Detecting Empty Parking Spots in a Parking Lot from Aero Footage

Fanchen Bao
CAP 6619 Deep Learning Term Project
Florida Atlantic University
Boca Raton, FL, USA

***Abstract*—The ability to quickly and reliably identify empty parking spots in a parking lot has various benefits for both the user and maintainer of the parking lot. Given the tremendous advance in image processing and machine learning, it is possible now to train a deep learning model on video footage of a parking lot to detect its empty spots. In this report, an end-to-end example of detecting empty parking spots is given, and many of the challenges and limitations also discussed.**

***Index Terms*—Parking, Deep learning, OpenCV, ResNet50**

I. Introduction

Parking has always been a pain point when travelling to downtown area, campuses, large sports or entertainment events, etc. The reason is straightforward: there is no good way for the user to know in real-time where the empty spots are. Consequently, the user is often faced with the situation where he has to circle around a parking lot many times before, if he is lucky enough, an empty spot shows up.

Simultaneously, parking is also a pain point for the maintainers of the parking lot. Managing parking lot is most of the time not a profit center for the maintainers, yet if parking service is not appropriate, the business can suffer. Unfortunately, there is also no good way for the maintainer to know in real-time the current or historical status of his parking lot. As a result, the maintainer often has little clue whether his parking lot is sufficient for the business needs.

There have been solutions to resolve this issue, the most common of which is the installation of sensors. For garages, such sensor can be a pressure sensor under the entrance and exit of a garage, or a distance sensor above each parking spot. The former is activated when a vehicle drives over it and can help update the overall availability of the garage, whereas the latter is able to detect whether a spot is occupied by calculating the distance between the sensor to the floor. For parking lots, a light-sensitive sensor can be installed in the center of each spot and detect whether it is occupied by the change in light exposure. While the sensor solution works and has been commercialized (e.g. the overall availability count in FAU's garages, spot-level availability indicator in Miami's Dolphin Mall, etc.), the downside is obvious: the infrastructure requirement to reach spot-level availability detection is huge. Not only do we have to install a sensor for each individual spot, but there is also the overhead to establish communication between sensors and a local server to upload sensing data for further analysis. And this is not even taking into account the cost of sensor failure and replacement. In a word, a sensor-based solution to detect empty parking spot is feasible yet costly.

A better solution should be scalable, which means the installation of one device is able to detect multiple empty parking spots. To this end, camera is a good candidate. A strategically placed camera can potentially capture all the parking spots in a garage, so the hardware implementation and maintenance cost could be low. Once the footage is available, the problem becomes a software one: how do we detect an empty parking spot from a footage? And since video footage is essentially a series of frame images, the problem further boils down to this: how do we detect an empty parking spot from an image?

Thanks to the advancement in image processing and machine learning, it is indeed possible to detect empty parking spots from an image of a parking lot. Generally speaking, there are two methods, as described below.

The first approach, hereby termed as the image processing approach, takes in one frame footage, uses image processing techniques to identify where the parking spots are, trains a deep neural network on the spots to identify empty or parked, and finally applies the model to the rest of the frames to recreate the footage with empty spots highlighted. The feasibility of the image processing approach has been demonstrated by Dwivedi [1].

The second approach, hereby termed as the object detection approach, uses a pre-trained object detection algorithm, such as YOLO, to identify where the cars are in a parking lot footage. Once a car stops moving for a period of time, the location of the car is considered a parking spot. If a parking lot has been filled with cars at any point during the footage, all the parking spot location and their bounding boxes can be obtained. To determine whether a spot is taken, one only needs to compare whether the spot's bounding box overlaps with a car's. The feasibility of the object detection approach has been demonstrated by Chatterjee [2].

The object detection approach is less restrictive, hence more scalable, than the image processing one, because it can be applied to any parking lot footage out-of-the-box. The image processing approach, however, requires manual work to tease out each parking spot from the footage. This means the model thus trained is not applicable to another parking lot, or the same parking lot but viewed from a different angle or light

condition (see V for other drawbacks). That said, one benefit from the image processing approach is that it can handle a sparsely occupied parking lot. Since the object detection approach relies on a parked vehicle to pinpoint the location of a parking spot, if a parking lot is never completely full, it won't be able to identify all the empty parking spots.

In this report, we will follow the image processing approach, despite its many shortcomings, because of two main reasons. First, the footage we can find for this report does not have all the parking spots occupied. And second, the image processing approach is less of a black-box solution, thus leaving us more room to tweak and learn.

The report is organized as follows. In Section II, the method used to detect empty parking spot in an aero footage is discussed step-by-step. In section III, a brief description of the data, i.e. the aero footage, is provided. In Section IV, the results of each step described in the methods are presented. In Section V, the difficulties of applying the image processing approach is listed. And finally, Section VI concludes the paper and points directions to future research.

II. Methods

The image processing approach consists of three main steps: parking spot localization, parking spot training, and prediction on video. Each step is discussed in detail below.

A. Parking Spot Localization

In the parking spot localization step, *OpenCV* [3] is used to perform image processing to locate the coordinates, in pixels, of each parking spot. All the APIs mentioned below come from *OpenCV* if not otherwise specified.

First, we use the *VideoCapture* API to capture a few frames representative of the whole footage. By representative, we mean that the frames captured spread across the footage evenly from beginning to end. We then pick two most clear and best contrasted frames to feed to the next stage.

The next stage is to pick out all the parking lines. To do so, we use the *inRange* API to select pixels whose RGB values fall within a specified range, which in our case, is a somewhat white color. All the pixels not selected are default to RGB $(0, 0, 0)$, which is black. After the white parking lines are picked out, we convert the image to gray scale to further contrast the white color and the rest. These two transformation prepares the image for edge detection.

Edge detection is the next stage, in which a gray scaled image with white parking lines selected goes through the Canny edge detector such that all the parking lines can be detected as edges. The Canny edge detector is implemented in the *Canny* API. After edge detection, the image becomes a matrix of binary values, where the edges are denoted as 1 and everything else 0. We then black out anything that does not concern parking, which leaves us with a smaller area of interest for further processing. Edge detection tells us visually where the parking lines are. The next stage is to reveal the pixel coordinates of each detected parking line.

To find out the pixel coordinates, we use the *HoughLinesP* API, which runs the Hough Transform to generate the pixel coordinates of the edges. Since not all edges detected belong to the parking lines, we set up additional constrains on the pixel to keep only the coordinates of the parking lines.

After the parking line coordinates are acquired, we are able to compute the coordinates of each parking spot, which is represented by a rectangle. Ideally, if the previous image processing steps are 100% successful, we will have all parking lines localized. However, in reality, only a few parking lines are available, and some of them, although identified, are not well characterized (i.e. there are overlaps, missing segments, etc.). Fortunately, all parking spots are uniform in dimension. Thus given the pixel width of a parking spot, we are able to localize each parking spot in a row as long as we can identify the coordinates of the left and right edge.

Once all the parking spot coordinates are available, we can crop each parking spot out from a specific frame and save it as its own image. This will become our training and validation data.

B. Parking Spot Training

After the parking spot localization step, we acquire a lot of cropped images corresponding to each parking spot. These images must be manually classified as either "empty" or "parked" to serve as the training and validation data. It is important to note that some cropped images do not represent either class yet also unavailable for parking (e.g. the image shows a piece of lawn). For these images, a third class "obstacle" is established. Therefore, eventually all cropped images are separated into three classes: "empty", "parked", and "obstacle".

We use *keras* [4] framework to build a deep neural network to train the parking spot images. Standard practice of data preparation is followed to retrieve the image data and split them into 80% training and 20% validation. In order to leverage the power of *keras*, we have decided to use a pre-trained deep neural network as our feature extractor. In other words, we will feed our image data to a pre-trained model and use its output (i.e. extracted feature) as input to our own model. The benefit of doing this is obvious: there is no need to design from scratch a most likely less performant neural network. In addition, it also speeds up training as no parameter tuning is needed for the pre-trained model.

We test out three pre-trained models: Xception [5], VGG16 [6], and ResNet50 [7]. After the image data goes through a pre-trained model, its features are fed to two layers of our own model: a flatten layer to turn the pre-trained model output into 1D array and a dense layer as output, which contains three nodes and softmax as activation function. Each model is trained for maximum 50 epochs. An early stopping constraint is established where if the model's validation accuracy does not improve more than 0.01 for over 10 epochs, the training is terminated. The best performing model of the three is saved for future use.

It is worth noting that since we are relying on the pre-trained model to extract features for us, no data augmentation (e.g. flip, rotation, distortion, etc.) is applied to the raw images before feeding the images to the pre-trained model.

C. Prediction on Video

Prediction on video is the final step of the image processing approach. It first loads the trained model described in Section II-B. Then it captures each frame from the original image, extracts all the parking spot images, feeds them to the trained model to obtain a predicted label, and based on the value of the label draws either a translucent green or blue rectangle on top of the parking spot. The predicted labels are 0 for an empty spot, 1 an obstacle, and 2 a parked spot. The translucent green rectangle indicates the spot is empty, whereas the translucent blue the spot is actually an obstacle. Drawing and translucency is achieved via the *rectangle* and *addWeighted* API. The newly drawn frames with green and blue rectangles are saved on disk.

Finally, the *VideoWriter* API is used to stitch the newly drawn frames together into an AVI video file. In the video, the empty spot detection can be observed in a dynamic setting, where vehicles move in and out of the parking spots. The video is uploaded to YouTube and can be accessed from this URL: https://youtu.be/WmK37HBfnQ8

III. Data Collection

The data used for this report comes from an aero footage of Walmart parking lot, shared on YouTube under this URL: https://www.youtube.com/watch?v=IZ8NPmp0LPk. It is selected because of the video quality, the completeness of the parking lot shown, and decent amount of action from vehicles parking and leaving. It is not the best overhead parking footage available, but the best footage has already been used in [1], so we have to settle for this one.

We then use https://www.y2mate.com/en30/convert-youtube to obtain the footage of the video in MP4 format.

IV. Evaluation

In this section, the results of each step described in Section II are presented and discussed.

A. Frame Selection

Figure 1 shows the two selected frames from the original footage that will go through all the image processing steps. These two frames are one of the best lit and contrasted in the video.

B. Parking Line Selection

Figure 2 shows the aftermath of parking line selection. Since parking line is white, we use *inRange* API to select pixels with RGB value between $(160, 160, 160)$ and $(240, 240, 240)$. Note that all pixels within this range are selected, thus we end up with not only parking lines but also other similarly colored objects in the image, such as the adjacent smaller parking lot. The RGB value range is manually selected based on trial-and-error. The quality of each value range choice is judged by human perception.

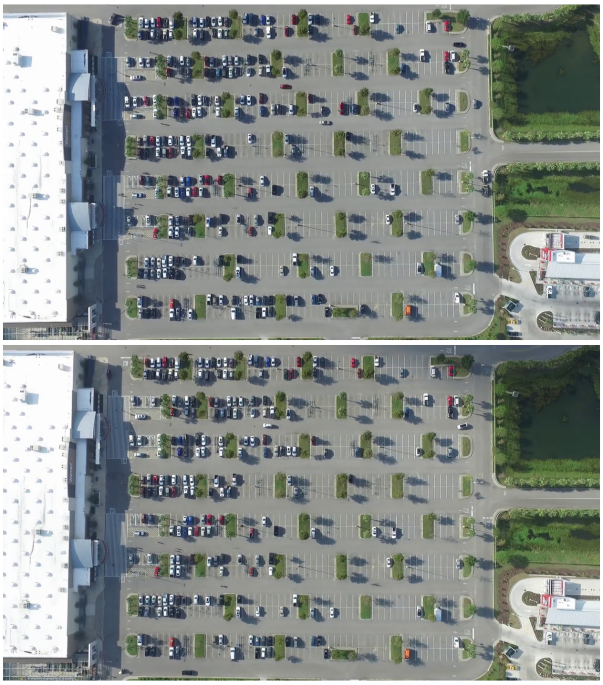

Fig. 1. Two Selected Frames from The Original Video Footage

After *inRange*, we also apply the *cvtColor* API to convert the image to gray scale, which further contrasts the parking lines to the background.

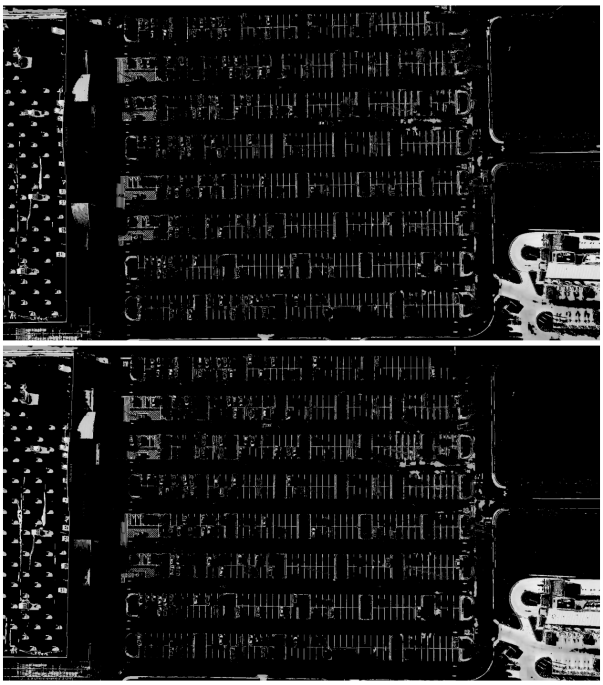

Fig. 2. Frame Images After Parking Lines Have Been Selected

C. Edge Detection

Figure 3 shows the aftermath of edge detection. Edge detection removes a lot of the details in the image that is not relevant to parking lines. The image is also simplified to a binary matrix, which helps the subsequent Hough Transformation.

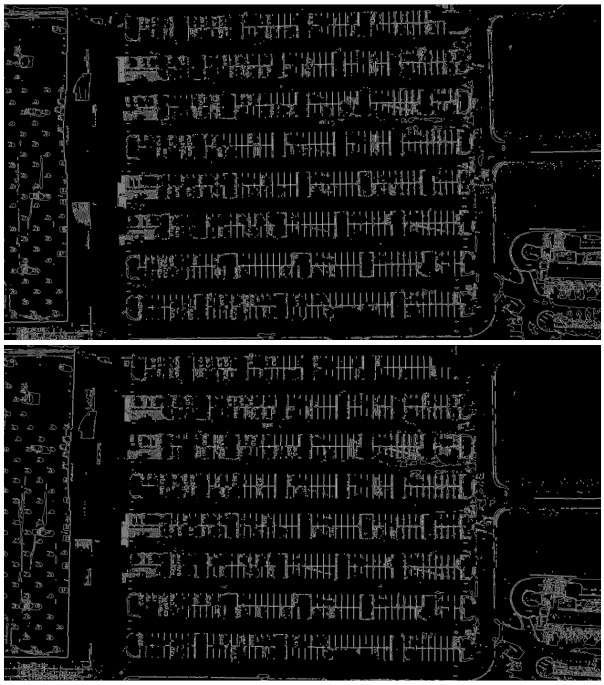

Fig. 3. Frame Images After Edge Detection Has Been Performed

D. Area of Interest

Figure 4 shows the aftermath of focusing on the area of interest. The purpose of this step is to further remove the unnecessary information on the image by blacking out the uninteresting area. This is achieved via the *fillPoly* and *bitwise_and* APIs. The coordinates of the area to be blacked out is obtained manually through trial-and-error.

E. Parking Line Localization

Figure 5 shows the result of parking line localization. Each edge-detected line's coordinates ((x_1, y_1) and (x_2, y_2)), which consists of two sets of coordinates corresponding to the two ends of a line, are obtained via Hough Transformation. To filter out lines not belonging to parking lines, only those lines satisfying $-1 \leq x_1 - x_2 \leq 1$ (i.e. vertical lines) are kept and drawn on the original frame image. Also drawn on the frame image are the coordinates of each line. In particular, the yellow coordinates on the top image represent the x-coordinates. In the bottom image, the green coordinates represent the top y-coordinates while the blue the bottom y-coordinates.

From Figure 5 it is clear that the parking line localization is not thorough. Furthermore, the overlap of coordinate text suggests that multiple lines have been detected on a single parking line. There are also a few mistakes sprinkled here and there. Therefore, it is not possible to completely rely on the

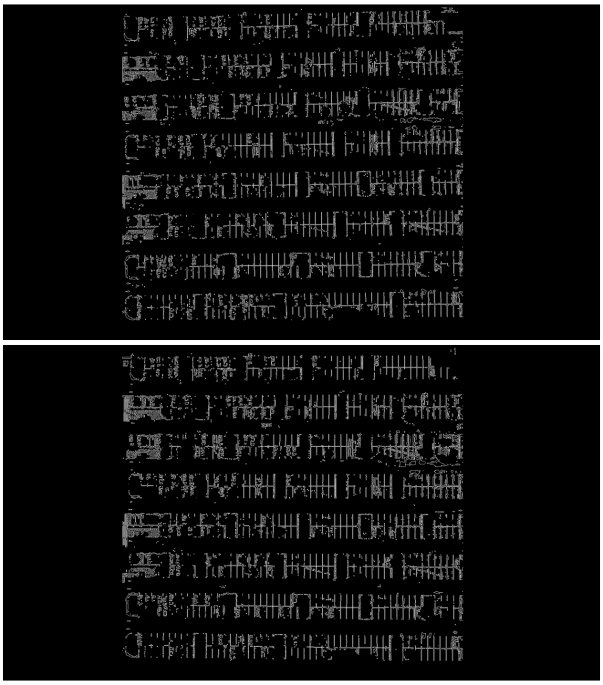

Fig. 4. Frame Images After Focusing on Area of Interest

available parking line localization to identify all the parking spots. We have to use the left and right edge of each parking row and an estimated parking spot width, 20 pixels, to estimate the location of the parking spots.

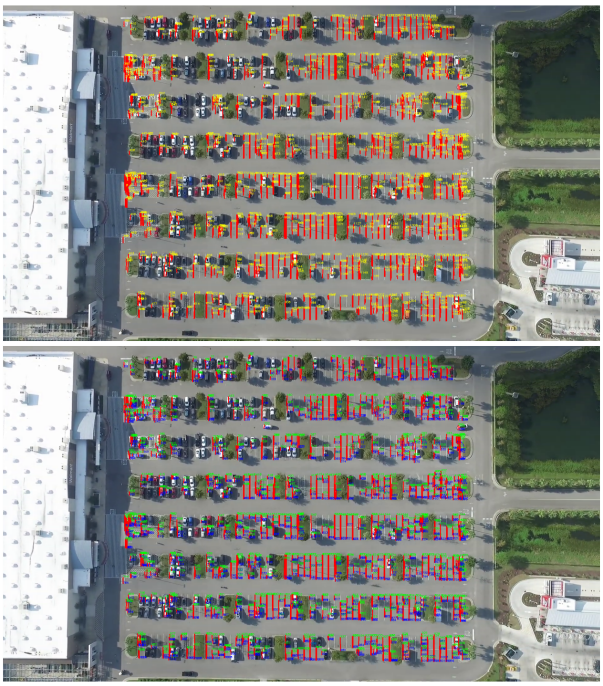

Fig. 5. Original Frame Images After Detected Lines Are Drawn

F. Parking Spot Localization

Figure 6 shows the result of parking spot localization. As mentioned above, we cannot use the detected parking line to obtain the location of all parking spots. Therefore, we estimate the parking spot location by taking the left and right edge of a parking row, and split the range in increments of 20 pixels. The result is good for some rows, but bad for the others. This is because the footage itself is not completely stable. According to the YouTube description, the aero footage was shot by a drone, hence the constant shifts in the shot. The consequence of an unstable footage is that any coordinate system obtained on one frame is inaccurate on another. And this is exactly what happens to us: the parking line localization obtained on one frame is not perfectly applicable to the others. However, this is something we have to accept, as stabilizing the footage via software is beyond the scope of this report.

Another important aspect to note is that the lawns and cart return station are also included in the estimated parking spot location. This is inevitable because we could not figure out an easy way to filter them out in the previous steps. Therefore, the solution is to include these obstacles as part of the parking spot but label them as "obstacles", such that the deep neural network can learn to not classifying these spots as empty.

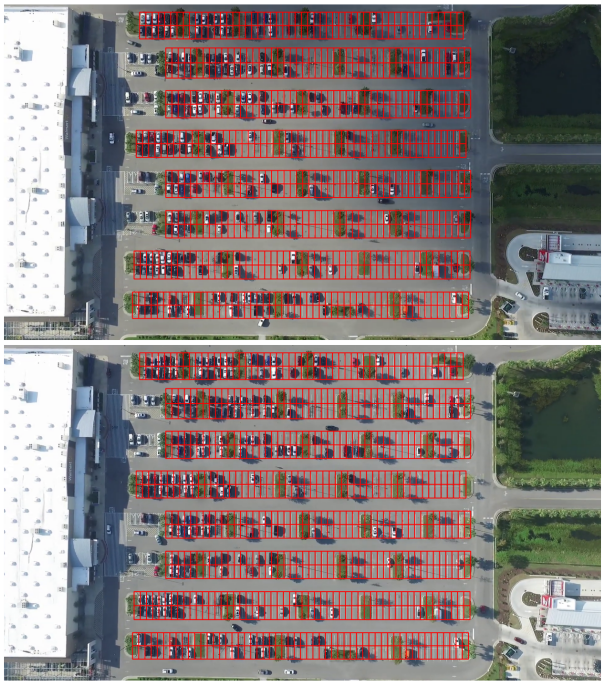

Fig. 6. Original Frame Images with Parking Spot Identified by Bounding Boxes

G. Training Data Generation

From the knowledge of the coordinates of all parking spots, we can crop each parking spot from a specific frame and save the cropped image as training data. In total, 818 cropped images are saved. They are then manually separated into three folders: "empty" (391 samples), "obstacle" (218 samples), and "parked" (209 samples).

H. Neural Network Training

Figure 7 shows the sample training data for the deep neural network. These images are cropped from a specific frame according to the parking spot localization and resized to shape $(72, 72, 3)$. The three classes are also one-hot encoded. The label "empty" is coded as $[1, 0, 0]$, "obstacle" $[0, 1, 0]$, and "parked" $[0, 0, 1]$. The misalignment issue mentioned above is clearly manifested here. For example, the image on the second row first column shows half an empty spot and half a lawn. It is labeled as "obstacle", because we want to err on the side of safety, which is to say we only label a spot as "empty" if it is almost certain that it is empty. Therefore, the ambiguity presented by the second row first column is treated as an "obstacle". Yet, the first row second column is considered as an "empty" spot because it is less ambiguous.

The same ambiguity rule is applied to "empty" vs. "parked". For instance, the third row first column is a misaligned spot where half of the image shows a parked car and the other half an empty spot. To err on the side of safety, we consider this image as "parked".

Notice the images on the second row second and third column. These two represent the cart return station, which in the case of Walmart is built on top of a parking spot but not available for parking. They are labeled as "obstacle".

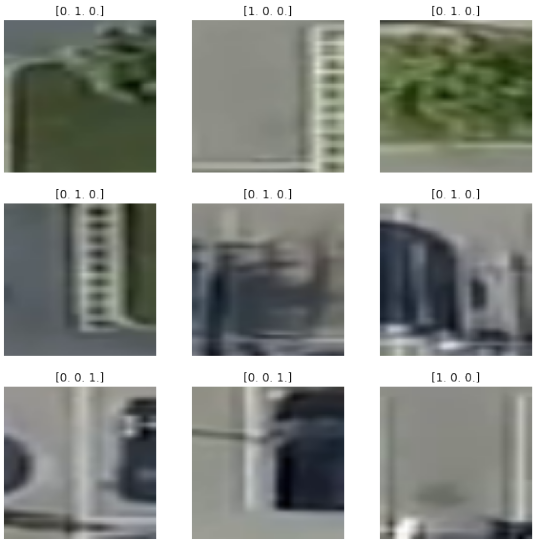

Fig. 7. Sample Training Data for The Deep Neural Network

Figure 8, 9, and 10 shows the training processes of the three models. To recap, we use three pre-trained models: Xception, VGG16, and ResNet50, to extract features from the training data and feed the extracted features as input to our

own simple neural network for classification. All three models hit early stopping criteria. The final validation accuracy using the Xception model for feature extraction is 0.8834, using the VGG16 model is 0.8834, and using the ResNet50 is 0.9387. Since using the ResNet50 model produces the best validation accuracy, it is saved for future use.

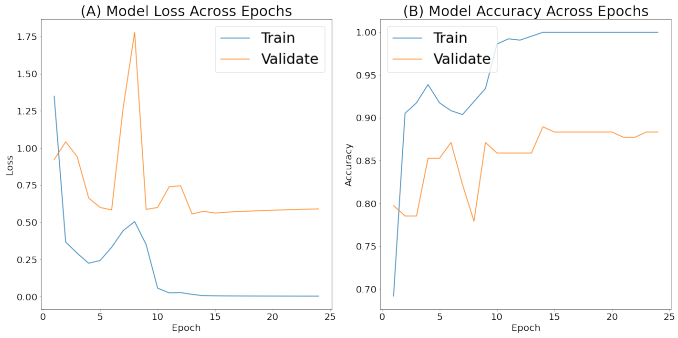

Fig. 8. Training Process Using The Xception Model for Feature Extraction

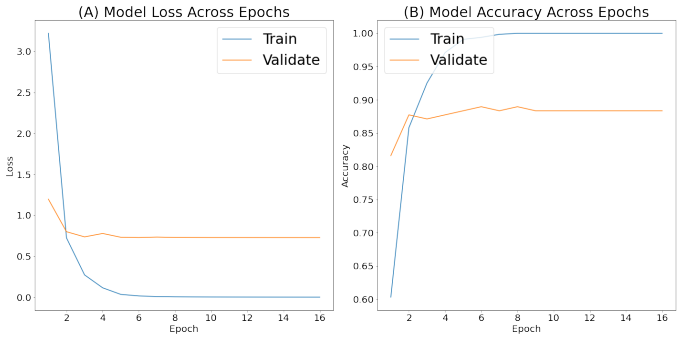

Fig. 9. Training Process Using The VGG16 Model for Feature Extraction

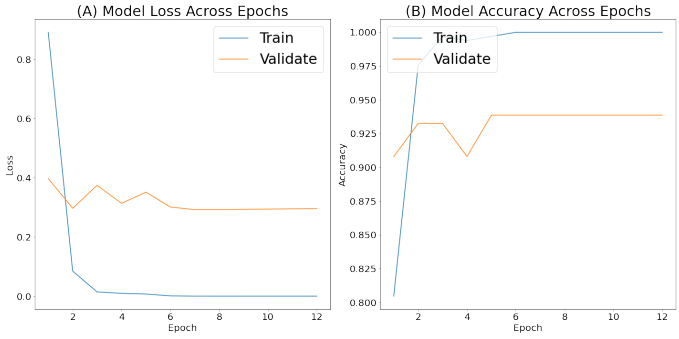

Fig. 10. Training Process Using The ResNet50 Model for Feature Extraction

I. Prediction on Video

Figure 11 shows the results of applying the trained model on still frames. The green rectangles indicates empty spot, whereas the blue obstacles. The parked spots are not indicated by any color. Close inspection of the prediction reveals that although the majority of the classification is correct, there are still errors. One source of error is the misalignment of the actual and the estimated parking spot locations. This usually leads to inaccurate prediction of a spot by the edge of an obstacle. Another source of error is the shadow cast by a parked vehicle onto an empty spot nearby. These shadows are dark and sharp enough to trick the model to classify an empty spot as parked.

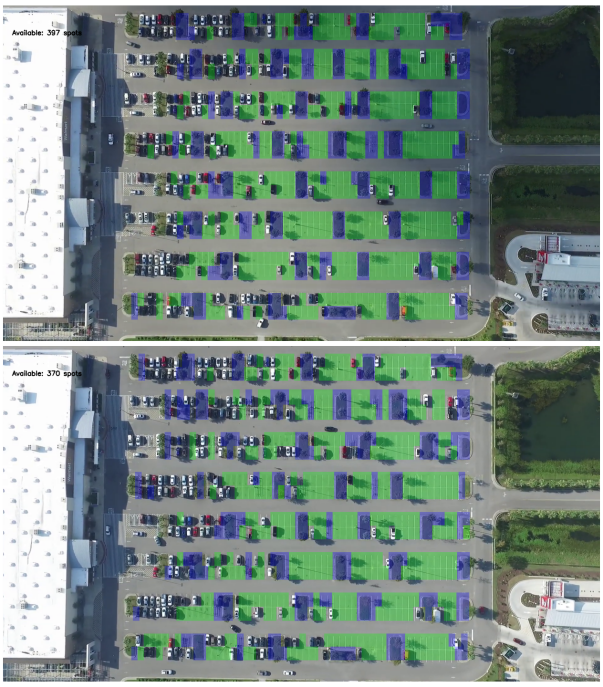

Fig. 11. Results of Empty Parking Spot Detection on Still Frames

To produce a new video with empty spot detection added, we capture and predict on 500 frames from the original video. We then use the first 360 frames to construct a 30-second *AVI* video file with 12 frames per second. The resulting video is available on YouTube at https://youtu.be/WmK37HBfnQ8.

V. Challenges

A couple of challenges present themselves in the project. The biggest challenge is the unsteadiness of the video footage. As mentioned earlier, unstable footage means the parking line coordinates obtained from one frame will be misaligned with the other frames. This misalignment issue directly leads to the difficulty in manually classify the training sample (e.g. when the cropped image of a parking spot actually spans two spots and one of the spot is taken, should we classify this image as “parked” or “empty”?) and empty spot detection (e.g. the errors happening on the spot right next to obstacles). Although this challenge is the main contributor to the error, it is also the easiest to resolve, as it is not related to the methodology itself. For instance, we can use footage from a wall- or pole-mounted camera.

The presence of shadow is another challenge, because it resembles dark vehicles when viewed from a high vintage point. We think the challenge with shadow might be resolved

by having more training data, because the occurrence of shadow in the current training data is sparse. However, it is worth noting that expanding training data is a highly labor-intensive work, as all the 818 cropped parking spot images need to be manually classified. With the misalignment issue added on top, this is certainly not a pleasant task.

The third challenge is the speed of model prediction. The initial plan for prediction on video is to do it in real-time. That is as the video is playing, a backend will grab a frame, run it through the model to obtain predictions, draw the predicted results on a new frame, and present the new frame on the video. Unfortunately, the current pipeline is not fast enough to perform real-time prediction. We believe the slow speed is due to both the deep neural network model and *OpenCV* handling the frame. Granted, the original footage is of high resolution (1920×1080). If the resolution is lower, it is possible that tour current pipeline can handle real-time prediction already. For the neural network at its current shape, there is not much to tweak as the majority of the workload is lifted by the pre-trained model. If we want to speed up the prediction, one possible way is to split a frame into multiple parts and make prediction on each part in parallel.

VI. Conclusion

In this report, we have gone through an end-to-end example of detecting empty parking spot in a parking lot using an aero footage. We rely on image processing to obtain the location of each parking spot, and train a deep neural network to classify each parking spot image as being empty or not. At the end, we are able to detect empty parking spot with decent accuracy, and the results of our detection have been stitched together into a video.

For future research, in addition to the areas mentioned in Section V, we also want to explore other ways to incorporate a pre-trained model into our own model. Currently, we run the feature extraction as a step outside our own model. That is to say we first run feature extraction, and then feed the output to our model. This is not ideal because the our model itself is not end-to-end, since it always requires a data pre-processing step outside the model. Furthermore, by not incorporating the pre-processing step into the model, we lose some of the optimization baked into the *TensorFlow* framework. Finally, when the pre-processing step is not in the model, the model fitting process cannot perform data augmentation on the input images at the beginning of each epoch. This could potentially hurt the performance of the model, especially in the case of shaky footage.

Another area we would like to investigate is whether the current pipeline can be faster when executed on GPU. This can be easily prototyped on Kaggle (https://www.kaggle.com/), a machine learning platform that provides free but limited access to GPU.

Finally, as mentioned in Section I, there are two general approaches to detecting empty parking spot. This report uses the image processing approach, and we have well experienced its limitation. Next step will be to try the object detection approach. We expect the object detection approach will involve a steeper learning curve regarding the usage of an object detection library, yet since it is a black-box method, we think the actual implementation would be easier.

References

[1] P. Dwivedi, "priya-dwivedi/Deep-Learning/parking_spots_detector," Dec. 2020, original-date: 2016-08-17T18:29:12Z. [Online]. Available: https://github.com/priya-dwivedi/Deep-Learning/tree/master/parking_spots_detector

[2] S. Chatterjee, "visualbuffer/parkingslot," Dec. 2020, original-date: 2019-05-20T12:16:33Z. [Online]. Available: https://github.com/visualbuffer/parkingslot

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[5] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 1800–1807, iSSN: 1063-6919.

[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015, arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778, iSSN: 1063-6919.