VEHICLE COUNTING IN PARKING LOT

Team Name: Team 1

Teammates: sv94, srisakth, sarojavu

Course: cse676. Date: May 3, 2025

University at Buffalo The State University of New York

Problem Description

In busy urban areas, drivers spend significant time and fuel searching for parking. This leads to:

- Increased traffic congestion
- Wasted fuel and emissions
- Driver frustration

There's a need for **automated real-time monitoring of parking lots** that provides visibility into free vs. occupied spaces especially for smart city integration and efficient land use.

Our Project Objective:

We built a fully functional prototype of a **real-time parking space monitoring system** that:

- **Detects** whether a parking space is **empty or occupied** using a **deep learning model**
- Uses **video input** and processes each frame in real-time
- **Displays the results visually** with colored boxes and count of available spaces
- **Serves a live web interface and REST API** using Flask

Background

Related Works:

1. Edge Detection Methods

Used contours and rules to detect cars — failed under poor lighting or occlusion; no deep learning.

2. SIFT-Based Detection

Compared SIFT features with reference frames — required static views and manual calibration.

3. YOLO-Based Approaches

Used object detection for vehicles — fast but needs full car bounding boxes and high compute power.

4. Commercial Solutions

Sensor-based systems — expensive, closed-source, and not research-friendly.

Background

Similarities:

- Uses deep learning based methods
- Real-time video frame analysis
- Detects parking occupancy

Differences:

- Fully **custom dataset** from our own annotated images
- Slot-level ROI classification (no full car bounding boxes)
- Lightweight **VGG16 transfer learning**, not object detection
- Built and deployed **end-to-end pipeline** (data → model → video → web UI)

Dataset

Type:

- Custom-labeled dataset for binary classification:
  Occupied (1) vs. **Empty (0)** parking slots

Structure:

- `train_data/train/occupied`, `train_data/train/empty`
- `train_data/test/occupied`, `train_data/test/empty`

Creation:

- Cropped ROIs from parking lot images using our **OpenCV annotation tool**
- ROI size: **120×45 px**, resized to **48×48 px**
- Labels assigned manually and sorted into train/test folders

Size:

- ~450 training images
- ~140 test images
- Class imbalance reflects real-world scenarios

Format:

- PNG, RGB, slot-level focus

Fig.2 Sample Images with labels 0 and 1 for "empty" and "occupied"

Dataset Preprocessing

Augmentation Techniques Used (via `ImageDataGenerator`):

- **Rescaling**: Pixel values normalized to [0, 1] using `rescale=1./255`
- **Horizontal Flip**: Simulates vehicle direction changes
- **Zoom Range (0.1)**: Mimics different camera distances
- **Width/Height Shift (0.1)**: Handles misalignment of slots
- **Rotation Range (5°)**: Captures slight camera angle variations
- **Fill Mode = 'nearest'**: Smooths transformed image gaps

Why it Matters:

- Increases dataset diversity without collecting more data
- Helps model generalize to real-world parking conditions
- Reduces risk of overfitting on small dataset

Inference : datacollection.py

Purpose: Manually label parking slots from `car2.png` to create a custom dataset.

Key Features:

- **ROI size:** 120×45 pixels
- **Image resized** to 1280×720 for uniform annotation
- **Left-click:** Adds a cropped ROI saved as `roi_<n>.png` in `cropped_img/`
- **Right-click:** Removes existing ROI
- **Live preview:** Draws purple boxes over selected ROIs
- **Persistent annotations:**
  - Coordinates saved to `carposition.pkl` via `pickle`
  - Enables multi-session labeling

Outputs:

- Cropped slot images (`cropped_img/`)
- Slot positions (`carposition.pkl`) for later inference

Fig.3 Training Set - Image Distribution per Class

Fig.4 Testing Set - Image Distribution per Class

Annotation Output – carposition.pkl

Purpose: Stores the coordinates of all manually annotated parking slots.

Details:

- Created and updated in real time during annotation (`mouseclick` function)
- Each ROI's (x, y) top-left position is saved in a list
- The list is serialized using Python's `pickle` module

Benefits:

- Enables **slot position reuse** across sessions without re-annotating
- Ensures **consistent slot locations** during both training and inference
- Acts as a **bridge** between dataset creation and real-time video inference

Used In:

- Data labeling (`datacollection.py`)
- Real-time inference (`test.py`, `main.py`) — slots are cropped using these stored coordinates

Methods (Transfer-Learning–Based Parking-Space Classifier)

Backbone: VGG16 (ImageNet-pretrained, include_top=False)

- Freeze first 10 convolutional blocks to retain low-level filters
- Fine-tune higher-level layers for “occupied” vs. “vacant” semantics

Custom Head:

- GlobalAveragePooling2D → Dense(256, ReLU) → Dropout(0.5) → Dense(2, Softmax)

Optimization:

- Loss: Categorical Cross-Entropy
- Optimizer: Adam with warm-up LR schedule + L2 weight decay
- Early stopping on validation-accuracy plateau

Data Augmentation:

- Random flips (H/V), rotations (±15°), shifts (±10%), zoom (±10%)
- Simulates occlusions, lighting changes, camera jitter

Inference Latency: < 50 ms per crop on CPU

University at Buffalo The State University of New York

Number of Training Samples: 432
Number of Validation Samples: 164
Image Dimensions: 48 x 48
Batch Size: 32
Epochs: 15
Number of Classes: 2

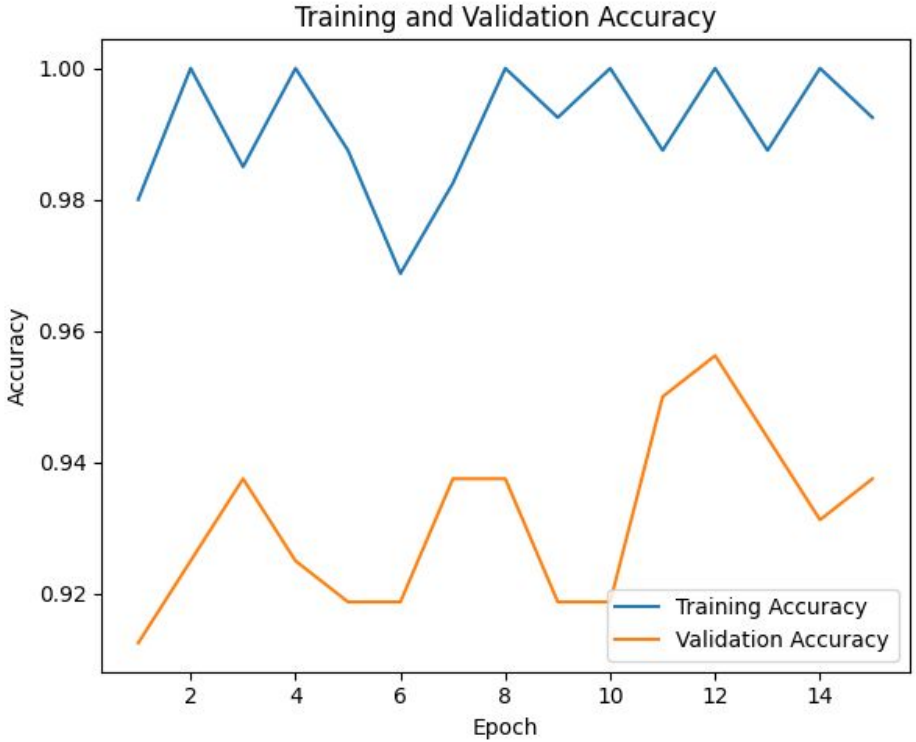
Total params: 29,431,430 (112.27 MB)

Trainable params: 14,715,714 (56.14 MB)

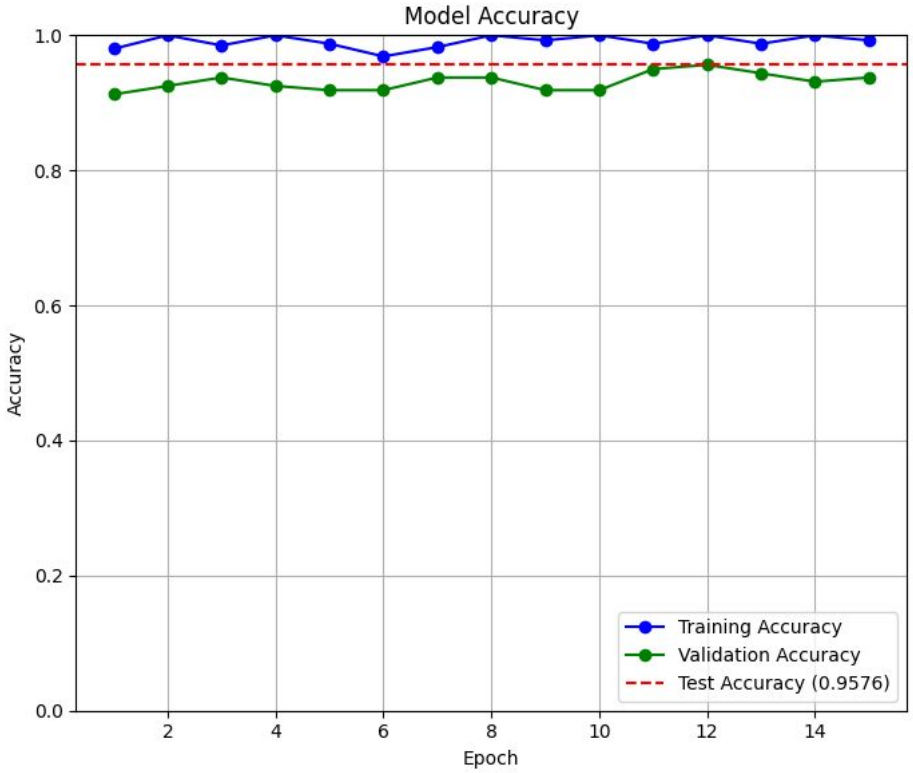
Non-trainable params: 0 (0.00 B)

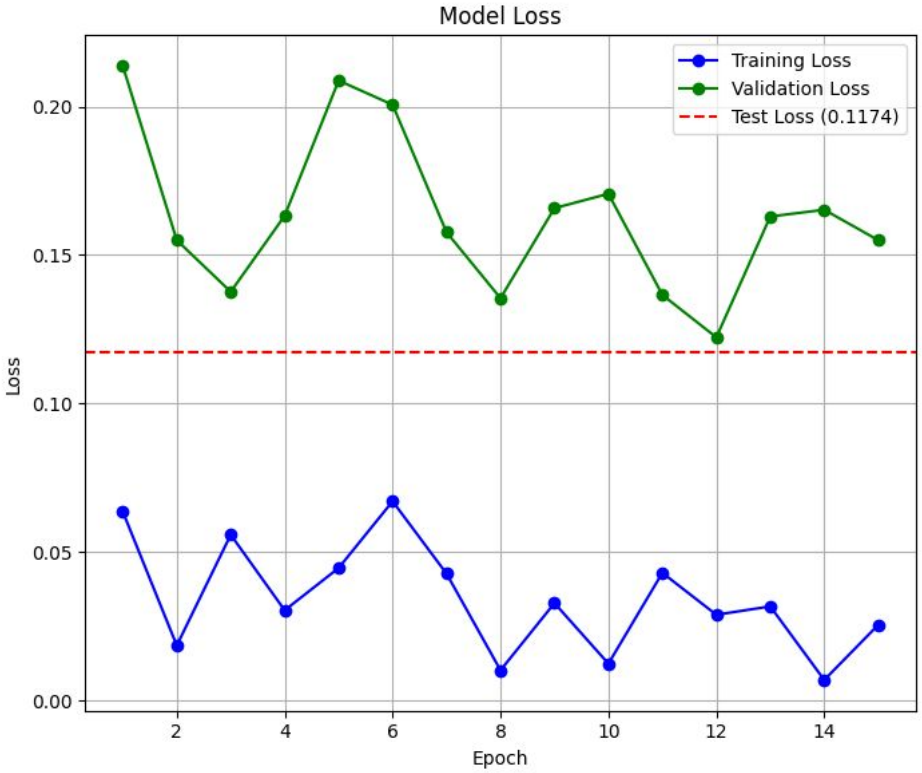
Optimizer params: 14,715,716 (56.14 MB)

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 48, 48, 3)	0
block1_conv1 (Conv2D)	(None, 48, 48, 64)	1,792
block1_conv2 (Conv2D)	(None, 48, 48, 64)	36,928
block1_pool (MaxPooling2D)	(None, 24, 24, 64)	0
block2_conv1 (Conv2D)	(None, 24, 24, 128)	73,856
block2_conv2 (Conv2D)	(None, 24, 24, 128)	147,584
block2_pool (MaxPooling2D)	(None, 12, 12, 128)	0
block3_conv1 (Conv2D)	(None, 12, 12, 256)	295,168
block3_conv2 (Conv2D)	(None, 12, 12, 256)	590,080
block3_conv3 (Conv2D)	(None, 12, 12, 256)	590,080
block3_pool (MaxPooling2D)	(None, 6, 6, 256)	0
block4_conv1 (Conv2D)	(None, 6, 6, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 6, 6, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 6, 6, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 3, 3, 512)	0
block5_conv1 (Conv2D)	(None, 3, 3, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 3, 3, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 3, 3, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten_2 (Flatten)	(None, 512)	0
dense_2 (Dense)	(None, 2)	1,026


Training and Validation Accuracy
Accuracy
Epoch
Training Accuracy
Validation Accuracy
1.00
0.98
0.96
0.94
0.92
2
4
6
8
10
12
14


Training and Validation Loss
Loss
Epoch
Training Loss
Validation Loss
0.20
0.15
0.10
0.05
0.00
2
4
6
8
10
12
14


Model Accuracy
Accuracy
Epoch
Training Accuracy
Validation Accuracy
Test Accuracy (0.9576)


Model Loss
Loss
Epoch
Training Loss
Validation Loss
Test Loss (0.1174)

Predicted: car

Predicted: no_car

test.py

Model & Labels

- Loads OUR MODEL Keras CNN (model_final.h5)
- Defines a simple class map: 0 → no_car, 1 → car

Region Definitions

- Reads parking-slot coordinates from carposition.pkl
- Uses fixed crop size (120×45 pixels) for each slot

Frame Acquisition

- Opens car_test2.mp4 via OpenCV
- Resets to first frame upon reaching the end

ROI Extraction & Preprocessing

- For each slot, crops the frame
- Resizes to 48×48, normalizes pixels to [0,1]

Batch Prediction

- Feeds all slot crops together into the CNN
- Obtains “car” vs “no_car” labels via argmax

Visualization & Counting

- Draws colored rectangles around each slot (green for free, red for occupied)
- Overlays text labels and tallies free spaces

Status Overlay

- Displays total free-space count on the frame

Real-Time Loop

- Continuously processes frames at ~10 ms intervals
- Exits on ‘q’ key and cleans up windows


Image
Space Count: 15
car
no_car

main.py

Flask Web Service

- Uses Flask to serve both a live video stream and a JSON API
- Runs in debug mode (app.run(debug=True))

Model & Slot Definitions

- Loads OUR MODEL CNN (model_final.h5)
- Reads parking-slot coordinates from carposition.pkl
- Defines label map {0: no_car, 1: car}

Video Capture

- Opens car_test2.mp4 via OpenCV's VideoCapture
- Frames resized to 1280×720 each iteration

checkParkingSpace(img) Function

- Crops each slot region (120×45), resizes to 48×48, normalizes to [0,1]
- Runs batch prediction, determines “car” vs “no_car” per slot
- Draws colored rectangles/text on the image
- Returns annotated image, free-space count, occupied-space count

Frame Generator (generate_frames)

- Reads frames in a loop, applies checkParkingSpace
- Encodes to JPEG and yields multipart MJPEG chunks

Endpoints

- / → renders index.html
- /video_feed → streams live MJPEG (multipart/x-mixed-replace)
- /space_count → returns { free: x, occupied: y } as JSON

Real-Time Loop & Cleanup

- Continuously serves new frames until the video ends
- No manual cleanup needed—Flask/WSGI handles connections

Demo!

Key observations/ Summary

- Reliable Detection: The pretrained CNN consistently distinguishes “car” vs. “no_car” across all defined slots.
- Efficient Pipeline: Batch-cropping, resizing, and normalization of ROIs keep per-frame latency low.
- Real-Time Streaming: MJPEG endpoint (/video_feed) delivers live video at interactive speeds.
- Programmatic Access: /space_count JSON API exposes free/occupied counts for dashboards or apps.
- Modular Configuration: Slot positions loaded from a simple pickle file—easy to adapt to new layouts.
- Lightweight & Portable: Single Flask script with minimal dependencies enables quick deployment.

Contribution

Name	Contribution
sv94	33.33
srisakth	33.33
sarojavu	33.33

Three of us were equally passionate about this project and worked equally hard to make it possible

THANK YOU!