Real-time image-based parking occupancy detection and automatic parking slot deliniation using deep learning: A tutorial

Debaditya Acharya* and Kourosh Khoshelham**

deb.acharya@rmit.edu.au and k.khoshelham@unimelb.edu.au

*Department of Manufacturing, Materials and Mechatronics, RMIT University, Victoria, Australia, 3000

**Department of Infrastructure Engineering, The University of Melbourne, Parkville, Victoria, Australia, 3010

Abstract

This chapter describes a simple method for parking occupancy detection and an automatic parking slot delineation method using CCTV images. These methods will be presented in the form of MATLAB tutorials with code snippets to allow the interested reader to implement the method and obtain results on a sample dataset. The first tutorial will involve fine-tuning a pre-trained deep neural network for vehicle detection in a sequence of CCTV camera images to determine the occupancy of the parking spaces. In the second tutorial, we perform spatio-temporal analysis of the detections made by a state-of-the-art deep learning object detector (Faster-RCNN) for automatic parking slot delineation. The dataset and the code is made public at https://github.com/DebadityaRMIT/Parking.

Keywords: Automatic parking slot delineation; Real-time parking occupancy detection; CCTV cameras; Deep learning; Tutorial

1 Introduction

Smart parking technologies are an indispensable part of urbanisation to facilitate a congestion-free traffic flow. The advantages include less emissions and less waiting periods for drivers. These facts have motivated the research community to develop smart parking technologies, and real-time parking occupancy detection has become one of the key elements for the design of tomorrow's smart cities. While different sensor technologies exist for occupancy detection, they are usually expensive and require regular maintenance. The vision-based methods for parking occupancy detection provide an economical yet reliable alternative to the costly counter-based and sensor-based counterparts.

The rest of the section describes the motivation, related works and challenges of parking occupancy detection and automatic delineation of parking spaces using deep learning. Section 2 provides a brief overview on the definitions and the theory of machine learning in general. This is followed by introducing the deep learning architectures used for occupancy classification and parking space detection. Additionally, the details of the dataset is presented in this section. Section 3 presents the first tutorial where a deep learning image classifier is fine-tuning to perform parking occupancy detection. Also, several deep learning architectures are compared based on their performances and runtimes. Section 4 presents the second tutorial where a deep learning object detector is used to perform automatic parking slot delineation. Subsequently, the model performance is improved by using spatio-temporal and statistical analysis of the detections. Section 5 concludes the observations of the tutorial.

1.1 Parking occupancy detection using vision-based methods

The vision-based methods consist of cheap cameras to cover the whole parking area. The closed-circuit television (CCTV) cameras used for surveillance can also be used for occupancy detection. The images taken from these cameras are subsequently processed to provide the occupancy information. For a comprehensive review of other sensors for parking occupancy detection, the reader is redirected to Chapter 10.

There are a two challenges that limit the broad applicability of the vision-based methods. The first challenge is the low detection accuracy of vision-based methods as compared to the count-based or sensor-based methods (Amato et al., 2017). This lack of precision for vision-based methods can be linked to many factors, such as diverse appearances of the vehicles, environmental factors such as shadows, reflections and haze (due to sun and rain), occlusion by other vehicles (or other objects) in the line-of-view and distortion due to the oblique view of the cameras.

The second challenge is the delineation of the parking slots in the images. This delineation is not necessary for counter-based methods as the number of parking slots are fixed, and for sensor-based methods each parking slot is physically visited once to install the sensor. A parking area can be covered by several cameras, and perhaps hundreds of cameras for on-street parking. Manually labelling each parking slot is a laborious task. Moreover, the parking boundaries can change from time-to-time. Another related challenge arises in areas where the parking slots are not marked, especially in developing countries, such as India. An equally important challenge is the detection of improperly or illegally parked vehicles, e.g., when a vehicle is parked on the markings between two spaces or when several cars are parked in a large parking space designated for buses. Therefore, automatic ways to delineate the parking slot boundaries (or parking zones for unmarked parking areas) is highly desirable for smart parking solutions.

Robust image representations help in the accurate detection of the parking slots, and the recent deep learning methods have showed promising results in this aspect (Amato et al., 2017; Acharya et al., 2018). Therefore, in the next sub-section we discuss the background of parking occupancy detection and automatic parking slot delineation using deep learning. The relevant definitions and theory related to the understanding of deep learning can be found in Section 2.

1.1.1 Parking occupancy detection using deep learning

Parking occupancy detection is usually formulated as an image classification problem, where each image is either empty or occupied by a vehicle. Image classification follows a standard pipeline of feature extraction, and comparing the extracted feature with features belonging to different classes. In the past these images features were hand-crafted (or engineered) and showed poor performance for "unseen" examples. For instance, De Almeida et al. (2015) generated a robust dataset containing parking of different parking slots and used hand-crafted textural descriptors, such as Local Phase Quantization (LPQ) to perform parking occupancy detection. They report an accuracy of over 99% while validating with the images from the same dataset, and around 89% while testing with images of a different dataset. Other such examples of parking detection with hand-crafted images features includes the works of True (2007), Ichihashi et al. (2009) and del Postigo et al. (2015).

With the advances in machine learning algorithms, especially the recent deep learning and convolutional neural networks (CNNs), the feature extraction from the images have been automated, and state-of-the-art accuracies in image classification are reported. In the context of parking occupancy classification with CNNs, several past works (Valipour et al., 2016; Amato et al., 2016, 2017; Acharya et al., 2018) exist that report excellent performance. These studies achieve greater than 99% accuracy for the task of occupancy detection, when being validated with unseen samples from the "same" dataset. The performance of the models on unseen samples from a "different" dataset is around $90 - 96\%$. This improvement in the "generalising" ability (or adaptability to unseen examples) of the CNNs demonstrates the robustness of the learnt features compared to the hard-crafted features.

Parking occupancy classification using CNNs can be done in two ways. The first approach involves fine-tuning a "pre-trained" CNN, like the approaches of Valipour et al. (2016); Amato et al. (2016, 2017). A pre-trained CNN contains the weights of a network that is trained on millions of images and is suitable for classification of several hundreds of classes. These pre-trained CNNs usually require weeks to train on graphics processing units (GPU) and perhaps can take years to train on normal CPUs. Therefore, using pre-trained networks saves the effort of training a network from scratch and can easily be "adaptable" to a particular problem by "transfer" learning.

However, the pre-trained networks are not suitable for a two class classification task, like parking occupancy, where a class is either "empty" or "occupied". Therefore, to adapt to the classification problem, we fine-tune the CNN by with some example images (often a couple of thousands) to adapt the weights of the network for the specific classification task. This is performed by back-propagating the loss using an objective loss function. For a classification task, cross-categorical loss is usually used. However, one of the disadvantages of this method is the computational power required for the fine-tuning process. It can take several minutes on GPU or several hours for fine-tuning a network with a couple of thousand images.

The second approach involves extraction of the image features using a pre-trained CNN and then performing classification using support vector machines (SVM), like the approach followed by Acharya et al. (2018). SVMs are a kind of machine learning algorithms that project the features into higher dimensional feature spaces to find the optimum hyperplane that separates the classes. This approach of training is faster as compared to fine-tuning the CNNs and the whole training can be performed with CPU within minutes. This reduction in the training time is due to the elimination of the back-propagation, and because the training time of SVM is considerably less (few seconds for couple of thousand samples). For a CPU-friendly MATLAB tutorial of this CNN + SVM method please follow our previous work (Acharya et al., 2018) which is available at `https://github.com/debaditya-unimelb/real-time-car-parking-occupancy`.

We present the fine-tuning approaches in Section 3. Additionally, we compare the performance in terms of precision and computational times of different CNN architectures. This information will help the audience to decide the trade-off between performance and computational need to check the suitability for real-time applications.

1.1.2 Delineation of the parking slots using deep learning

Delineation of parking slots (knowing the locations of the parking slots) is required prior to accurate parking occupancy detection. Currently, delineation is performed manually (Cai et al., 2019; Khan et al., 2019; Sairam et al., 2020; Paidi et al., 2020). These studies use deep learning-based object detectors such as Faster-RCNN (Ren et al., 2015) to detect the vehicles, subsequently, compare the location of the detections with the manually delineated parking slots to estimate the occupancy.

To eliminate the manual delineation of parking slots, some researchers (Ahmad et al., 2019; Ding and Yang, 2019) have used automatic object detectors to detect the vacant and the occupied parking slots directly in the images using Faster-RCNN, Mask-RCNN (He et al., 2017), Retina-Net (Lin et al., 2017) and YOLO (Redmon and Farhadi, 2018). However, such methods do not take into account of the actual number of the parking slots available in the area, rather report number of detections (both empty and occupied) made by the object detector. Because these object detectors always miss some of the detections, the parking estimates might not be practical for all applications. Moreover, object detection pipeline involves localising the objects in the images. This particular step is computationally expensive as the system has to process thousands of proposals to identify the correct detection. For instance, one forward pass through ResNet50 (He et al., 2016) for image classification needs approximately 0.1 seconds on CPU, and processing the same image with the same ResNet50 in Faster-RCNN framework on CPU needs around 12 seconds in MATLAB.

Differently, there are few other approaches that perform automatic parking slot delineation. Vítek and Melničuk (2018) propose an automatic method of delineation of the parking spaces in a multi-camera framework using histogram of oriented gradients (HOG) and a sliding window to perform vehicle space classification using SVM. The authors do not report the detection accuracies, and in addition the HOG features are susceptible to lighting changes. Nieto et al. (2018) use satellite images to manually register the parking slots and use an input of the number of parking slots to automatically delineate the parking slots. However, the method is not completely automatic as it needs input from a skilled operator to actually count the number of parking slots and for entering three common points (ground control points).

Another research direction of automatic delineation of parking slots, can be found in the works of Jung et al. (2009); Suhr and Jung (2013); Zhang et al. (2018), however they are vehicle centric and rely on the cameras installed in the vehicles to detect the parking slot marking automatically. However, such methods have not been applied yet for parking slot delineation from fixed camera, and is a future research direction.

1.2 Contributions

The following are the main contributions of the chapter:

- A pre-trained CNN is fine-tuned with PKLot dataset for parking occupancy detection, and is tested with Barry Street dataset. Different CNN architectures are compared in terms of their accuracies and run-times to demonstrate the suitability for real-time applications on both CPU and GPU.

- A novel off-line method is proposed for automatic parking slot delineation by performing spatio-temporal analysis of the detected vehicles using state-of-the-art object detector Faster-RCNN (Ren et al., 2015). Compared to the previous approaches, the proposed pipeline eliminates the requirement of localising the objects on-line, and generates parking slots that would have been otherwise be manually delineated. This off-line method eliminates the requirement of localising the vehicles, and reduces the object detection problem to an image classification problem that significantly reduces the computational requirements. We present the approach and the related tutorial in Section 4 to prove the concept.

Section 2 introduces the prerequisites of the tutorials, such as definitions, related theory, software toolboxes and subsequently the dataset. Section 3 demonstrates fine-tuning a pre-trained network, and compares different network architectures in terms of achievable accuracies with the sample dataset and the run-times. Section 4 demonstrates the automatic parking slot delineation using vehicle detector. Section 5 concludes the finding of the tutorial.

2 Prerequisites

In this section, we start by presenting the theory and definitions that are used throughout the tutorials. For completeness, we have repeated some of the related theory already presented at the beginning of the chapter.

Subsequently, we introduce the software and toolboxes required running the tutorials. Lastly, we describe the dataset that we used for the tutorials, which we made public.

2.1 Definitions and theory

Machine learning are a set of computer algorithms that build a mathematical model based on a training data, which can be used to make predictions or decisions of unseen data based on learnt representation of the features. Neural networks and Deep learning are included in this class of algorithms. Machine learning can be broadly classified as supervised machine learning, un-supervised machine learning and reinforcement learning. In this chapter we use supervised machine learning approach, where we provide the samples of training data with their respective labels (ground truth annotations).

Neural networks are networks inspired from the biological neural networks of the brain and are composed of artificial neurons (containing weights and biases) to perform many complex operations, such as classification. These networks learn a feature representation automatically and eliminates the manual feature selection process. These networks are composed of connected layers, where each layer contains many neurons and the training process involves back-propagation by minimising an objective loss function.

Deep learning refers to the machine learning algorithms which deal with neural networks that contains many layers of neurons. Adding increased depth to the neural networks provide the networks ability to perceive complex operations that are not possible by their "shallow" counterparts. Recently, deep convolutional neural networks have achieved the state-of-the-art accuracies in classification and object recognition task, sometimes even surpassed the human ability.

CNNs consists of many layers of image "convolutions" containing learnable kernels that convolute the whole image and hence create a hierarchy of increasing complex image features. These image features are learnt automatically thereby eliminating the need of fragile hand-engineered image features. These learnt image features are unique representations of the images, and are often used for image classification and object detection. In this chapter we have used many CNN architectures, v.i.z. AlexNet (Krizhevsky et al., 2012), GoogleNet (Szegedy et al., 2015), MobileNet v2 (Sandler et al., 2018), ResNet50 (He et al., 2016), SqueezeNet (Iandola et al., 2016) and VGG-16 (Simonyan and Zisserman, 2014). We selected these networks to demonstrate the effects of network run-time and the achievable accuracy with limited training data. For the scope of the tutorials we only explain ResNet50 in the following lines.

Pre-trained networks are trained with millions of images of publicly available image datasets containing different classes. The training can take a couple of weeks, depending on the network architecture and training data. To save the immense training effort before using deep networks, these pre-trained models are often used for other tasks by fine-tuning them.

Fine-tuning refers to the process of training a pre-trained network with relatively small examples to adapt to a different task. This is achieved by the process called transfer learning.

Transfer learning is the process of applying learnt knowledge in one domain to solve a different but related problem. For instance, a pre-trained network trained to perform image classification of 1000s of classes that contains vehicles, cats, dogs can be used to differentiate between types of insects, a task that it was not trained to do.

Training data refers to the samples that are used during training the model.

Test data refers to the sample that needs to be classified.

Over-fitting refers to a condition where the classification accuracy of the trained model is excellent on the training data, but its performance is poor on test data. Therefore, during the training process, a validation data is generated as a subset of training data which is used for evaluating the accuracy of the trained model independently.

Loss function also known as cost function or objective loss function that we try to minimise during the learning process by back-propagating. The simplest form of this function is the difference between observed and the actual values. For classification problems, a cross-entropy loss function is often used.

Back-propagation refers to the process of propagating the gradients from output to input to update the weights of the network for the intended operation. The back-propagation is achieved by using an optimiser and the weights of the neurons are updated using a hyper-parameter called learning rate.

Optimisers are the iterative methods of optimising the loss function by calculating the gradients (or rate of change) and they connect the weights of the individual neurons with the loss function with the help of learning rate. The objective here is to reach the global minima or the minimum possible value of the loss function. The most commonly used optimiser is stochastic gradient descent and its variants.

Learning rate refers to the rate of update of the gradients for each individual neuron throughout the network. A ~~higher learning rate might help to reach the minima fast, but can lead to a local minima~~. The target of the optimisation is to reach the global minima, and hence learning rate is one of the key training parameters of a neural network.

Epoch refers to the training interval when the neural network is trained with one complete dataset. Usually, a neural network needs to be trained on several epochs of data before it converges to an optimal solution.

Learning curve refers to the graphical representation of the model learning with the amount of training data. This curve often contains the training loss, validation loss, training accuracy and validation accuracy, and is used to identify whether the model is over-fitting.

ResNet50. One of the challenges of deep CNNs and deep learning in general is the problem of vanishing gradients, where the gradients during the back-propagation becomes infinitely small for the shallow layers. To address this challenges, residual networks have been proposed in the literature, and ResNet50 is a variant of a deep residual network, as shown in Figure 1.

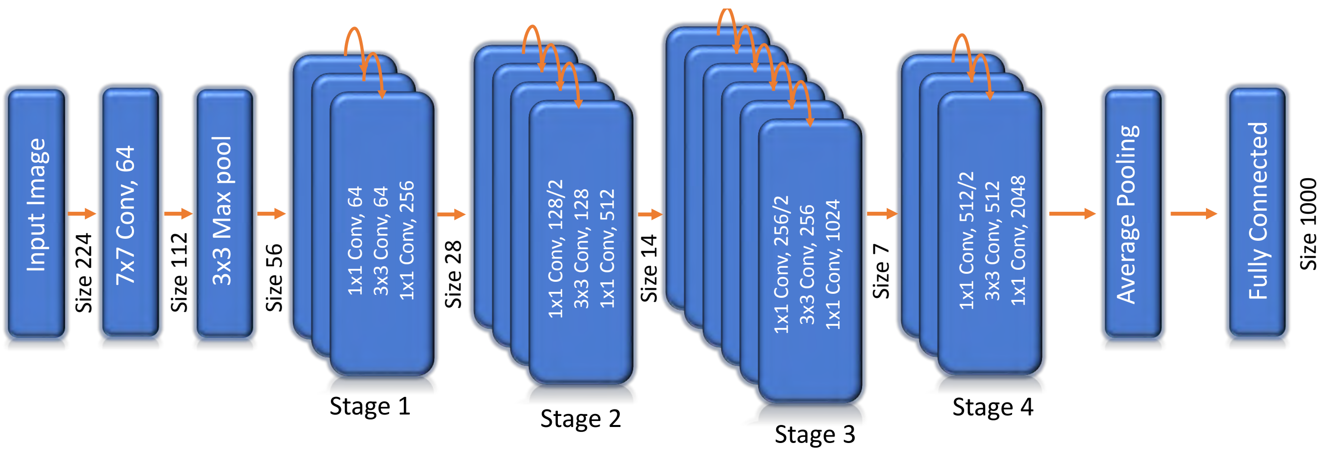

Figure 1: The architecture of ResNet50 containing 50 layers. Stages 1-4 contains blocks of length 3, 4, 6 and 3 respectively, where each block consists of three convolutional layers.

The main innovation in this architecture is the presence of the "skip connections" or the identity mapping (orange curved lines on the top of blocks), where the output of a previous block is connected to the next block. This skip connection, helps to alleviate the vanishing gradient problem by skipping one or more layers. The result is a deep network with the state-of-the-art accuracy in image classification. The input to the network is an image of 224 x 244 pixels and the output is a 1000-dimensional feature vector.

Faster-RCNN (Ren et al., 2015)) is an object detection algorithm that performs the task of localising objects on the images and its subsequent classification. This algorithms needs a CNN as its backbone for operation, and is shown in Figure 2. The CNN extracts features and generates a feature map using "*activation_40_relu*" layer. This feature map serves as the input to the Region Proposal Network (RPN) that generates the object proposals. The RPN searches for potential objects throughout the image at regularly gridded anchor points using anchor boxes of different shapes and sizes. The object proposals from RPN are used to create Region of Interest (ROI) pooling on the feature map to extract the features of the potential objects. The final bounding boxes and the classes are predicted using a bounding box regressor and a Softmax classifier.

t-SNE algorithm is a non-linear dimensionality reduction algorithms that is used for visualisation of the higher dimensional data, and is often used to assess the quality of the features for the task at hand.

2.2 MATLAB and toolboxes

The tutorials are intended to run on MATLAB 2020a, although the code can run in MATLAB versions higher than 2018a. Additional toolboxes might be required to run the experiments that include computer vision toolbox, statistics and machine learning toolbox, deep learning toolbox, signal processing toolbox and automated driving toolbox. For running the live script smoothly, please ensure to increase the java heap memory of MATLAB, as demonstrated at the start of the live script.

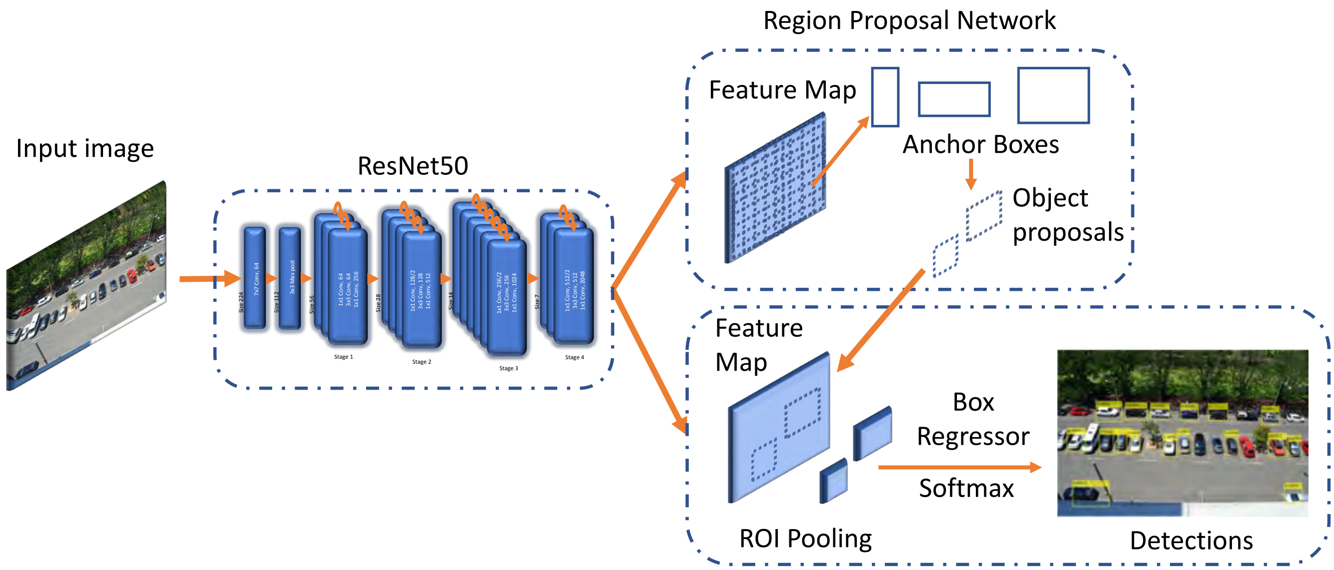

Figure 2: The architecture of Faster-RCNN containing (ResNet50 backbone).

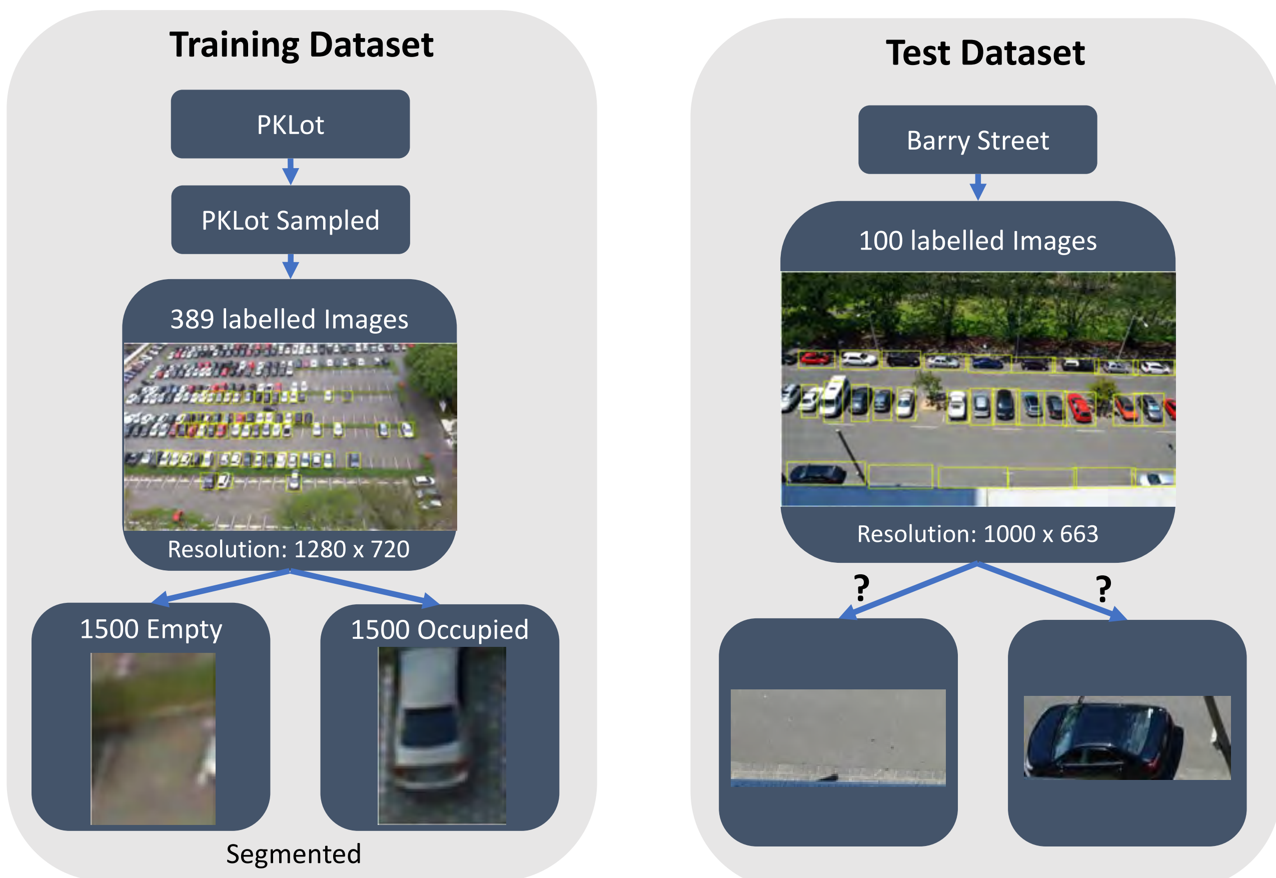

Figure 3: The training and the test datasets. The CNNs are trained with PKLot dataset and are tested entirely on Barry Street dataset.

2.3 Dataset description

The code is made available at Github (https://github.com/DebadityaRMIT/Parking). The provided file is in form of a MATLAB live script (.mlx file) that contains all the outputs embedded within the script. Therefore, the user can view the results of the experiment without running the experiments. By changing the default configurations, an interested reader can run the tutorials on-line. For running the experiments, the data (including the code) can be downloaded from Figshare (https://rmit.figshare.com/ndownloader/files/24753887). Three datasets, v.i.z. BarryStreetData, PKLotSampled and PKLotSegmentedSampled along with the trained models and supporting files are present in the archive. Figure 3 shows the training and the test datasets.

BarryStreetData: was captured by the authors from the rooftop of Faculty of Business and Economics Building, The University of Melbourne, and shows on-street parking spaces along Barry Street, Melbourne, Australia. This dataset was taken by a camera at different intervals throughout the day (except night) having an image resolution of 1000 x 663 pixels. We created a subset of the dataset containing 100 images for these tutorials. This serves as our test data throughout the tutorials. We also provide the ground truth annotations of the parking slots delineations (28 slots) and the occupancy (2800) for evaluating the accuracy.

PKLotSampled: contains 279 randomly sampled images (having resolution of 1280 x 720 pixels) from original PKLot dataset (De Almeida et al., 2015)) and an additional 90 images that have been rotated to remove the dataset bias, totalling the number of images to 389. Dataset bias often happens due to the presence of a particular pattern in the training dataset, and in turn biases the classifier to make wrong prediction on unseen data. In the current context, the vehicles in the PKLot dataset (parking area PUCPR) were parked only in up-down orientation. Therefore, we rotated them to make the orientation of the vehicles left-right. The ground truth annotations of the parking slot delineations are provided for fine-tuning the vehicle detector using Faster-RCNN.

PKLotSegmentedSampled: contains 3000 randomly sampled (1500 empty and 1500 occupied) image crops from the original PKLot dataset of varied resolutions ranging from 32 x 39 pixels to 68 x 63 pixels. These images are used to fine-tune the CNNs.

3 Tutorial 1: Parking occupancy detection by fine-tuning a pre-trained CNN

Figure 4 shows the pipeline of the tutorial, where we will fine-tune CNNs (pre-trained with ImageNet dataset) with 3000 segmented images of PKLot dataset. We will demonstrate the fine-tuning process for ResNet50 in Section 3.1. Subsequently, we will test the fine-tuned ResNet50 with Barry street dataset to check the generalising (adaptability to other dataset) ability of the CNN in Section 3.2. Also, we benchmark the accuracies of different CNN architectures and report their run-times in Section 3.3.

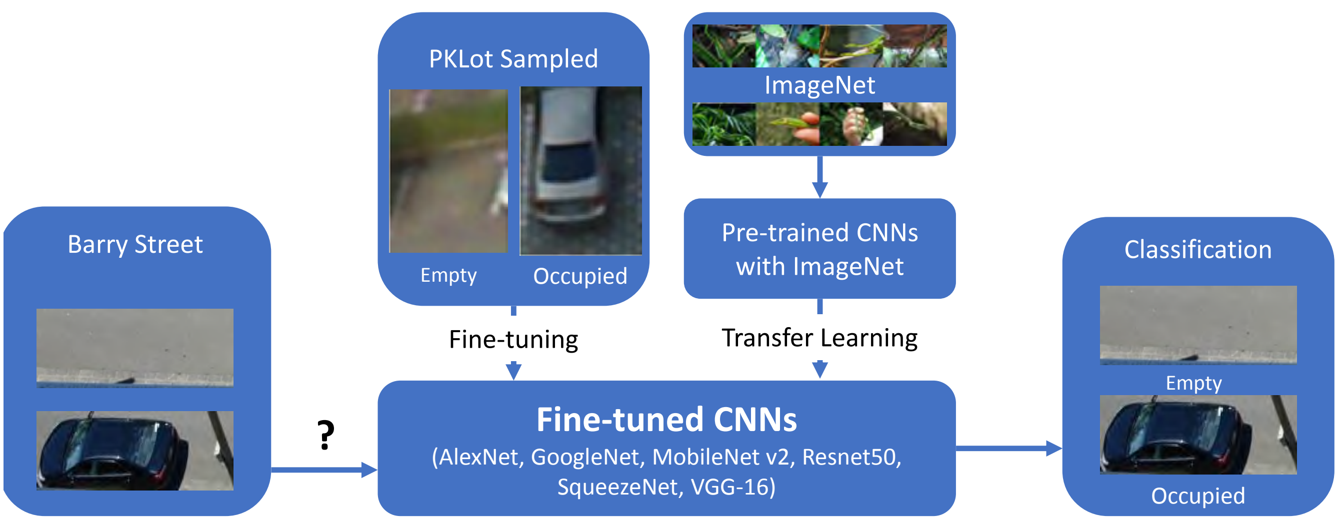

Figure 4: The pipeline of Tutorial 1, where we fine-tune pre-trained CNN with PKLot dataset and test with Barry Street dataset.

3.1 Fine-tune ResNet50 network with PKLot dataset

```
TrainOnline = false;
.
else
load('TrainedDetectorResnet50.mat');
load('trainingInfoResnet50.mat');
```

By default, "TrainOnline" is set as false, and therefore, the fine-tuned CNN is loaded directly without performing fine-tuning on-line. This option can be changed to perform the training on-line. GPU should be used for fine-tuning the network and it takes around 30 minutes (in NVIDIA Tesla P100).

"TrainOnline" being set to true, we start by loading pre-trained ResNet50 and the segmented images into an image datastore. A datastore contains the list of the file-names, and does not actually load the images into memory. The datastore also creates labels automatically based on folder names. For instance, it creates "Empty" and "Occupied" labels for each images automatically. Subsequently, we split the images into training (70%) and validation sets (30%) using the following lines of codes:

```
% load the pre-trained model in the workspace
load('Resnet50FeatureExtractor.mat');

% Create image datastore from folder and label by folder name
imds = imageDatastore([pwd '/PKLotSegmentedSampled/'], ...
'IncludeSubfolders',true, 'LabelSource','foldernames');

% Randomly split the trainig set (70%) and the validation set (30%)
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
```

The pre-trained ResNet50 contains 1000 classes, and currently it is unsuitable for making occupancy predictions. Therefore, we need to replace the classification and the fully-connected layers with the two classes which correspond to "Empty" or "Occupied". Subsequently, we extract the connections of the newly created graph and connect them to form a new CNN.

```
numClasses = 2; % Number of classes: Occupied and Empty

% replace the classification and the fully connected layers
lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);

% extract connections of the new graph
connections = lgraph.Connections;

% connect graph and new layers
lgraph = createLgraphUsingConnections(layers,connections);
```

To reduce over-fitting and to improve the generalisation ability of the CNN, data augmentation is performed on the fine-tuning dataset. Data augmentation involves transforming the fine-tuning images without changing the total number of images for each epoch. Here we perform two transformations 1) reflection along X and Y axes and 2) change the X and Y scales of the images. Also, we need to resize the fine-tuning images according to the input size of the CNN, which is fixed for each CNN architecture. Subsequently, we generate the validation dataset to validate the performance of the CNN.

```
% set range of chaging the scales of the images along X and Y axes
scaleRange = [0.9 1.1];

% define a data augmenter with steps to perform
imageAugmenter = imageDataAugmenter( ...
.
'RandYScale',scaleRange);

% define augmented fine-tuning dataset
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
'DataAugmentation',imageAugmenter);
```

In the next step we start the fine-tuning process, by setting up the training options. We set the optimiser to stochastic gradient descent with momentum with an initial learning rate of 0.005 that we reduce at every 5 epochs by a factor of 0.5. The maximum numbers of epochs is set to 20. This high initial learning rate helps the model to converge fast, otherwise it might have taken more epochs to reach the same level of performance. We reduce the learning rate slowly to avoid reaching the gradient descent to a local minima. We also set the batch size as 10 (this depends on the memory of the GPU). Increasing the batch size speeds up the fine-tuning, however, higher learning rate should be used, as large batch size usually provides a strong regularisation. Also, we shuffle the training data at every epoch to remove any dataset bias due to image sequences. To check the performance of the fine-tuning we set the validation frequency as 3. We could perhaps use higher frequency, but that would slow the fine-tuning process without any improvement.

```
% define validation frequency
valFrequency = 3;

% set fine-tuning options
options = trainingOptions('sgdm', ... % stochastic gradient descent with momentum
'MiniBatchSize',10, ... % number of samples to train together
'MaxEpochs',20, ... % maximum number of epochs
'InitialLearnRate',5e-3, ... % learning rate
'Shuffle','every-epoch', ... % shuffle data to reduce over-fitting
'LearnRateDropFactor', 0.5, ... % factor to reduce learning rate
'LearnRateDropPeriod', 5, ... % epoch after learning rate dropped
.
.
% Fine-tune the newly created CNN and save training information
[net,traininfo]  = trainNetwork(augimdsTrain,lgraph,options);
```

Fine-tuning approximately takes 30 minutes to complete on GPU, and we can now plot the training loss and accuracy curves, where variable “traininfo” contains the details of the training. After post-processing the data, we can visualise the curves using the following lines:

```
plot(TrainLoss, 'lineWidth', 2); hold on; % to show two plots
plot (ValidationLoss, 'lineWidth', 2)
legend('Training loss', 'Validation loss')
xlabel('Epochs'); ylabel('Loss');
```

3.2 Test the fine-tuned network with Barry street dataset

We have fine-tuned the CNN in the previous step, and now we will test it with Barry street dataset. By default RunOnline is set to false, therefore, the results of classification are loaded into the workspace without running the CNN on-line.

```
RunOnline = false;
.
.
else
load('BarryStreetTestResults.mat');
```

“RunOnline” being set to true will read the 100 Barry street images and will crop out individual 28 parking slots of each image. Subsequently, these cropped images will be passed to the fine-tuned CNN for classification. We start by setting path to the directory of the Barry Street images. Note that “pwd” refers to present working directory in MATLAB. In the next step we load the ground truth of Barry Street dataset containing the occupancy status (2800) and delineation of the parking slots (28). This delineations are in form of bounding boxes (variable “ParkingSlots”) that are used to crop the individual parking slots. A bounding box is defined by $[x, y, w, h]$, where $[x, y]$ represents the one corner of the the box and w and h denotes the width and height of the boxes. These cropped images are further resized to suit the input size of the CNN. Running on-line the classification should take around 4 minutes on CPU to complete, or approximately 2.5 seconds for each image (for classifying 28 parking slots). On a GPU the whole process takes a couple of seconds. The results of the classification are saved in the variable “YPred” and the classification scores are saved in variable “probs”. Subsequently, we check the accuracy of the classification

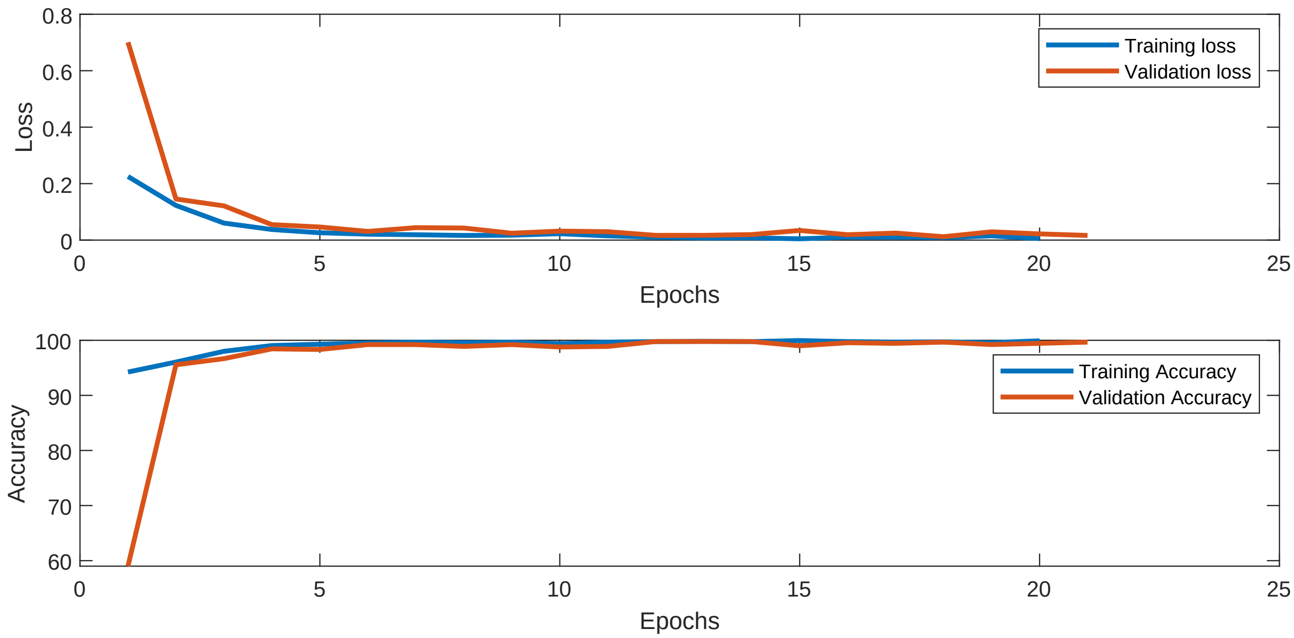

Figure 5: Training curves of the fine-tuning process. (a) The fine-tuning and validation loss vs epoch. (b) The fine-tuning and validation accuracy vs epochs.

by comparing YPred with "AnnotationTable", where variable AnnotationTable contains the ground truth parking occupancy. In the last step we plot the confusion matrix and visualise some of the wrongly classified image.

```
% set the directory containing the Barry street dataset
imageName = dir(fullfile(pwd,'BarryStreetData\', '*.JPG'));

% load Barry St occupancy and parking slot delineation ground truth
load('GroundTruthBarryStreet.mat')

% crop individual parking slots from the Barry Street image
cropImage = imcrop(BarryStreetImage, ParkingSlots(m,:));

% resize each cropped image to suit the input size of CNN
imdsIm = imresize(cropImage, inputSize(1:2));

% Predict the occupancy status using the fine-tuned CNN.
[YPred(count),probs(count,:)] = classify(net,imdsIm);

% plot the confusion matrix
plotconfusion (categorical(AnnotationTable), YPred);
```

From Figure 6 we observe that only one occupied parking slot is classified as empty, and 21 empty slots have been wrongly classified as occupied. Therefore, the fine-tuned CNN is slightly less precise while classifying empty parking slots. Upon visualising some of the wrongly classified parking slots in Figure 7 we observe most of them contain a part of a vehicle inside the image crop. Also, it is observed that the images that does not contain the vehicles have low classification scores. Finally, we visualise the occupancy of the parking slots in Figure 8.

3.3 Time and accuracy benchmarking using different CNN architectures

In this section we compare different CNN architectures in terms of their runtime and accuracies achievable to help the audience to in choosing the right architecture for their needs. For the experiments, an i7 5600U CPU @2.6 GHz was used and the GPU was NVIDIA Tesla P100 with 12 GB of memory. Note we used only one core of the CPU for the experiments. We compared ResNet50 with AlexNet, GoogleNet, MobileNet v2, SqueezeNet and VGG-16.

Table 1 shows the accuracy achieved by fine-tuning different CNN architectures, their run-times and train-times

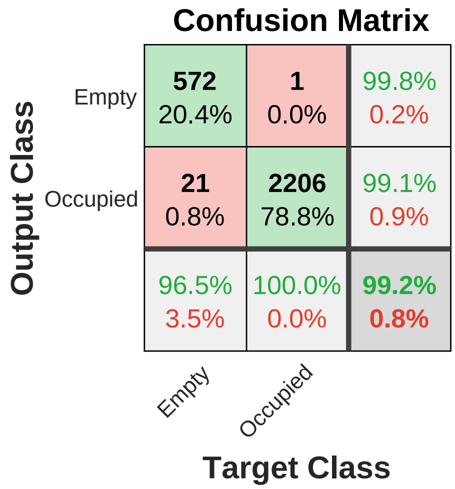

Figure 6: The confusion matrix of the classification, showing that the overall accuracy is 99.2%. The misclassifications are largely due to wrong classification of empty parking spaces as occupied ones (21 instances).

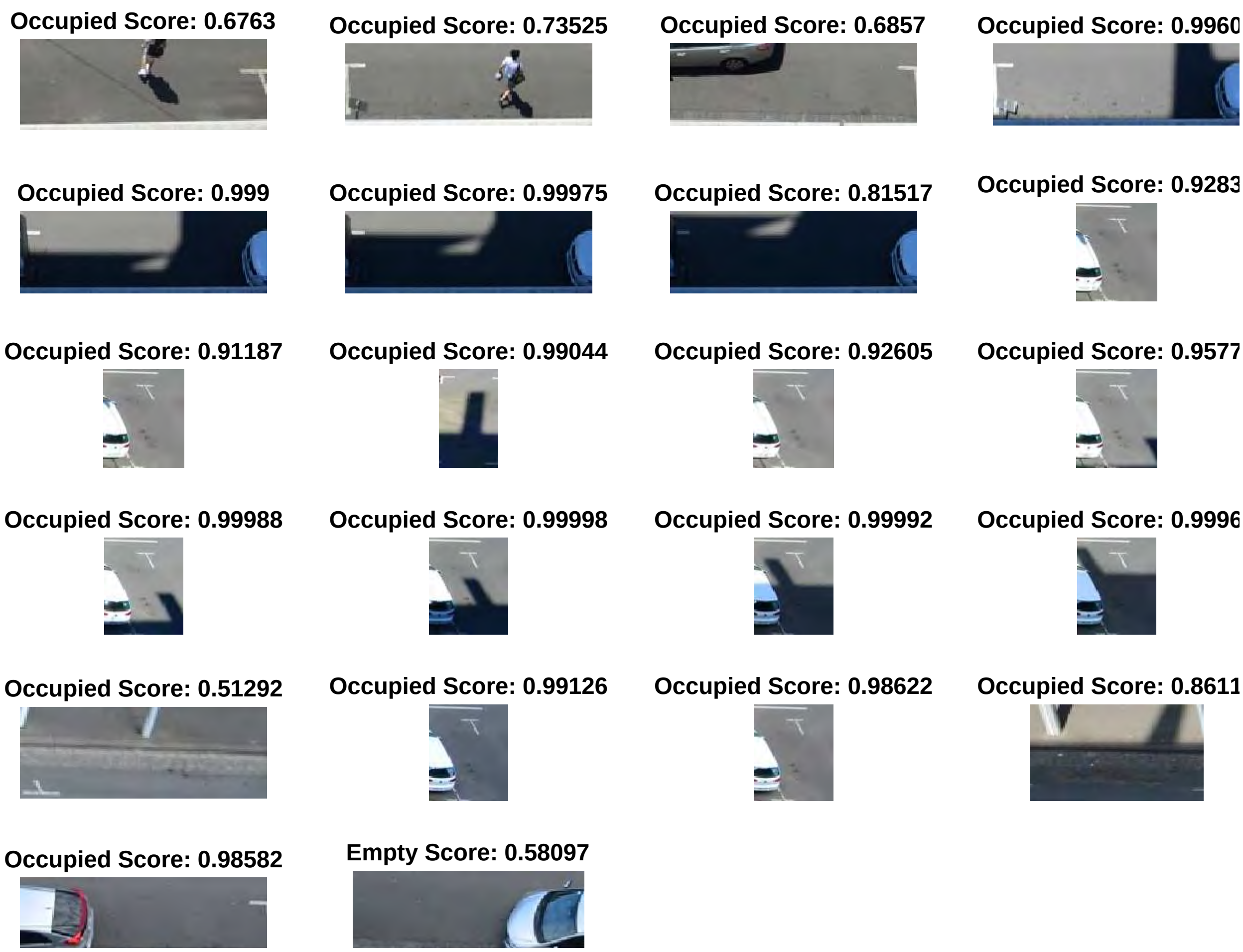

Figure 7: Some of the wrongly classified image patches with their respective classification scores.

with GPU. It is observed that the generalising ability of all the fine-tuned networks are excellent, however, we see slight low accuracies for AlexNet, MobileNetv2 and VGG-16 networks. In terms of run-time we see SqueezeNet is the fastest one and the slowest one being VGG-16 network. Note these run-times are average times and include overheads such as reading files from the disk and cropping images. Therefore, SqueezeNet can process approximately 50 parking lots in one second with CPU alone with 99.2% accuracy. In regards to training time, all of the CNN are

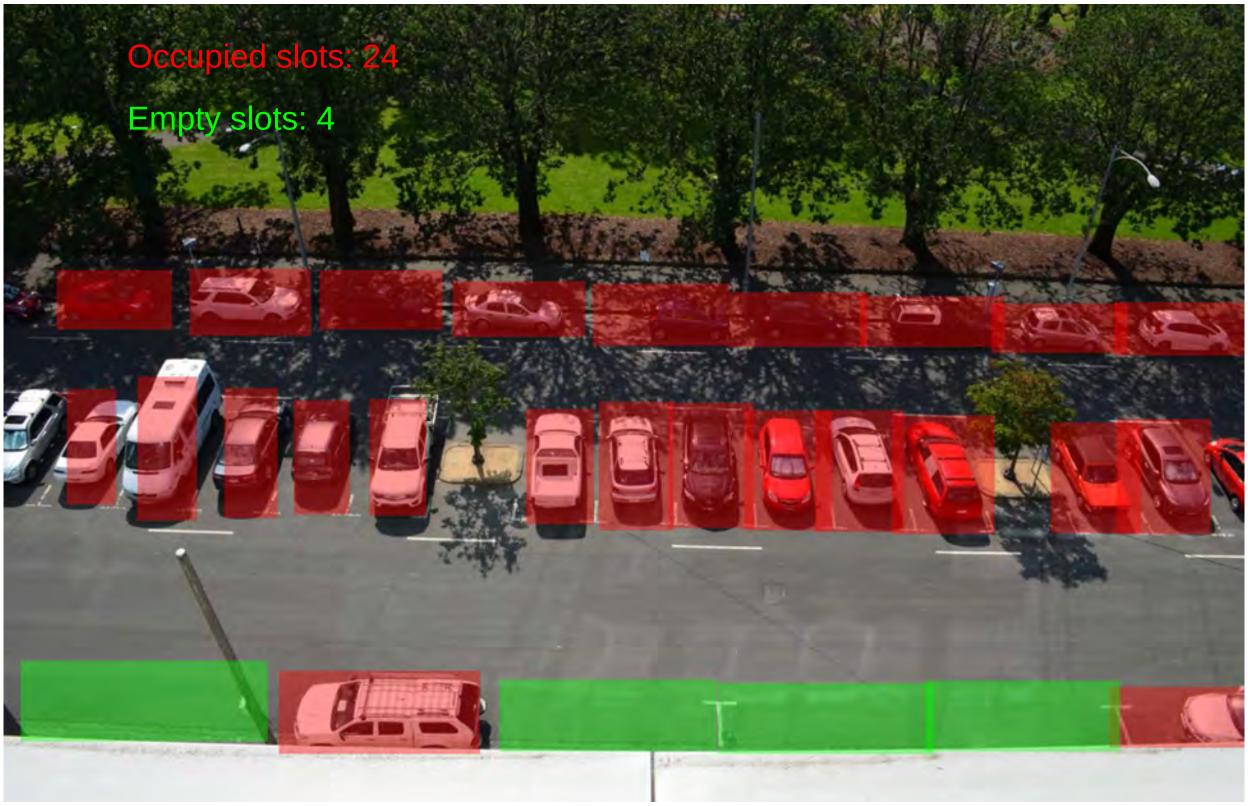

Figure 8: The visualisation of the parking occupancy, where red colour denotes the occupied and green colour denotes vacant spaces.

fine-tuned under 10 minutes on the GPU. Note that different learning rates were used for fine-tuning the different networks.

Table 1: The comparison of the achievable accuracy and the run-times for different CNN architectures by fine-tuning the networks on GPU. The fine-tuning data was PKLot segmented images and test data was Barry street data.

Network	Accuracy	Run-time on CPU (msec)	Train time on GPU (min:sec)	Base learning rate
AlexNet	97.3%	33.6	1:46	0.001
GoogleNet	99.2%	55.4	6:27	0.001
MobileNet v2	98.6%	62.7	9:23	0.005
ResNet50	99.2%	100.1	8:09	0.005
SqueezeNet	99.2%	19.3	2:42	0.005
VGG-16	98.5%	400.0	4:56	0.001

4 Tutorial 2: Automatic parking slot delineation using deep learning

In Tutorial 1, we have used the ground truth delineations of the parking slots in the images. As shown in Figure 9, in this Tutorial, we describe a novel method to for automatic parking delineation of Barry Street data by fine-tuning Faster-RCNN object detector with PKLot dataset. This delineation is performed using spatio-temporal analysis of the detected vehicles in Barry Street dataset. We detect vehicles in Barry Street images for many frames, and then cluster the detections to individual parking slots using a robust density-based clustering algorithm (Ester et al., 1996). Subsequently, we estimate the coordinates of the parking slots by weighing the coordinates of the individual detections with the scores of the detections. Further we refine and improve the coordinates of the parking slot delineations using the statistics of all the detections.

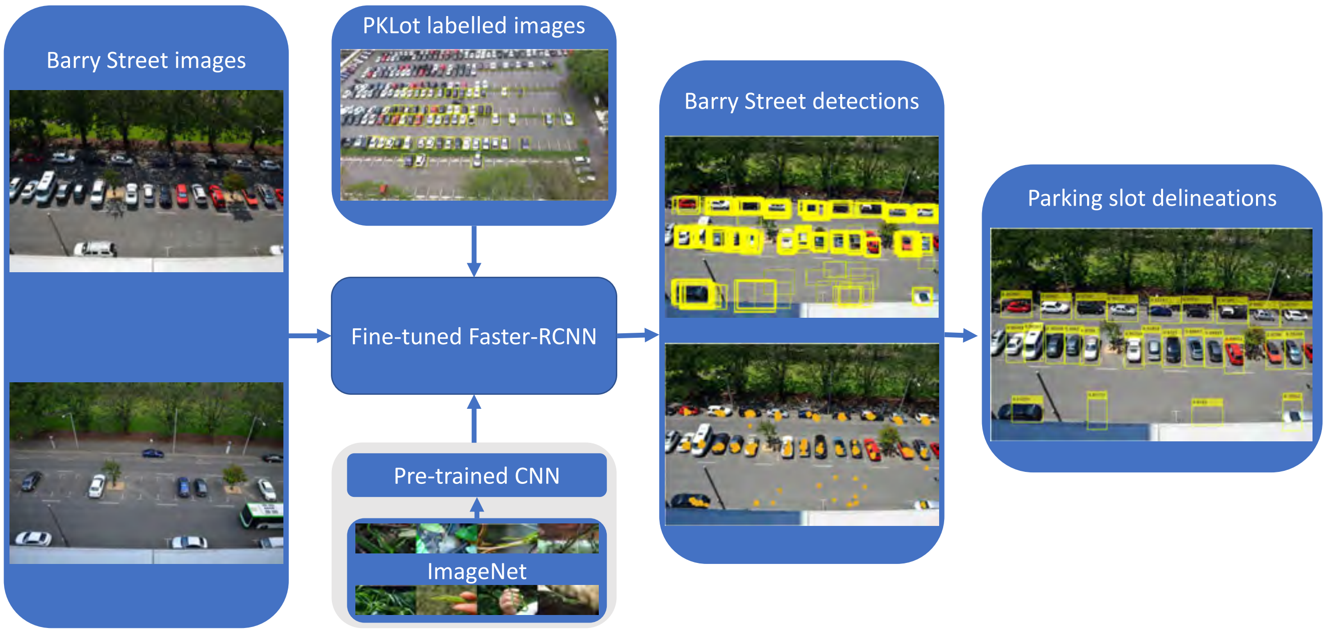

Figure 9: The pipeline of Tutorial 2, where we use a fine-tuned Faster-RCNN object detector to automatically delineate the parking slots.

4.1 Assumptions and limitations

The key assumption of the method is that the 1 vehicle detections made by Faster-RCNN will cluster more often in the actual parking slots, as compared to other parts of the image, such as on the roads. This is a valid assumption as the vehicles are parked longer than they are actually on road. Therefore, we should be able to estimate individual parking slots by combining all the detections in each cluster. 2 The second assumption of the method is that a vehicle takes around 80% space of the parking slot. Therefore, each parking slot is approximately ~~1.2~~ 1.25 times the length of the parked vehicle. 3 The last assumption is that the size of the parking slots remain approximately the same throughout the camera view.

Coming to the limitations of the assumptions made. Some of the parking slots might be missed as a result of low parking rate. For instance, the vehicles might be parked less often in reserved parking slots (like for people with special needs), or parking slots that are far from the entrance of the parking area. Additionally, dense traffic during peak hours might also result in several detections on road. Although, these detections might not form dense clusters as compared to the actual parking slots, the possibility of delineation of parking slots on roads can not be neglected. Lastly, for a very oblique view of the cameras, the vehicles that are far away from the camera appear smaller than the vehicles that are closer.

4.2 Vehicle detection using Faster-RCNN

We start the tutorial by loading a pre-trained Faster-RCNN detector that is trained on images of highway taken from a camera inside a moving vehicle. This pre-trained detector performs well to detect vehicles on highway, however, its performance to detect vehicles in CCTV images that has been taken from an oblique view is questionable. Therefore, we will fine-tune this detector with PKLot dataset that includes the bounding boxes of the detections.

By default, “train” is set to false and the Faster-RCNN detector fine-tuned with PKLot dataset is loaded into the workspace.

```
train = false;
.
else
load('Fine-tunedFRCNNResnet50.mat'); % load the trained model
load('trainingInfoFRCNNResnet50.mat'); % load training info
```

ok

Setting "train" to true will start training the Faster-RCNN detector with PKLot dataset. We set batch size to 1 to accommodate the model to GPU memory. We set the Negative Overlap Range to {0 0.3} and Positive Overlap Range to {0.6 1}. This means that a detection is considered as negative detection when the Intersect over Union (IoU) falls between 0 and 0.3, and is considered positive when it falls in the range of 0.6 and 1. The IoU is the area of intersection of two bounding boxes. The following lines fine-tunes the network.

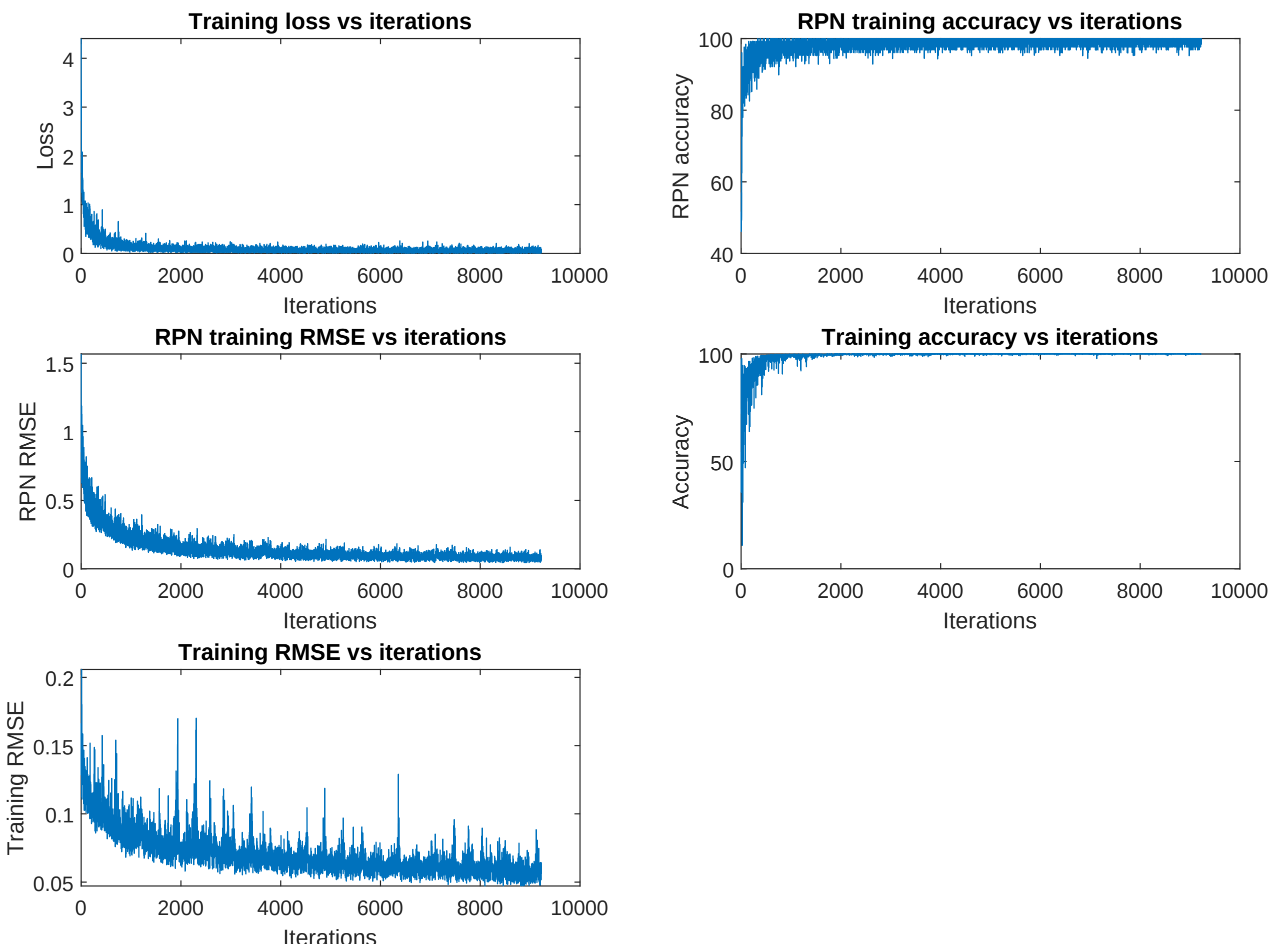

Figure 10: Training curves of Faster-RCNN fine-tuned with PKLot dataset.

Figure 10 shows the training curves. Note there are 369 images in each epoch, and we trained the network for 25 epochs, hence resulting in approximately 10000 iterations. We observe that training RMSE continues to improve upto 9000 iterations. Subsequently, we test the fine-tuned network with Barry Street images again, and Figure 11 shows the results of the detections.

```
options = trainingOptions('sgdm', ...
'MiniBatchSize', 1, ...
.
.
% Train an R-CNN object detector
[rcnn,traininfo] = trainFasterRCNNObjectDetector(vehicleDataset, detector, options, ...
'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.6 1]) % train the model
```

Note that the image size of the images in PKLot dataset was 1280 x 720 pixels, whereas for Barry Street dataset it is 1000 x 663 pixels. Ideally, we should use same image size to reduce any bias. Therefore, we pre-process the Barry street images to a resolution of 1280 x 720 pixels without distorting the aspect ratio. This is done by adding the Barry Street images to a blank 1280 x 720 image. We use this image as an input the Faster-RCNN detector.

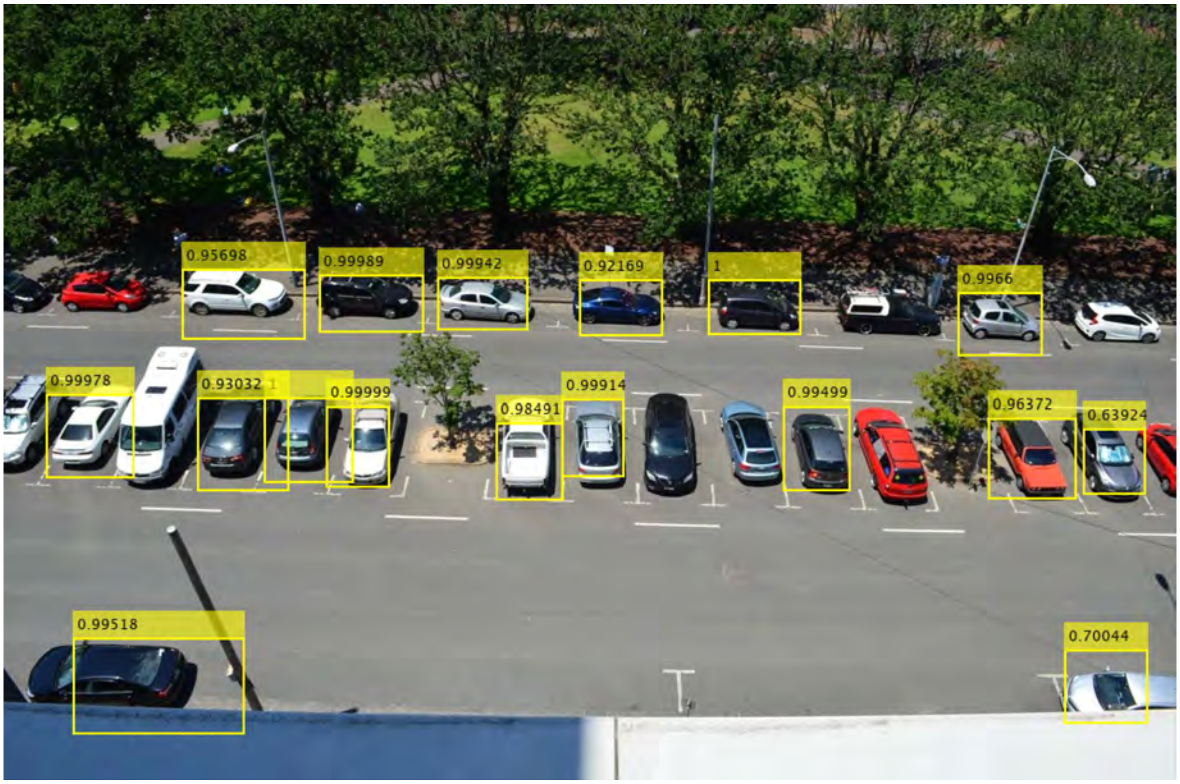

Figure 11: The detections of one Barry Street image with Faster-RCNN fine-tuned with PKLot dataset.

```
% Preprocess the Barry street frame (663 x 1000) to original training image size (720 x 1280)
BarryStreetImageProcessed = uint8(zeros(720,1280,3));
BarryStreetImageProcessed(58:720, 141:1140,:) = BarryStreetImage;

% Run the trained detector on Barry street image. This step takes 1 minute on CPU and a second on ...
    GPU.
[bboxes,scores] = detect(rcnn,BarryStreetImageProcessed);
```

662 1020

The outputs of the detector are the bounding boxes and their respective scores. From Figure 11 we observed that 17 vehicles are detected. However, we notice that all the vehicles are not detected. Therefore, in the next step we run the fine-tuned network for all the 100 Barry Street images. Figure 12 shows all the detections and Figure 13 shows the centre of the detections. Subsequently, we save all the bounding boxes and scores of the detections for post processing.

Next, we use a robust density-based clustering algorithm (Ester et al., 1996) to estimate the number of clusters based on the spatial distance between the neighbouring points and the number of occurrences.

```
% Use density-based algorithm
idx = dbscan(bboxesTotal,20,4);
Classes = unique(idx,'rows');
```

cluster

idx: 好像是cluster的ID，會重複

Classes: 就是idx中unique的，共27個。one additional class as unclassified having a value of -1

Ideally the spatial distance should be equal to the standard deviation of the detections, and is a function of the image size and the size of the detections. We found this experimentally to be 20 pixels. Also, the threshold for the number of neighbourhood occurrences is set to 4. This value can be set based on the total number of images the detection is made, which in this case is 100 images. The output of the function is the number of clusters, and it identifies the number of parking slots. A total number of 26 clusters were identified by the algorithm. In the next step we estimate the final bounding boxes of each class by weighing them with their respective scores. This

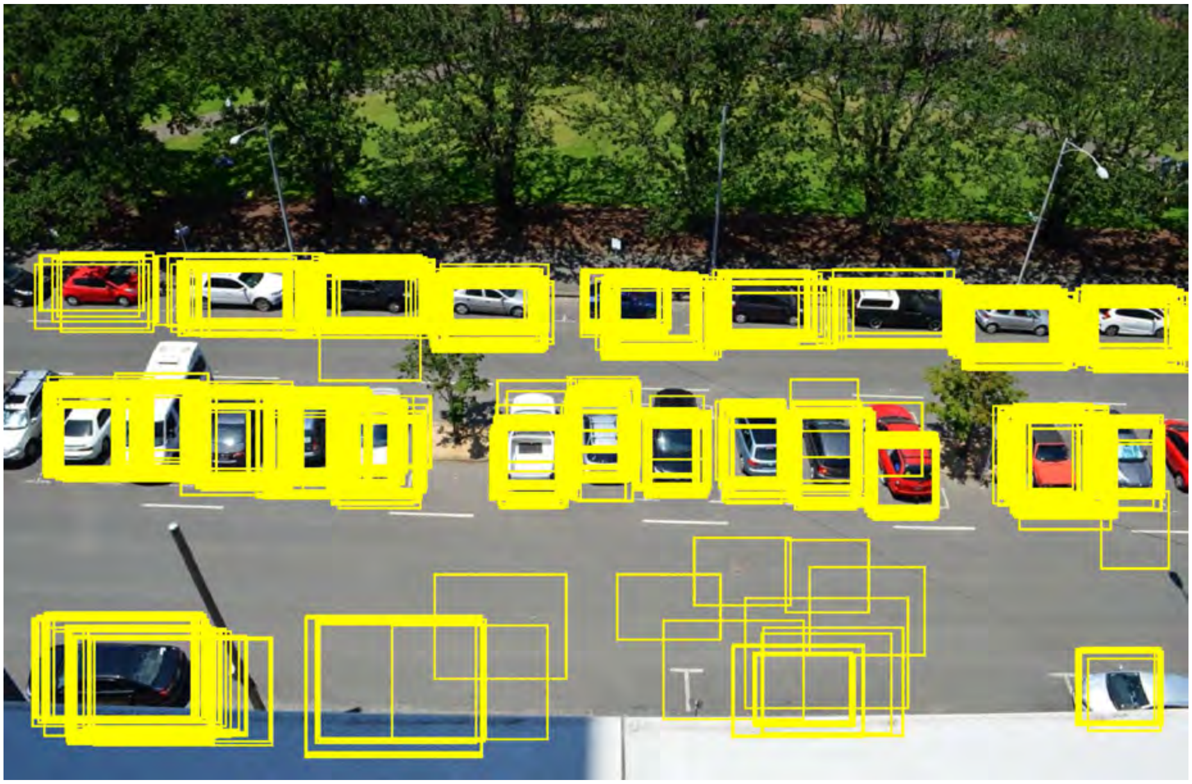

Figure 12: The detections of all Barry Street images with Faster-RCNN fine-tuned with PKLot dataset.

Figure 13: The centre of detections of all Barry Street images with Faster-RCNN fine-tuned with PKLot dataset.

estimation is achieved by the following equation, where $*$ represents element-wise multiplications:

$$Bbox^{class} = \frac{Bbox * BboxScore}{\sum BboxScore} \tag{1}$$

Where $Bbox^{class}$ represents the bounding box of a particular class (parking slot), $Bbox$ represents the array containing all the bounding boxes of the particular class, and $BboxScore$ is the matrix containing the respective scores of the $Bbox$. This is achieved in the following lines of code:

```
% Estimate the bounding boxes of each class as the parking slot
classifiedMean(n,:) = [(classified{n}(:,1)'*classifiedScore{n}) ...
.
.
(classified{n}(:,4)'*classifiedScore{n})/(sum(classifiedScore{n}))];
```

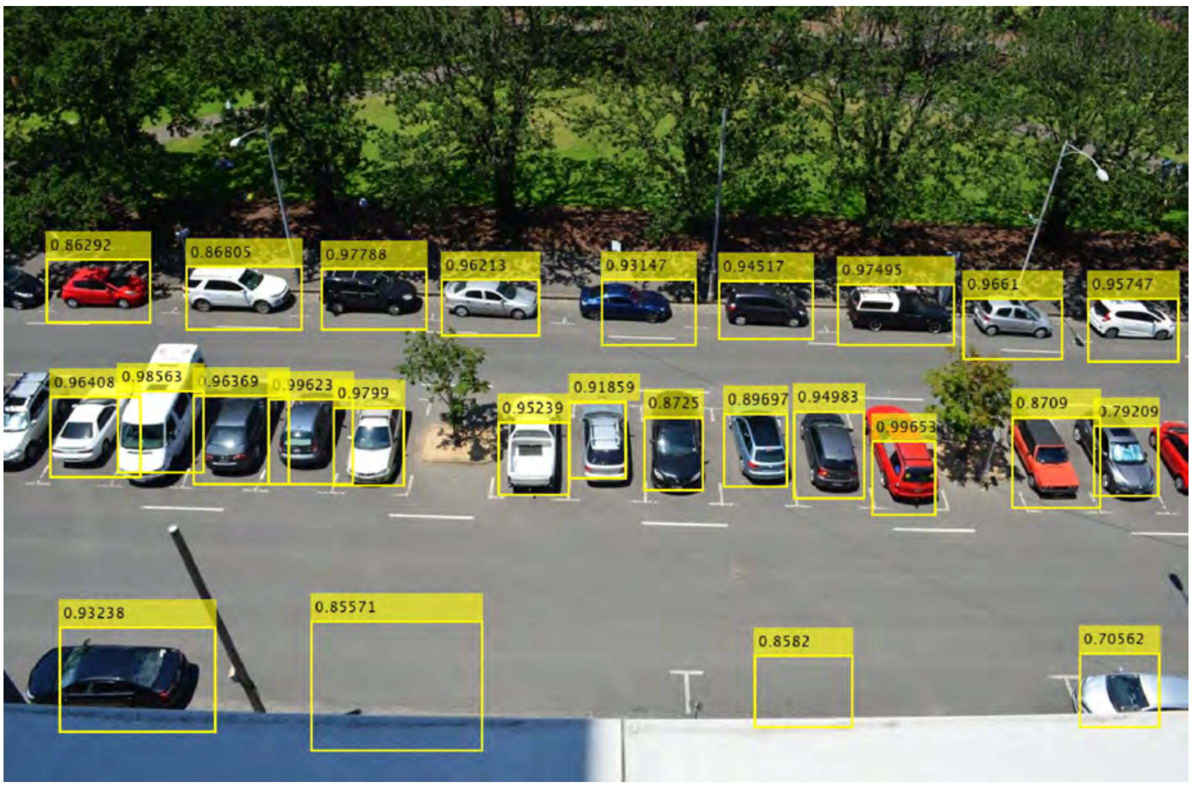

Figure 14: Bounding boxes derived from each cluster representing individual parking slots.

Figure 14 shows the bounding boxes of each classes with their respective average scores, and Figure 15 shows the average precision of the detections. The average precision with 50% IoU (also refereed as AP50) is [redacted] This low average precision is due to a couple of problems. Firstly, size of the detections are not uniform, for instance see the third row of Figure 14. Secondly, the lengths of the detected parking slots are smaller as compared to the whole parking slots, as the detector detects vehicles that are smaller than the parking slots. Therefore, to improve the delineations of the slots we further post-process the detections.

We start by calculating the average length and width of the parking slots in the above lines of code. We interchanged the length and the width for the slots whose aspect ratio was less than 1. Figure 16 shows the visualisation of the parking slots. Later, we calculated the average length and width of the parking slots to be [redacted] pixels, and therefore, and the average aspect ratio is [redacted]. Next, as per our assumption (in Section 4.1), we increase the length of the parking slots by [redacted]% and therefore, resize all the parking slots to [redacted] pixels.

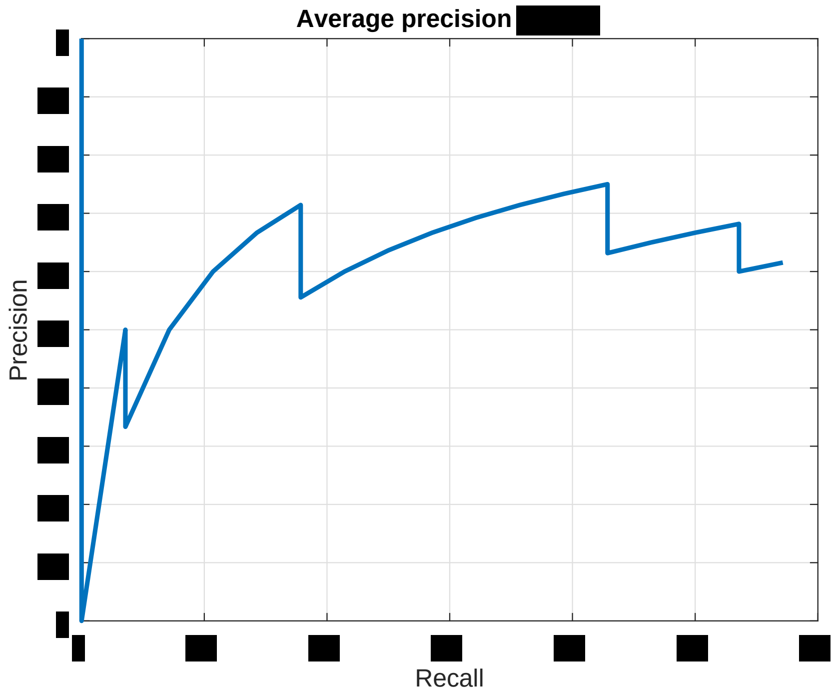

Figure 15: The average precision of all the bounding boxes delineated by clustering is [redacted]

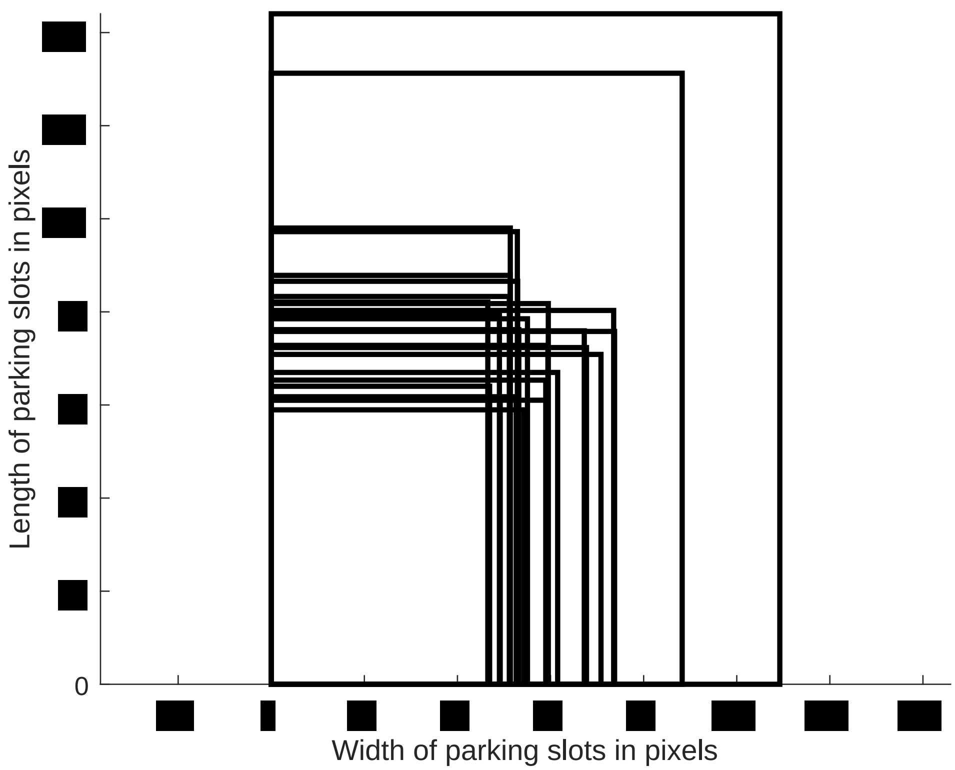

Figure 16: Visualisation of the delineated parking slots after clustering. The length and width of the bounding boxes having an aspect ration greater than 1 were interchanged.

```
for n=1:length(classifiedMean2)
if (classifiedMean2(n,3)/classifiedMean2(n,4) > (1/w)) % aspect ratio constraint
differenceX = classifiedMean2(n,3) - 1*[redacted]; % 80% assumption
.
.
else
.
.
classifiedMean2(n,4) = classifiedMean2(n,4) - differenceY;
end
end
```

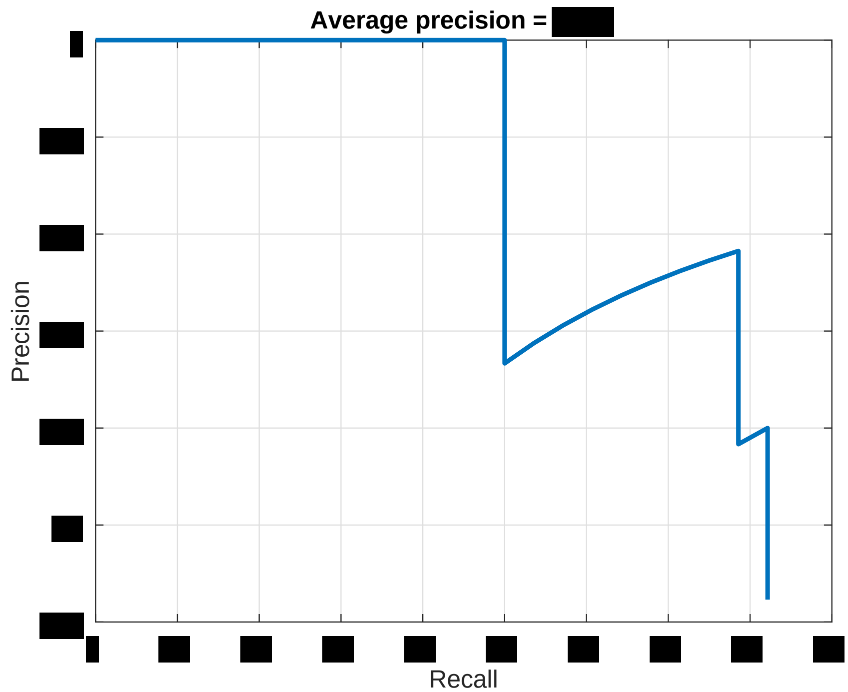

Figure 17: The average precision of the bounding boxes after post-processing.

Next we plot Figure 17 that shows the average precision of the detections after post-processing the bounding boxes. We observe an excellent improvement in the average precision AP50 from [redacted]% to [redacted]%, which is around [redacted]% improvement. For reference, the mAP50 (mean average precision for multi-class objects) of Faster-RCNN is approximately 59% (Redmon and Farhadi, 2018) with state-of-the-art feature extractor (ResNet-101-FPN).

In the next steps, we visualise the delineated parking slots of the Barry Street image in Figure 18, and we visualise them along with the ground truth bounding boxes in Figure 19. In Figure 18 and 19 we can see that all of the detections in the top and middle row are performed correctly. However, two parking slots in the bottom row of the parking area are missed. These missed detections can be explained on the basis of less parked vehicles in those parking slots that resulted in the lack of detections in those areas (Figure 12 and 13). Also, the bounding boxes of two of the detections in the bottom row have inverted dimensions. This anomaly can be explained based on the presence of the wall that occludes the vehicles, and hence change the dimensions (and hence the aspect ratios) of the bounding boxes for the detections. The change in the aspect ratio results in the inversion of the bounding box dimensions during the post-processing.

Therefore, once the parking slots are delineated, we no longer need the computationally expensive Faster-RCNN for the detections. We can directly use the delineations to perform classifications with the method explained in Tutorials 3. Therefore, the object detection problem reduces to the problem of image classification only. This is an advantage of the proposed method as compared to the methods that perform parking occupancy detection using Faster-RCNN directly.

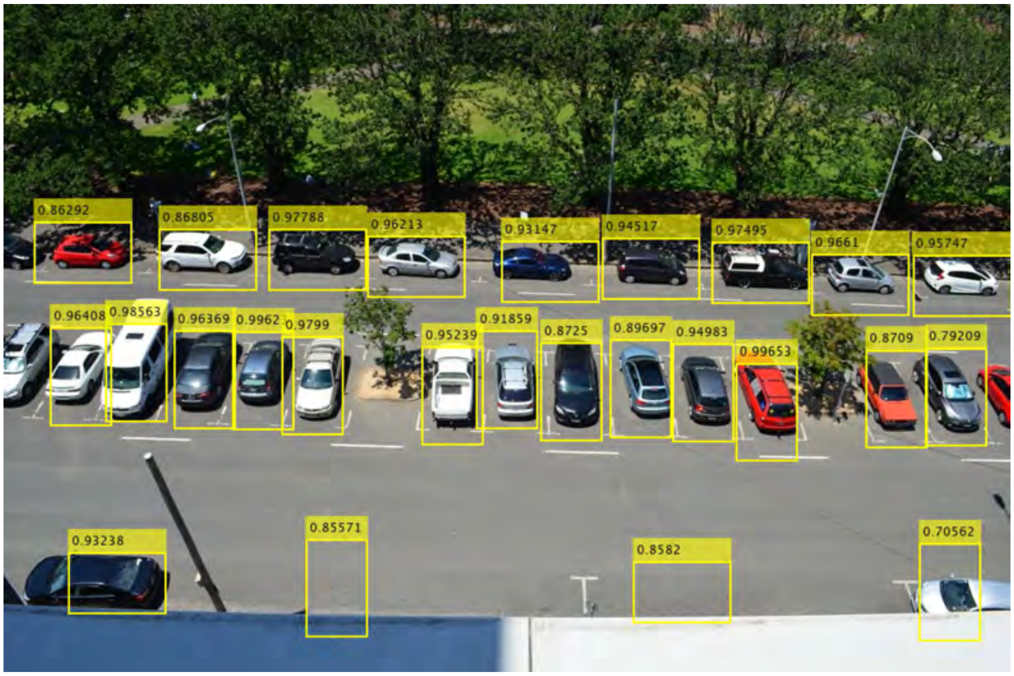

Figure 18: The final delineations of the parking slots after post-processing.

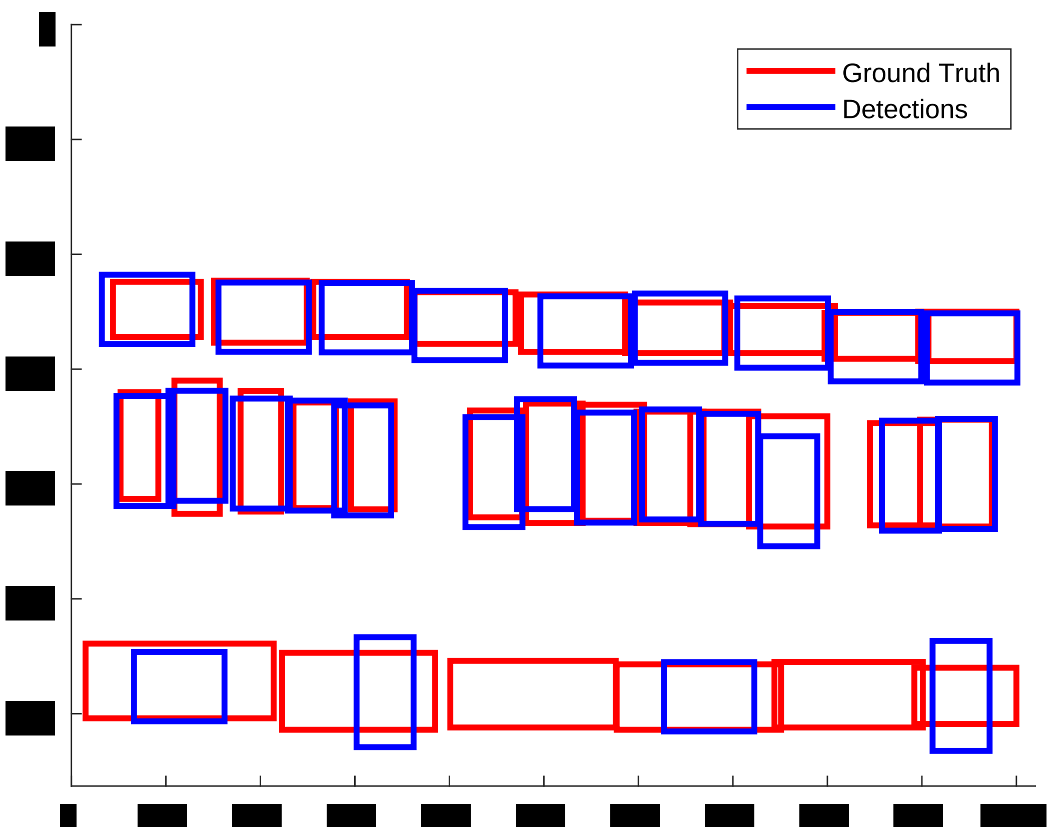

Figure 19: Visualisation of the delineated parking slots and the actual ground truth for the Barry Street dataset.

4.3 Applications to unmarked open parking spaces

The methodology demonstrated in Tutorial 2 can be extended to unmarked open parking spaces as well, or in areas where the parking delineations do not exist (for instance in India). Instead of allocating individual bounding boxes to the parking slots, we could perhaps allocate continuous "parking zones" where the vehicles are most likely to park. We can also have information on the orientation of parking of the vehicles. This continuous parking zone can be broken down according to the standard sizes of the vehicles plus a buffer zone to calculate the number of empty parking slots. Figure 20 shows a visualisation where 4.5 parking spaces could be identified in parking zone 5 using standard parking slot sizes. Therefore, more exploration in this context is needed and is a promising research direction.

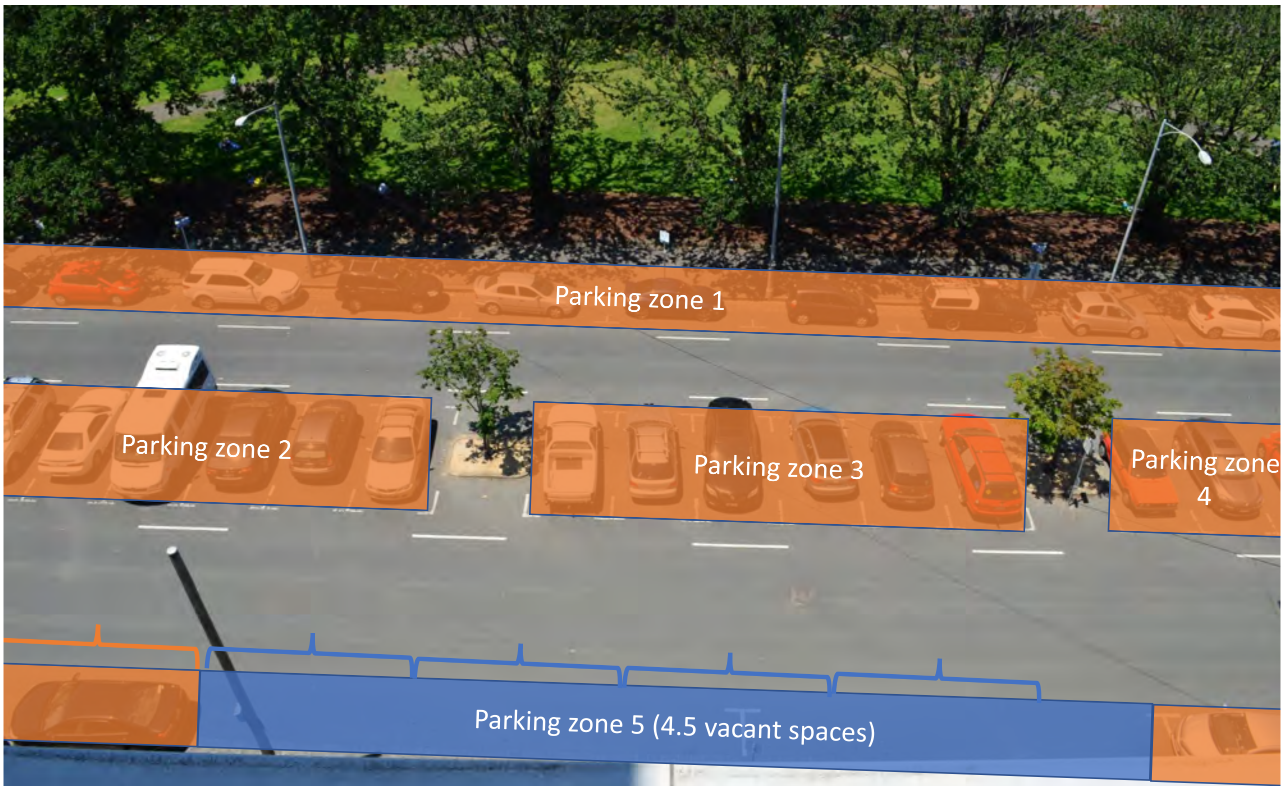

Figure 20: Visualisation of the parking zones for unmarked open parking spaces, showing the areas where the vehicles are likely to park. Using standard vehicle sizes the parking occupancy can be estimated.

4.4 Training and testing time of Faster-RCNN with ResNet50 backbone

The fine-tuning process took approximately 5 hours on a NVIDIA Tesla P100 GPU. It is infeasible to perform the fine-tuning process on CPU. However, once the model is fine-tuned it can operate with CPU in approximately 55 seconds (or 0.6 seconds on GPU). Therefore, for the automatic delineation of the parking slots, GPU is not mandatory, and the system can run for a couple of hours on CPU to produce the results. This computational overhead can be reduced by using smaller networks like SqueezeNet, however, the accuracy of the detector might be compromised.

5 Conclusions

This chapter presents two tutorials, one for detecting image-based parking occupancy and the other for automatic delineation of the parking slots. In the first tutorial we fine-tune a pre-trained network on a subset of publicly available PKLot dataset and checked the generalising ability of the network by testing with Barry street dataset. Also, we provide insights on the training hyper-parameters, training and testing times, accuracies and visualise some wrong classifications.

In the second tutorial we demonstrate a novel method to automatically delineate the parking spaces using a state-of-the-art vehicle detector (Faster-RCNN). We fine-tuned Faster-RCNN with a subset of PKLot data and detected vehicles in the Barry Street images. We combined the detections in multiple frames and performed spatio-temporal analysis of the parking slots to automatically delineate the parking slots. We used a robust density-based clustering algorithm to find the centre of the parking slots, and then weighted the bounding boxes according to the detection scores (confidence). We further post-processed the delineations to improve the detection accuracy and achieve better results than reported in the literature. We conclude that occlusions can effect the detections and can reduce the accuracy of automatic delineations of parking slots. Moreover, the results discussed in this tutorial can be extended to unmarked open parking spaces and points towards an interesting future direction.

References

Acharya, D., Yan, W., and Khoshelham, K. Real-time image-based parking occupancy detection using deep learning. In *Proceedings of the 5th Annual Conference of Research@Locate*, volume 2087, pages 33–40, April 2018.

Ahmad, J., Lewis, Z., Duraisamy, P., and Mcdonald, T. Parking lot monitoring using mrcnn. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–4. IEEE, 2019.

Amato, G., Carrara, F., Falchi, F., Gennaro, C., and Vairo, C. Car parking occupancy detection using smart camera networks and deep learning. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 1212–1217. IEEE, 2016.

Amato, G., Carrara, F., Falchi, F., Gennaro, C., Meghini, C., and Vairo, C. Deep learning for decentralized parking lot occupancy detection. *Expert Systems with Applications*, 72:327–334, 2017.

Cai, B. Y., Alvarez, R., Sit, M., Duarte, F., and Ratti, C. Deep learning-based video system for accurate and real-time parking measurement. *IEEE Internet of Things Journal*, 6(5):7693–7701, 2019.

De Almeida, P. R., Oliveira, L. S., Britto Jr, A. S., Silva Jr, E. J., and Koerich, A. L. Pklot–a robust dataset for parking lot classification. *Expert Systems with Applications*, 42(11):4937–4949, 2015.

del Postigo, C. G., Torres, J., and Menéndez, J. M. Vacant parking area estimation through background subtraction and transience map analysis. *IET Intelligent Transport Systems*, 9(9):835–841, 2015.

Ding, X. and Yang, R. Vehicle and parking space detection based on improved yolo network model. In *Journal of Physics: Conference Series*, volume 1325, page 012084. IOP Publishing, 2019.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Ichihashi, H., Notsu, A., Honda, K., Katada, T., and Fujiyoshi, M. Vacant parking space detector for outdoor parking lot by using surveillance camera and fcm classifier. In *2009 IEEE International Conference on Fuzzy Systems*, pages 127–134. IEEE, 2009.

Jung, H. G., Lee, Y. H., and Kim, J. Uniform user interface for semiautomatic parking slot marking recognition. *IEEE transactions on vehicular technology*, 59(2):616–626, 2009.

Khan, G., Farooq, M. A., Tariq, Z., and Khan, M. U. G. Deep-learning based vehicle count and free parking slot detection system. In *2019 22nd International Multitopic Conference (INMIC)*, pages 1–7. IEEE, 2019.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

Nieto, R. M., García-Martín, Á., Hauptmann, A. G., and Martínez, J. M. Automatic vacant parking places management system using multicamera vehicle detection. *IEEE Transactions on Intelligent Transportation Systems*, 20(3):1069–1080, 2018.

Paidi, V., Fleyeh, H., and Nyberg, R. G. Deep learning-based vehicle occupancy detection in an open parking lot using thermal camera. *IET Intelligent Transport Systems*, 2020.

Redmon, J. and Farhadi, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

Sairam, B., Agrawal, A., Krishna, G., and Sahu, S. P. Automated vehicle parking slot detection system using deep learning. In *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, pages 750–755. IEEE, 2020.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Suhr, J. K. and Jung, H. G. Sensor fusion-based vacant parking slot detection and tracking. *IEEE transactions on intelligent transportation systems*, 15(1):21–36, 2013.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

True, N. Vacant parking space detection in static images. *University of California, San Diego*, 17:659–662, 2007.

Valipour, S., Siam, M., Stroulia, E., and Jagersand, M. Parking-stall vacancy indicator system, based on deep convolutional neural networks. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 655–660. IEEE, 2016.

Vítek, S. and Melničuk, P. A distributed wireless camera system for the management of parking spaces. *Sensors*, 18 (1):69, 2018.

Zhang, L., Huang, J., Li, X., and Xiong, L. Vision-based parking-slot detection: A dcnn-based approach and a large-scale benchmark dataset. *IEEE Transactions on Image Processing*, 27(11):5350–5364, 2018.