Automated Parking Space Detection System

A comprehensive computer vision system for detecting parking space occupancy using machine learning. The system supports training on the PKLot dataset and provides real-time detection capabilities through camera feeds.

Features

- **Dataset Support**: Compatible with PKLot dataset format
- **Machine Learning**: SVM classifier with feature engineering
- **Real-time Detection**: Live camera feed processing
- **Interactive ROI Selection**: Click-and-drag parking space selection
- **Performance Evaluation**: Comprehensive metrics and visualizations
- **Model Persistence**: Save and load trained models
- **Visualization**: Annotated output images with bounding boxes

Requirements

- Python 3.7+
- OpenCV 4.x
- Scikit-learn

- NumPy, Matplotlib, Seaborn
- PKLot dataset (optional, for training)

Installation

Option 1: Local Installation

1. Clone the repository:

bash

```
git clone <repository-url>
cd parking-detection-system
```

2. Install dependencies:

bash

```
pip install -r requirements.txt
```

Option 2: Docker Installation

1. Build the Docker image:

bash

```
docker build -t parking-detector .
```

2. Run the container:

```bash
# For training (mount your dataset)
docker run -v /path/to/dataset:/app/dataset -v /path/to/output:/app/output parking-detect

# For real-time detection (with camera access)
docker run --device=/dev/video0 -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix park
```

Dataset Preparation

PKLot Dataset Structure

```
dataset/
├── PKLot/
│   ├── PUCPR/
│   │   ├── Cloudy/
│   │   │   ├── 2012-09-12/
│   │   │   │   ├── image001.jpg
│   │   │   │   ├── image001.xml
│   │   │   │   └── ...
│   │   └── ...
│   └── ...
```

The XML annotations should contain rotated rectangle coordinates for each parking space with occupancy labels.

Usage

Training Mode

Train a new model on the PKLot dataset:

```bash
python parking_detection_system.py --mode train --dataset /path/to/pklot/dataset --model r
```

Parameters:

- `--dataset`: Path to PKLot dataset root directory
- `--model`: Output path for trained model (default: parking_model.pkl)
- `--output`: Directory for evaluation results and visualizations

Testing Mode

Evaluate a trained model on test data:

```bash
python parking_detection_system.py --mode test --dataset /path/to/test/dataset --model my_
```

Real-time Detection Mode

Run real-time detection using a camera:

```bash
python parking_detection_system.py --mode realtime --model my_model.pkl --camera 0
```

Parameters:

- `--camera`: Camera device ID (default: 0)
- `--model`: Path to trained model file

Interactive Controls

- **Click and drag**: Select parking space regions (ROIs)
- **'r' key**: Reset all selected ROIs
- **'q' key**: Quit the application

API Usage

Basic Usage

python

```python
from parking_detection_system import ParkingSpaceDetector

# Initialize detector
detector = ParkingSpaceDetector()

# Training
image_paths, rois, labels = detector.load_pklot_dataset('/path/to/dataset')
detector.train_classifier(image_paths, rois, labels)
detector.save_model('my_model.pkl')

# Prediction
detector.load_model('my_model.pkl')
predictions, probabilities = detector.predict_spaces(image, parking_rois)
```

Real-time Detection

python

```python
from parking_detection_system import ParkingSpaceDetector, RealTimeParkingDetector

# Load trained detector
detector = ParkingSpaceDetector()
detector.load_model('parking_model.pkl')

# Initialize real-time detector
rt_detector = RealTimeParkingDetector(detector)

# Set predefined parking regions (optional)
parking_rois = [[x1, y1, x2, y2], ...]  # List of bounding boxes
rt_detector.set_parking_regions(parking_rois)

# Start detection
rt_detector.detect_from_camera(camera_id=0)
```

Model Architecture

Feature Extraction

- **Statistical Features**: Mean intensity, variance, texture
- **Edge Features**: Sobel edge detection for boundary information
- **Preprocessing**: Histogram equalization, Gaussian blur

Classification

- **Algorithm**: Support Vector Machine (SVM) with RBF kernel
- **Features**: 4-dimensional feature vector per parking space
- **Scaling**: StandardScaler for feature normalization
- **Class Balance**: Weighted classes to handle imbalanced data

Output Files

Training Output

- `parking_model.pkl`: Trained classifier and scaler
- `confusion_matrix.png`: Classification performance visualization
- `empty_spaces_comparison.png`: Actual vs predicted empty spaces
- `*_annotated.jpg`: Annotated images with detection results

Performance Metrics

- **Precision**: Positive predictive value
- **Recall**: Sensitivity/True positive rate
- **F1-Score**: Harmonic mean of precision and recall
- **Accuracy**: Overall classification accuracy

Configuration

Model Parameters

You can modify the SVM parameters in the `train_classifier` method:

```python
self.classifier = SVC(
    kernel='rbf',         # Kernel type
    C=1.0,               # Regularization parameter
    class_weight='balanced',  # Handle class imbalance
    probability=True,     # Enable probability estimates
    random_state=42      # For reproducibility
)
```

Feature Extraction Parameters

Adjust preprocessing parameters in the `preprocess_image` method:

```python
# Gaussian blur kernel size
blurred = cv2.GaussianBlur(equalized, (5, 5), 0)

# Sobel kernel size for edge detection
sobelx = cv2.Sobel(region, cv2.CV_64F, 1, 0, ksize=3)
```

Troubleshooting

Common Issues

1. **Camera not detected**:

   bash

   ```
   # List available cameras
   ls /dev/video*
   # Try different camera IDs
   python parking_detection_system.py --mode realtime --camera 1
   ```

2. **XML parsing errors**:
   - Ensure XML files match the expected PKLot format
   - Check file permissions and paths
3. **Memory issues with large datasets**:
   - Process dataset in batches
   - Reduce image resolution if needed
4. **Poor detection accuracy**:
   - Collect more training data
   - Adjust feature extraction parameters
   - Try different SVM parameters

Docker Issues

1. **Camera access in Docker**:

   bash

   ```
   # Add camera device access
   docker run --device=/dev/video0 ...
   ```

2. **Display issues**:

   bash

   ```
   # Enable X11 forwarding
   xhost +local:docker
   docker run -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix ...
   ```

Performance Optimization

Speed Improvements

- Use smaller ROI regions
- Reduce image resolution
- Implement multi-threading for batch processing
- Consider using faster classifiers (e.g., Random Forest)

Accuracy Improvements

- Collect more diverse training data
- Implement data augmentation
- Use deep learning models (CNN)
- Add temporal consistency for video streams

Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/new-feature`)
3. Commit changes (`git commit -am 'Add new feature'`)
4. Push to branch (`git push origin feature/new-feature`)
5. Create a Pull Request

License

Citation

If you use this system in your research, please cite:

```bibtex
@software{parking_detection_system,
  title={Automated Parking Space Detection System},
  author={Your Name},
  year={2024},
  url={https://github.com/your-repo/parking-detection-system}
}
```

Acknowledgments

- PKLot dataset creators for providing the benchmark dataset
- OpenCV community for computer vision tools
- Scikit-learn developers for machine learning algorithms

Contact

For questions and support, please open an issue on GitHub or contact [your-email@example.com].