Detection of Objects in Video Streams Using Deep Convolutional Neural Networks

Zhankun Luo

Ben Matone

Andrew Uzubell

Academic Advisor: Bin Chen

Departments of Engineering

College of Engineering and Sciences

Purdue University Northwest, Hammond Campus

April 24, 2019

Executive Summary

This report presents a new approach on how to make the burden of finding a parking spot a thing of the past. This method of parking lot occupancy detection will lead to easier parking and less unneeded traffic in a parking lot. In a day and age where time is money, every second counts, and nobody should have to spend valuable time hopelessly searching for a spot to park.

Programming cameras to detect specific objects has many different benefits and applications. One practical use of this technology is to implement it into cameras in a parking lot in order to calculate the number of available parking spaces. Then, display the number of open parking spaces for the motorists to view. That way motorists will know prior to entering a parking lot if there is an open spot for them to park. This will lead to a better overall parking experience by eliminating unneeded traffic and searching in parking lots.

There are already a few different ways that parking lot occupancy is calculated and displayed. However, programming cameras to calculate the number of open parking spaces will be more efficient than the current methods. This method will be more accurate and require less money and maintenance than the current solutions. It also has the added benefit of detecting parking spaces that are occupied by something other than a vehicle.

The new design builds off of current object detection techniques in order to fine tune the detection process. This design utilizes a binary classifier to determine whether or not a parking spot is occupied or empty. In addition to this, the techniques of image registration, voting, and perspective transformation have been applied to the videos of the parking lots. This makes the program more consistent for low angle videos and for parking spaces that are hard to see. The proposed solution is easy to install and requires little to no upkeep. Most importantly, it’s more accurate and reliable than the current methods.

Abstract

The purpose of this report is to streamline the parking process by having parking lot cameras see, detect, and display whether or not a parking spot is occupied or empty. Current parking spot occupancy detection utilizes ground sensors, or bars with infrared sensors, among other expensive, high maintenance, and impractical methods. These pitfalls prevent the aforementioned methods from being a realistic solution. There are some parking spot occupancy techniques that use computer vision. However, this project improves upon those techniques by introducing an original design that is tested with videos that have many different angles, lighting, stability, weather conditions, and distance.

The design detailed in the report utilizes a convolutional neural network (CNN or ConvNet) in order to capture and store the features of thousands of cars. Parking lot videos are then segmented in order to turn the video into all of its frames. The first frame of the of the parking lot video then has all of its parking spaces labeled and their coordinates stored. These coordinates are pulled from the program and run through the CNN binary classifier to determine whether or not the spot is occupied. If it is determined to be occupied it is highlighted red, if it is empty it is highlighted green. The overall number of occupied and empty spaces are displayed on the video along with the highlighted parking spaces.

Table of Contents

Introduction 1

Background 1

Objectives 1

Constraints 2

Environmental Impact 2

Approach 3

- Obtain Data Sets 3
- Design and Evaluate the CNN Architecture 4
- Acquire Appropriate Hardware 5
- Obtain Parking Lot Videos 6
- Apply Mask to Parking Lot Videos 6

Results 11

Conclusion 25

References 26

Appendix A: Milestone Log 27

Appendix B: Project Schedule 28

List of Figures

Figure 1. Images Contained in the Training Data Set 2
Figure 2. Example of Filters Developed with CNN 3
Figure 3. Image in Data Set Before Rescaling and After Rescaling 4
Figure 4. Snip of the Code Displaying the Accuracy for the Classification 6
Figure 5. Outputs for the Small Test Images 7
Figure 6. Labeled Parking Spaces 8
Figure 7. Snip of the Coordinates of the Parking Spots. 8
Figure 8. Original Image of a Parking Space in the Video 9
Figure 9. Rescaled Image of a Parking Space in the Video 9
Figure 10. Outputs for one of the Parking Lot Images 9
Figure 11. Original Image, Mask, and Original Masked 11
Figure 12. Confusion Matrix 12
Figure 13. Label Accuracy Pie Chart 12
Figure 14. Annotation of VOC data set 13
Figure 15. VOC data set with Bounding Box 14
Figure 16. Arrays read by YOLO 14

List of Tables

Table 1. Accuracy Comparison after Improvements

Table 2. Results from UMEA University Project

Table 3. Accuracy Comparison Between Projects

Introduction

Two popular methods currently being used to calculate parking lot occupancy are completed using either barriers with infrared sensors or using ground sensors. However, neither method is ideal, and each come with their own set of negatives. For one, using barriers and infrared sensors is not practical in every situation. Parking lots with high traffic flow or that experience a lot of vehicles entering and not parking would struggle using this method. This method also does not take in to account spaces that may be occupied from other obstructions other than vehicles like bicycles, snow, large debris, etc. Ground sensors excel in these aforementioned categories, but have their own downfalls. Using ground sensors requires a sensor in every parking spot and the maintenance of each individual sensor. Installing all of these sensors and maintaining them becomes expensive.

A need for an efficient and affordable method to detect parking space occupancy is needed. Because of this, our design plans to build off of the newer technique of detecting available parking spaces through the use of convolutional neural networks.

Background

The use of ConvNets has been a huge advancement in the field of artificial intelligence because they are particularly great at analyzing visual imagery. While they are great at analyzing visual imagery, they also require a lot of processing power. Because of this, they have not been able to be widely implemented until about the last ten years when the GPU technology advanced enough to compute these heavy algorithms. This makes the field of computer vision relatively new when it comes to using CNNs to analyze images

Objectives

There have already been attempts at designing a ConvNet architecture that can detect open parking spaces. However, this technology is relatively new so there is room for improvements to be made. The main goal of this project was to present an accurate and cost efficient method of displaying the number of open parking spaces in order to streamline the parking process. This was completed by utilizing a binary classifier developed from a CNN. This classifier is applied over each parking spot and calculates whether or not the spot is occupied. If the spot is occupied, then it is highlighted red. If the spot is empty, then it is highlighted green. In addition to this, the total number of occupied and empty parking spots in the parking lot are displayed on the video.

Constraints

At the start of the project, the team had limited knowledge regarding CNNs. The team researched how CNNs function, learned the basics of programming in Python using the numpy and PyTorch packages, and learned the application techniques of CNNs. These steps were required before the design process could begin.

The programming techniques used in this project required a lot of GPU space. This led to a lot of revision of techniques in order to get the program to run properly. Early on in the project, the deep learning technique YOLO was developed in order to detect moving cars in addition to parked cars. However, the YOLO program required a large amount of GPU space in order to run. After many revisions, this problem still persisted. Because of this, it was decided to put the YOLO technique aside and instead improve upon the CNN classifier.

Video camera footage of a parking lot was needed to test the program but proved harder to obtain than expected. The project team had the luxury of being able to work with the Purdue Northwest Police Department to download some of the Purdue Northwest (PNW) security camera footage. However, the security footage obtained was at a low angle, which resulted in a lot of overlapping of parked cars. This made it extremely difficult to detect cars accurately since their view was blocked by other parked cars. Because of this, it was decided to capture original parking lot footage of the PNW parking lot north of Porter Hall using the camera purchased with funding from the Student Research Office. This camera would solve the poor angle problem. However, it did not arrive until March 18, 2019, which did not allow for much time to test the program.

It was decided to order a drone with the grant money obtained in the second semester of the project. This drone arrived too late in the project schedule in order to implement it into our project. However, a senior design group that builds off of this project's results could utilize this drone in various ways for their project.

Environmental Impact

Searching for a parking spot can seem like a never-ending chore. The average motorist in the U.S. spends 17 hours and $345 a year searching for a spot to park. The total amount for all of the U.S. motorists is about $72.7 billion [10]. These costs come from the wasted fuel consumed when searching for a spot. With this wasted fuel comes wasted emissions from the vehicles. The goal of implementing this project's parking spot occupancy technique is to cut down on the parking time and cost for drivers. This would result in a major decrease in wasted emissions and would help cut down on vehicle emissions.

Approach

The project was divided into the following stages:

1. Obtain Data Sets
2. Design and Evaluate the CNN Architecture
3. Acquire Appropriate Hardware
4. Obtain Parking Lot Videos
5. Apply Mask to Parking Lot Videos

Below is an overview of each stage.

Obtain Data Sets

The first step was to obtain data sets [3]. This was done first that way we knew how go about designing our ConvNets. The images obtained were of individual parking spaces that were either occupied or free and consisted of different angles, colors, and lighting. It was important to obtain a wide variety of parking space images in order to ensure our CNN model could work in various scenarios. The images in these data sets were converted into a numeric matrix where each number represents a pixel value of that image. The data set gathered was separated into three categories: train, validate, and test. These data sets were independent of each other to ensure no overfitting occurred.

Training Data Set

The training data set was the largest consisting of 6171 images representing occupied or open parking spaces (Fig. 1). The training data set is the largest of the three because the model requires a lot of unique examples in order to be able to pick out important features from images.

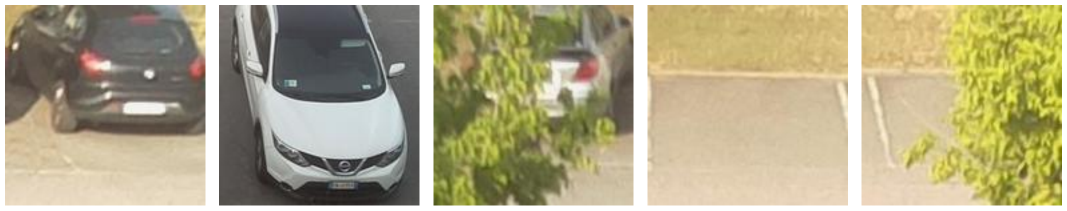


Figure 1. Images Contained in the Daytime Training Data Set

Validate Data Set

The validation data set only contains 48 images. These images are similar to the training images but are independent of the training data. The validation data set is used to evaluate our CNN architecture. This way any adjustments that need to be made to the weights of our hyperparameters can be made before the final testing stage.

Test Data Set

The test data set contains 49 images. These images are similar to the other data sets but are unique from them. This ensures that an honest result is obtained from testing the test data set.

Design and Evaluate the CNN Architecture

The CNN architecture was developed using the Python Programming language. In addition to Python, the programming libraries torch and Numpy[1][2] were utilized to create the CNN architecture. Numpy allows users to create large multi-dimensional arrays and matrices. Torch is a computing framework that allows users to utilize the power of their GPU to compute large arrays and matrices.

This stage was divided into 3 subsections:

Design the CNN Architecture

A CNN architecture was developed in order to “teach” our program what features to look for in an image. These features would then be represented as weights that would in turn make up our filters (Fig. 2). These filters would later be used to detect certain key features seen in an image and determine whether or not the image we are viewing is in fact occupied or open.

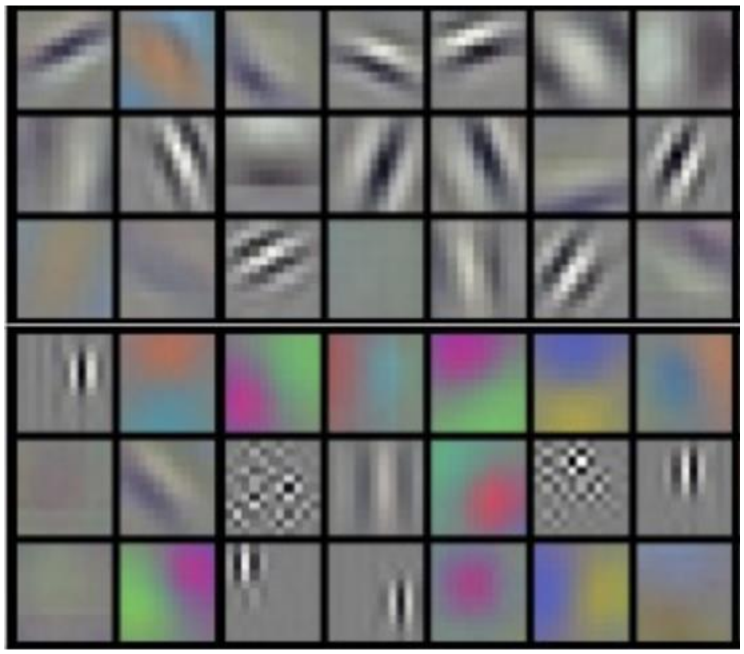


Figure 2. Example of Filters Developed with CNN

This was completed by designing a CNN architecture that consisted of an input image’s raw pixel values, 4 convolutional layers, 1 fully connected layer and a binary output indicating whether the image represented a free parking space or one that was occupied. The results of all the images were then compared with the expected result of the classification to determine the models efficiency. The filter parameters calculated from the CNN are then stored. The stored parameters can then be used to detect for features in images without having to retrain the CNN. This saves a lot of time and memory of GPU, which are some of the major benefits of using CNNs for object detection.

It was decided to omit a pooling layer in our model. Pooling layers are often used to downsize the spatial dimensions of our input image so that different features can be extracted. Instead of using pooling layers, our design uses a stride that has a value of 2. This effectively reduces the size of our image in the convolutional layer of our network without having to add in pooling layers.

Training the CNN

After the CNN model is designed, it is necessary to train the CNN on what features to look for in order to determine if a spot if occupied. The training data set along with their corresponding labels are fed into the CNN model multiple times so the classifier can improve its accuracy in detecting those features. The number of times the whole data set has been inputted into the CNN is called EPOCH. During the training step, we also introduced a loss function to quantify the accuracy of our CNN model. It is ideal to get the loss function as close to zero as possible. The loss function basically compares the predicted value of the classification with the actual value of the image. After the loss is computed, the parameters of the CNN are altered in order to obtain the most accurate CNN model.

Evaluate Design

Once the CNN is trained it must be evaluated in order to see how reliable it is at classifying new images introduced into the system. This is done by inputting the validation data set into the CNN model and calculating the accuracy of classifying these new images. The reason this is performed is because the CNN model may be really accurate when classifying the training images but not accurate enough when trying to classify new images. If the accuracy of classifying the val data set is low, then the EPOCH must be adjusted to avoid overfitting. After the val data has been verified to be classified accurately, the test data is introduced as the final accuracy check.

Acquire Appropriate Hardware

To further test the program, a video camera was required. The video camera was used to capture original parking lot video at an angle the program can work with. A microSD card was needed to store and transfer the video and a tripod was needed to stabilize the camera. Lastly, a 2TB hard drive was needed to download and transport the security camera footage obtained from the PNW Police Department.

Video Camera

After researching various video cameras, the group decided on the Sony Alpha 6000. This camera records video in 1080p, which allows for the program to analyze the video with greater accuracy because there are more pixels to extract data from. Another benefit of this camera is that it records in mp4 format, which can be easily read by the program.

MicroSD Card and Tripod

With the chosen camera, real-time streaming was not feasible, so a 32 GB microSD card was purchased to store and transfer the video recordings. The camera fit onto a tripod that was provided, so it was not necessary to purchase one. The tripod was essential for ensuring the accuracy of the program when reading the video because shifts in the image were seen to influence proper detection of whether a parking spot was occupied or not.

Hard Drive

A very large external hard drive was needed to download and transport the PNW security camera footage. It was decided that a 2TB hard drive would provide enough storage to store all the needed security videos.

Obtain Parking Lot Videos

In order to determine how well the CNN classifier works in real world applications, it needed to be applied to many videos with varying angles, distance, weather, and lighting conditions.

Three different types of videos were collected:

PNW Police Department Security Videos

The PNW Police Department was contacted and provided security camera footage of the parking lot on the west end of the Lawshe building. The low angle of the footage made it difficult for the program to detect parking spaces because the vehicles closer to the camera overlapped the other vehicles and parking spaces.

Original PNW Parking Lot Video

Using the video camera that was ordered, the Porter parking lot of PNW was recorded. The recording was done from the top of the parking garage to allow for a full view of the parking lot and to prevent overlapping of the parking spaces. This video captured vehicles pulling into the parking spaces, which allowed for testing of the binary classifier.

YouTube Videos

The PNW security footage and the original parking lot videos did not include all of the desired conditions that were to be tested. Because of this, several different YouTube videos were gathered. Each video included a different type of condition that was to be tested that was not present in the PNW security footage and original parking lot video.

Apply Mask to Parking Lot Videos

After the CNN model was finalized, the next step was to apply it to a video of an entire parking lot. In this step, the CNN was applied on each parking spot in the video and it determined whether or not the space was occupied or empty. These spots were then indicated to be occupied by placing a red rectangle over the occupied spots and a green rectangle over the empty spots.

The steps followed to complete these conclusions are as follows:

Perspective Transformation

The technique of perspective transformation was applied to all of the parking lot videos that were captured at a lower angle. The lower angle view makes it very difficult to box the parking spaces when labeling, since the box cannot avoid overlapping into the next parking spot (Fig 3). Applying perspective transformation to these videos will transform the video from having an angled viewpoint, to one that appears to be from above. The video is transformed by multiplying the original coordinates with the perspective matrix M (Fig 4). In figure 4, [u,v,w] represents the coordinates in the original image, the matrix containing 'a' is the perspective matrix, and [x',y',w'] are the coordinates after transformation. After this transformation is applied and the classification is determined, the video is transformed back with inverse matrix of M to its original angle. This eliminates the problem of overlapping when labeling the parking spaces in the videos (Fig 5).

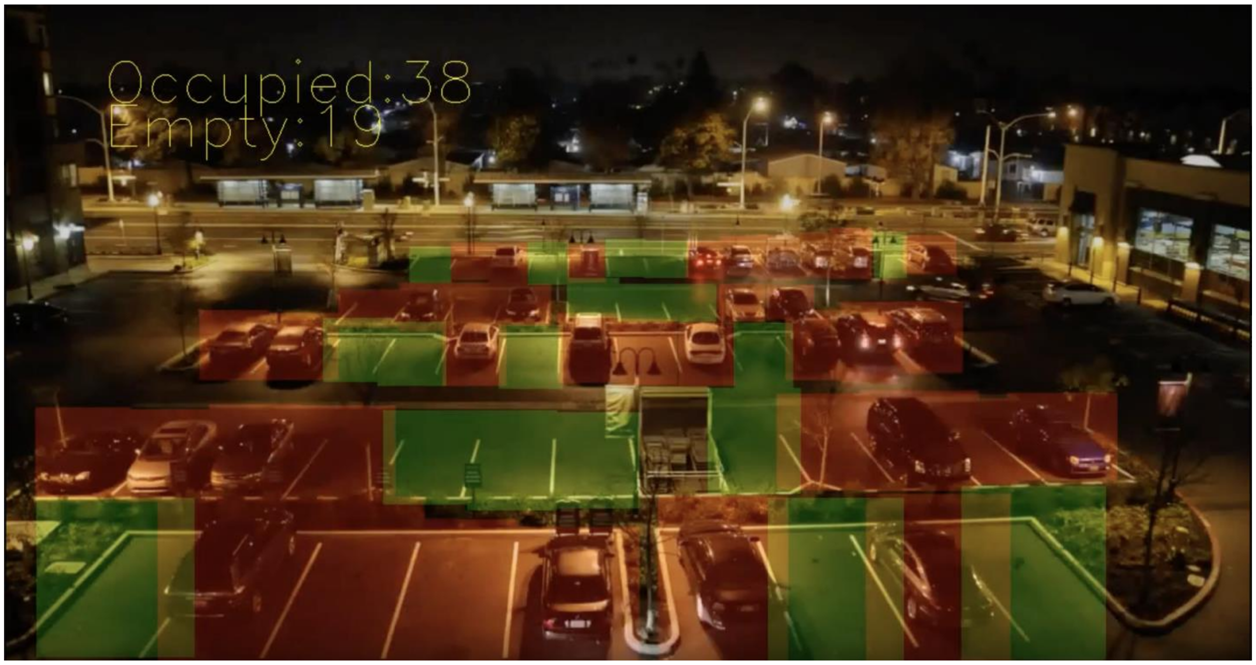


Figure 3. Mask Applied to Video with No Perspective Transformation

$$[x', y', w'] = [u, v, w]\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Figure 4. Perspective Transformation Calculation

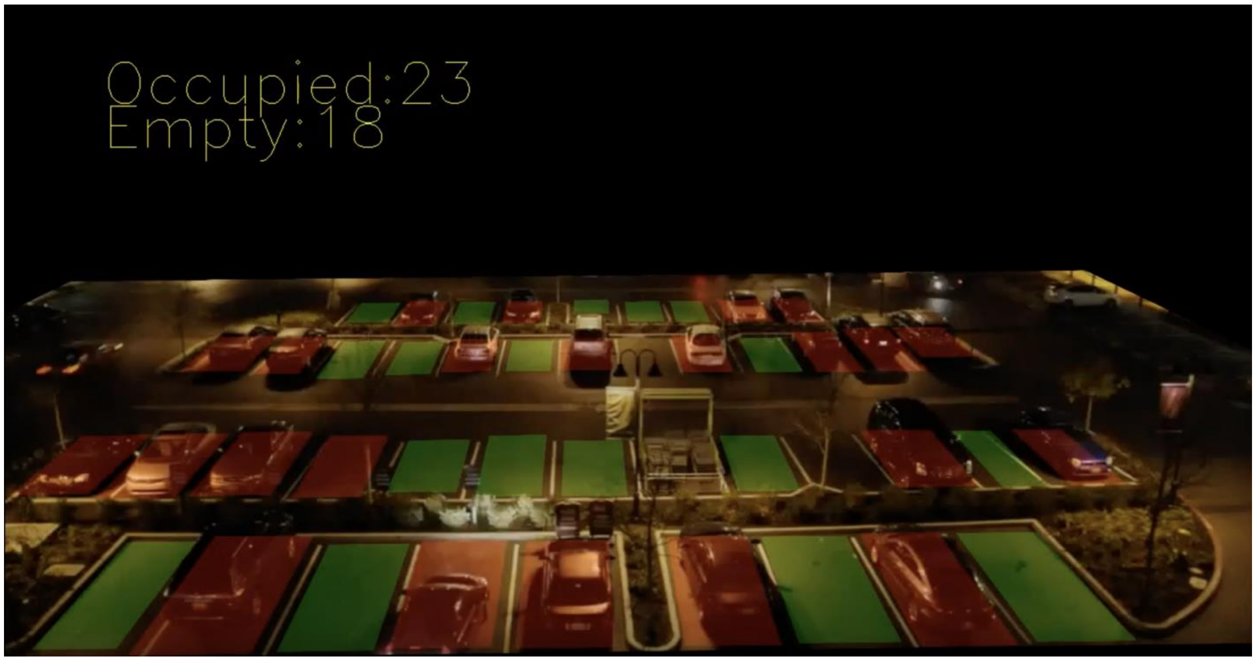
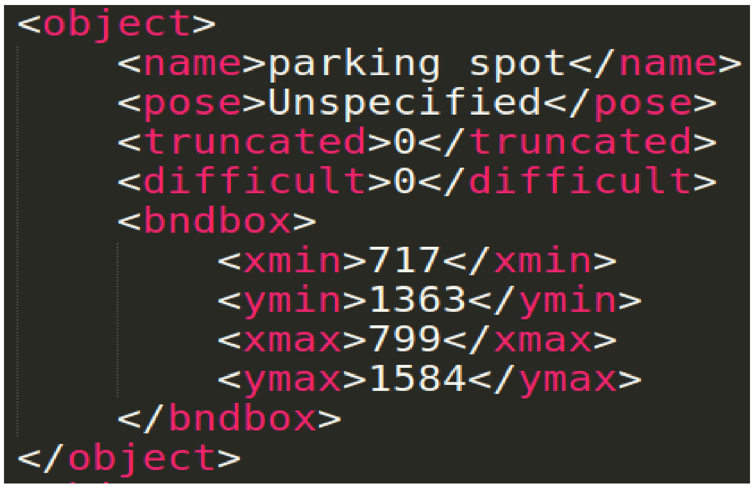


Figure 5. Mask Applied to Video with Perspective Transformation

Labeling the Parking Spots

In order to apply the classifier to each parking spot, the program needs to know where each parking spot is located within the video. The first step to find the locations of all the parking spots in the video, is to cut up each video frame by frame. This was done using OpenCV. The first frame of these videos were used to complete the labeling of each parking spot in the parking lot picture. Using various image annotation tools (like RectLabel and Labelme), each parking spot was labeled. They were labeled by boxing each parking spot within the image and labeling them “parking spot”. Every time a parking spot was boxed, the annotation tool would save the pixel coordinates. Each box’s coordinates were represented by four numbers: [xmin,ymin,xmax,ymax]. All the parking spot coordinates for one image were then saved to an XML file (Fig. 6).



```
<object>
    <name>parking spot</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
        <xmin>717</xmin>
        <ymin>1363</ymin>
        <xmax>799</xmax>
        <ymax>1584</ymax>
    </bndbox>
</object>
```

Figure 6. Example of one Labeled Parking Spot in the XML File

Video Stabilization

Most of the videos that were gathered had some movement. Since they are not very stable, each frame alters the parking spot locations slightly. This causes problems when trying to apply the mask to all of the parking spaces, since they will have different locations. To ensure that the mask is accurately applied to each frame, the technique of image registration was introduced. Image registration was introduced by first finding the coordinates of a landmark within the video that does not change (i.e. tree, building, etc.) The location of this landmark in the first frame is then compared to its location in all of the following frames. The offsets of these locations are calculated, and the frames are shifted to the same location as the landmark in the first frame. This will result in all of the parking spots in each frame having the same locations.

Creating and Applying the Masks

Now that the coordinates of all of the parking spaces are known for the given video, the CNN classifier can be applied to each space. This step is performed by taking the first image of the hundreds of frames that make up the video. Then, each parking space is cropped out of the image from the known coordinates of the spaces that were stored while labeling the parking spots. These cropped images are transformed into a list, more specifically, a Numpy array. This array is then converted into a tensor and the images are reshaped into (3, 32, 32) so they can be used in the CNN model. Each of these parking spot images are then fed into the CNN to determine whether or not the spot is occupied. The output will be 1 if the spot is occupied and 0 if the spot is open. The parking spaces that were determined to be occupied will then have their respective boundary box filled in red and the empty spaces will be filled in green on the mask of that image of the video. The overall number of occupied and empty spots will also be displayed on each frame of the video. This process will continue for all of the frames of the video. All the masks are then laid over their respective image. All of these images with masks over them are put back together which results in a video representation of the CNN classifier in action.

Voting

The technique of voting was implemented into the classification stage to improve the accuracy and consistency of the classifiers. Voting works by classifying every frame in the video. It then calculates the average classification of the previous 2N-1 frames. If over half - N frames are classified as occupied, the parking spot is determined to be occupied and a red mask is put over the parking spot. Otherwise, the green mask is added to the parking spot (Fig. 7). This technique cuts down on the sporadic classification that can happen in low light and obstructed views. Voting helps in these scenarios by not changing the classification of a spot just because it could not be seen very well for a couple frames. It also helps cut down on incorrectly classifying a spot as occupied when a car drives by a parking spot for a few frames.

$$a[n] = Net\left(img[n]\right), \quad n \text{ indicates number of image frame}$$

$$a[n] = \begin{cases} 1 & \text{Net "thinks" the parking spot is occupied} \\ 0 & \text{Net "thinks" the parking spot is empty} \end{cases}$$

$$a[n-(2N-2)] + a[n-(2N-3)] + \cdots + a[n] = b[n]$$

$$\text{output}[n] = \left\lfloor \frac{b[n]}{N} \right\rfloor = \begin{cases} 1 & \text{Add a Red mask to the image} \\ 0 & \text{Add a Green mask to the image} \end{cases}$$

Figure 7. Example of the Voting Code

Results

This completed version of our CNN architecture has the ability to detect open and occupied spots from the videos that were acquired.

The results are explained below in the following order:

- Rescaling of Data Sets
- Evaluating Design
- Segmenting Video
- Creating the Mask
- Evaluating Results
- Testing Additional Videos
- Improving Program and Retesting
- Comparing to Other Programs

Rescaling of Data Sets

The images in all of the data sets that were gathered had to be resized into the same size in order to work with the CNN model. Before rescaling was done, the images had a size of (150, 150, 3) and had a size of (32, 32, 3) after rescaling (Fig. 8).

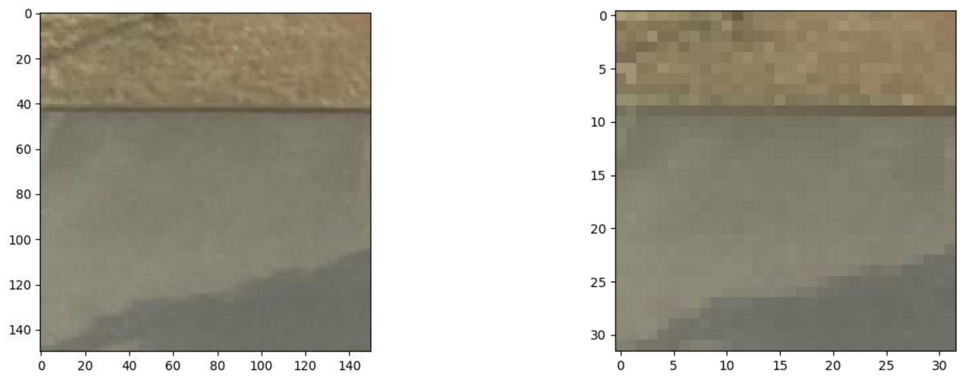
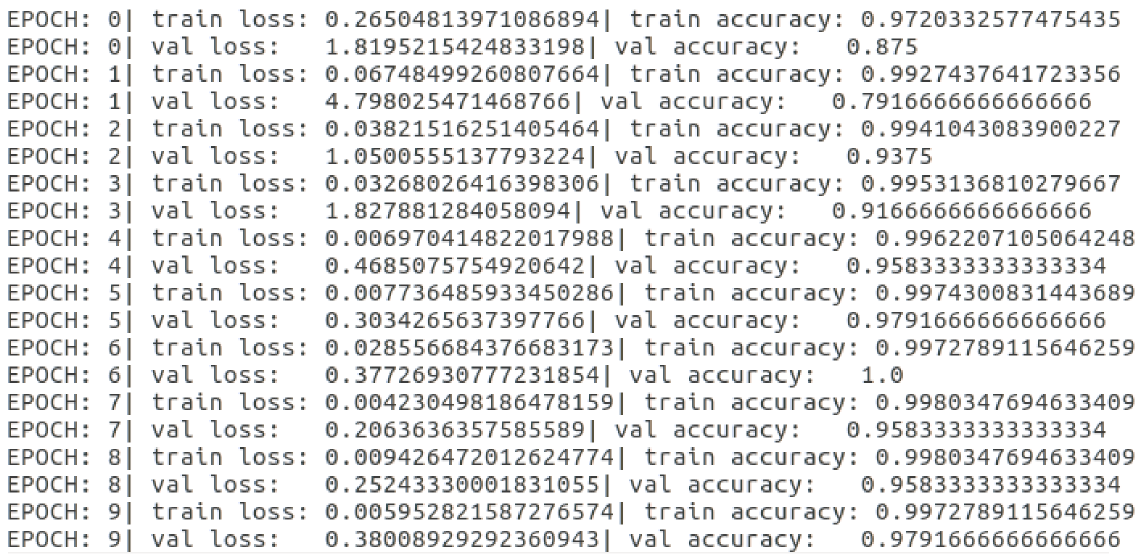


Figure 8. Image in Data Set Before Rescaling (left) and After Rescaling (right)

Evaluating Design

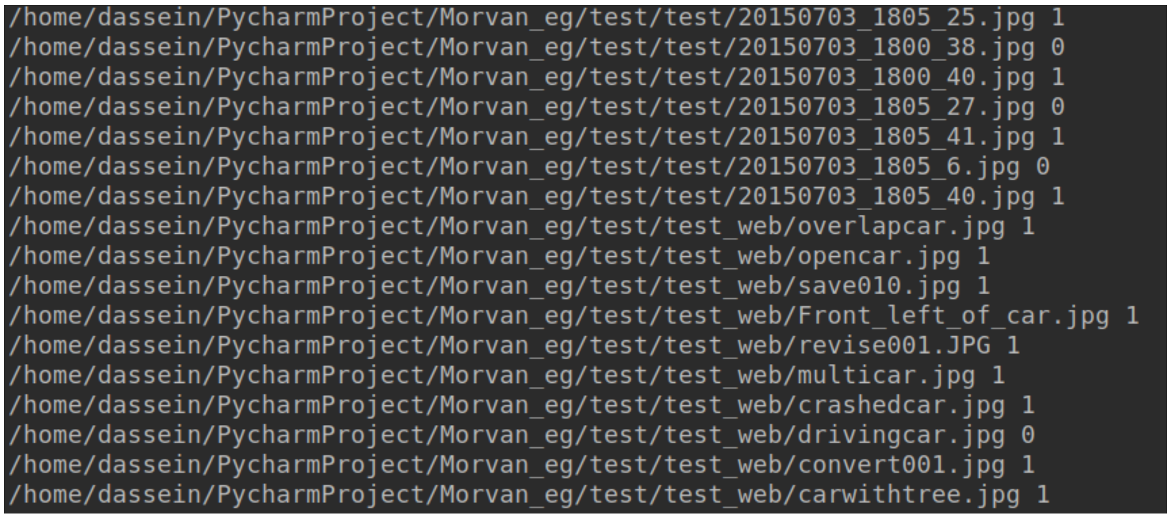
After the Binary Classification model was designed it had to be tested to ensure its accuracy. This was completed by training the CNN model with the resized images from the training data set. After the CNN model was determined to be accurate, the val data set was introduced in order to calculate the CNN's accuracy when classifying these new images. The code detailing the accuracy of the classification for each EPOCH of the training and val data sets was written so the accuracy could be evaluated (Fig. 9).



```
EPOCH: 0| train loss: 0.26504813971086894| train accuracy: 0.9720332577475435
EPOCH: 0| val loss:   1.8195215424833198| val accuracy:   0.875
EPOCH: 1| train loss: 0.06748499260807664| train accuracy: 0.9927437641723356
EPOCH: 1| val loss:   4.798025471468766| val accuracy:   0.7916666666666666
EPOCH: 2| train loss: 0.03821516251405464| train accuracy: 0.9941043083900227
EPOCH: 2| val loss:   1.0500555137793224| val accuracy:   0.9375
EPOCH: 3| train loss: 0.03268026416398306| train accuracy: 0.9953136810279667
EPOCH: 3| val loss:   1.827881284058094| val accuracy:   0.9166666666666666
EPOCH: 4| train loss: 0.006970414822017988| train accuracy: 0.9962207105064248
EPOCH: 4| val loss:   0.4685075754920642| val accuracy:   0.9583333333333334
EPOCH: 5| train loss: 0.007736485933450286| train accuracy: 0.9974300831443689
EPOCH: 5| val loss:   0.3034265637397766| val accuracy:   0.9791666666666666
EPOCH: 6| train loss: 0.028556684376683173| train accuracy: 0.9972789115646259
EPOCH: 6| val loss:   0.37726930777231854| val accuracy:   1.0
EPOCH: 7| train loss: 0.004230498186478159| train accuracy: 0.9980347694633409
EPOCH: 7| val loss:   0.2063636357585589| val accuracy:   0.9583333333333334
EPOCH: 8| train loss: 0.009426472012624774| train accuracy: 0.9980347694633409
EPOCH: 8| val loss:   0.25243330001831055| val accuracy:   0.9583333333333334
EPOCH: 9| train loss: 0.005952821587276574| train accuracy: 0.9972789115646259
EPOCH: 9| val loss:   0.38008929292360943| val accuracy:   0.9791666666666666
```

Figure 9. Snip of the Code Displaying the Accuracy for the Classification

The classification of these data sets turned out to be highly accurate. The parameters of the CNN were then saved to be used later on the test data. Next, the test data was inputted into the CNN model and the classification of the test images was displayed (Fig. 10). When the output is ‘1’ the spot is classified as occupied and when it is ‘0’ the spot is classified as open.



```
/home/dassein/PycharmProject/Morvan_eg/test/test/20150703_1805_25.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test/20150703_1800_38.jpg 0
/home/dassein/PycharmProject/Morvan_eg/test/test/20150703_1800_40.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test/20150703_1805_27.jpg 0
/home/dassein/PycharmProject/Morvan_eg/test/test/20150703_1805_41.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test/20150703_1805_6.jpg 0
/home/dassein/PycharmProject/Morvan_eg/test/test/20150703_1805_40.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/overlapcar.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/opencar.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/save010.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/Front_left_of_car.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/revise001.JPG 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/multicar.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/crashedcar.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/drivingcar.jpg 0
/home/dassein/PycharmProject/Morvan_eg/test/test_web/convert001.jpg 1
/home/dassein/PycharmProject/Morvan_eg/test/test_web/carwithtree.jpg 1
```

Figure 10. Outputs for the Small Test Images

Segmenting Video

The video of the entire parking lot had to be hand labeled (Fig. 11).

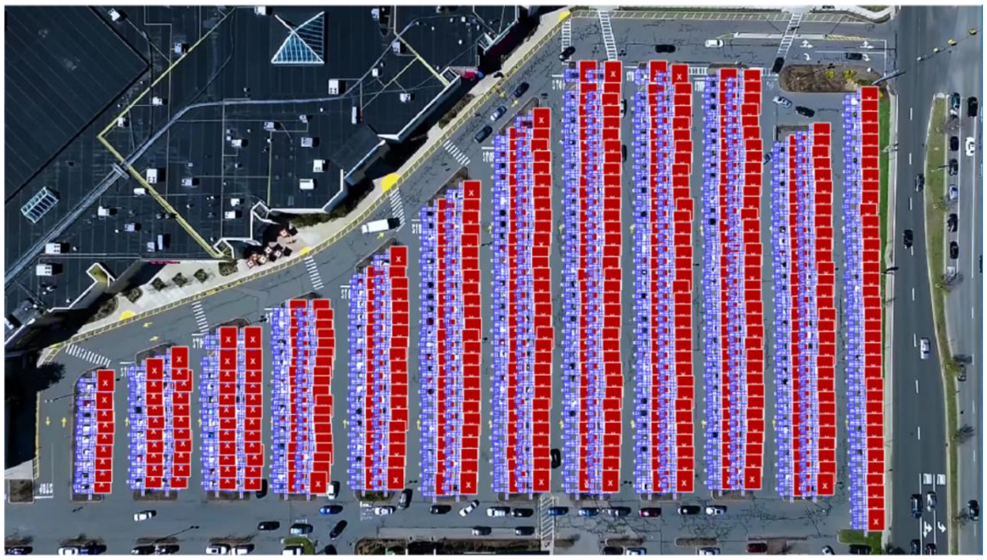
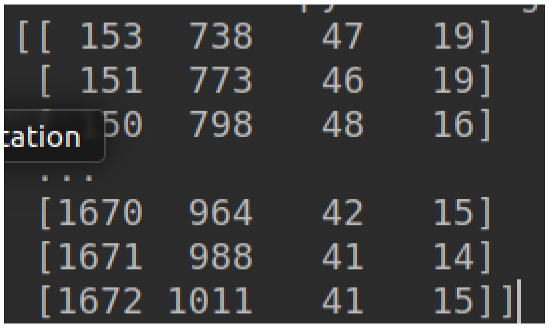


Figure 11. Labeled Parking Spaces

After labeling the spaces, the coordinates of each parking spot was recorded and saved to be used later for cropping the images and creating the masks (Fig. 12).



```
[[ 153  738   47   19]
 [ 151  773   46   19]
  50  798   48   16]
 ...
 [1670  964   42   15]
 [1671  988   41   14]
 [1672 1011   41   15]]
```

Figure 12. Snip of the Coordinates of the Parking Spots.

Creating the Mask

The coordinates of each parking spot in the video is known. A mask of the parking spaces in the video can now be made utilizing these coordinates. First, these coordinates were used in order to crop out the individual parking spaces' images (Fig. 13).

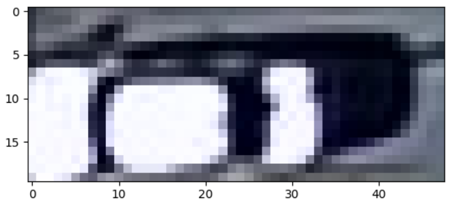


Figure 13. Original Image of a Parking Space in the Video

These images were of size (19,47,3). All the images had to resized to the dimensions (32, 32, 3) in order to work with our CNN model (Fig. 14).

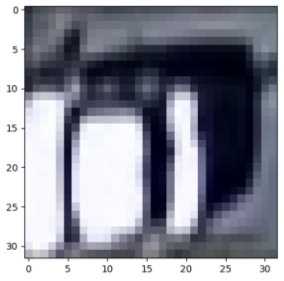
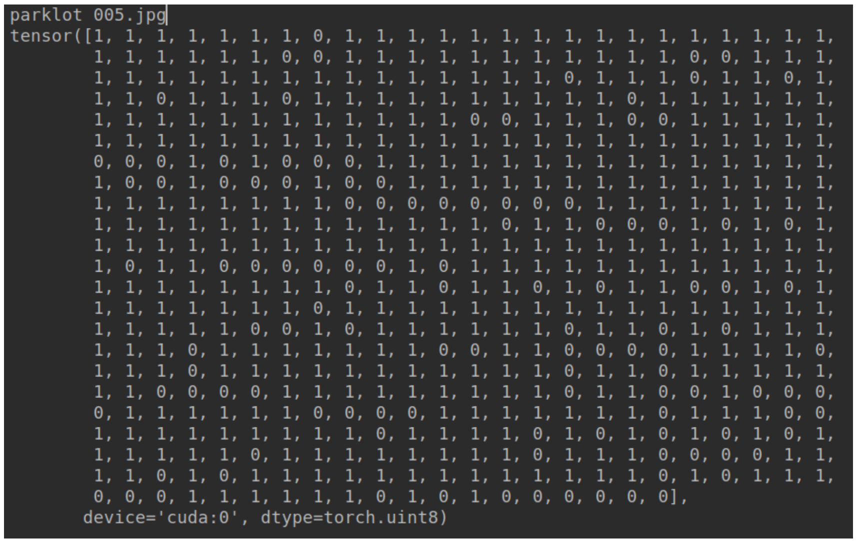


Figure 14. Rescaled Image of a Parking Space in the Video

The resized images were then saved in a list containing all the resized images for the overall image of the entire parking lot. This list was then converted into a tensor and inputted into the CNN model. The output obtained showed the classifications of each parking space in the entire parking lot image (Fig. 15)



```
parklot 005.jpg
tensor([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
        0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1,
        1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
        0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
       device='cuda:0', dtype=torch.uint8)
```

Figure 15. Outputs for one of the Parking Lot Images

This information was then used to make a mask for the entire parking lot. The parking spaces occupied were colored red and the open spaces were colored green. After the mask was created for each image of the entire parking lot, the mask was put on top of the original image of the parking lot to highlight the parking spaces (Fig. 16). This resulted in the masked original image.

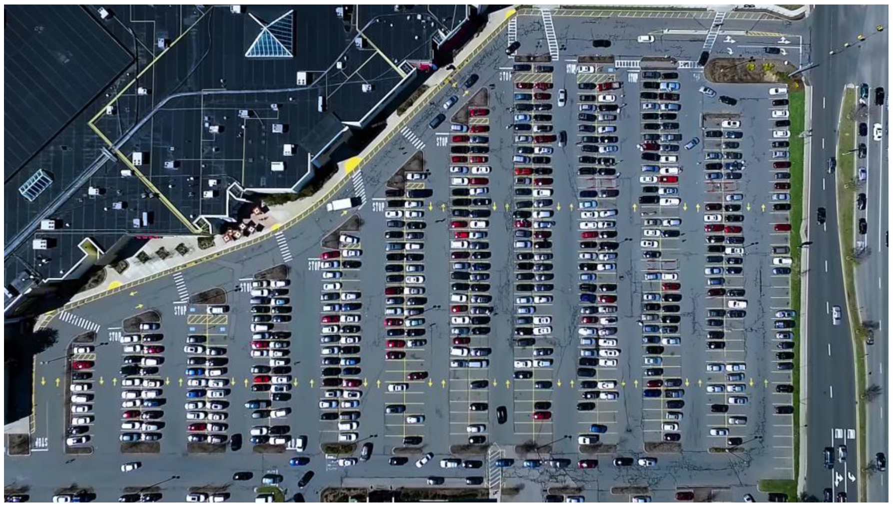
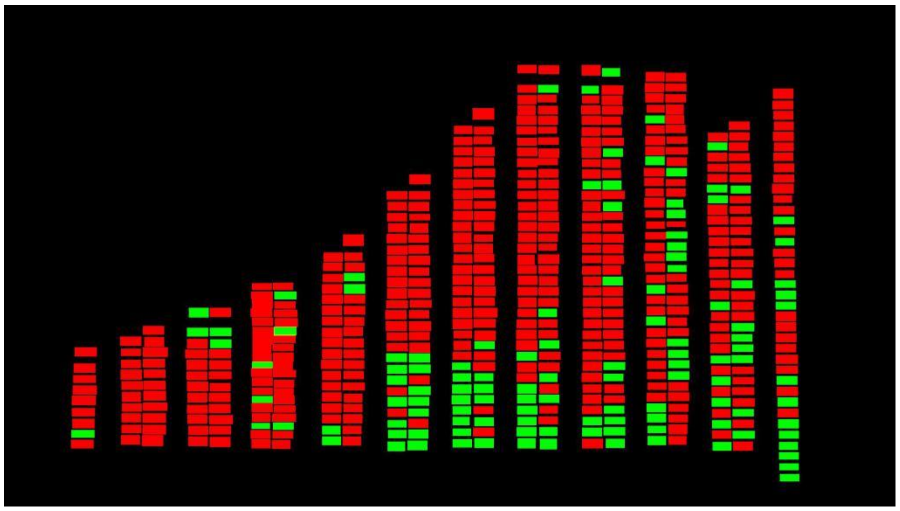
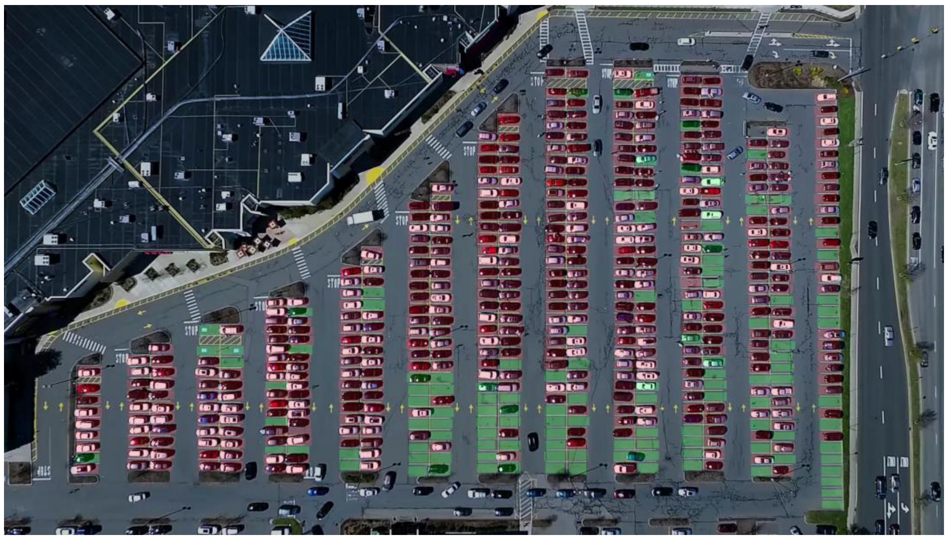
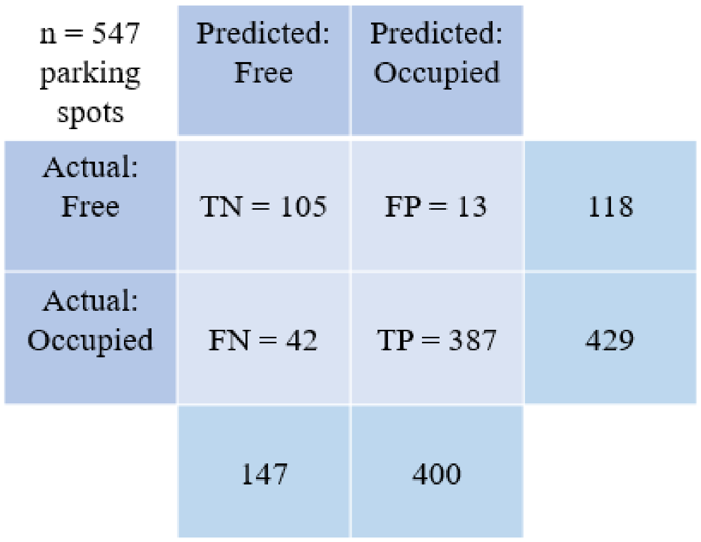




Figure 16. Original Image (top), Mask (middle), and Original Masked (bottom)

Evaluating Results

The video was split into 200 images and each one was inspected to verify the accuracy of the program. A confusion matrix (Fig. 17) was created based on the average true positives (accurately predicted occupied), true negatives (accurately predicted free), and their counterparts (false positives/negatives).



n = 547 parking spots	Predicted: Free	Predicted: Occupied	
Actual: Free	TN = 105	FP = 13	118
Actual: Occupied	FN = 42	TP = 387	429
	147	400	

Figure 17. Confusion Matrix

The accuracy, which is shown below (Fig. 18), was calculated using the following formula: 1 – (FP + FN) / total number of parking spots.

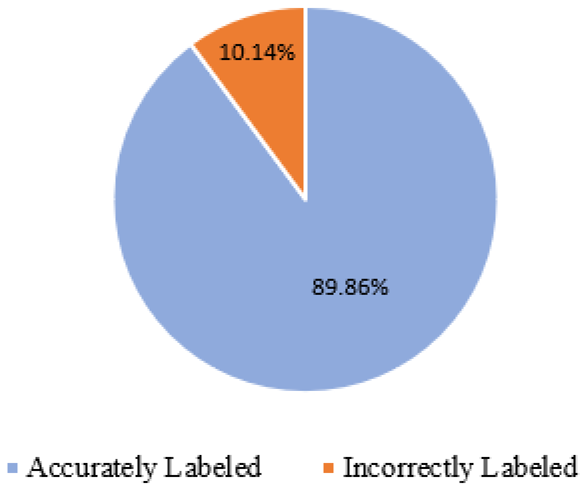


Figure 18. Label Accuracy Pie Chart

Testing Additional Videos

The process of segmenting the video, creating the mask, and evaluating the results was repeated for five additional videos. Testing these additional videos made it possible to check the program’s accuracy under different conditions such as video recorded at night, cloudy weather, and varied camera angle. A masked image and the confusion matrix for each video is shown below, and the accuracy is stated.

The first video was recorded at about a 45 degree angle. The angle of the video caused vehicles to sometimes obstruct the parking spot next to it, which had a slight affect on the accuracy. Even with this issue, the program was able to detect the vehicles and empty parking spots with an accuracy of 93.1%.

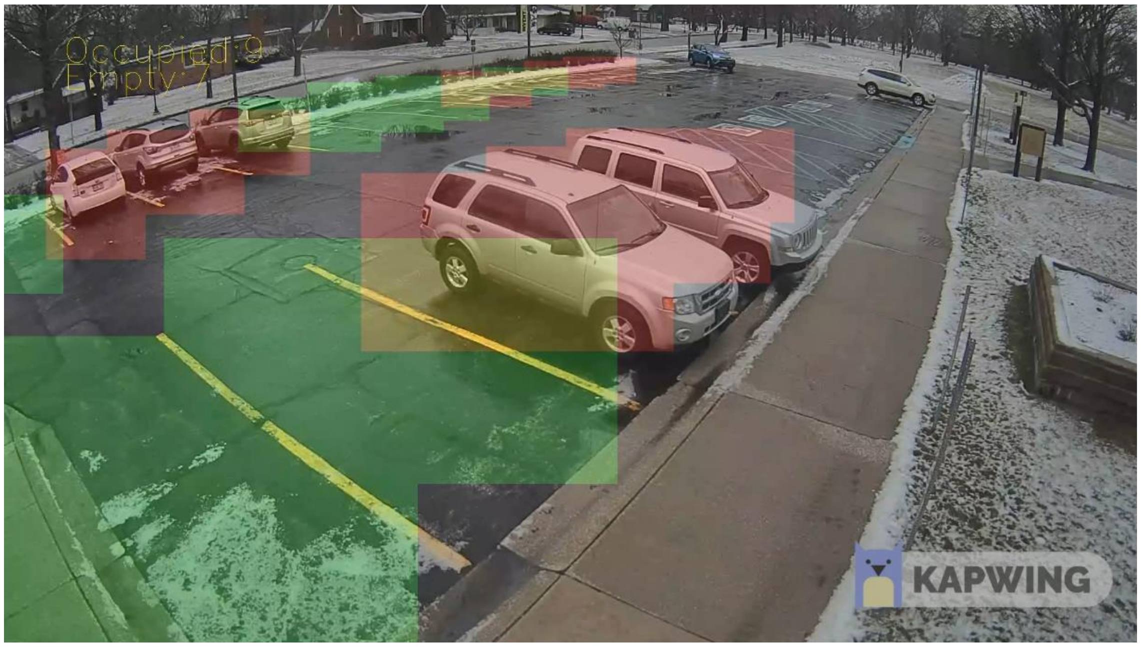
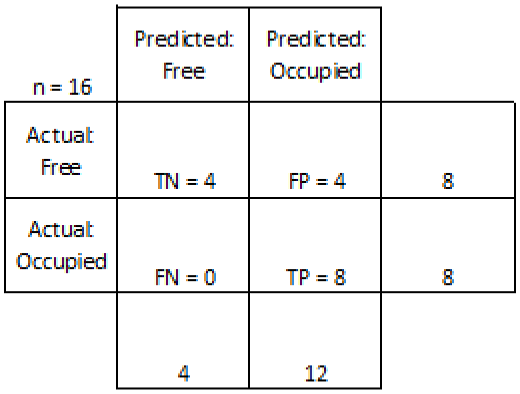


Figure 19. Purdue Northwest NW Staff Parking Lot Masked



n = 16	Predicted: Free	Predicted: Occupied	
Actual Free	TN = 4	FP = 4	8
Actual Occupied	FN = 0	TP = 8	8
	4	12	

Figure 20. Confusion Matrix for NW Staff Parking Lot

The next video analyzed was also taken at about a 45 degree camera angle. There was a shadow over approximately half of the parking spots, which is believed to have affected the accuracy of the program’s detection as that was where most of the false positives and false negatives occurred. The program’s detection accuracy for this video was 91.2%.

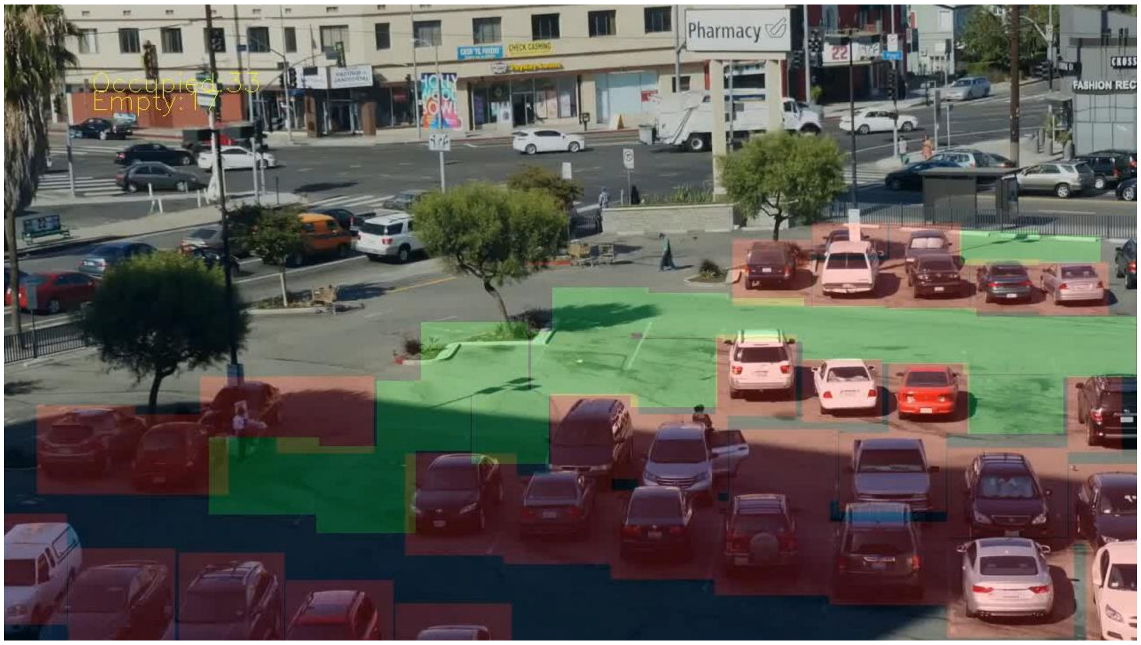
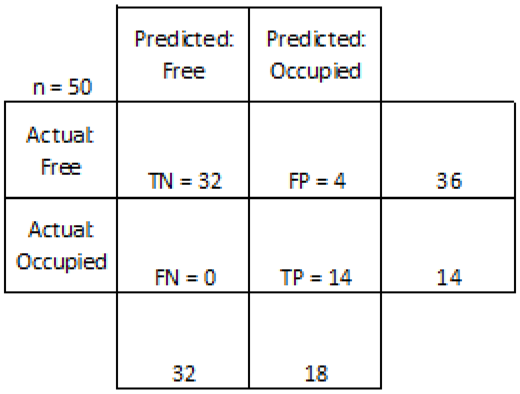


Figure 21. Shadowed Parking Lot Masked



n = 50	Predicted: Free	Predicted: Occupied	
Actual Free	TN = 32	FP = 4	36
Actual Occupied	FN = 0	TP = 14	14
	32	18	

Figure 22. Confusion Matrix for Shadowed Parking Lot

The third video was recorded at night and at a 45 degree angle. The program performed very well at determining when the parking spot was occupied, but it had issues with the spots that were empty. This issue was likely caused by the low lighting conditions and shadows triggering false positives. The program's accuracy for this video was 81.3%, which is significantly lower than the accuracy for the daytime videos.

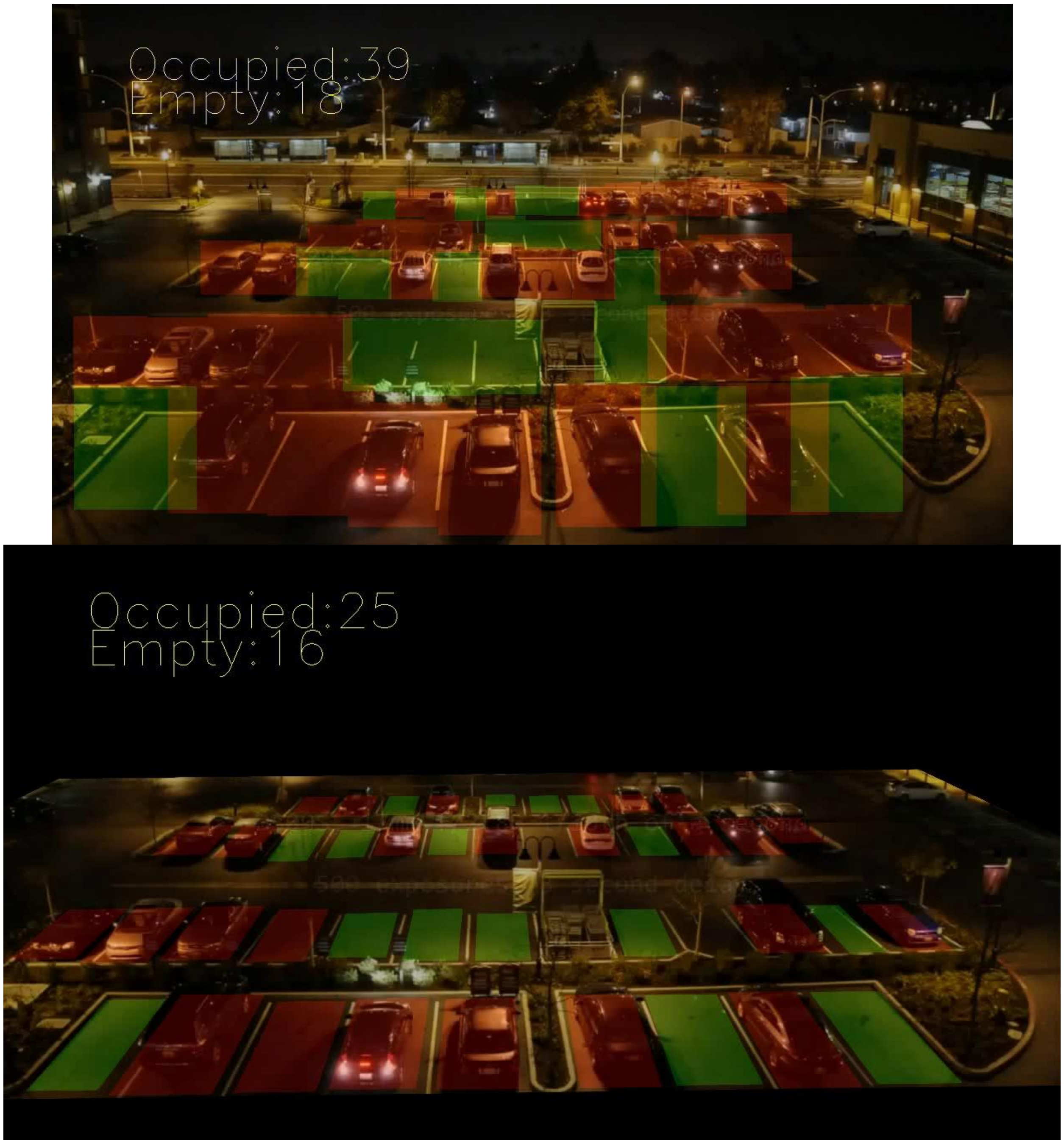
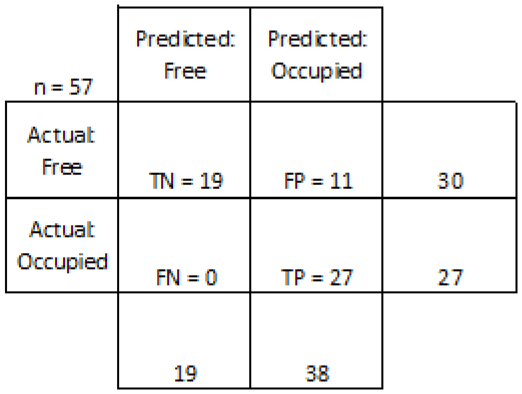


Figure 23. Night Parking Lot Masked without Perspective Transformation(top). Night Parking Lot Masked with Perspective Transformation (bottom).



n = 57	Predicted: Free	Predicted: Occupied	
Actual: Free	TN = 19	FP = 11	30
Actual: Occupied	FN = 0	TP = 27	27
	19	38	

Figure 24. Confusion Matrix for Night Parking Lot

The fourth video was also taken at night, but from a top-down perspective. The program's detection accuracy for this video was 93.7%. This accuracy for this video was significantly higher than the accuracy for the other nighttime video. This was likely due to the fact that the program was trained with a large amount of images that had a top-down view of vehicles.

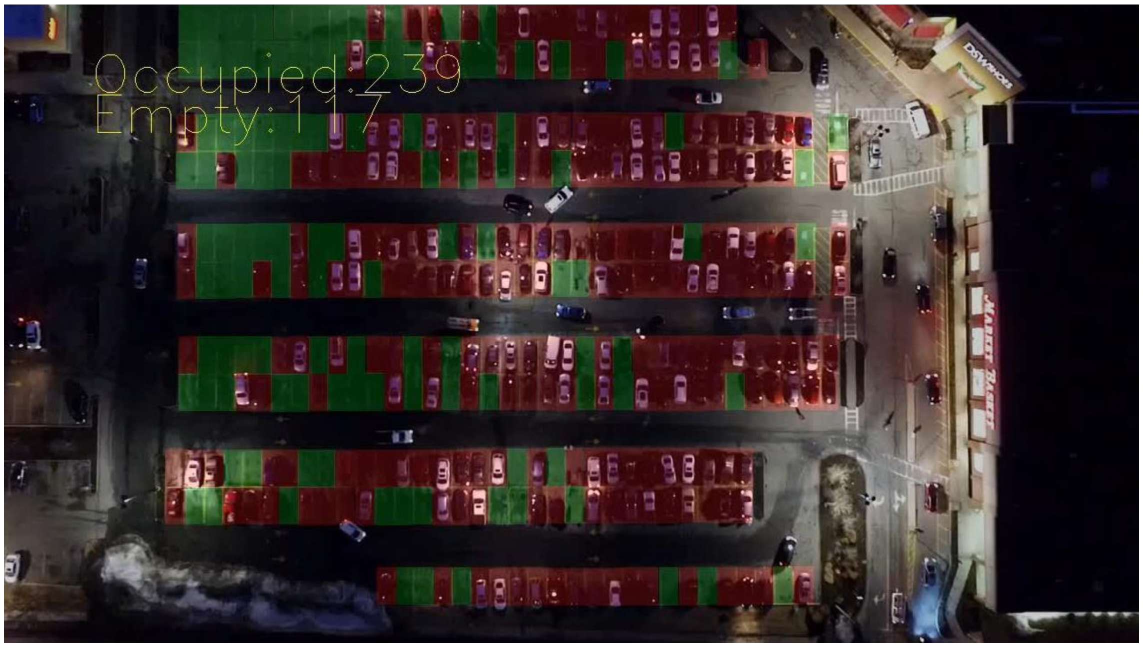


Figure 25. Night Parking Lot Above View Masked

n = 346	Predicted: Free	Predicted: Occupied	
Actual Free	TN = 94	FP = 16	110
Actual Occupied	FN = 6	TP = 230	236
	100	246	

Figure 26. Confusion Matrix for Night Parking Lot Above View

The fifth video used for testing was recorded at about a 30 degree camera angle, had trees obstructing the camera's view of some of the parking spots, and had frequent weather changes from sunny to cloudy. The program's detection accuracy for this video was 88.1%. The cloudy weather had a big impact on the accuracy for this video, especially near the end of the video, where the shadow over the parking spaces got darker causing more false positives. The obstructing trees also had an effect on the accuracy, causing the program to detect those spaces as free when they were actually occupied.

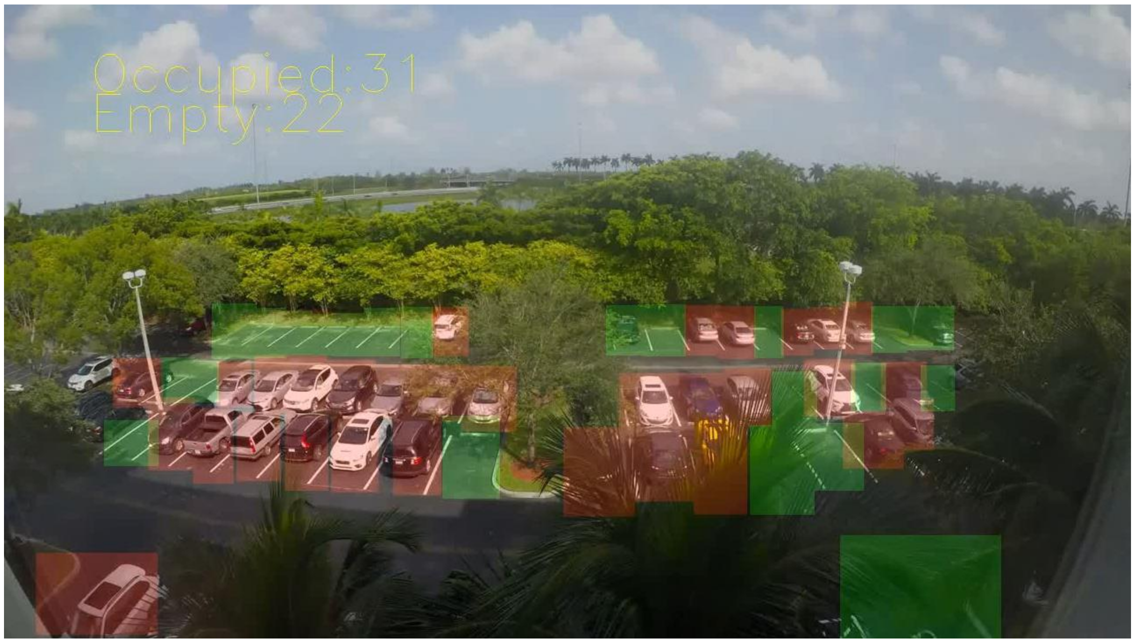
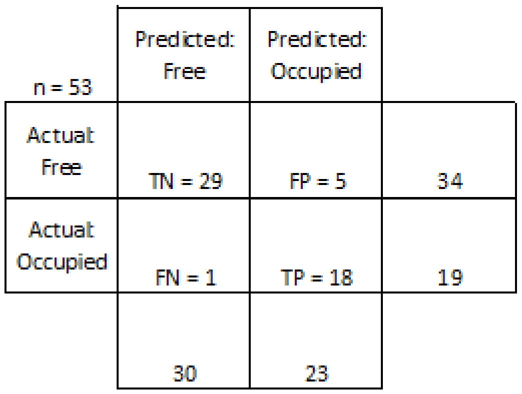


Figure 27. Obstructed View Parking Lot Masked



n = 53	Predicted: Free	Predicted: Occupied	
Actual: Free	TN = 29	FP = 5	34
Actual: Occupied	FN = 1	TP = 18	19
	30	23	

Figure 28. Confusion Matrix for Obstructed View Parking Lot

Improving Program and Retesting

The program was improved by applying a voting technique to it. This allowed for the program to compare the classification of multiple frames and then vote on whether the parking spot was occupied or not based on which classification was true for a greater number of frames. A perspective technique was also implemented and applied to the Night Parking Lot video. This technique shifted the video to appear from a top-down view, which helped to overcome the vehicles overlapping parking spots and improve accuracy.

Below is a table showing how the accuracy changed from the original testing of the five videos when the voting and perspective techniques were applied.

Video Name (Shortened)	**Original Testing Accuracy**	**Voting Technique Accuracy**	**Perspective Technique Accuracy**
NW Staff Lot	93.06%	93.27%	N/A
Shadowed Lot	91.16%	92.63%	N/A
Night Lot	81.29%	80.48%	87.6%
Night Above Lot	93.66%	92.9%	N/A
Obstructed Lot	88.05%	87.62%	N/A

Table 1. Accuracy Comparison After Improvements

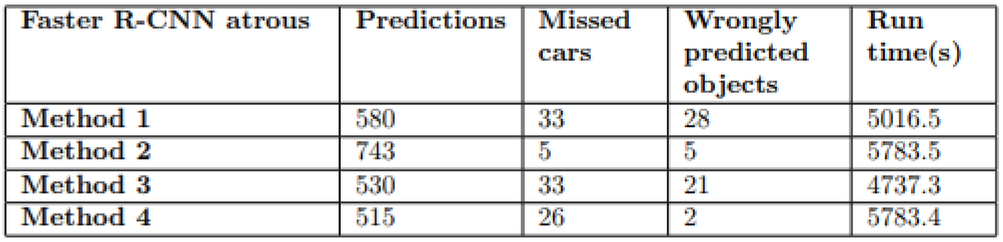
The voting technique improved the accuracy slightly for the NW Staff Lot and the Shadowed Lot, but decreased the accuracy for the other three parking lots that were tested. While it did help decrease the rapid changes of detection where a parking spot would change from occupied to empty or visa versa, it caused a slight delay when a vehicle entered or left the parking spot before the new classification of that spot was applied. This delay was responsible for the slight decrease in accuracy for the bottom three videos.

With the perspective technique applied to the Night Lot, the accuracy improved by about 6.3% from the original test. The delay from the voting technique still affected the accuracy slightly, but the shifting of the video prevented the vehicles from overlapping the other parking spots, which had a larger effect on the accuracy.

Comparing to Other Programs

The program was compared to two similar projects. The Umea University project took place in June of 2018 and the University of Tartu project took place in 2018 (the specific date was not specified).

The Umea University project tested 100 images of the same parking lot at different times, instead of using a video, and only detected vehicles, not empty spaces. They applied four methods: the first was their original trained model, the second was a model with additional training and no box optimization, the third was with no additional training but with box optimization, and the fourth was with both additional training and box optimization. Box optimization is where the bounding boxes overlap by 50%, and the box with the lower accuracy is removed. Their results are shown below[#].



Faster R-CNN atrous	**Predictions**	**Missed cars**	**Wrongly predicted objects**	**Run time(s)**
Method 1	580	33	28	5016.5
Method 2	743	5	5	5783.5
Method 3	530	33	21	4737.3
Method 4	515	26	2	5783.4

Table 2. Results from Umea University Project[#]

The accuracy of the Umea University project's results was calculated using the formula 1 – (Missed cars + Wrongly predicted objects) / Predictions. The comparison of their accuracy to the accuracy of this project's program is shown in the Table 3.

The University of Tartu's project also only detected vehicles and not empty parking spaces. Their testing was conducted using 833 images and resulted in an accuracy of about 95%[#].

Table 3 below shows the accuracy comparison of this paper's program to the Umea University and The University of Tartu's projects. The asterisks mark the testing that was done with the program from this project, with the highest accuracy achieved shown in the table.

Video/Project Name	**Accuracy**
Original Test Video*	89.86%
NW Staff Parking Lot*	93.27%
Shadowed Parking Lot*	92.63%
Night Parking Lot*	87.60%
Night Above Parking Lot*	93.66%
Obstructed Parking Lot*	88.05%
Umea University Method 1	89.48%
Umea University Method 2	98.65%
Umea University Method 3	89.81%
Umea University Method 4	94.56%
The University of Tartu	95%

Table 3. Accuracy Comparison Between Projects

As can be seen from Table 3, this program performed with an accuracy close to that of Umea University's program and The University of Tartu's program. It should be noted that both Umea and Tartu's programs only detected vehicles and not empty parking spaces, and that they were not applied to different conditions such as nighttime video, weather changes, and camera angle.

Conclusion

This program determines if a given parking spot is occupied by applying the binary classifier on the parking spot location. The spot is highlighted red if occupied and green if it is open. The program was tested on six different videos with varied traits, with the highest accuracy achieved being 93.66% and the lowest accuracy achieved being 87.6%. From the accuracy data, the program was shown to perform well from a top-down or angled view and also during different lighting/weather conditions.

Improvements were made by utilizing registration to handle situations where the camera video is unstable, the voting technique to improve the precision of the classification, and the perspective technique to handle situations in the angled camera view where the vehicles overlapped the parking spots behind them. Compared to similar programs, the accuracy achieved from this program was slightly lower, but those programs only detected vehicles and not empty parking spaces. This program also has the advantage of being able to detect both vehicles and empty parking spots at night, during different lighting conditions, and various camera angles.

This program is a viable replacement for the current methods such as ground sensors and infrared sensors, which are costly and require routine maintenance. The program has the advantage of determining whether the parking spot is occupied or empty and can then display the empty locations to make finding a parking spot quick and convenient.

References

1. “PyTorch-Tutorial.” https://github.com/MorvanZhou/PyTorch-Tutorial (10 Oct. 2018)
2. “Pytorch-book.” https://github.com/chenyuntc/pytorch-book (31 Jan. 2019)
3. “CNRPark+EXT.” http://cnrpark.it/ (31 Jan. 2019)
4. “The PASCAL Visual Object Classes Challenge 2007.” http://host.robots.ox.ac.uk/pascal/VOC/voc2007/
5. “Vehicle Detection.” https://github.com/tatsuyah/vehicle-detection (31, Jan. 2019)
6. “Keras-yolo2.” https://github.com/experiencor/keras-yolo2 (31, Jan. 2019)
7. “YOLO: You only look once (How it works).” https://www.youtube.com/watch?v=L0tzmv--CGY&t=178s (31, Jan. 2019)
8. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. "You Only Look Once: Unified Real-Time Object Detection." *2016 The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016).*
9. Redmon, Joseph and Ali Farhadi. “YOLO9000: Better, Faster, Stronger.” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)*.
10. “Searching for Parking Costs Americans $73 Billion a Year.” http://inrix.com/press-releases/parking-pain-us/ (2 Feb. 2019)
11. Holmstrom, Alexander. “Counting Cars and Determining the Vacancy of a Parking Lot using Neural Networks” (Master’s Thesis). Umea University, 6 June 2018.
12. Plemakova, Viktoria. “Vehicle Detection Based on Convolutional Neural Networks” (Master’s Thesis). University of Tartu, 2018.

Appendix A: Milestone Log

MILESTONE LOG

TASKS	**TARGET DATE**
1. Research	
A. Research How CNNs Function	10/26/18
B. Learn Essential Programs and Programming Languages	11/02/18
C. Learn Different Application Techniques	10/26/18
2. Acquire Hardware	
A. Choose Camera and Accessories	01/28/19
B. Choose Development Board	02/06/19
3. Design Program	
A. Getting Image Data Sets and Preprocessing Them	11/12/18
B. Using Images to Develop Basic CNN Model	11/26/18
C. Tackling Video Streams with OpenCV	12/08/18
4. Improve and Test Software	
A. Test Program Using Camera Video	02/22/19
B. Test Program Using Old Security Video	03/22/19
C. Propose Program to Faculty	04/19/19
D. Develop Accompanying Application	04/19/19
5. Document	
A. Begin Documentation	10/01/18
B. End documentation	04/24/19
C. Complete design review	12/01/18
D. Complete instruction manual	04/24/18
E. Complete final report	04/24/19
F. Complete final team presentation	04/26/19

Appendix B: Project Schedule

SCHEDULE

TASKS	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
1. Research		A C	B					
2. Acquire Hardware					A	B		
3. Design Program			A B	C				
4. Improve and Test Software						A	B	C D
5. Document		A		C				BD EF