display =
                document.getElementById('confusion-matrix-section').style.display
                document.getElementById('improvement-section').style.display = 'bl

                // Populate model details
                populateModelDetails(data.model_details);
```

```
            // Populate performance metrics
            populatePerformanceMetrics(data.performance_metrics);

            // Populate detection stats
            populateDetectionStats(data.detection_stats);

            // Create confusion matrix
            createConfusionMatrix(data.confusion_matrix);

            // Populate improvement opportunities
            populateImprovementOpportunities(data.improvement_opportunities);
        } else {
            document.getElementById('loading-message').textContent = 'Error lo
        }
    } catch (error) {
        console.error('Error fetching model performance:', error);
        document.getElementById('loading-message').textContent = 'Error loadin
    }
}

function populateModelDetails(details) {
    if (!details) {
        document.getElementById('model-details-section').innerHTML = '<div cla
        return;
    }

    document.getElementById('model-name').textContent = details.model_name;
    document.getElementById('creation-date').textContent = details.created_at;
    document.getElementById('weights-file').textContent = details.weights_file

    // Add configuration details if available
    const configDiv = document.getElementById('model-config');
    if (details.config && Object.keys(details.config).length > 0) {
        let configHtml = '<h3>Configuration</h3><table><thead><tr><th>Paramete

        for (const [key, value] of Object.entries(details.config)) {
            configHtml += `<tr><td>${key}</td><td>${value}</td></tr>`;
        }

        configHtml += '</tbody></table>';
        configDiv.innerHTML = configHtml;
    } else {
        configDiv.innerHTML = '<p>No configuration details available.</p>';
    }
}
```

```
function populatePerformanceMetrics(metrics) {
    if (!metrics) {
        document.getElementById('performance-metrics-section').innerHTML = '<d
        return;
    }

    // Format metrics values as percentages
    document.getElementById('precision-value').textContent = (metrics.precisio
    document.getElementById('recall-value').textContent = (metrics.recall * 10
    document.getElementById('map50-value').textContent = (metrics.mAP50 * 100)
    document.getElementById('map50-95-value').textContent = (metrics.mAP50 * 1

    // Create per-class performance chart if data available
    if (metrics.per_class && Object.keys(metrics.per_class).length > 0) {
        const ctx = document.getElementById('per-class-chart').getContext('2d'

        const labels = Object.keys(metrics.per_class);
        const precisionData = labels.map(label => metrics.per_class[label].pre
        const recallData = labels.map(label => metrics.per_class[label].recall

        new Chart(ctx, {
            type: 'bar',
            data: {
                labels: labels,
                datasets: [
                    {
                        label: 'Precision',
                        data: precisionData,
                        backgroundColor: 'rgba(52, 152, 219, 0.7)',
                        borderColor: 'rgba(52, 152, 219, 1)',
                        borderWidth: 1
                    },
                    {
                        label: 'Recall',
                        data: recallData,
                        backgroundColor: 'rgba(46, 204, 113, 0.7)',
                        borderColor: 'rgba(46, 204, 113, 1)',
                        borderWidth: 1
                    }
                ]
            },
            options: {
                responsive: true,
                maintainAspectRatio: false,
                plugins: {
                    title: {
                        display: true,
```

```
                        text: 'Per-Class Performance'
                    },
                    legend: {
                        position: 'top',
                    }
                },
                scales: {
                    y: {
                        beginAtZero: true,
                        title: {
                            display: true,
                            text: 'Percentage (%)'
                        },
                        max: 100
                    }
                }
            }
        });
    } else {
        document.getElementById('per-class-chart').innerHTML = '<p>No per-class
    }
}

function populateDetectionStats(stats) {
    if (!stats) {
        document.getElementById('detection-stats-section').innerHTML = '<div c
        return;
    }

    document.getElementById('total-recent').textContent = stats.total_recent;
    document.getElementById('verified-count').textContent = stats.verified_cou
    document.getElementById('correction-rate').textContent = stats.correction_

    // Create correction chart if data available
    if (stats.species_corrections && Object.keys(stats.species_corrections).le
        const ctx = document.getElementById('correction-chart').getContext('2d

        // Sort by correction rate descending
        const sortedSpecies = Object.entries(stats.species_corrections)
            .sort((a, b) => b[1].correction_rate - a[1].correction_rate)
            .slice(0, 10); // Top 10

        const labels = sortedSpecies.map(item => item[0]);
        const correctionRates = sortedSpecies.map(item => item[1].correction_r

        new Chart(ctx, {
            type: 'bar',
```

```
            data: {
                labels: labels,
                datasets: [
                    {
                        label: 'Correction Rate (%)',
                        data: correctionRates,
                        backgroundColor: 'rgba(231, 76, 60, 0.7)',
                        borderColor: 'rgba(231, 76, 60, 1)',
                        borderWidth: 1
                    }
                ]
            },
            options: {
                responsive: true,
                maintainAspectRatio: false,
                plugins: {
                    title: {
                        display: true,
                        text: 'Top Species Requiring Corrections'
                    },
                    legend: {
                        position: 'top',
                    }
                },
                scales: {
                    y: {
                        beginAtZero: true,
                        title: {
                            display: true,
                            text: 'Correction Rate (%)'
                        },
                        max: 100
                    }
                }
            }
        });
    } else {
        document.getElementById('correction-chart').innerHTML = '<p>No correct
    }
}

function createConfusionMatrix(data) {
    if (!data || !data.matrix || !data.class_names) {
        document.getElementById('confusion-matrix-section').innerHTML = '<div
        return;
    }
```

```javascript
// Create confusion matrix visualization
const ctx = document.getElementById('confusion-matrix-chart').getContext('

// Use Chart.js matrix with heatmap
const matrixData = [];
for (let i = 0; i < data.matrix.length; i++) {
    for (let j = 0; j < data.matrix[i].length; j++) {
        matrixData.push({
            x: data.class_names[j],
            y: data.class_names[i],
            v: data.matrix[i][j]
        });
    }
}

// Create custom chart
new Chart(ctx, {
    type: 'scatter',
    data: {
        datasets: [{
            label: 'Confusion Matrix',
            data: matrixData,
            backgroundColor: function(context) {
                const value = context.raw.v;
                const maxValue = Math.max(...data.matrix.flat());
                const alpha = value / maxValue;
                return `rgba(52, 152, 219, ${alpha})`;
            },
            pointRadius: function(context) {
                const value = context.raw.v;
                const maxValue = Math.max(...data.matrix.flat());
                return 10 + 20 * (value / maxValue);
            },
            pointHoverRadius: function(context) {
                return context.raw.v > 0 ? 15 : 0;
            }
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        plugins: {
            tooltip: {
                callbacks: {
                    label: function(context) {
                        const dataPoint = context.raw;
                        return `Predicted: ${dataPoint.x}, Actual: ${dataP
```

```
                        }
                    }
                },
                legend: {
                    display: false
                },
                title: {
                    display: true,
                    text: 'Confusion Matrix (Predicted vs. Actual)'
                }
            },
            scales: {
                x: {
                    type: 'category',
                    position: 'bottom',
                    title: {
                        display: true,
                        text: 'Predicted'
                    }
                },
                y: {
                    type: 'category',
                    title: {
                        display: true,
                        text: 'Actual'
                    }
                }
            }
        }
    });
}

function populateImprovementOpportunities(data) {
    if (!data) {
        document.getElementById('improvement-section').innerHTML = '<div class=
        return;
    }

    // Populate suggestions list
    const suggestionsList = document.getElementById('suggestions-list');
    suggestionsList.innerHTML = '';

    if (data.improvement_suggestions && data.improvement_suggestions.length > 
        data.improvement_suggestions.forEach(suggestion => {
            const li = document.createElement('li');
            li.textContent = suggestion;
            suggestionsList.appendChild(li);
```

```
                });
            } else {
                suggestionsList.innerHTML = '<li>No specific improvement suggestions a
            }

            // Populate underrepresented species table
            const underrepresentedTable = document.getElementById('underrepresented-tal
            underrepresentedTable.innerHTML = '';

            if (data.underrepresented_species && Object.keys(data.underrepresented_spe
                // Sort by count ascending
                const sortedSpecies = Object.entries(data.underrepresented_species)
                    .sort((a, b) => a[1] - b[1]);

                sortedSpecies.forEach(([species, count]) => {
                    const row = document.createElement('tr');
                    row.innerHTML = `<td>${species}</td><td>${count}</td>`;
                    underrepresentedTable.appendChild(row);
                });
            } else {
                underrepresentedTable.innerHTML = '<tr><td colspan="2">No underrepresen
            }

            // Populate problem species table
            const problemTable = document.getElementById('problem-table').querySelecto
            problemTable.innerHTML = '';

            if (data.problem_species && Object.keys(data.problem_species).length > 0)
                // Sort by correction rate descending
                const sortedProblems = Object.entries(data.problem_species)
                    .sort((a, b) => b[1] - a[1]);

                sortedProblems.forEach(([species, rate]) => {
                    const row = document.createElement('tr');
                    row.innerHTML = `<td>${species}</td><td>${rate.toFixed(1)}%</td>`;
                    problemTable.appendChild(row);
                });
            } else {
                problemTable.innerHTML = '<tr><td colspan="2">No problem species found
            }
        }
    </script>
</body>
</html>
```

3.2. Add Model Performance Route to Static Routes

Modify the file: `/api/app/routes/static_routes.py`

Add this new route to the file:

```python
@static_pages.route('/model-performance')
def model_performance():
    """Serve the model performance page."""
    return render_template('model_performance.html')
```

3.3. Add Button to Dashboard Page

Modify the file: `/api/app/templates/dashboard.html`

Find the "Admin Tools" card in the dashboard page and add a new button right after the "Database Admin" button:

```html
<a href="/model-performance" class="button" style="background-color: #1abc9c;">
    <i class="glyphicon glyphicon-stats"></i> Model Performance
</a>
```

4. Integration with Training Notebooks

To make the Model Performance page fully functional, your training notebooks need to generate additional output files that the Model Performance service can read. Follow these steps to add this functionality to your training notebooks:

4.1. Add Class Metrics Export to Training Notebooks

Add this code at the end of your model evaluation cell in `/notebooks/training/wildlife_model.ipynb`:

```python
# Save class metrics for model performance dashboard
import json
import os

class_metrics = {}
for i, c in enumerate(names):
    # Extract per-class metrics from evaluation results
    class_metrics[c] = {
        'precision': performance_metrics['precision'][i] if i < len(performance_metric
        'recall': performance_metrics['recall'][i] if i < len(performance_metrics['rec
        'mAP50': performance_metrics['mAP_0.5'][i] if i < len(performance_metrics['mAP_
    }

# Save to class_metrics.json in model output directory
output_dir = os.path.join('../../models/trained', run_name)
os.makedirs(output_dir, exist_ok=True)
with open(os.path.join(output_dir, 'class_metrics.json'), 'w') as f:
    json.dump(class_metrics, f, indent=2)

print(f"Class metrics saved to {os.path.join(output_dir, 'class_metrics.json')}")
```

4.2. Add Confusion Matrix Export

Add this code to generate a confusion matrix in the same notebook:

python

```python
# Generate and save confusion matrix
import numpy as np

# Create a confusion matrix
conf_matrix = np.zeros((len(names), len(names)))

# Populate from validation predictions
# This is a placeholder - you would populate this with real confusion matrix data
# based on your model validation results

# Save to confusion_matrix.json in model output directory
confusion_data = {
    'matrix': conf_matrix.tolist(),
    'class_names': names
}

with open(os.path.join(output_dir, 'confusion_matrix.json'), 'w') as f:
    json.dump(confusion_data, f, indent=2)

print(f"Confusion matrix saved to {os.path.join(output_dir, 'confusion_matrix.json')}"
```

5. Testing the Implementation

After implementing all the components, follow these steps to test the Model Performance feature:

1. Run one of your training notebooks to generate the required model output files (results.csv, class_metrics.json, confusion_matrix.json).
2. Start the Flask server:

   bash

   ```bash
   cd api
   python run.py
   ```

3. Access the dashboard at http://localhost:5000/
4. Click on the "Model Performance" button to view the model performance page.
5. Verify that all sections are populating with the correct data:
   - Model Details
   - Performance Metrics
   - Detection Statistics
   - Confusion Matrix
   - Improvement Opportunities

6. Additional Improvements

Once the basic implementation is working, consider these additional improvements:

1. Add model comparison functionality to compare different model versions.
2. Implement model performance over time tracking.
3. Add threshold adjustment features to optimize precision vs. recall tradeoffs.
4. Implement automatic model improvement suggestions based on error analysis.
5. Add direct links to problematic images for further analysis.

This implementation provides a comprehensive view of model performance and integrates seamlessly with your existing Wildlife Detection System. The performance metrics and improvement suggestions will help researchers understand the current state of the model and identify areas for enhancement.