ISSN 2464-4617 (print)
SSN 2464-4625 (DVD)
Computer Science Research Notes
CSRN 3101
WSCG 2021 Proceedings

Autonomous Parking Spot Detection System for Mobile Phones using Drones and Deep Learning

Guilleum Budia Tirado
Department of Computer Sciene
University of Colorado Colorado Springs
Colorado Springs, CO, 80918, USA
guillemgbt@gmail.com

Sudhanshu Kumar Semwal
Department of Computer Sciene
University of Colorado Colorado Springs
Colorado Springs, CO, 80918, USA
ssemwal@uccs.edu

ABSTRACT

Many parking lot facilities suffer from capacity overloads and many times there are no monitoring tools to provide feedback. As a consequence, the people, wanting to park, become frustrated as there is considerably loss of time. In this paper, we present a novel prototype of an automatic parking-lot analysis platform using image-based machine learning to (a) guide a drone autonomously; and (b) to process useful information to be handled into a smartphone application to communicate with the parking lot users.

We have collected a reasonable amount of test images to build a classification model using Convolutional neuronal networks (CNNs) to classify parking lot images, and build different object detection models to identify free and occupied parking spots. Those models have been exported to the back-end module of our platform so it can control the drone and record the computed information to its database. In addition, we have implemented an iOS application that requests and displays the parking lot status and its empty spots.

We have been able to prove that this prototype is feasible, functional, and opens a path towards future improvements and refinements. The flight control and the data classification algorithms have been shown to work using the machine learning models. In summary, we found a clear and and concise way to display useful information in real time to our users.

Keywords

Drones, AR, Parking Lot, Navigation, Deep Learning

1 INTRODUCTION

It is really a universal experience that we have all faced parking lots frustrations, such as spending time looking for an available free spot, or even realizing that there are no spaces after visiting all of the parking spot. There are several related papers: [1, 2, 3, 4]. In most of these cases, static cameras pointing to the parking lot are used, resulting into non-variant images. Then build manually a mask to segment each parking space into the parking lot and then, after segmenting the original image, individual parking lot space images are used as input to a classifier, using mainly Convolutional Neuronal Networks [5] to determine if those are free or occupied. Traditionally, in robotics, sensor information such as LIDAR [9], sonar and other technologies that give proximity information could be used as well. Yet, our drones are the cheapest available in the market and these drones do not provide this kind of sensors. Other approaches are depth maps computed from drone images. One of the most well known models is the Faster Regional Convolution Neuronal Network (*Faster-RCNN*) [6]. The difference between Faster R-CNN and the other R-CNN techniques is that instead of using Selective Search it uses Region Proposal Networks (RPN) [7]. Other object detection approach very different from Faster-RCNN are the *You Only Look Once* (YOLO) models [8]. The research found in [9] concluded that they could autonomously guide a drone through indoor corridors only using the front camera and without any GPS or other sensor information. To detect and classify parking spaces given dynamic images we will use object detection models to localize and identify two classes, the free parking lots and the occupied parking lots. To achieve good results, we will train different object detection models defined in the Tensorflow Object Detection API such as Faster R-CNN or SSD. We would evaluate the goodness of fit for each model and its speed to choose the final models we will export to use in our back-end system. Figure 1 shows the working of our overall algorithm. The central server will be developed with Python and Django framework. This way, our algorithm integrates the Tensorflow models. The server will be responsible for controlling the drone and receive images from it. Furthermore, serve will also use the trained object detection and flight prediction models to process all the data and store useful information and compute accurate flight instructions for the drone. We have also designed and built a simple iOS application developed with Swift that interacts and retrieves information from the server's database and displays it to the user so we can have a complete prototype of the desired system.

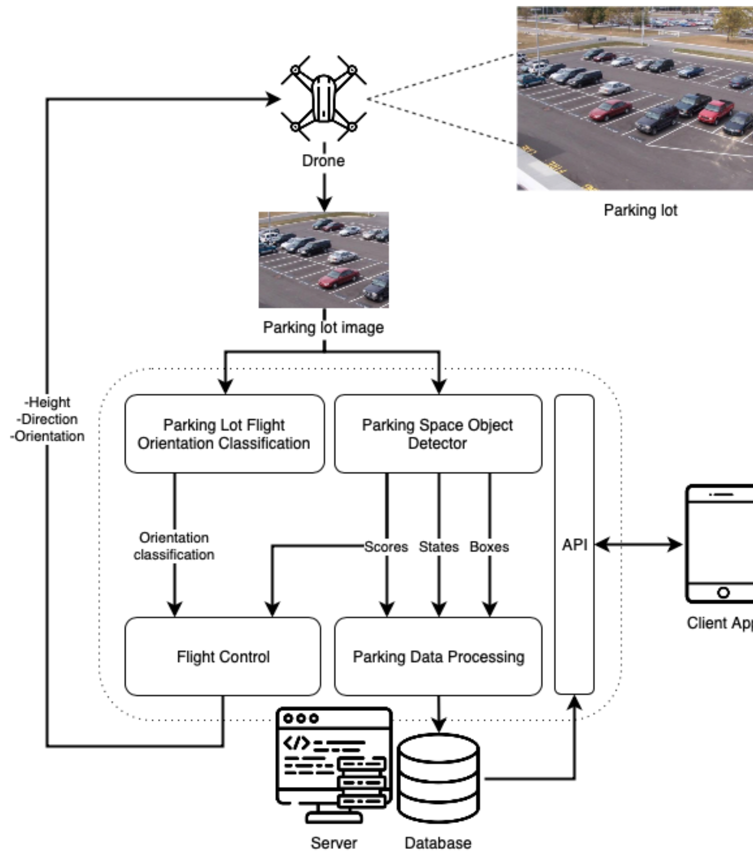

Figure 1: Overall system structure diagram.

2 DATA SETS

All of the machine learning models we wanted to develop receive an input of a parking lot image. That is why we started by collecting them. We created a Python program that could control the drone manually from the computer and display its video stream in the screen. Using different keyboard keys we could fly the drone around parking lot 228, 7 and the different sections of the lot 224. The available instructions to the drone are listed in the Table 2. After 50 takes, we achieved 1129 RGB 960 by 720 images.

Graphics	Top	In-between	Bottom
Tables	End	Last	First
Figures	Good	Similar	Very well

Table 1: Table captions should be placed below the table

Figure 2: Examples of captured images.

This is a binary image classification problem where the positive class is the image that contains a parking lot and the negative are the images that don't. Since it is

Key	**Action**
Up arrow	Move forward
Down arrow	Move backwards
Left arrow	Move left
Right arrow	Move right
W	Move up
S	Move down
A	Rotate counter clockwise
D	Rotate clockwise
T	Take-off
L	Land
C	Start/Stop capturing images

Table 2: Manual drone flight instructions.

a supervised machine learning scenario, we manually label the drone images we collected so we can transfer the human experience into observations and transfer knowledge into our classification algorithm. To make things easy, we developed a simple Python script that goes through each captured images placed in the "Takes" directory mentioned earlier, displays each image one by one and asks if that current image is a parking lot or not through the terminal console. After going through the original 1129 drone images, we classified 1096 as valid observations. 737 observations are set as positive parking lot (67,3%) and 358 were found as negative parking lot (32,7%).

In order to retrieve more visual information to guide the drone automatically, and obtain parking spot information from images we implement different parking spot object detection models. And to train them, we need a labelled parking spot object detection data-set. The "PKLot" data set [11] contains nearly 13.000 images of different angle and conditions of the parking lots of the Federal Univeristy of Parana and the Pontical Catholic University of Parana. From the previous mentioned images, nearly 700.000 segmented images of the parking spaces are obtained too. For each parking lot image there is an XML file that contains the position of each bounding box of the parking spaces and the occupancy status.

The bounding boxes have a rhomboid shape as shown in Figure 1. The majority of object detectors work with rectangular boxes aligned with the image axis. That is why we converted each bounding box found on the XML files into a aligned-rectangular shape before being recorded into the CSV file. This method basically chooses min and max points to create a rectangle.

3 CREATING MODELS

In order to guide a drone over a parking lot automatically we will require image information. We decided that one of those key pieces would be a binary image classification machine learning algorithm developed with a Convolutional Neuronal Network. Due to the specifications of our own parking lot classification data-set, the network is required to have as input tensor of 128 by 128 grayscale image. Furthermore it has to be able to be used by our back-end service implemented with Python and Django. The resulting network needs to present acceptable prediction results since it will have a primary relevance in the navigation system of the drone. That means that we are aiming for an accuracy of 90% and above.

There are many different models to choose and perform fine-tuning in the *Tensorflow Object Detection API*. We will test two different families of algorithms. The Faster-RCNN family, known for their speed over the previous RCNN models and its accuracy, and the Single Shot Detector family, known for their speed and real time applications. The difference between Faster R-CNN and the other R-CNN techniques is that instead of using Selective Search it uses Region Proposal Networks (RPN) [7]. This change improves the speed of the network. The first model we have chosen is the faster-rcnn-resnet50-coco pre-trained model since it has a good balance between speed and accuracy between all other Faster-RCNN models found in the library. This model in particular is been pre-trained with the Common Objects in Context (COCO) dataset and uses *Residual Networks* (ResNets) in its implementation [13]. The second model is the faster-rcnn-inception-coco pre-trained model. This model uses the *Inception* architecture [14]. The main idea behind the Inception architecture is that in a same layer of convolution, different filter sizes are being used and then combined to get the output tensor. The *GoogLeNet* CNN developed by Google(TM), found also in [14] uses 1 by 1 filters, 3 by 3 filters and 5 by 5 filters working together in the same layer. This model is also pre-trained over the COCO data-set. Single Shot Detector (SSD) model takes a different approach to the object detection problem [15]. Previous state-of-the-art object detection models were composed by two steps. For each location, K default boxes, called anchors, with multiple shapes and aspect rations are considered. For each anchor, the network will predict the probability of each class for the object that may fall into the box and also four offsets for the anchor to get an accurate bounding box for the object. The first SSD model we have chosen is the **ssd-mobilenet-v2-coco** pre-trained model. It is a balanced model among all the pre-trained SSD models. This model is also pre-trained over the COCO data-set and as a feature extraction architecture uses the one proposed by MobileNetV2 CNN architecture [17], a fast classification CNN developed by Google, found in many portable and real time systems. The second SSD model we will test is the **ssd-inception-coco** pre-trained model. It combines the SSD detection architecture with the Inception feature extraction architecture mentioned above.

Before testing the resulting object detection models, we can observe that the Faster-RCNN ones tend to converge at a lower loss levels than the SSD ones.

Comparing them with other reported loss metrics of other fine tuned models, we can see that are quite low values. It is a sign that our model is being able to fit correctly to the test data. Yet, we can see that the Faster-RCNN models presents less loss than the SSD models. In particular, the Resnet50 version is achieving remarkable results. Besides, among the SSD models, the MobilenetV2 seem to present better results in terms of loss than the Inception one.

4 BACK-END

The central functional piece of our system is the back-end, it handles multiple responsibilities. In the first place in here we define the information models of our system as SQL tables. Furthermore it establishes end-points so that HTTP clients, such as our iOS application, can retrieve data and send actions to the server and trigger actions. Those two responsibilities are part of the RESTful API block of our back-end. In the other hand, our server is responsible of flying the drone automatically and intelligently by integrating and using the previously exported image-based machine learning models. Besides, given the drone images and the model predictions, it has to be able to process and retrieve

the required information to create and update the required database models so we can serialize them and send them over HTTP requests and provide the clients with the needed information. We have used Python as the programming language since it has many resources and frameworks to support an incredible number of possibilities. In particular, we are using the popular Django framework to easily design the REST API and the structure of the program. In addition, we are using the Tensorflow graph framework to load the exported machine learning models and use them for the drone flight control and the data retrieving from images.

5 INFORMATION MODELS

When creating the database models of information, we need to do a design effort and figure out the basic blocks of information that better represent the problem we want to represent and resolve and how they interact. Django provides a smooth way to define those models. It allows us to define those models as Python classes by sub-classing the *Model* class found in the framework. That subclass will generate the particular SQL table for us. Then, each attribute of that class will map to its table's columns. When creating instances of that class, filling the attributes and calling its *save* method, we are recording a new row in the table. Each model inherits the **id** property that represents the primary key of the table's row.

5.0.1 Lot model

The lot model represents any parking lot we want to monitorize in our system. It consists of metadata information and useful data regarding the state of it. The first attribute, or database field is the **name** given to it as a char field. For example, one of our analyzed lots is names *lot 224* . The **created** and **updated** attributes are *Date* fields representing when the lot has been created and when it has been modified. The *image* attribute is an image field representing the path where a descriptive image of the parking lot is found on the static files directory of the server. The **occupancy** is a Float value representing the percentage of occupancy of the lot recomputed every time new spots are recorded. In a similar way, the **tendency** is a discrete attribute of five possible string values that represent how the occupancy is changing over time, if the lot's occupancy is decreasing, increasing or maintaining given different predictions over time.

5.0.2 Spot model

The spot model represents a single parking spot found in a specific parking lot that we are analysing. This entity also has a **created** date field that represents when the model is created, that is, when it has been detected. Since we want to provide a descriptive image for each spot, we include a *image* field too. A Boolean field named **is_free** is used to determine if the spot is occupied or free. In addition, the Integer field **lot_id** is a *foreign key* that matches each spot with its relative parking lot where it belongs.

5.0.3 FlightState model

The FlightState model represents a more abstract entity in our system. In this structure, we represent the state of the flight algorithm. It is used as a central piece of communication between the HTTP client and the flight algorithm through the REST API. The **state** field represent each possible state the flight algorithm can be. It consists into *LANDED*, *STARTING*, *SCANNING*, *STOPPING* and *ERROR*. The **lot_id** also is the foreign key to the parking lot we are analysing. The **enabled** Boolean flag is manually set if the flight is enabled. And finally the **created** date field is specified too.

6 ENDPOINTS

The endpoints are defined URLs that allow the HTTP clients to interact and request data with the back-end. Each URL is linked with a particular action called *View*. The View is responsible to perform its defined action and return a HTTP response with the data body and status of the request.

7 MACHINE LEARNING MODEL INTEGRATION

As mentioned in previous chapters, we used Tensorflow framework to build, test and export the image-based machine learning models proposed by this project. Now is time to deploy them and make them useful. We exported them in a standard way where in a single file the network structure, operations and weights were defined. This file is called *frozen graph*. To make use of the frozen graphs of our lot classification model and the different parking spot object detection models, the Tensorflow framework must be present in our back-end too. We will use it to recreate the models in memory reading from the disk files. To make our code modular and intuitive to use, we encapsulated the model business logic into Python class definitions and build useful methods so we define a simple interface to work with the image predictions in our algorithms. All the files regarding the model class definition and the actual frozen graphs are placed in a sub-module inside our back-end.

7.1 Lot Classification Model

The first class containing all the classification model logic is called *IsLotCNN*. In the initialization phase of that class the Tensorflow graph is loaded into memory. As a default parameter we pass the path to where the

frozen graph is placed. Once the graph is loaded into memory, we define the input tensors to the network so we can pass new image to predict. The first tensor is the actual **input**, it consists of a tensor of size *[None, 128, 128, 1]*. That means that the network can handle an undefined number of 128 by 128 grayscale images. The second input tensor is the *keep probability* of the dropout technique explained in [9]. This class defines a method to predict the probability of an input image to be a parking lot.

7.2 Spot Object Detection Models

When predicting the parking spots given a drone image, we use the defined method called *detect_drone_img*. This method runs the output tensors *boxes, classes and scores* in the defined session with the drone image as the input value from the input tensor of the network. The output tensors return arrays of boxes (x,y minimum and maximum) defining the bounding boxes of each spot detection, the class of each detection and the confidence score of each detection respectively. Yet, to work easily with the predictions, we defined a class called *PKLotBox* that keeps all the information of a single detection so we can return a single array that each element contains all the detection information instead of a collection of arrays.

8 PARKING LOT DATA PROCESSING

Before handling the flight control algorithm, we decided to prepare first the components that retrieve information from the images and predictions since it is a smaller component that can be easily tested and prepared before facing the main algorithm of the system.

The parking lot data processing module expects that we already have an incoming image and we have already computed the spot detection on that image. Those two elements combined with the previous information in the database regarding the state of the parking lot model and the corresponding spots found in early iterations are the inputs for this process. As for the output, we expect to have registered the new classifications found in the incoming image transformed into Spot database models. Furthermore, once the new spots have been recorded, we need to recompute the new occupancy and tendency of the Lot model we are analyzing.

To compose the Spot models from spot predictions, we are segmenting the original image and storing it into the static files directory so we can access it over HTTP requests, assign this image to the Spot model, retrieving the class of the detection and save the model into the database. Once we recorded all the new spots in the database, retrieve all stored spots given the lot we are analyzing. The new occupancy will be estimated given the ratio of all occupied spots over the total spots. Then, quantizing the difference between the old and new occupancy we obtain the new tendency category.

9 FLIGHT CONTROL – NOVEL ALGORITHM

The flight control algorithm is the core process of our system. It is responsible of the drone guide and communication, including the retribution of images over the UDP video streaming that the drone provides. Furthermore it uses the imported image classification model and multiple object detection models to perform decisions and retrieve information using the defined classes *IsLotCNN* and *PKLotDetector*. Besides it uses the *PKLotDataRetriever* class to obtain information from the spot predictions and update the database. In addition, the algorithm is responsible to update the *FlightState* database model so we can have its information centralised. Also it will check the *FlightState* to know if the HTTP client requested for the flight to stop.

As we mentioned before, this process will be triggered from the action related to the *flight/lot/<int:pk>/start/* endpoint exposed by our REST API module. This means this will be triggered by the client iOS application once we detect that the user wants to know the actual situation of a parking lot. Since the HTTP responses of our back-end have to respond quickly to the client petition, we will need to execute this algorithm in a background thread from our back-end Python process.

9.0.1 Algorithm Overview

Find in Figure 3 the main skeleton of the flight control algorithm. In the first step we initialise all the resources that the algorithm needs, starting with retrieving the needed database models, initialising the machine learning models and setting the drone connection, video streaming and taking off.

9.0.2 Implementation

In Algorithm 1 can be seen the pseudo-code implementation of the **start** method, that is, the main body of the algorithm. It maps the overview algorithm diagram we described above. Yet it delegates important tasks to other functions of the class as explained in the *FlightControl* class diagram to better structure the code. Below we explain the most important methods used by the main algorithm.

The algorithm 2 shows the pseudo-code for the **is_pointing_to_lot(image)** method. It can be seen that we use the lot classification model **cnn** to compute the probability if the input image of being a parking lot or not. Following with the description of internal methods, the Algorithm 3 illustrates how we adjust the initial pose of a scanning iteration when at that point we are not properly pointing to a parking lot. We will do that by performing small and random movements. Afterwards, we will check again if we are pointing to a lot or not. Eventually the drone may find a good position and proceed with the scanning process.

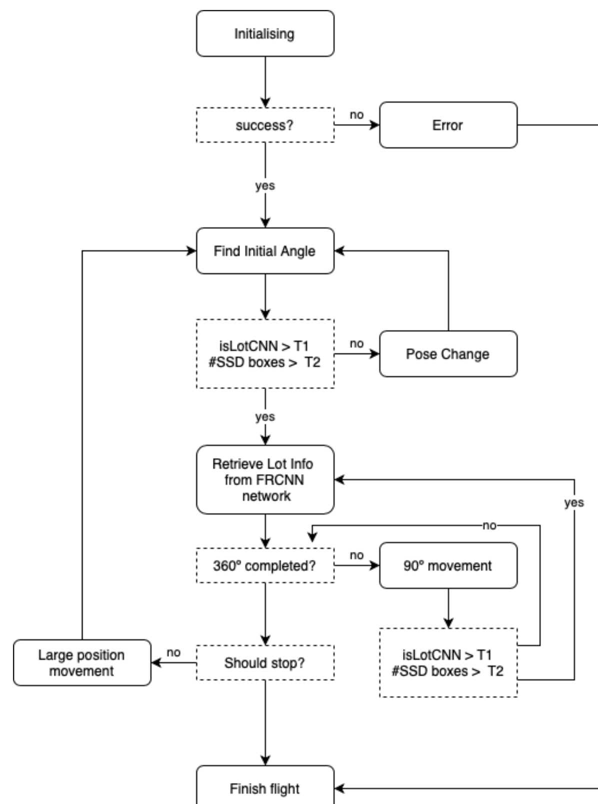

Figure 3: Flight control algorithm diagram

Algorithm 1 Flight control main algorithm (start method)

flight_state := get_flight_state();
lot := self.get_lot();
set_up_networks();
set_up_drone();
set_initial_position();
set_state_to(SCANNING);
While True
While not is_pointing_to_lot(image=stream.frame)
adjust_initial_pose();
retrieve_info_from(image=stream.frame);
for angle in [90, 180, 270] move_to_next_angle();
If is_pointing_to_lot(image=stream.frame) retrieve_info_from(image=stream.frame);
should_finish_flight() break
change_initial_position();
finish_flight()

First, we will determine the rotation direction, then between 20 and 180 degrees we will find a random rotation angle magnitude. The forward movement magnitude will be found between 200 and 300 cm. The Algorithm 4 shows how the algorithm uses the FRCNN model to predict the spots found in a given image and then pass the image and the spot detections to the *PKLotDataRetriever* module explained above. Finally, in the Algorithm 5 we can see the implementation of how we change to the next initial scanning position after a complete iteration. First, the forward distance is set to 500cm. Then we rotate the drone with a random angle until we are certain that the drone is pointing correctly to the parking lot. This way we make sure that the drone is moving forward in a direction where more probably the parking lot extends.

Algorithm 2 is_pointing_to_lot(image) method

image is_lot
lot_prob := cnn.predict_drone_img(image=image);
lots = ssd.detect_drone_img(
image,confidence=ssd_detect_conf);
lot_count := length(lots);
is_lot := (lot_prob >= is_lot_thr) AND (lot_count >= ssd_count_thr);

Algorithm 3 adjust_initial_pose() method

rotation_direction := random(0, 1);
rotation_angle := random(20, 180);
forward_distance := random(200, 300);
rotation_direction drone.rotate_clockwise(rotation_angle);
drone.rotate_counter_clockwise(rotation_angle);

Algorithm 4 retrieve_info_from(image) method

```
detections := frcnn.detect_drone_img(image, confidence=frcnn_detect_conf)
info_retriever.retrieve_data_from(lot_image=image, predictions=detections)
```

Algorithm 5 change_initial_position() method

```
forward_distance := 500;
WhileTrue
rotation_direction := random(0, 1);
rotation_angle := random(20, 180);
Ifrotation_direction
drone.rotate_clockwise(rotation_angle);
drone.rotate_counter_clockwise(rotation_angle);
Ifis_pointing_to_lot(image=stream.frame) break;
drone.move_forward(forward_distance);
```

9.0.3 Testing the flight control

The Figures 4 through 7 show four real tests over different points of the University of Colorado at Colorado Springs parking lot 224. In each test we tested a different combination of object detection models for the tasks of guide the drone and the retribution of information to see which one is performing better. In each one of the tests a complete scanning loop iteration has been recorded and analysed.In the 90 degree angle analysis the classification CNN detected that it may not be a parking lot even though the SSD model was detecting some spots yet some of them where wrong classification. That image was a hard one to analyse due to the occlusions so the CNN did a good job by skipping it. In the rest of the images the drone was pointing to the parking lot correctly and the CNN gave high probability values for those, meaning that it is working. The SSD MobilenetV2 model is doing a good job by analysing fast the drone images for possible spots. Regarding to the third test found int Figure 6, we are using the FR-CNN Inception model for both information and guide tasks. It is performing a bit slower than the first test since the SSD models compute faster. Yet the timings are acceptable for our task. In this test can be seen that our detection models struggle most with the detection of empty parking spots. Again, the CNN is working as expected by giving a high probability to the lot images and a low one for the images not pointing to it. Finally, the last test found in Figure 7 we are using the FR-CNN Resnet50 model for the data computing task and the FRCNN Inception for the fast spot analysis. Again, using the Resnet50 model is resulting into a non acceptable timing in the scanning iterations. Yet in this test, we are able to detect a good amount of spots, including the empty ones.

10 CLIENT APLICATION

At this point, we have already put all the pieces together to obtain parking lot state information and detecting their spots by controlling the drone automatically, processing its images with our own built machine learning models and recording it to the database. Yet we are missing the piece that would allow to make this system useful by displaying the information we are processing. In our case, this piece is the iOS application. It serves as a client of our back-end and requests data through the defined endpoints to display parking lot information in an intuitive way. To implement it we have used Apple's native tools. We used *Swift 5* as the programming language and XCode as integrated development environment.

11 SUMMARY OF RESULTS AND FUTURE RESEARCH

In total, we tried four different model architectures so we could evaluate their performance and have multiple options at the time of the flight control algorithm implementation. Those models were: two Faster-RCNN family models, one using Residual Networks and the second one using the Inception technique as a feature extractor. The other two models were from the Single-Shot-Detector family, one using MobileNetV2 as feature extractor and the other using also the Inception technique. As a result we achieved really precise models with a higher time complexity, which can only improve with better infrastructure such as faster networks and computers in future. We implemented methods which were usr selected and could be more precise yet are capable of working on real time. So we feel that we have provided good balance in our research. As mentioned earlier, it is not only necessary to build algorithms to retrieve information from parking lots. The resulting model could classify with a 90% of accuracy over the test images and it predicts with an speed of half a second. After deploying that model in the flight control algorithm, the parking lot classification model was able to provide confident predictions, resulting into nice drone navigation results with 90% accuracy on limited training.

12 ACKNOWLEDGEMENTS

We want to thank WSCG anonymous reviewers for providing us feedback so that our paper has improved. In addition, first author can be contacted for code for our applications as our research has benefitted from open source data sets and deep learning algorithms.

13 REFERENCES

[1] Qi Wu, Chingchun Huang,Shih-yu Wang,Wei-chen Chiu, Tsuhan Chen. ROBUST PARKING

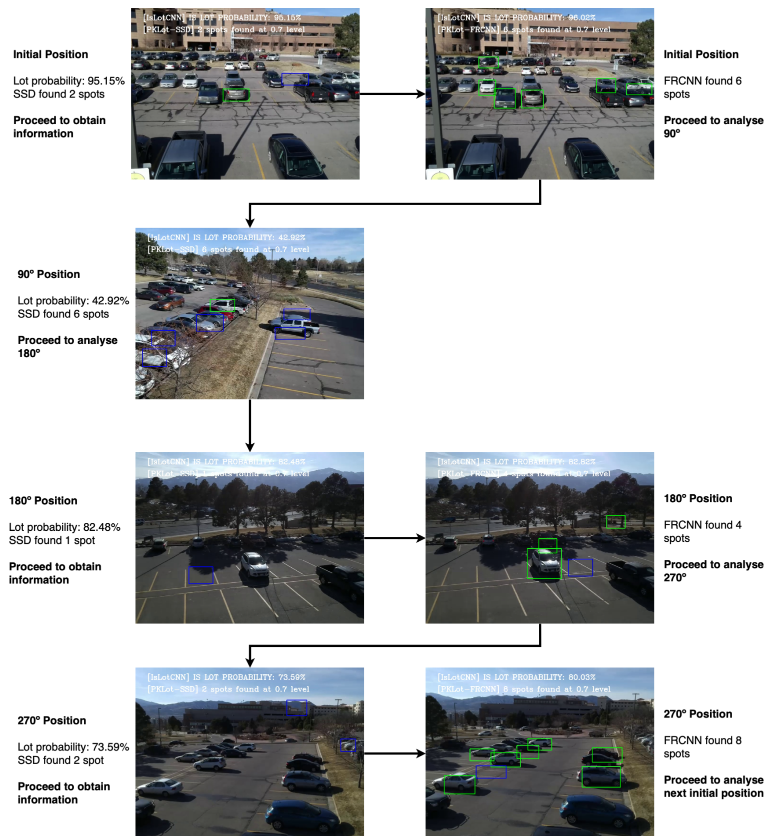

Figure 4: Flight test with FRCNN Inception and SSD MobilenetV2

Figure 5: Flight test with FRCNN Resnet50 and SSD Inception

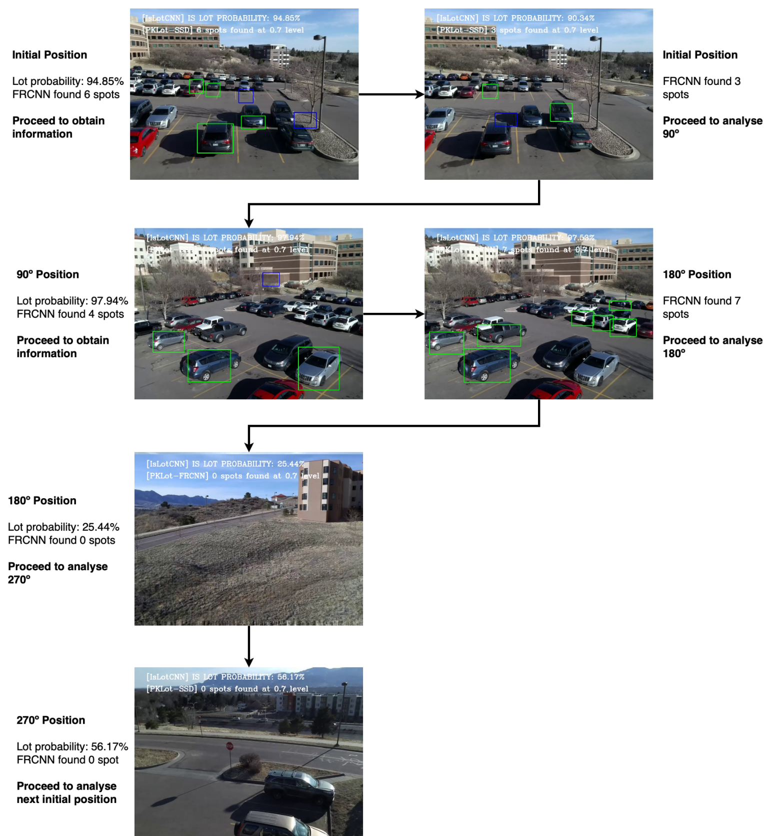

Figure 6: Flight test with double FRCNN Inception

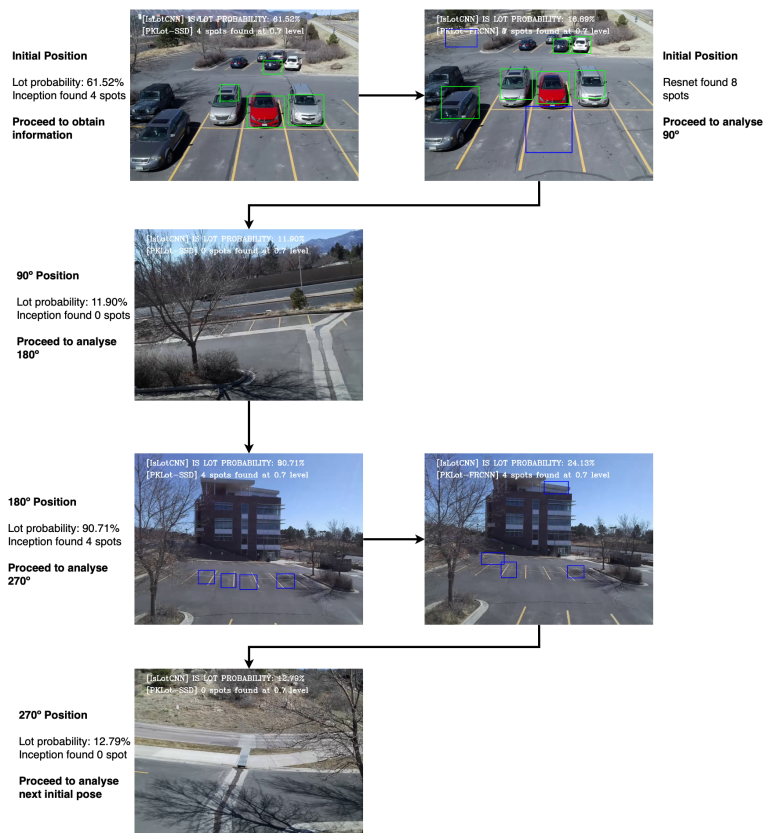

Figure 7: Flight test with FRCNN Resnet50 and FRCNN Inception

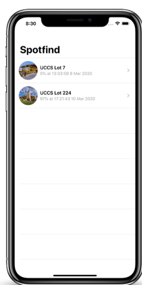

Figure 8: Sample of Lot list

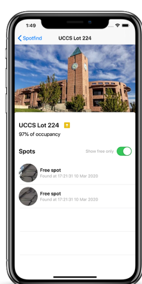

Figure 9: Pull to refresh data

SPACE DETECTION CONSIDERING INTER-SPACE CORRELATION, IEEE International Conference on Multimedia and Expo (2007)

[2] MartinAhrnbom ,Kalle Amstron and Mikael Nilsson. Car Parking Occupancy Detection Using Smart Camera Networks and Deep Learning, IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2016.

[3] Giuseppe Amato, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro and Claudio VairoCar Parking Occupancy Detection Using Smart Camera Networks and Deep Learning.)2016 IEEE Symposium on Computers and Communication (ISCC), 2016

[4] Julien Nyambal, Richard Klein, Pattern Recognition Association of South Africa and Robotics and Mechatronics, Automated Parking Space Detection Using Convolutional Neural Networks, 2017.

[5] Yunchao Wei, Wei Xia, Min Lin, Junshi Huang, Bingbing Ni, Jian Dong, Yao Zhao and Shuicheng Yan, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, HCP: A Flexible CNN Framework for Multi-Label Image Classification, 2016

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, Microsoft Research, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016.

[7] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, Xiaolin Hu, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), High Performance Visual Tracking With Siamese Region Proposal Network, 2018.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), You Only Look Once: Unified, Real-Time Object Detection, 2016

[9] Guilleum Budia Tirado, Automatic Parking Lot Detection with Image-Based machne learning, drones, and mobile platforms, MS thesis, Department of Computer Science, University of Colorado Colorado Springs.CO, USA. Advisor: Sudhanshu Kumar Semwal, pp. 1-104, 2020.

[10] Punarjay Chakravarty, Klaas Kelchtermans, Tom Roussel, Stijn Wellens, Tinne Tuytelaars and Luc Van Eycken, IEEE International Conference on Robotics and Automation (ICRA), CNN-based Single Image Obstacle Avoidance on a Quadrotor, 2017.

[11] Widodo Budiharto , Alexander A S Gunawan , Jarot S. Suroso , Andry Chowanda , Aurello Patrik and Gaudi Utama, International Conference on Computer and Communication Systems, Fast Object Detection for Quadcopter Drone using Deep Learning, 2018.

[12] Paulo Almeida, Luiz S. Oliveira, Alceu S. Britto Jr, Eunelson J. Silva Jr Alessandro Koerich, PKLot - A Robust Dataset for Parking Lot Classification, 2015.

[13] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Proceedings of the IEEE, Gradient-based learning applied to document recognition, 1998.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Deep Residual Learning for Image Recognition, 2016.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Going deeper with convolutions, 2015.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Google AI, SSD: Single Shot MultiBox Detector, 2016.

[17] Jonathan Huang, Vivek Rathod, Chen Sun, Speed/accuracy trade-offs for modern convolutional object detectors, 2016.

[18] Mark Sandler Andrew Howard Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen, IEEE/CVF Conference on Computer Vision and Pattern Recognition, MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2018.