ParkSense: Using Computer Vision and Deep Learning to Monitor Parking Availability

Aaron Alden
Martin, Tennessee, USA
aardalde@ut.utm.edu

Spencer Karpati
Martin, Tennessee, USA
spebkarp@ut.utm.edu

Zachary Rose
Martin, Tennessee, USA
zacerose@ut.utm.edu

ABSTRACT

Insufficient parking spaces on our campus, exacerbated by a new building occupying a large section of a previous lot, have forced students to park farther away. Consequently, this has led to increased travel times to classes, resulting in disruptions and attendance issues.

To address this challenge, we present an IoT solution utilizing OpenCV, YOLOv8, a Raspberry Pi, and a camera module. Our custom-built object detection software, leveraging OpenCV's real-time computer vision and YOLOv8's deep learning capabilities, focuses on identifying available and occupied parking spots.

The project's scope involves real-time parking availability detection using computer vision and a camera stream. The software, trained on a tailored dataset for precision, runs on a Raspberry Pi, handling data collection, calculations, and pattern recognition results.

Users can access a website interface for accurate, periodic updates on parking spot availability. While the technology's implementation is the primary focus, future iterations may explore additional features, such as insights into optimal parking times. However, we acknowledge the speculative nature of this aspect and emphasize delivering the core functionality of real-time parking detection.

1. INTRODUCTION

The purpose of developing ParkSense is to detect and deliver relevant information about parking to Students and Faculty in real-time. Using the design philosophy *The Internet of Things*, we intend to show how it's possible to use machine learning and computer vision technologies to make things more convenient in a cost-effective manner. By training a new model, this technology can be applied to many different aspects of life.

The technologies in use are primarily software-based. A camera feed is used for data input, and a Raspberry Pi can be used for data processing and delivery. We intend to show how to build a data set for training a model, how to train said model using the data set, and how you can use the model to generate a useful set of data to feed to a user interface. We want to show how to do all of this with the selected modules that we will use for this project: OpenCV, YOLOv8, and RoboFlow.

2. BACKGROUND

When we were choosing what we wanted to do for our project, we decided that it may be most effective to choose an idea that would correspond with an ongoing issue on campus. One issue that had been prevalent not only for the current semester, but has been recurring time and time again was low availability of parking spaces. This, consequentially, lead to students arriving late to class which most probably led to class interruptions.

This issue has only exacerbated further recently. This is due to the construction of a new STEM building on campus which was finally completed last semester. Before construction had begun, much of the area consisted of a parking lot that provided an ample of spaces for staff and students attending Sociology and Humanities classes. Once construction had started, much of that available parking had been taken for space needed for the new building.

Figure 1. Aftermath of New Building Construction

3. TECHNOLOGY

Parksense makes use of several technologies for monitoring the lot and presenting data. A Raspberry Pi and its associated camera module were chosen for its practicality. They are portable and inexpensive, allowing for quick deployment wherever needed. Several open license Python packages were used for building and training the model, and for real time video processing.

OpenCV will provide computer vision for real-time processing for our video feed while YOLOv8 will provide a deep learning algorithm for training on our dataset and producing weights for our model once training is complete. RoboFlow will help with labelling each image of our mock-up with taken and empty instances as it provides extensive image annotation tools needed to produce a training data-set that is accurate and efficient. We will record mean average confidence, recalls, and object loss across different epoch values to show how the algorithm produces more accurate results with more training. [1] [4]

4. DEEP LEARNING

4.1 What is Deep Learning?

Deep learning is a subset of machine-learning, which itself is a subset of artificial intelligence (AI). Machine learning wants to allow machines to learn and adapt through experience

and iteration. Deep learning takes this a step further by layering algorithms and computing units, also known as *neurons*. The difference that this makes is that where machine learning requires human assistance, usually in the form of humans manually feeding it relevant features from data, deep learning can learn these relevant features and improve its understanding of these features through simple iteration without human assistance, thus streamlining the learning process. [2]

4.2 Why Would We Need Deep Learning?

We would want deep learning for this very reason. We can create a data-set (the data being pictures of a lot in this case) that we can give to our deep learning algorithm. Then, the deep learning algorithm can then iterate over our data-set hundreds of times to learn and extract relevant features from our data. Finally, after the training process is complete, the algorithm will then form a model for us to use for object detection tasks.

5. COMPUTER VISION

5.1 What is Computer Vision?

Computer vision is a subset of artificial intelligence (AI) that allows machines to "see". Computer vision is a necessity for our newly generated custom deep learning model as the model needs to leverage OpenCV's ability to perceive visual data to be able to train properly. In simple terms, how is our deep learning model supposed to train on an object if it cannot "see"? In this case, you would need something like computer vision to solve this issue.

5.2 How Does it Work?

So how does computer vision work exactly? First, it needs to receive an inputted image or video for processing. Once it has received the input, it will then transform this input into a format that our deep learning model expects. This usually includes resizing the input to a size of 640 x 640. Then other prepossessing techniques such as sharpening and image restoration are applied to the input before training begins. Once the training process is completed, OpenCV will further refine the results by trimming out low confidence detection while annotating successful ones using a bounding box and a corresponding class label which, in this case, would be `Taken` or `Empty`.

6. THE PARKING LOT

6.1 Mock-up or Real Lot?

For our project, we decided that we would use a mock-up instead of collecting data from a real parking lot. This is because we believed that a mock-up would be more convenient as it allows us to implement different lot scenarios in a faster and more efficient manner than if we actually collected this data from a real parking lot. If we wanted to use an actual lot on campus, we would have to find a reliable spot for our Raspberry Pi to capture this data. This means that 1. the Raspberry Pi has a sufficient viewing angle over a lot and 2. was safe from varying and extreme weather conditions. This was difficult to achieve. Even if we found a good spot, we would have to scrounge through hours of footage to find lot scenarios that we would want. With a mock-up, we can simply create these scenarios by simply moving prop cars around.

6.2 Advantages of Using a Mock-up

The mock-up is relatively straight forward. It is a whiteboard with gray canvas paper applied over it. The parking spaces were created using a simple white marker and the cars used are just Matchbox cars. The mock-up proved surprisingly sufficient for providing relatively accurate photos as, when under harsh lighting, each photo of the mock-up looked effectively similar to an actual lot during the day. So, although cheap and not exactly professional, the mock-up proved extremely vital for collecting data to be processed and just for our project overall.

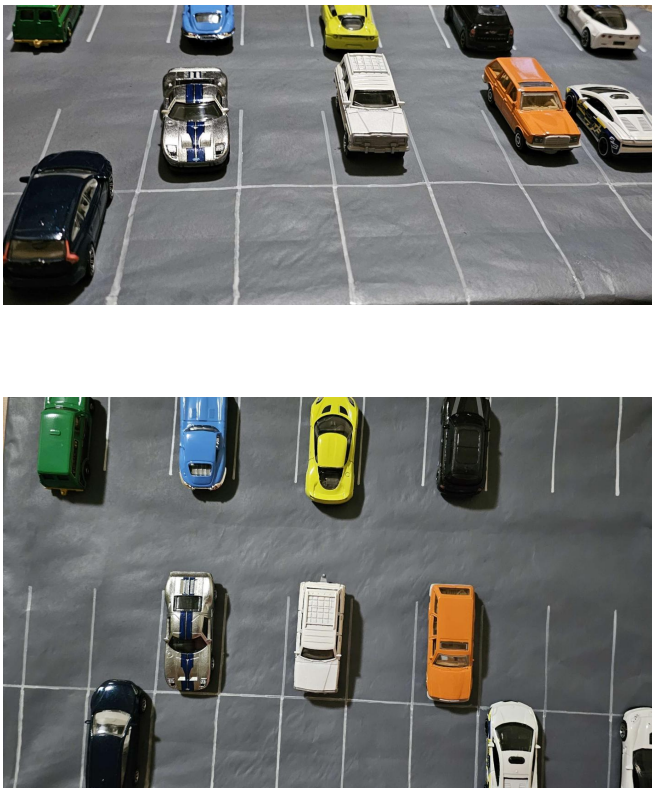

Figure 2. Images of our Mock-up

7. THE DATA-SET

The data-set that we trained our model with consists of only 20 images. Usually, when training a model, you would need it to iterate over hundreds of images. So, in most cases, only 20 images would not be sufficient for training, but, in most cases, each image of the object that you are training your model to detect will usually have only one instance of said object. In our case, each one of our images are of a parking lot, which contains many instances of both `Empty` and `Taken` parking spots. The total amount of instances that are prevalent in the entire data-set total to about 550 instances. So, although we do not have many images in our data-set, the total amount of instances in the data-set are sufficient enough. [5]

8. THE ANNOTATION PROCESS

8.1 Why Annotation?

Every instance within each image of the data-set must have an annotation. This is done by drawing what's called a bounding box over said instance. We want to use a bounding box as it serves as a guide for our deep learning model since it tells our model which sections of an image it needs to train over. This is possible because our model receives the coordinates of our bounding boxes for an image that is preceded by a class representation of 1's and 0's (1 meaning taken while

0 means empty). The coordinates are used as a parameter for what the model should only train over. This ensures that our model observes only what we want it to, which limits other objects and distractions from interfering or botching the training process. [3]

8.2 Roboflow

We decided to use Roboflow to annotate each image of our data-set as it provides convenient tools that we would want when annotating an image. The most useful tool would be the *polygon* tool. This allows us to be able to draw a bounding box line by line instead of clicking and dragging. This proves useful as it helps with preventing annotation overlap, which can occur often when annotating photos that 1. are at an angle and 2. have many instances close together. This tool ensures that our annotations remain accurate while limiting confusion for our model. Finally, Roboflow allows us to export our annotated data-set in YOLOv8 format, which we would obviously need since we are using YOLOv8.

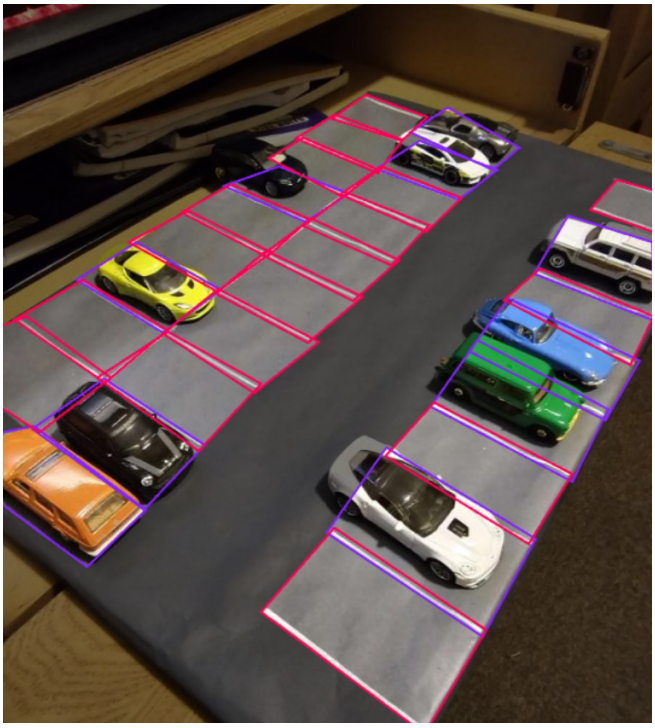

Figure 3. Annotated Image from our Data-set Using Roboflow

9. THE MODEL

9.1 Our Current Model

The model that is most stable and up-to-date that we have is named "testmodel250". This is because we had it train over our data-set for over 250 iterations or *epochs* as its called during the training process. We deemed it not necessary to train a model past 250 epochs as the score for successful detection, known as Mean Average Precision (mAP), was reaching its maximum value which was 1.0 in this case.

9.2 What is a Model and Why Do We Need It?

A model in terms of object detection can be described as a manual or reference for OpenCV that states what objects it actually needs to detect and annotate. The model is created after the training process is complete. This model is crucial for our object detection software as it provides needed learning parameters that are known as *weights*. Weights represent the strength of a connection between two neurons in our deep learning model. Weights are tuned such that our model can confidently detect the object(s) that it has trained over.

9.3 How are Weights Determined?

Before the training process begins, each weight is completely randomized and is applied to a picture in a matrix-like fashion. Once the training process begins and epochs are made, the weights are then fine-tuned using an optimization algorithm until our model begins to extract relevant features from our images like corners, edges, textures, etc. Once the training process is complete where the last epoch has occurred, the newly-tuned weights are then saved and stored in our model for use in object detection tasks.

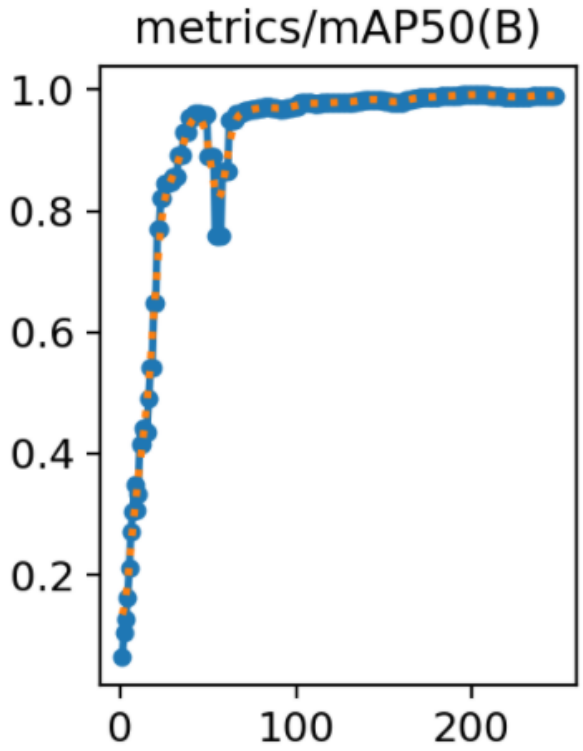

Figure 4. mAP Scores After 250 Epochs

10. THE SERVER

10.1 Node.Js

In order to update the web-dashboard in real time, a simple webserver is needed. Node.Js was used to monitor for updated information from the computer vision script, as well as to update the dashboard accordingly. Once the new information is made available in a file, the server communicates it to the client.

10.2 Websocket Communication

The communication is done through a websocket connection. When the client connects to the dashboard, a websocket is formed between the client and the server. When the client

receives the new data through the websocket, it's able to update the dashboard. Like this, the client can remain up to date as information is available, without needing to refresh.

10.3 Security Concerns

A websocket works well for demo purposes, but it's far from the best option. For starters, a two way communication is unnecessary for the web dashboard, as it has nothing it needs to communicate to the server. Furthermore, the connection opens up the server for attack, and in practice a more secure option would be used.

10.4 Cloud Computing

In our current demo model, the server and the object detection script are both running on the Raspberry Pi directly. Besides the aforementioned security issues, the script runs slowly on the limited hardware, taking over 30 seconds of processing before completion. In order to improve this issue, we looked into delegating our computation to a dedicated server or the cloud. If the Raspberry Pi sent an image to be processed remotely, the server would be more secure and the dashboard could be updated closer to real time.

11. CHALLENGES

Throughout development on the project, there were several challenges we had to overcome. This was the first time we had worked with deep learning, computer vision, and the Raspberry Pi, so there was a large adjustment period as we experimented with new technologies. Finding the right software to annotate the data set with was especially difficult. Getting the Raspberry Pi environment set up involved solving compatibility problems between Python packages and the hardware.

11.1 The Raspberry Pi

The Raspberry Pi was difficult to work with. The hardware is minimal, so the speed at which you can compile is slow. This made setting up the Python environment challenging, since each attempt took hours to try. An option we want to try next time is cross compilation, where you can compile for the Raspberry Pi on a normal computer, speeding the process up greatly. The Raspberry Pi 2B, which we started with, was also incompatible with Yolo due to its 32 bit hardware. It took a while to find out exactly what the problem was before we were forced to upgrade to the Raspberry Pi 3.

11.2 The Model

Building a model with Yolo that works well under varied conditions was an issue we focused our time on. Our model was built under the assumption that our data would be collected from a constant camera angle, such as a security camera in a parking lot. However, a data set with only one angle and consisting lighting is not resilient to variance in the input to the script. Creating a more varied data set increases the accuracy of the object detection under different weather and lighting.

12. FUTURE WORK

With the successful implementation of our proof-of-concept, there is considerable potential for enhancing ParkSense in the future. One prominent real-world application involves developing a mobile application for students, faculty, and staff to monitor real-time parking availability. Our model will continuously update the remaining and vacant spaces for each tracked parking lot. Additionally, incorporating push notifications would provide users with alerts when a specific lot is nearing full capacity or vice versa. Over time, the accumulation of data will enable us to conduct statistical analyses on parking trends.

REFERENCES

1. G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
2. IBM Data and AI Team. Ai vs. machine learning vs. deep learning vs. neural networks: What's the difference? July 2023.
3. Dhanashree. Image processing and bounding boxes for ocr. August 2022.
4. Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics, January 2023.
5. Spencer Karpati. Parking lot dataset. https://universe.roboflow.com/university-of-tennessee-at-martin/parking-lot-saxzu, sept 2023. visited on 2023-10-31.