Spring 5-20-2019

Detecting Cars in a Parking Lot using Deep Learning

Samuel Ordonia
San Jose State University

Detecting Cars in a Parking Lot using Deep Learning

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Samuel Ordonia

May 2019

The Designated Committee Approves the Project Titled

DETECTING CARS IN A PARKING LOT USING DEEP LEARNING

by

Samuel Ordonia

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2019

Dr. Chris Pollett Department of Computer Science

Dr. Jon Pearce Department of Computer Science

Dr. Teng Moh Department of Computer Science

ABSTRACT

DETECTING CARS IN A PARKING LOT USING DEEP LEARNING

by Samuel Ordonia

Detection of cars in a parking lot with deep learning involves locating all objects of interest in a parking lot image and classifying the contents of all bounding boxes as cars. Because of the variety of shape, color, contrast, pose, and occlusion, a deep neural net was chosen to encompass all the significant features required by the detector to differentiate cars from not cars. In this project, car detection was accomplished with a convolutional neural net (CNN) based on the You Only Look Once (YOLO) model architectures. An application was built to train and validate a car detection CNN as well as track the quantity of cars in a parking lot across a period of time. A separate service called Vision maps and standardizes the input data formats for the CNN as well as sanity checks the bounding box labeling with ground truth image annotations. Finally, another serviced called Skynet analyzes and summarizes the car count statistics over time on a series of parking lot images.

ACKNOWLEDGEMENT

I would like to thank my family for all their love and support, Professor Pollett for guiding me throughout the research process, my committee members for their time now and in the classes I took with them, Robbie for inspiring me to take on this challenging parking lot car detection problem, my manager Bo for mentoring me in engineering best practices as well as computer vision, and friends for keeping me grounded.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

This project addresses the difficult task of detecting the locations of dozens of cars in a single image. Object detection of cars in a parking lot is a two-part problem. First, objects of interest need to be identified in the image; this is referred to as localization. Second, the objects of interest need to be identified as cars; this is referred to as a classification. Localization of several objects of various sizes in the foreground and background of an image is quite difficult. Depending on the dataset content, some cars in the foreground can be quite large, while other cars in the background encompass a fraction of the pixels of their foreground neighbors. Extensive deep learning research has been done already in object localization with respect to iconic images--frames containing few objects, with the handful of objects in each one usually the center of the image. In the space of car detection, there has been research done to detect cars moving through a toll station with neural networks in the 1990s [1].

CNNs have proven effective at learning significant features in images since its usage in LeCun's digit classifier [2]. In recent years, various CNN architectures applied to object detection have converged into two camps. The first overarching architecture of CNN object detector is the single-shot detector (SSD) approach [3]. This involves feeding the entire image into the neural net at once instead of subsections of the image. Moreover, the model detects all the car bounding boxes in the same pass as

the classification, thus having significantly shorter runtimes than R-CNNs. It is this approach of single-shot CNN detectors that this project bases its work on, specifically the You Only Look Once (YOLO) ones [3].

The other camp applies a sliding window approach that harkens back to classical sliding window approaches to object detection [4]. The image is divided into subsections that each pass through the CNN for object classification, and the corresponding location of the subsection from the image is used to locate the object in the frame. This is the regional CNN, or R-CNN. Subsequent iterations have worked to improve the runtime by determining multiple regions of interest simultaneously. Nevertheless, this region proposal network approach suffers longer runtime, which can exacerbate parking lot detection latency. And in other use cases involving cars, such as driverless automobiles, the latency can prove detrimental.

This project shows the feasibility of using the latter camp's SSD CNN architecture to detect dozens of cars in a parking lot image. This report will review the key components and theories applied to the YOLO CNN models. This project's various applications will then be presented and how they were implemented with technologies such as Docker and Pytorch to drive the research effort and lay the foundation for a practical parking lot application. The experiments design, results, and analysis will then be covered. A section will also review the effectiveness of the parking

lot car counting application Skynet. Finally, concluding remarks reiterating the project learnings as well as next steps will be discussed.

2. BACKGROUND

2.1 Fundamental Building Blocks of the CNN

This project's entire car detection deep learning model relies on the fundamental building block of every CNN, which is the convolutional layer itself. It is often applied to grid-like inputs such as the matrix representation of images. The layer's behavior is akin to the mathematical convolution linear operation that accepts two real valued parameters, the input and the kernel--or filter, as referred to in deep learning [5]. Tensors are inputs to and outputs from the convolutional layer and obey transformation rules defined elsewhere in the model. The output tensor from a set of convolution layers is sometimes referred to as a feature map.

The convolution operation is equivariant to certain other transformations, specifically translation. This means that the output of convolution and translation would match the output of translation and convolution. E.g.,

$$f(g(x)) = g(f(x))$$

Given two images of cars, the CNN should detect the same car in both images, even if it has moved between the frames.

In the simplified case of a grayscale image matrix, the convolution operation is applied by the kernel across the entire input matrix. This differs from a fully connected perceptron layer in that the convolutional layer benefits from parameter sharing and sparse connectivity. The kernel is usually several times smaller than the input. And the

kernel is applied across the input in smaller linear transformations that are not overly memory heavy compared to operations that would be performed in a fully connected layer. The output tensor of the convolution operation may also be smaller than the input, reducing the memory requirements as well.

There are cases when it is not desirable that the output size be reduced by the convolution operation. If there are several convolutional layers, each reducing the size of the output, the feature map can reduce to a point where no new features can be learned by the model, and the gradient vanishes, which ends the training process. Also, by nature of the convolution operation, the kernel is applied against the edges of the input less than other parts of the input. To counteract this, convolution layers tend to add padding of zero values around the input to preserve dimensionality and apply the kernel more evenly across the input.

Another setting often applied in convolutional layers is stride. Stride can be considered as a "skip" variable that applies the kernel over fewer subsections of the input. Instead of applying the kernel across the entire input, stride skips over part of the input by a whole number delta to apply the kernel further down the input.

2.2 Intersection over Union

As applied in this project, Intersection over Union (IOU) is a ratio for comparing and contrasting two states--prediction and ground truth. The intersection defines the

space that both states share in common, while the union defines the total encompassing space.

$$IOU = \text{Intersection Space} / \text{Union Space}$$

This metric gauges how accurate a model determines the location of an object. During training, weights are updated based on how close the predicted bounding box matches the coordinates of the ground truth one. The IOU ratio of the predicted and ground truth bounding boxes provides a quantitative score on whether a model's prediction is correct or needs to be updated. During detection, IOU is also compared against a threshold hyperparameter to surface the ones in which we have sufficient confidence.

2.3 Anchor Boxes

The anchor box is another key component to car detection in a parking lot and is directly related to how classification and bounding box regression is done in this project. An anchor box is a hyperparameter "bounding box" derived from the ground truth shapes the model intends to detect. The anchor box enables a deep learning model to specialize in a specific set of classes for detection by defining standard shapes objects of interest take. For example, cars tend to have very consistent shapes across models. Cars are longer than they are taller, so the model should learn those shapes as opposed to other shapes that may be more appropriate for people or trees. During bounding box regression, the model should narrow down what predictions it surfaces to those that fit the defined anchor box shapes for cars.

Another advantage of using anchor boxes is to use the same pixel in the input to detect multiple, different classes. If a model needed to detect and differentiate people from cars, it could compare all predicted bounding box IOU ratios to the anchor boxes. If there were a match for the car anchor box as well as the person anchor box, then the model would predict that both a human and a car were located in that portion of the image. This saves computation time as well as allocates the rest of the pixels input for bounding box regression.

Often, several anchor boxes are defined for a model prior to training. As they are hyperparameters, anchor boxes can be defined ad hoc based on the objects of

interest for detection. Anchor boxes can be determined manually from the ground truth, perhaps computed as an average. The YOLO paper took a standardized approach to anchor box definitions by applying k-means clustering to the training set in order to determine the appropriate bounding boxes.

Unfortunately, one disadvantage of anchor boxes is their difficulty in differentiating the *same* object class when comparing bounding boxes [6]. This can be an issue with cars in parking lots, where certain camera angles result in highly occluded shots of parked cars behind other ones that are parked at the same angle. Because the proposed bounding boxes for both objects would be quite similar, the model would need a tie-breaker or other method to resolve the closely scored IOUs.

2.4 Non-max Suppression

Non-max suppression (NMS) has been applied for decades, since its inception in edge detection. For that use case, it determined the most likely edge at a pixel on an image based on the local maxima of gradients around it, and it "suppresses" all other gradients by setting them to zero. This approach can be applied in car detection with deep learning when determining the predicted car bounding boxes.

NMS provides a way to resolve multiple predicted bounding boxes by picking the one most likely to represent the object of interest [6]. When an image is divided into a square grid during CNN processing, the resulting output could be multiple grids detecting objects of interest. And each grid would have its corresponding bounding

boxes based on the anchors in order to differentiate different object classes in the same grid cell.

The NMS algorithm involves a few steps defined by threshold parameters. First, objects below the confidence probability threshold are discarded. Then, the remaining predicted bounding boxes are compared against the anchor boxes. The bounding boxes with the highest IOUs are chosen per anchor box. This means that there can be more than one maximized IOU bounding box per anchor, so more than one class may be detected depending on the anchor boxes defined.

2.5 YOLO Detection - You Only Look Once

The YOLO object detection algorithm has a single forward pass on the entire input image tensor, making its runtime quite fast compared to multi-stage detection architectures [4]. To do so, the YOLO CNN incorporates all the concepts covered thus far. Although the YOLO CNN has gone through three iterations, this project focused on the last two versions. Nevertheless, the original implementation deserves elaboration as well.

The first version began with a novel approach to use a single deep network to detect objects. The bounding box predictions were treated as the overarching problem, which once isolated could be used to determine what classes occupied them. This approach differed from the other ones of its time. Previously, a sliding window classifier was used to localize different objects in an image. Naturally, this solution suffered long runtimes that would never realize real-time requirements. Other approaches involved splitting up the detection problem into localization or segmentation, followed by the classification of the cropped image containing just that object proposal. These approaches often included CNNs in the pipeline, but despite efforts to optimize them, they still suffered from long runtimes because of their multi-step approaches.

The YOLO CNN conducted bounding box regression and did classification simultaneously from the feature map extracted by the convolution layers prior to the

detection step. The network's final fully connected layers would be defined based on how many classes YOLO was trained on. Each grid cell in the detection layer is responsible for classifying one object. Localization is determined via NMS by comparing the predicted bounding boxes to the ground truth and deciding based on the highest IOU.

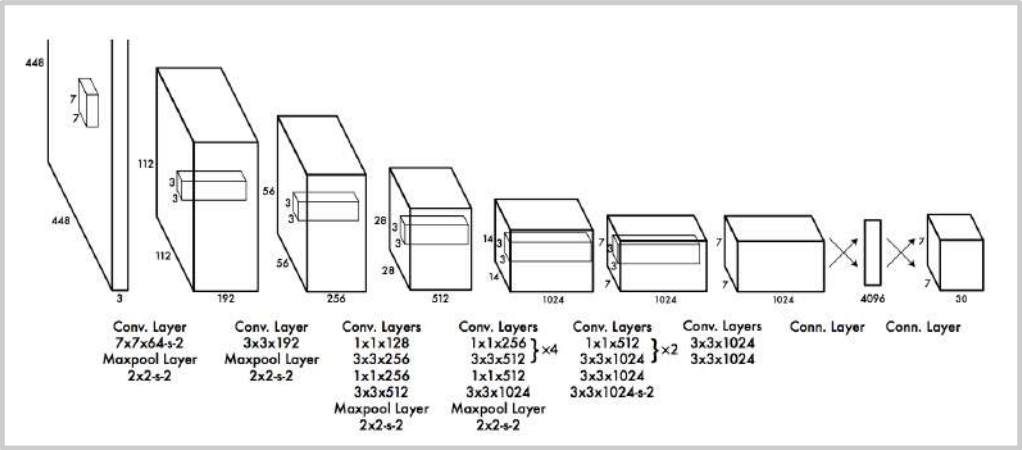

Figure 1: CNN architecture of the first YOLO model. [3]

The YOLO loss function captures classification and localization loss, so the model can train for object detection all at once.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Figure 2: Loss function for the first YOLO iteration. [3]

The classification loss is the squared error of the class conditional probabilities for every grid that detects an object. The localization loss terms resolve to zero everywhere except for the box that is responsible for detecting the object; the square root predictions for the bounding box is compared to the ground truth one.

2.7 YOLO v2 - YOLO9000

The most noteworthy innovation in the second YOLO version is the update to utilize anchor boxes during detection. Rather than compare predictions to the ground truth coordinates, the model is trained and modified based on offsets between the predictions and anchor boxes. The authors of YOLO9000 determined from the start to use k-means clustering to determine the best-fitting anchor boxes for their objects of

interest [6]. This saw the transformation of the detection layer to be defined by the anchor boxes themselves in addition to the number of classes to detect.

The YOLO9000 model was named so because it successfully detected nine thousand different classes [6]. For example, it could distinguish different types of dogs or birds because it could perform hierarchical classification. Although this project only aimed to detect cars in general in a parking lot, given a more granular dataset, it could be trained to distinguish sedans, trucks, vans, buses, etc., from one another.

Finally, because of the separation of the classification and localization loss terms established since the first YOLO iteration, the CNN could be trained for detection with one dataset while augmenting its classification accuracy with supplemental ones that may not provide bounding box labels. For YOLO9000, the model was trained primarily on the Microsoft COCO dataset for detection that contains images with multiple objects in each one [7]. And its classification accuracy was augmented by thousands of more images from ImageNet, including cars. This would prove helpful to some extent in this project, since the weights of this project are based on the publicized ones from Redmon's work.

2.6 YOLO v3 - Detection at Three Levels

The latest iteration on the YOLO detection model improves its accuracy by sacrificing to some extent both memory and time. Version three outperformed many

of its contemporaries that had previously beaten its previous version in accuracy, including some models from the multi-stage camp of CNNs [8].

The latest version is deeper with many more layers to load into main memory, and as a consequence of being deeper, each pass through the networks is longer than its predecessors. Features that were included into this latest iteration from its predecessor include the overall goal of a single-shot detector that does localization and classification at once, the anchor box approach to bounding box regression, non-max suppression of IOUs, and augmented classification training.

The model underwent a large refactor in its feature extraction network as well as its approach to detecting objects from the CNN. Instead of the much smaller v2 network consisting of nineteen feature extraction layers, the v3 CNN bases its feature extraction portion on the Darknet-53 CNN trained on ImageNet. Likewise, the detection portion of the v3 model is much deeper in order to support detection at multiple layers.

The addition of more detection layers was motivated by the v2 model having difficulty detecting small objects. The feature map that the v2's detection layer consumed was dependent on the input tensor dimensionality. This approach was ideal for iconic images such as those in ImageNet, where the objects of interest usually took up the majority of the frame. However, many real life applications, including cars in a parking lot, would need to support detection of small objects.

Darknet-53 downsampled the image input by three factors during feature extraction. When training on COCO data at 416x416 size, the input was downsized to various dimensions depending on the stride; the lower the stride, the more the downsampling occurs at that hidden layer.

With the deeper Darknet-53 feature extraction portion, the newer YOLO v3 model could support detection at multiple different levels of upsampling. Upsampling is the technique of increasing the size of an image. By doing so, smaller objects can be more easily detected, since more cells exist to perform detection [8].

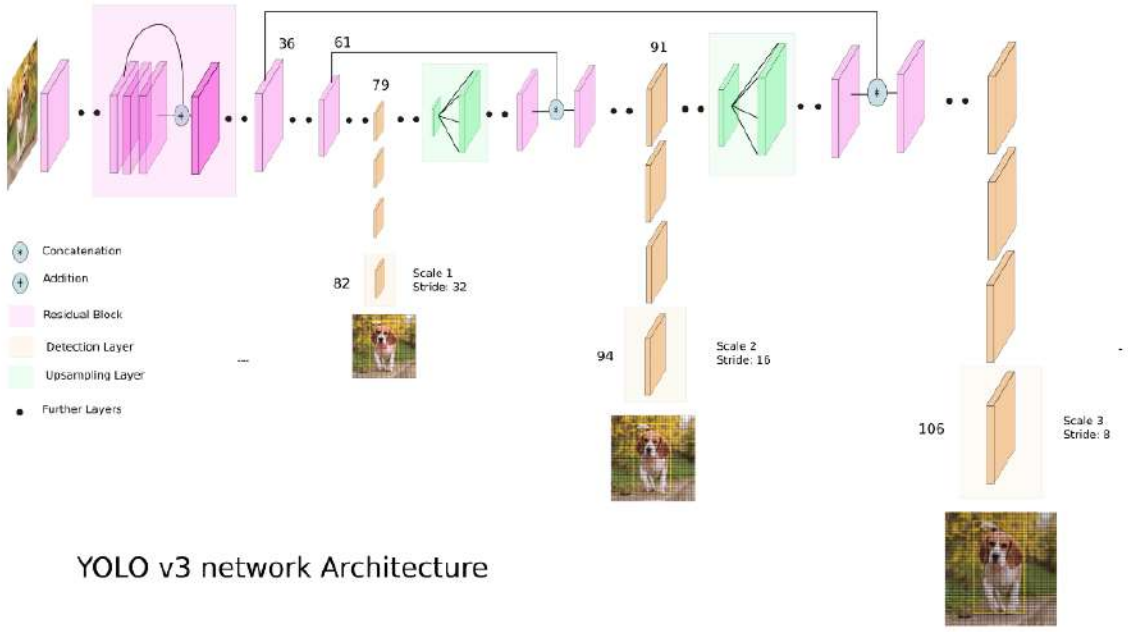

Figure 3: Visual representation of upsampling during detection of a COCO image [9].

Route layers are used to feed feature maps at different downsample points in the Darknet-53 portion to the corresponding detection section of the network. These

feature maps are represented as concatenated tensors from the specified feature extraction layers. Following the route layer, there are a final series of convolutional layers at the intended stride factor for the desired upsampling scale. Finally, the tensor is fed into the YOLO detection layer, where the classification and bounding box regression occurs.

3. IMPLEMENTATION

3.1 Vision - Open CV in Docker

Vision is the pre-CNN application this project utilized for compiling and validating each parking lot image set. It is a Docker application with Open CV installed in it and supports a wide variety of functions and tools for operating on images, such as resizing, rotating, and other affine transformations. Vision also supports bounding box annotation of objects within an image given an input of ground truth labeling. This annotation of images proved quite useful in validating a dataset's labeling.

There were several datasets used for training. Each one differed in its labeling format and file structure. An independent application from the deep learning portion was necessary to process the different datasets and compile them into a uniform format that the deep learning model can consume. Further, after processing a dataset into the standardized format the CNN could consume, Vision was used to draw the ground truth bounding boxes on the cars in each image. These Vision-annotated images would confirm whether the conversion of the ground truth labeling format to the standardized one was successful.

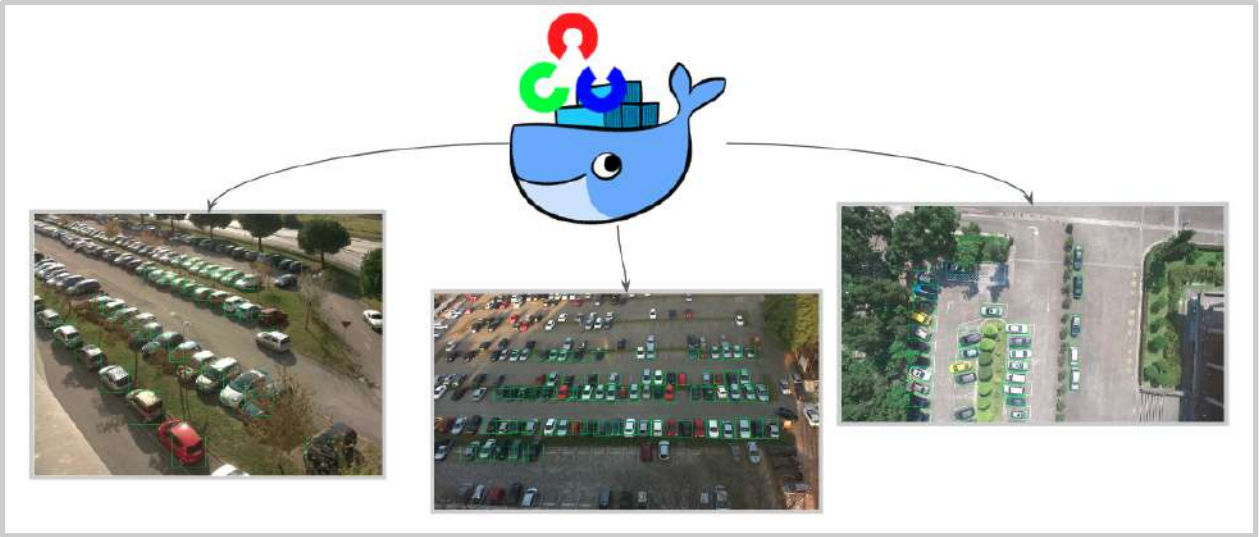

Figure 4: Vision application employs Open CV in a Docker container to process and analyze the parking lot images.

Open CV is a sizable image framework that provides several image transformation functions as well as various levels of processing. Nearly every standard image processing algorithm is included in the framework. Unfortunately, because it builds and installs a range of dependencies, Open CV tends to overwrite existing packages, libraries, and other binaries already installed on the host machine. This can lead to a non-working development environment with mismatched package dependencies and overwritten install or command paths.

Docker is Linux containerization as a service. Although containerization has existed for some time on the Linux kernel, Docker wraps the logic in concise syntax and provides a plethora of documentation on its features. Containerization of the Open CV framework enabled installation and execution of the image processing framework while also leaving untouched all the packages and install paths of the host machine.

Containerized processes can be started, stopped, or destroyed at will. And unlike a virtual machine, a Docker application reduces performance cost by sharing the Linux kernel with the host machine, instead of running its own operating system.

3.2 Parking Lot Car Detector - Pytorch Research Environment

The Pytorch car detector application supports many different configurations of deep learning models. It has comprehensive modules for training and testing detection. It also supports the exploration of various neural network layouts with a factory-like collection of generalized layer types, such as a convolution, route, or detection layer. Finally, the research environment houses a collection of utility functions and supplemental modules to aid the entire research process.

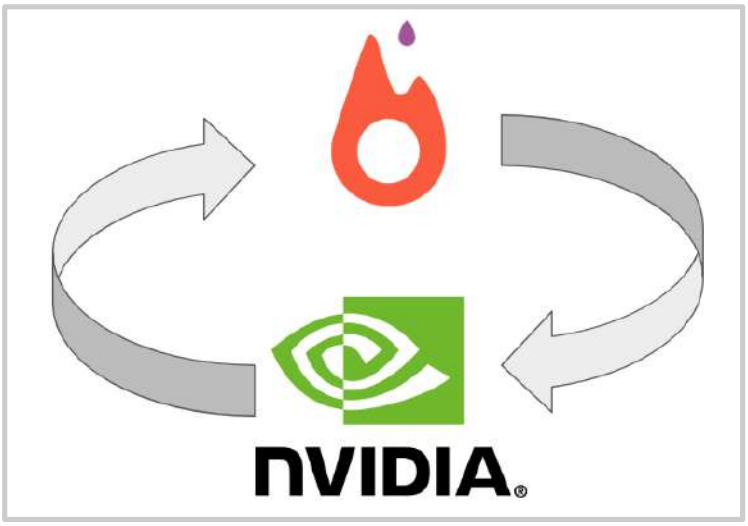

Figure 5: The project's research environment is powered by the Pytorch deep learning Python framework. The training process is accelerated greatly by NVIDIA's CUDA GPU framework.

Pytorch is a Python deep learning framework geared towards research in its flexibility and level of abstraction. Based off the Torch deep learning framework in Lua, Pytorch enjoys much of the libraries, readability, and other advantages that Python as a language provides. Graphs in Pytorch are constructed dynamically, so debugging breakpoints can be triggered as the graph is constructed to determine more easily any problems with the setup. This is also a benefit of Pytorch executing its code dynamically instead of having an intermediate compile step.

Pytorch also draws the line at the right levels of abstraction to empower researchers with the tools to create and iterate their deep learning models with both flexibility and terseness. Deep learning staples such as backpropagation, activation functions, and tensor instantiation are handled by the framework, and researchers can focus on defining forward passes, any custom layers, and iterating over hyperparameters of the model.

Training a deep learning model is computationally expensive, requiring sometimes several gigabytes of memory. Moreover, the majority of the tensor operations can be optimized with additional processes; unfortunately, most CPUs have very few cores to spread the load. Researchers always turn to GPUs to train effectively. And Pytorch has excellent CUDA support as part of its binaries. There is no need for researchers to set up CUDA or write CUDA code to interface with the GPU, as Pytorch handles all of that. A simple flag on tensor instantiation or the model will run that

piece of Pytorch code on the GPU. This ease of mind and separation of responsibilities quickly became a hallmark of why Pytorch was chosen as a deep learning framework for this project.

3.3 Models as Configuration Files

The deep learning community builds and iterates their neural network models in a variety of programming languages and frameworks for research and production. The YOLO implementation benchmarked in the published papers was written in Darknet, which took advantage of optimized deep learning algorithms in C and CUDA. Even neural networks written in Python may be using a different deep learning framework other than Pytorch, such as Tensorflow, Keras, or Caffe.

The deep learning community solved this issue by defining and sharing their model definitions in configuration (cfg) files. These cfg files follow a human-readable syntax to define model-level hyperparameters as well as individual layer properties. Each layer receives its own block, and recognized attributes are included within each one. For example, a convolutional layer will have included in its block stride, activation function, and filter attributes. The YOLO detection layer includes an anchor box list attribute and IOU threshold.

The Pytorch implementation of this project needed to support a factory wrapper that generated all the individual neural net layers given the cfg input. Pytorch has several fail-fast safeguards to ensure a constructed network is valid (i.e., tensor inputs

and layer definitions did not result in mismatched dimensions). Because of Pytorch's imperative nature, debugging issues at the cfg level were not overly cumbersome or confusing.

The cfg file affords a great deal of flexibility and may not even require any subsequent modifications to the Pytorch code, unless a new feature is required. This reduces friction in the research iteration process.

3.4 Tensorboard

Tensorboard is a visualization tool for deep learning metrics. It is primarily used in this project to monitor model convergence as loss reduces over training time, as well as other scalars of interest such as epoch. It can also be used to track layer-specific metrics such as weights and biases.

Although Tensorboard is a tool in the Tensorflow library, its use case is to read in and display charts for any deep learning training session, regardless of the framework used for the actual training. Tensorflow provides straightforward, generic functions to log metrics such as loss to a binary format. These code snippets are called within the Pytorch training modules.

Tensorboard displays these metrics via a web server, which in this project is hosted locally. It updates the view at regular intervals defined in the Pytorch training module.

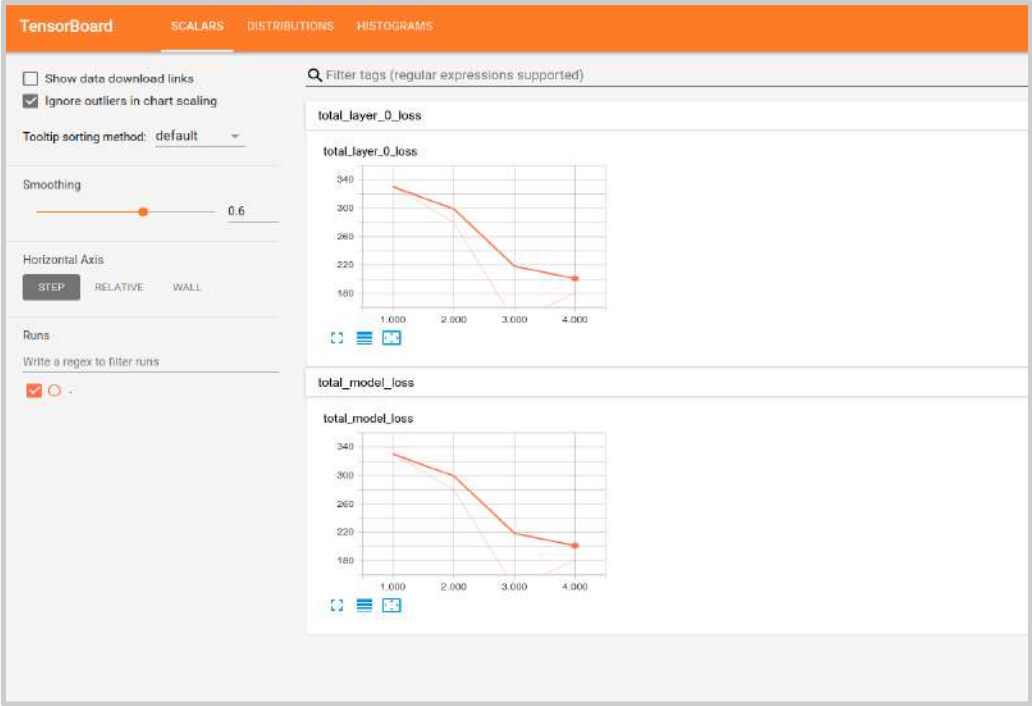

Figure 6: Tensorboard metrics displayed in the browser.

3.5 Skynet - An end-to-end parking lot occupancy application

Skynet is an extension to this project beyond merely detecting cars in a parking lot. It tracks the cars detected in each frame and generates statistics such as mean, minimum, and maximum car count in the input image set. In addition, it plots the car counts over the input images to visualize a parking lot capacity at a given time.

To separate Skynet from the core parking lot car detection Pytorch code, it is yet another Docker application based on the Vision Open CV one. It supports visualizations such as the plotting of parking lot capacity over time. It also writes out summary statistics and more detailed figures into comma-separated files. Conceivably,

these statistics resources can be converted into a server-driven API that provides clients with this information as JSON.

Figure 7: Skynet orchestrates the Pytorch detection given a set of pre-trained weights. It also does additional analysis on the detection results, such as time-series plotting, via a separate Docker application.

4. EXPERIMENTS

4.1 Design

The Pytorch application supports many different configurations of CNN models. This was necessary to support the exploration of various neural network layouts. Experiments included iterations on both the second and third iterations of the YOLO CNN, as well as modifications to both networks. The configuration files for the CNNs were based off the same ones used by Redmon for his work.

Each model iteration required training on the various datasets of parking lots. The accuracy did not improve beyond a hundred epochs of training, partly because of the limited dataset size. Moreover, because of the variable sizes of each dataset, training could take from a few days to a week in order to complete the session, even on an NVIDIA GPU. The training time quickly became the bottleneck in the research progress.

The initial goal was to benchmark how well the YOLO models could detect dozens of cars in a single parking lot image. The baseline tests were the versions two and three of the YOLO CNN. The layer definitions and hyperparameters were kept consistent. Moreover, both for these experiments and the subsequent iterations, the training sessions were based off the same weights that Redmon generated from training on COCO and ImageNet for his research.

The next phase was to explore various ways of improving the detection accuracy for objects in the extreme foreground and background of the parking lot. The version three model was extended to support a fourth detection layer corresponding to a different level of upsampling. Anchor boxes tailored to the dataset were calculated based on the same k-means approach that Redmon had used.

Table 1: CNN model architectures in this project and their corresponding hyperparameters.

Model	**YOLO v2**	**YOLO v3**	**YOLO Fourth Detection**
Number of Layers	31	106	118
Number of Detection Layers	1	3	4
Anchor Boxes	5	9	12
Learning Rate	0.001	0.001	0.001
Convolution Activation Functions	Leaky ReLU	Leaky ReLU	Leaky ReLU
Cost Optimizer	Stochastic Gradient Descent	Stochastic Gradient Descent	Stochastic Gradient Descent
Routing Layers	2	4	6
Filters Applied	1x1, 3x3	1x1, 3x3	1x1, 3x3

4.2 Datasets

There are two principle datasets used in these experiments. Each one differed in its effective labeling, resolution, or challenges. And each showcased the accuracy as well as limitations of this project's single-shot detection approach.

The CNR dataset contains generally favorable resolution images of parking lots with a fixed camera angle [10]. Cars may have different pose or suffer slight occlusion behind other cars, trees, or other obstacles. In addition, cars further back in the image are smaller with potentially fewer significant features extracted than cars up front in the image frame. However, the CNR dataset does not annotate every car in an image; in fact, a set of ground truth bounding box labels often miss several cars per parking lot image. Another drawback with the CNR dataset is that the bounding box annotations do not fully encompass the cars.

Figure 8: Vision annotation of a fixed camera angle image from the CNR dataset.

The CNR dataset was primarily intended for car classification. It included additional folders in which the cars in each image had been individually cropped into their own independent files. These cropped cars could be run through a classifier. This use case explains why these datasets did not thoroughly label every single car per parking lot image. Nonetheless, this dataset provided insights into the various experiments involving different models--namely, that the data quality can be just as important as the neural network architecture itself.

The other principle dataset, the CARPK dataset proved a highly reliable, high-resolution image set of cars in a parking lot collected not by fixed cameras but drones

[11]. It consists of over a thousand images of top-down parking lots. Because of the image capture angle, the model did not contend with challenging aspects such as differences in depth, pose, or occlusion. In addition, the dataset was very thoroughly annotated without a single car in an image that lacked a bounding box label.

Figure 9: Top-down drone images from the CARPK dataset. All cars are labeled with perfectly fitted bounding boxes.

4.3 Parking Lot Detection Results

The detection results are covered across all variations of models tested: YOLO version two, the latest YOLO version three, adjusted image sizing, and the fourth detection layer. In all the accuracy percentage charts, the accuracy was calculated based on the prediction error. I.e.,

$$\text{Accuracy} = (1 - |\text{predicted} - \text{actuall} / \text{actual}) \times 100$$

4.3.1 YOLO version two

The initial results with the version two YOLO models were promising to some degree, but they also highlighted areas for improvement. The detection threshold had to be quite low to detect cars, and by lowering the threshold so much, the number of false positive detections also increased. When detecting cars with a fixed camera angle of the parking lot, the detection accuracy was so low that the model does not seem a viable solution to detecting cars in a parking lot. The difficulties of pose, occlusion, and depth put the model at a distinct disadvantage to the human eye. In the case of the CARPK dataset, the detection accuracy was more accurate, though cases of false negatives would plague the results.

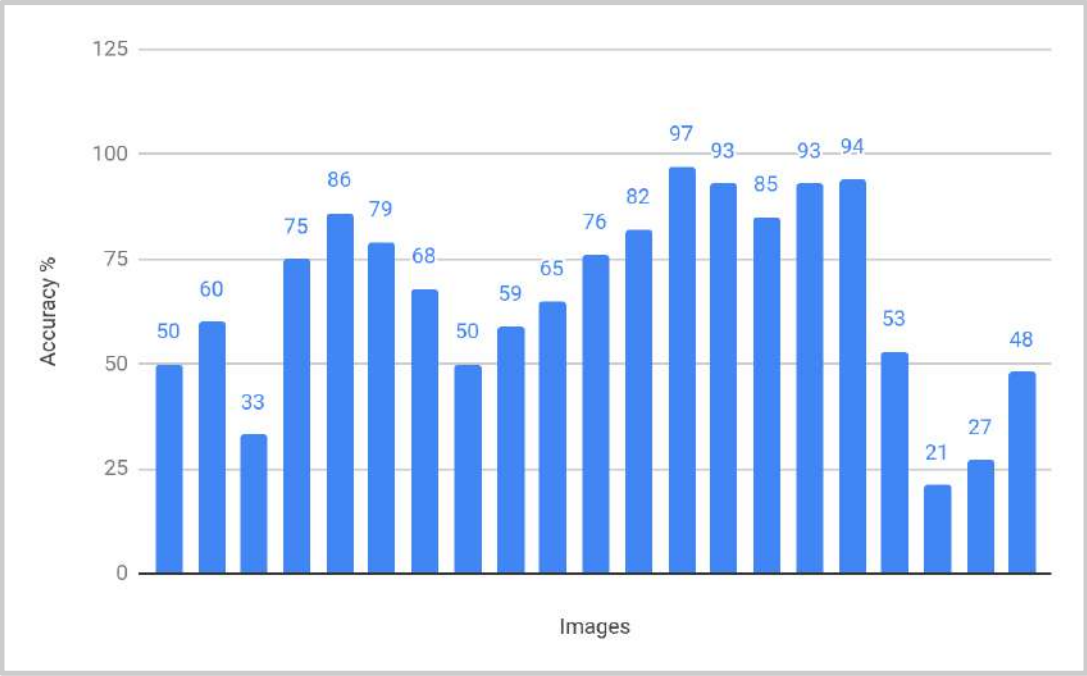

Figure 10: Car count accuracy percentage for fixed camera angle images based on the YOLO version two CNN.

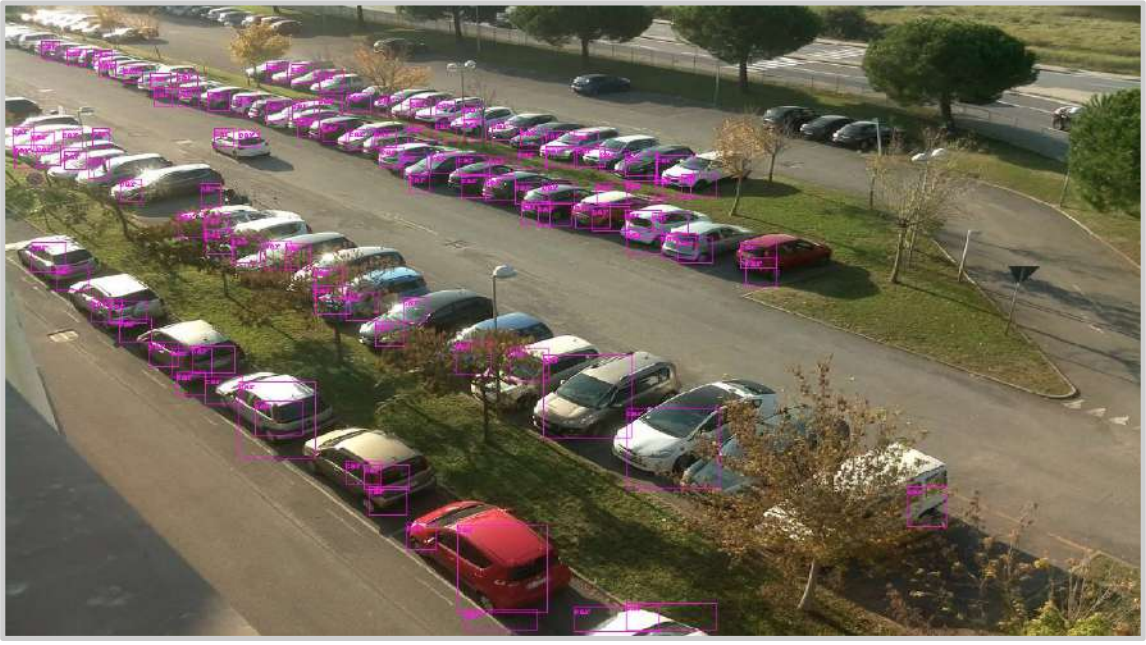

Figure 11: Poor detection with fixed camera angle. There are too many false positives with the threshold so low (5% in this case). At higher thresholds, few or none of the cars are correctly identified.

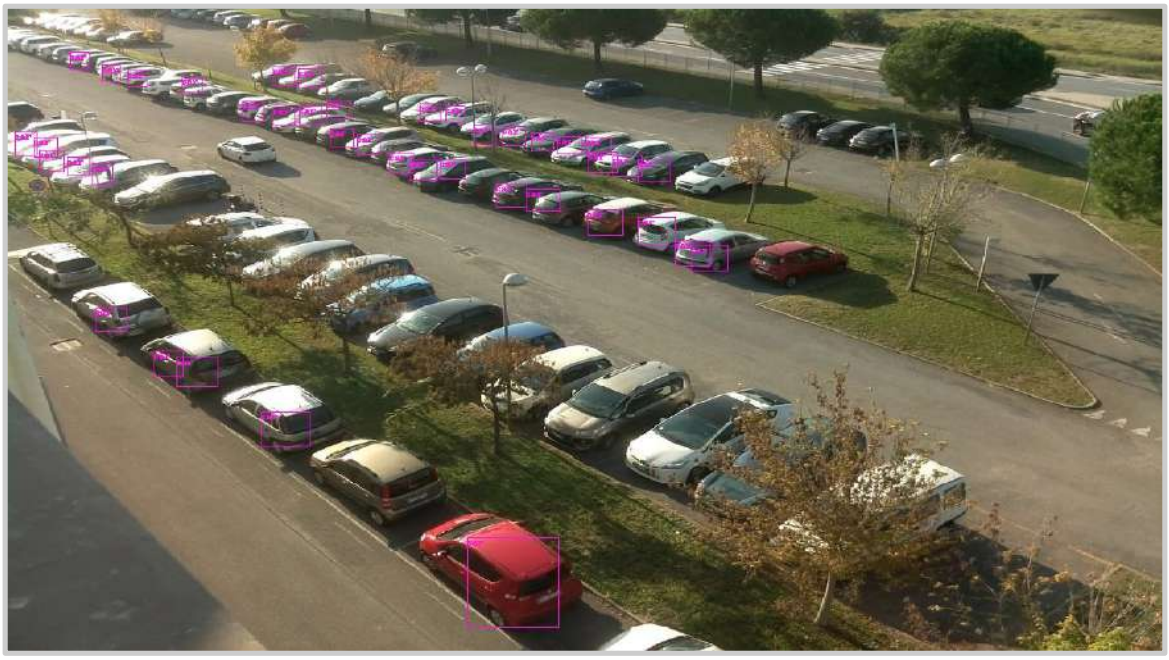

Figure 12: The same model and weights for the fixed camera angle, except with a slightly higher threshold (10%). The majority of the cars are not detected.

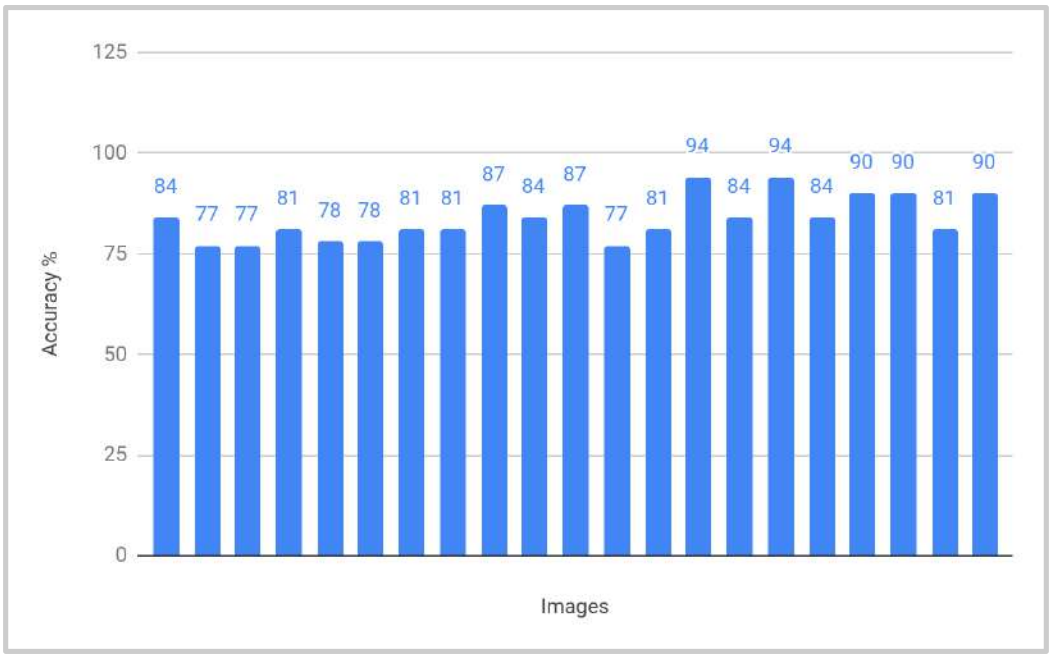

Figure 13: Car count accuracy percentage for top-down drone images based on the YOLO version two CNN.

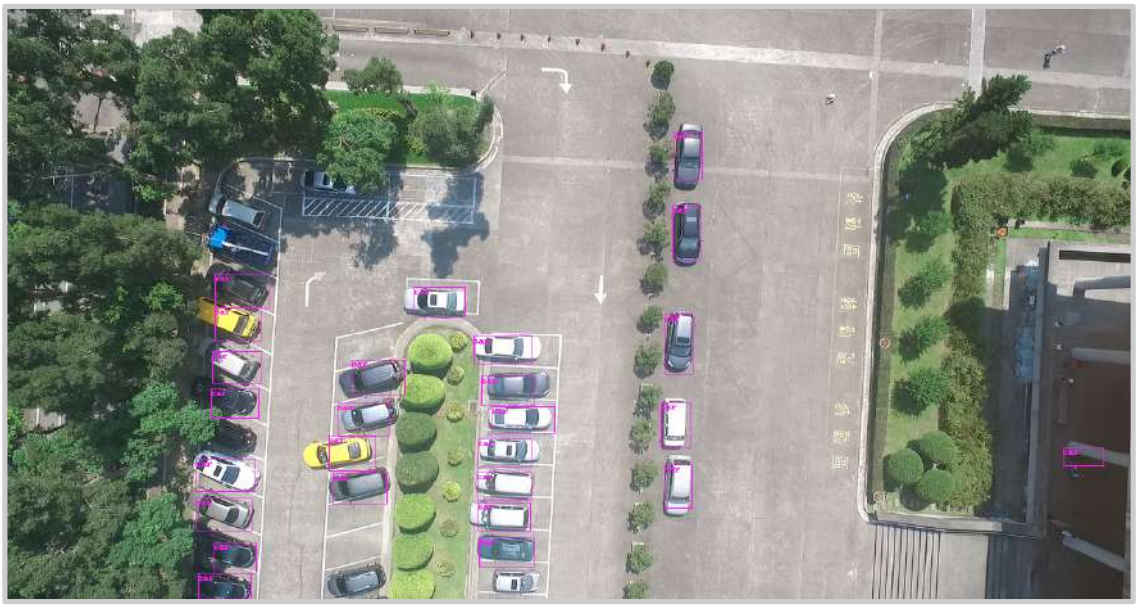

Figure 14: Detection of top-down drone perspective. The majority of the cars are detected, but there are a notable number of false negatives as well as unusual false positives.

4.3.2 YOLO version three

The results for parking lot detection using the latest version three models were much more promising. Overall, the confidence threshold was higher than that of the version two experiments, thus reducing the false positive counts while still having respectable car detection counts.

The fixed camera angle perspective showed significant improvement in detection accuracy at different depths and pose, although occlusion behind trees or other cars still proved challenging for the model. The CNN would consistently detect fewer cars in the image than there actually were. This could also be a product of the ground truth labeling on the original dataset, which did not place bounding boxes on every car. Nonetheless, the model would track well with the overall capacity of the parking lot as more cars parked in it.

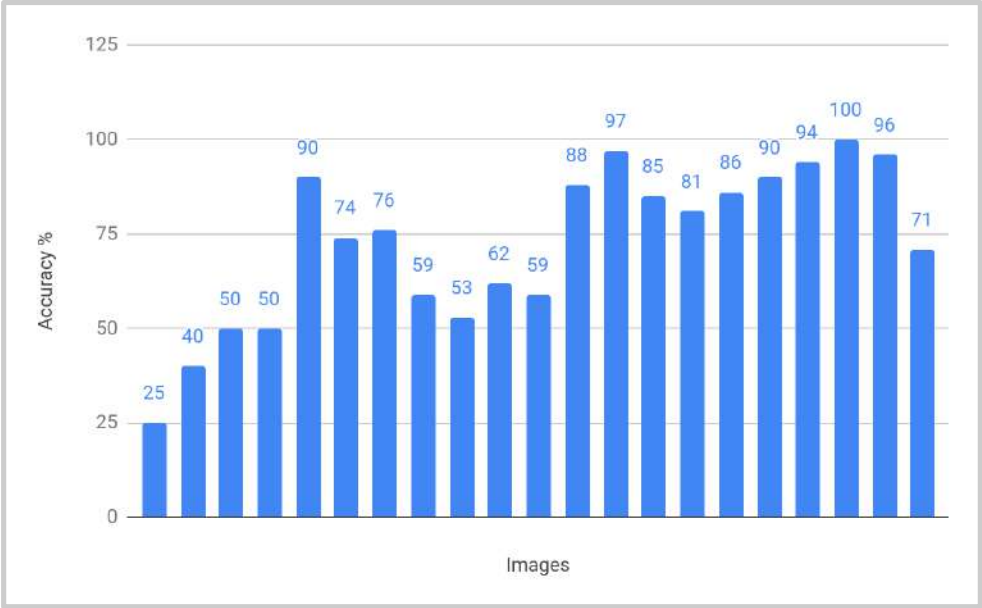

Figure 15: Car count accuracy percentage for fixed camera angle images based on the YOLO version three CNN.

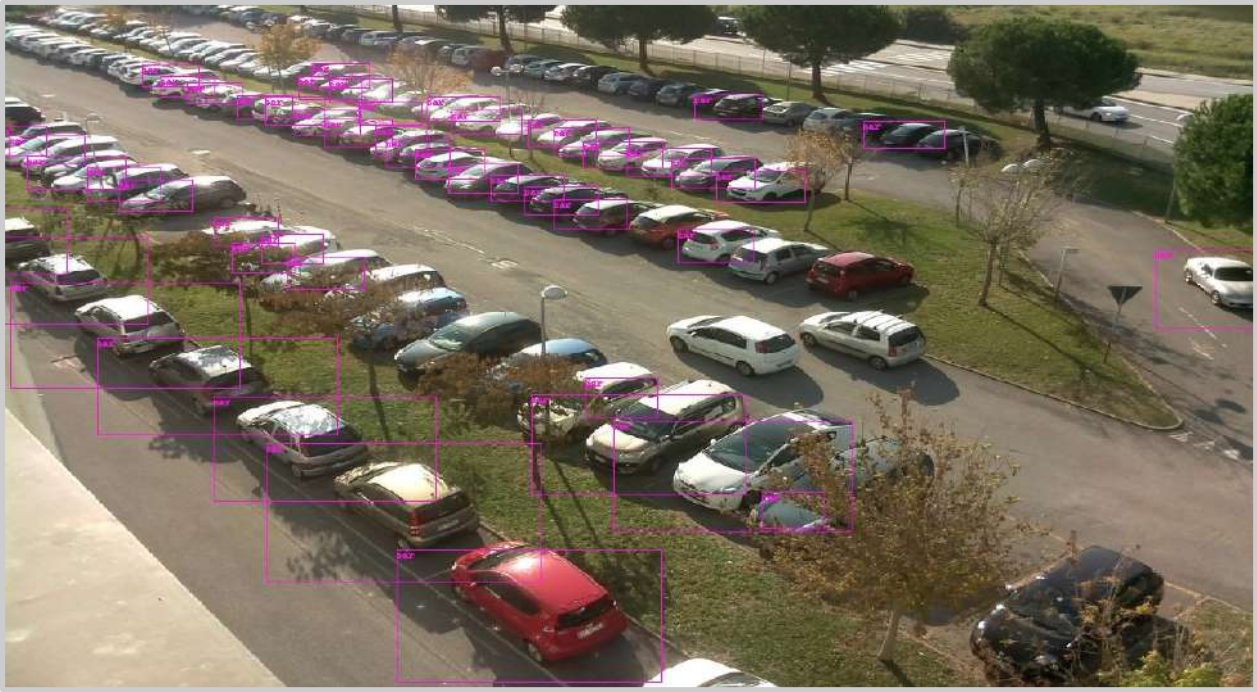

Figure 16: Fixed camera angle parking lot car detection with CNN based on YOLO version three. Unfortunately, the model did not detect the cars in the far background, since they were not labeled as cars in the ground truth.

In the case of the top-down parking lot images, the detection accuracy proved even better and ranged between 90-100% for parking lot images. This is most likely because of the consistent depth as well as minimal occlusion. The cars do have different poses, but the variability is not as high as the fixed camera angle parking lot images.

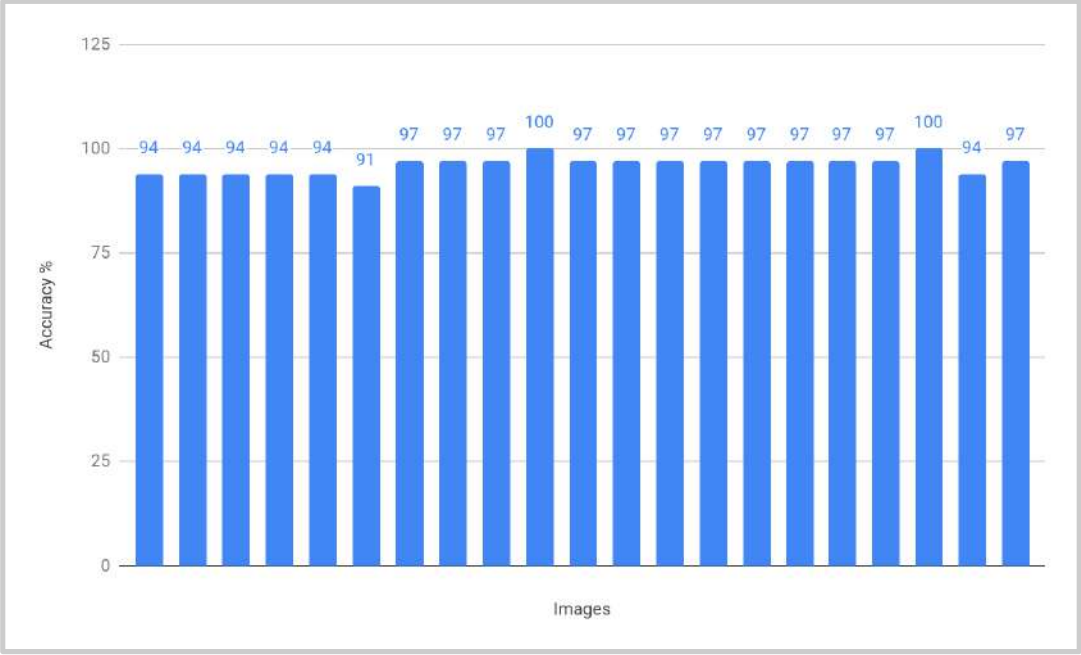

Figure 17: Near-perfect car count accuracy for top-down drone images based on the YOLO version three CNN.

Figure 18: Perfect accuracy of car detection, even with occlusions such as trees and different poses of car orientations.

4.3.3 The Fourth Detection Layer

In the case of the fourth detection layer, the CNN did improve as intended. Moreover, with the fourth detection layer, a more granular set of anchor boxes were created to suit detection at each upsampling level. This meant that the detection layer that had previously in version three been responsible for detecting the smallest of objects could now detect medium-small objects. And the layer responsible for detecting the largest of objects could delegate that responsibility to the detection layer in the first level of upsampling.

The feature extraction portion of the CNN was not extended like the authors of YOLO version three had done to support the three detection layers. Rather, the existing Darknet-53 convolutional layers were preserved, and a relatively "shallow" layer was routed to the corresponding fourth detection layer. Even without additional feature map extraction, accuracy can already be improved.

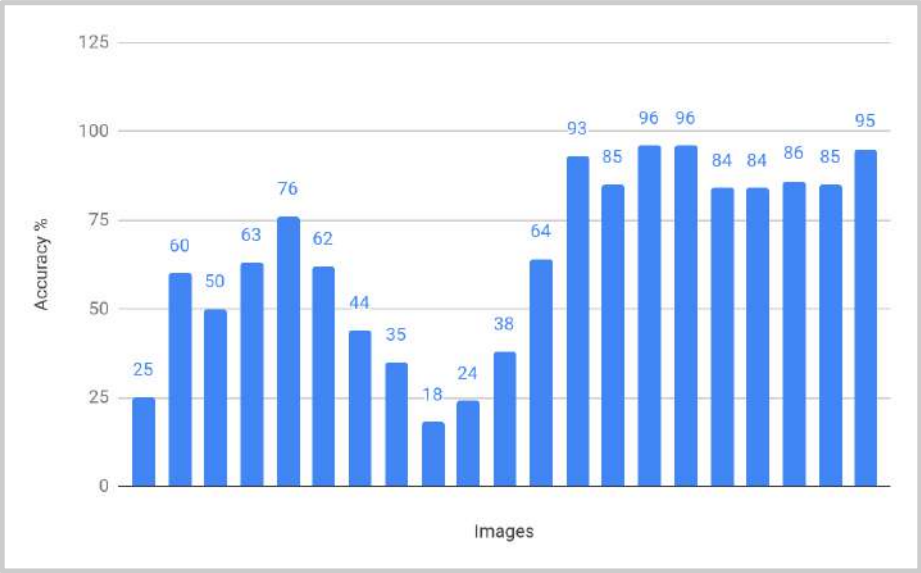

Figure 19: Car count accuracy with fourth detection layer. The seeming drop in accuracy in the middle is due to incomplete ground truth labeling of all cars in those images. In other words, the model detected cars that were not labeled during training.

Figure 20: Fourth detection layer helps the model detect even more cars.

Regarding top-down images, there was no detection accuracy increase, as expected. The appropriate upsampling was already accomplished in the current version three CNN of YOLO, since all the cars are going to share similarly sized anchor boxes. There are no cars in the forefront or background, as they are all similarly sized given the vantage point.

Figure 21: Top-down perspective with fourth detection layer.

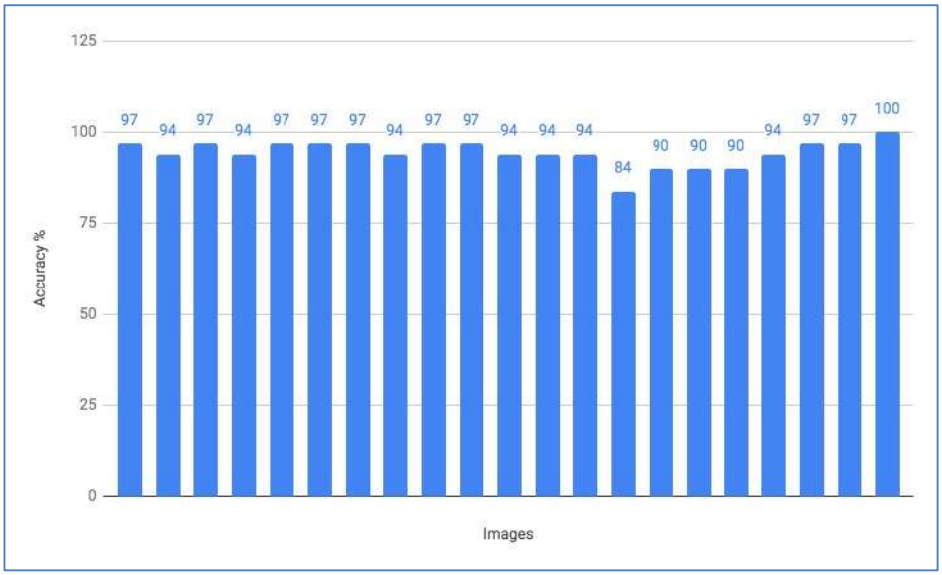

Figure 22: Detection accuracy of fourth detection layer on CARPK.

4.4 Analysis and Next Steps

The most recent YOLO version holds much promise. At the sacrifice of some additional runtime and memory overhead, the accuracy and confidence thresholds are much higher for parking lot car detection. Of course, the models based on the preceding version do have shorter runtimes, but they suffered in accuracy across all datasets. The previous version of YOLO was good at object detection of a handful of objects in an image, but when applied to densely packed objects like cars in a parking lot, YOLO version two simply does not handle well at all. The additional detection layers at different levels of upsampling that version three provides definitely help with the detection of cars at different depths.

The following are confusion matrices for the various models and the datasets they were trained and tested against. Note that the true positives are based on the ground truth labeling. With respect to counts alone, the models overall tracked progressively better with the ground truth car counts as more detection layers were added.

Table 2: Confusion Matrix for CNR car detection counts.

Model	**YOLO v2**		**YOLO v3**		**Fourth Detection Layer**	
	TP	TN	TP	TN	TP	TN
PP	26	2	24	4	27	1
PN	0	0	0	0	0	0

Table 3: Confusion Matrix for CARPK car detection counts.

Model	**YOLO v2**		**YOLO v3**		**Fourth Detection Layer**	
	TP	TN	TP	TN	TP	TN
PP	29	2	31	1	31	3
PN	0	0	0	0	0	0

If the parking lot images were captured by top-down perspectives, whether a drone or well-placed fixed camera, the latest version of YOLO should be sufficiently accurate to detect cars in the parking lot. The drawback with drones is the higher maintenance cost and initial sunk cost to purchase them. A fixed camera would be significantly less expensive, but its coverage of the parking lot would be limited. Several of these top-down fixed cameras would need to be installed throughout the parking lot, and the Skynet system would have to compile but not overlap the detection results from a series of different camera feeds. To minimize cost and overhead, a single camera should have a view of the entire parking lot, and the model should be optimized to account for the challenges behind an angled image capture of the lot.

The additional detection layer is the right direction for further iteration. Indeed, adding a fourth detection layer alone did improve car detection accuracy. It detected objects at yet another scale, and by doing so, it could detect more cars in the

foreground and background. The CNN even learned additional cars that were not included in the original ground truth labeling.

The latest iteration of the YOLO model added three layers of detection in light of having a much deeper feature extraction portion of the CNN. This enabled the authors to support detection at different levels of upsampling. Each detection layer would have tensor routing from a corresponding feature map in the extraction portion. Therefore, one next step to take is to extend the feature extraction portion of the CNN. Moreover, by having a deeper CNN, the novel feature map for the fourth detection layer would have even greater granularity--perhaps exactly what the model needs to detect the furthest background cars in the parking lot images.

4.5 Skynet Performance

Skynet provides data-driven insights into a parking lot's occupancy levels. In the context of this use case, it excels in tracking rises and falls in parking. A parking lot manager or data analyst would utilize the time-series plots of car counts to track when the lot capacity rises and falls. If it is a business center, there could be valuable insights gained by knowing when the parking lot is full versus nearly empty. Near-perfect accuracy in this use case would not be necessary, so a fixed camera taking periodic images of the parking lot should be sufficient for tracking car counts.

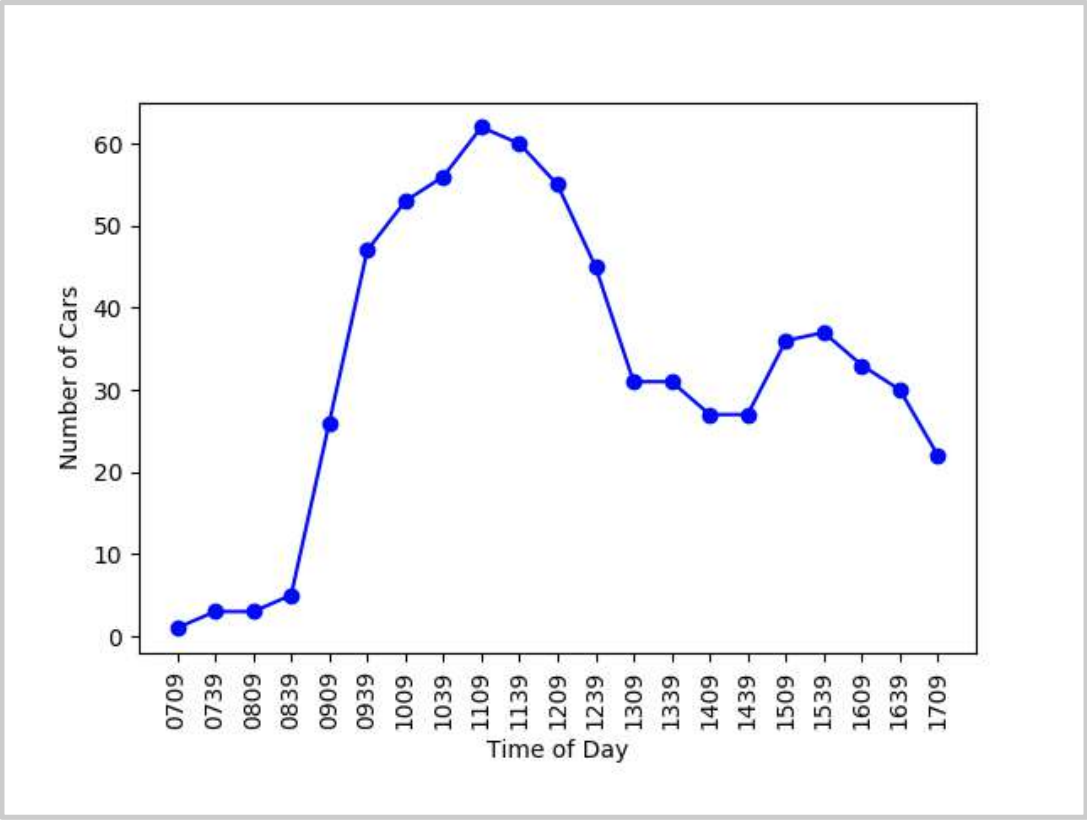

Figure 23: Skynet's output time-series plot of car counts for the fixed camera angle perspective.

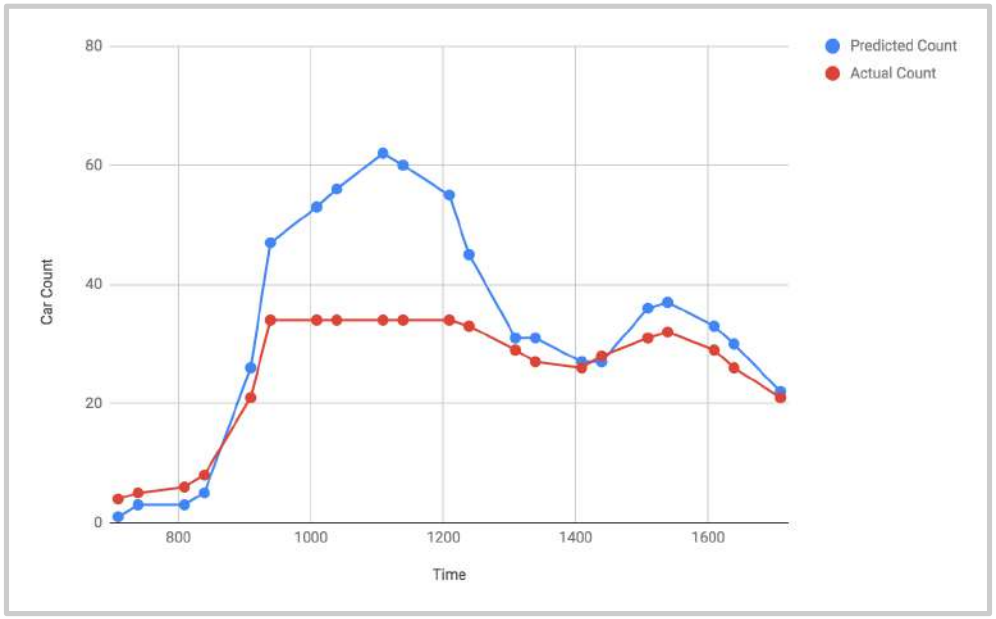

Figure 24: Predicted car counts and ground truth car counts over time at a fixed camera angle. (Note that the "ground truth" here does not account for all cars in the lot).

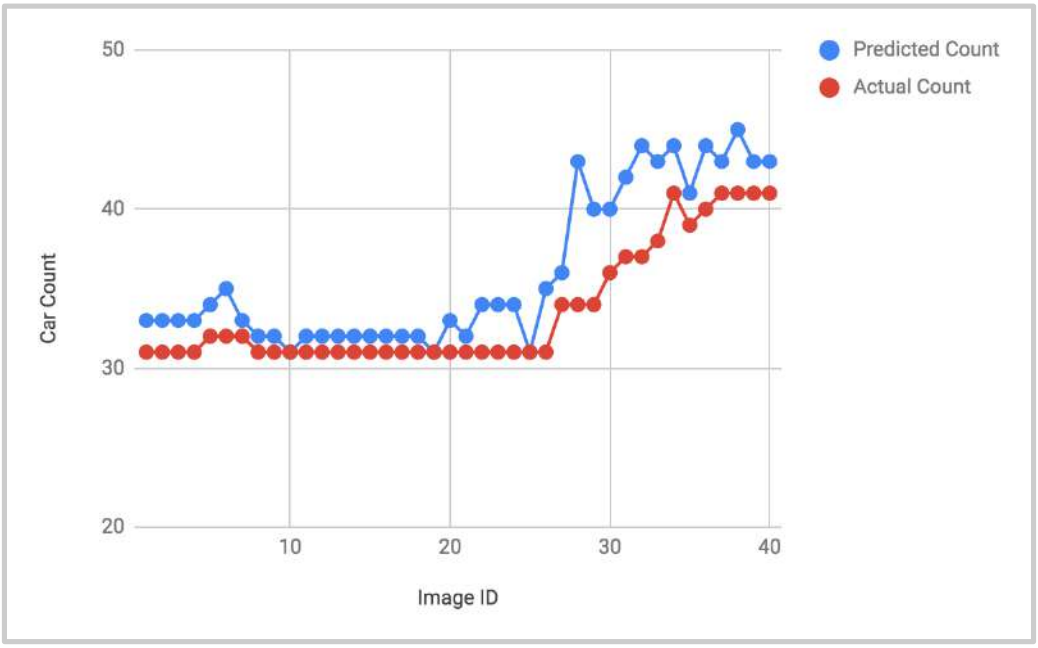

Figure 25: Predicted car counts and ground truth car counts over time with a top-down perspective.

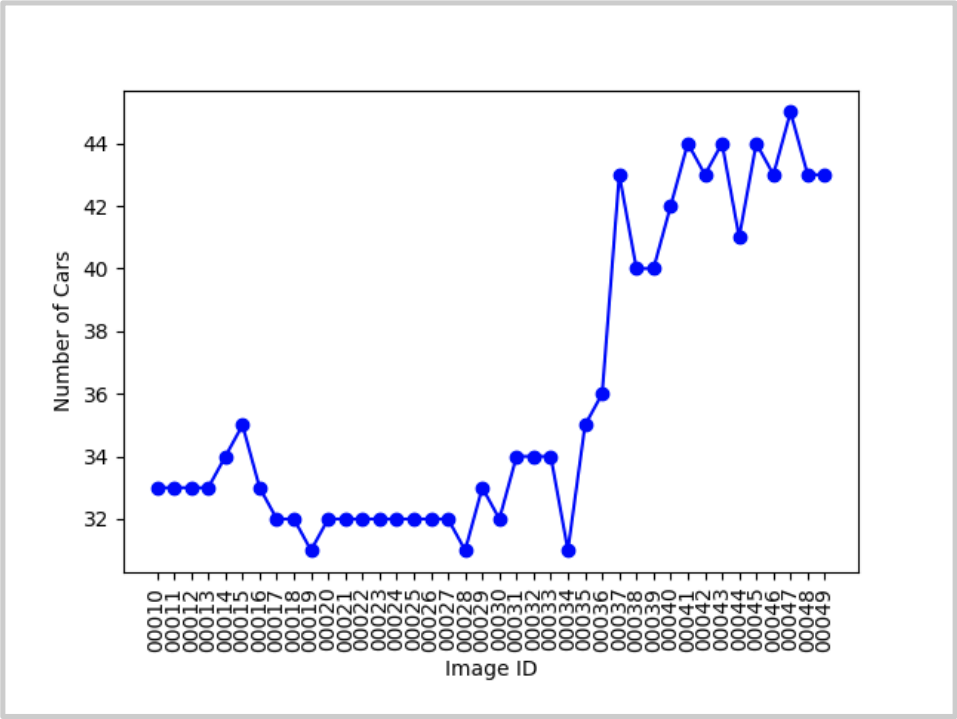

Figure 26: Skynet's output time-series plot of car counts for the top-down camera perspective.

5. CONCLUSIONS

This project has shown as a proof of concept the effectiveness of a single-shot detector CNN such as YOLO can be a baseline to evolve a model intended to detect cars in a parking lot. It highlights the effectiveness of containerization of responsibilities during training, where image processing and compilation occurs concisely and efficiently in a separate service. Finally, this project shows the application of car detection as a means to track throughput and capacity in parking lots over time without the expenses of hardware such as GPUs or labor of parking lot managers patrolling at regular intervals.

The level of accuracy in detecting cars in a parking lot depends greatly on the training set and by extensive the method of data collection. Fixed camera angles present greater challenges in object detection than top-down drone vantages. For near-perfect accuracy, drone footage should be captured to minimize the effects of pose, depth, and occlusion that the CNN would be challenged to work past. However, if the use case was to track general throughput and capacity at regular time intervals, the model could make fairly accurate detections of cars in the parking lot, thus still providing valuable data and insight into the parking lot.

There are several next steps in the research process to work towards higher detection accuracy, particularly with regards to addressing the challenges highlighted by the fixed camera angle experiments. Namely, the next leap forward for this CNN

setup is deepening the feature extraction portion even further than the version three did. This would require extensive research and retraining on standard object detection datasets such as VOC and COCO along with classifier-specific ones such as ImageNet. The end result will be a slower runtime, but ideally, there will be more comprehensive feature maps for the fourth (or even fifth) detection layer to detect either really large or small cars in the foreground and background, respectively.

There are also next steps in the maturation of the practical application behind this project's parking lot car detector. Skynet can be modified to run continuously on a server that accepted image inputs. This would remove a significant runtime bottleneck, the instantiation of the entire CNN upon every execution of Skynet. Further, Skynet can further be augmented with an option to read input from a video stream, which would convert the input at regular intervals into a tensor for the neural network to consume.

Object detection as a field will continue to evolve. There may be a single-shot detector with higher accuracy than the YOLO-based ones or a different approach to representing car locations' ground truth, and perhaps those innovations will be the next leaps forward in detecting cars in a parking lot.

LIST OF REFERENCES

[1] D. Bullock, et al, "A Neural Network for Image-based Vehicle Detection," Pergamon Press Ltd., 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0968090X9390025B

[2] Y. LeCun et al, "Object recognition with gradient-based learning," Proc. IEEE, 1998. [Online]. Available: http://yann.lecun.com/exdb/publis/index.html#lecun-98.

[3] J. Redmon, et al, "You Only Look Once: Unified, Real-Time Object Detection," arXiv, May 2016. [Online]. Available: https://arxiv.org/pdf/1506.02640.pdf

[4] R. Girshick, "Fast R-CNN," Microsoft Research, 2015. [Online]. Available: https://arxiv.org/pdf/1504.08083.pdf.

[5] I. Goodfellow et al, Deep Learning, MIT Press, 2016, pp. 3. [Online]. Available: http://www.deeplearningbook.org.

[6] J. Redmon, A. Farhadi, "YOLO9000: better, faster, strong," in Conf. on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017. [Online]. Available: https://arxiv.org/pdf/1612.08242.pdf.

[7] TY Lin et al, "Microsoft COCO: Common Objects in Context," Microsoft Research, Feb 2015. [Online]. Available: https://arxiv.org/pdf/1405.0312.pdf.

[8] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv, 2018. [Online]. Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf

[9] A. Kathuria, "What's new in YOLO v3?," 2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[10] G. Amato et al, "A Dataset for Visual Occupancy Detection of Parking Lots," 2015. [Online]. Available: http://cnrpark.it/

[11] M. Hsieh, Y. Lin, W. Hsu, "Drone-based Object Counting by Spatially Regularized Regional Proposal Networks," ICCV 2017. [Online]. Available: https://arxiv.org/abs/1707.05972