IEB3037: STUDENT INDUSTRIAL (SIP) REPORT

Name	Hong Jowell
Student ID	18002394
Program	Electrical & Electronics Engineering
UTP Supervisor	Dr Saiful Azrin B Mohd Zulkifl
Host Company (HC) Supervisor	Dr Anis Laouiti

Host Company Verification Statement

VERIFICATON STATEMENT

I hereby verify that this report was written by

Hong Jowell

and all information regarding this company and the projects involved are NOT CONFIDENTIAL / CONFIDENTIAL (strikethrough not relevant).

Host Company Supervisor's Signature & Stamp	
Name:	Dr Anis Laouiti
Designation:	Professor
Host Company's:	
Date:	

Contents

1. Abstract

The invention of vehicle has been a great leap for the humanity as it saves time for people to travel around. Almost every home owns a vehicle today, a car, a truck, a motorcycle or even a bicycle. As population increases, the number of vehicles increasing as well. The demand for parking spaces increases every day. People spend a lot of time finding parking spaces in parking lots. However, there are already parking sensors to help people determine occupancy. Sensors not only have high cost of building and maintenance, but they are also usually inaccurate. This paper aims to improve the system by replacing these sensors using computer vision with a parking space detection algorithm. The methodology of developing this algorithm includes training 2 models, VGG16 and YOLOv5 model. The results of the program will be further discussed later, but overall, the advantage of VGG16 is accurate while YOLOv5 is fast. When both models are combined together, a semi-automated parking space is developed, can be deployed and implemented in almost every parking lot.

2. Introduction

2.1. Background of study

As technology advances, owning a car is becoming more affordable and accessible to people nowadays. The number of cars in cities increases as the population increases. However, the number of parking spaces remains the same, making searching for parking spaces a challenge. Drivers often have to test their luck by circling the parking lot to identify an empty space. They simply just do not have exact information about them. In the past, to tackle this problem, parking lot owners would hire some people to manage the parking lot. These persons might not have a total view of the parking lot therefore cannot determine the number of empty spaces for the drivers. As time went by, researchers and scientists came up with many parking space detection algorithms using gadgets like sensors to determine. They usually put individual sensors on top of every parking space, a light indicator to tell the occupancy. This data is then sent to a hub to display the total of empty spaces in the parking lot at the entrance. This technology has been in the market for several years now.

2.2. Problem Statement

The target of the system is mostly in open areas, the condition applies to all open parking lots, such as supermarkets, universities and even shopping malls. Implementing a sensor-based approach to automate parking space detection has been very mature, however, to integrate these sensors is very time consuming, and also it will cost more paperwork. Most of the time, these parking lots are equipped with surveillance cameras for security purposes. These pre-quipped gadgets should be utilised for the detection of parking spaces.

2.3. Scope of work

The scope of this work includes a lot of reading to understand the concept behind VGG16 convolutional neural network, YOLOv5 object detection algorithm. Before any work is carried out, basic machine learning knowledge is required, especially transfer learning as it is needed to develop the parking space detection model. Trials and errors are necessary in the process especially during minor configuration on the models to optimise the program. The final program is targeted to inference on Jetson Nano, deploy this inference with a camera to detect parking spaces of a parking lot.

3. Objective

This research is crucial as it tries to simplify and automate parking space detection as humanly possible. By deploying only cameras and some algorithms, sensors can be removed and save cost of maintenance.

3.1. Current Technology

The current technology uses in-ground vehicle detection sensors:

3.1.1. Infrared sensors

Figure 1 Infrared sensors

These sensors can be found in applications where particular gestures or movement patterns need to be detected such as cameras for autofocus, drones, controllers. The principle of using an infrared sensor is simple, to determine the Time-of-Flight (ToF), where the emitter sends a photon to the target and waits for the photon to bounce back to the sensor. The time of this journey will be measured to determine the distance.

$$Measured\ distance = \frac{Time\ travelled\ by\ Photon}{2} \times speed\ of\ light$$

The advantages of ToF sensor include it is not sensitive to the target object material. It has better ambient light rejection and cover glass crosstalk compensation. However, it is expensive and energy inefficient. It can drain much higher current due to constantly emitting infrared rays. Most importantly, it is very sensitive to optical obstructions. For example, if

there is a tree near the empty space, the branch extends on top of the parking, the sensor will mistake the branch as a vehicle, hence it is not accurate.

3.1.2. Ultrasonic ranging sensors

Figure 2 Ultrasound sensors on car bumper

Instead of emitting photons, these sensors send ultrasound waves using the same principle as ToF. Ultrasound sensors can be widely found in cars nowadays for parking assistance purposes. These proximity sensors, or parking sensors can be found in front and rear of car bumpers. They use this algorithm to detect obstructions in short range, aiding the driver to identify obstruction on his blindspots.

It is better than infrared sensors as it emits a wider beam, making measurement more integral and less prone to random errors from reflections. However, it is also very power hungry. As this sensor covers much bigger mechanical dimensions and also high current drain, making this surface-mounted sensor thin is challenging. The casing needs to include a big emitter, a receiver and also need to accommodate a relatively large battery to operate the sensor.

3.1.3. Radar ranging sensor

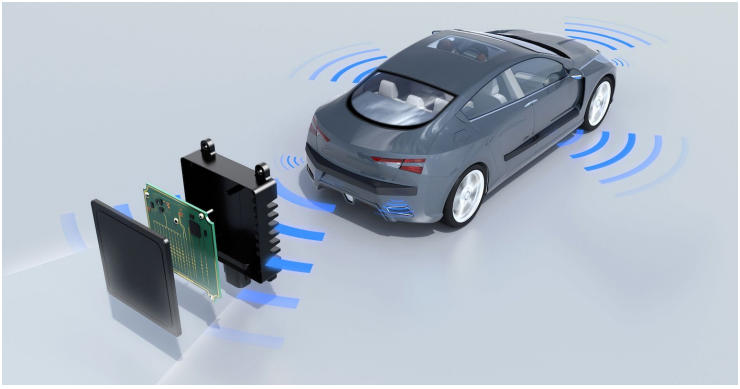


Figure 3 Radar sensors

Radar sensors are commonly used in extremely long range applications, such as air, naval defence systems, aviation. Radar uses the same principle of ToF as well, but instead of sending photons and ultrasound waves, it emits radio waves. The wavelength of radio waves is much greater than sound waves, this characteristic makes it travels extremely long distances, which is useful for long range applications.

With a lower frequency, the radio waves can penetrate thin obstructions. It emits a wider beam like an ultrasonic sensor, it is less prone to random reflections. However, it is more expensive and it requires special circuit knowledge during installation. Other than that, radar signal processing requires powerful machines, which makes this implementation not ideal anymore.

3.2. New Approaches Today

As Artificial intelligence and computer vision advance today, this technology can be the most straightforward approach.

3.2.1. Computer vision

Computer vision is one of the fields in artificial intelligence (AI) that allows computers to consume meaning information, data from digital images, video footage and other visual inputs and derive actions or predict recommendations based on consumed information. If AI is the brain of a computer, computer vision is the eyes of the computer, by working together, it can observe and understand.

Computer vision trains machines to perform these algorithms before deploying with cameras. The system runs a series of analyses of data over and over until it discerns

distinctions and finally recognizes images. To train computer vision to recognize an object, the system needs to be fed with a good amount of images of that particular object, from every angle. The system will then learn the differences and recognise the object without any defects.

Computer vision is a combination of two essential technologies, which are machine learning called deep learning and convolutional neural network (CNN). Machine learning essentially uses algorithmic models that enable a computer to teach itself about the context of visual data. With enough data fed through the model, the system is able to learn itself to recognise one image from another. It is different from conventional programming where developers make the system recognise the image, machine learning is actually learning the image itself.

Addition of CNN enables these models to "see" better by breaking images down into pixels with tags and labels. These tags and labels represent the identity of every image, representing their significant differences from one another. CNN uses labels to perform convolutions, a mathematical operation on two functions to generate a third function, then make predictions about what is "seen". The neural network runs convolutions depending on different models and checks the accuracy of its predictions in a series of iterations until the result of predictions becomes true. Good prediction results mean the system is seeing images similar to humans.

There are several tasks can be accomplished with computer vision:

1. Image classification

Task where the system classifies the objects in an image, such as a dog, apple or even human's face. It can accurately predict that the given image belongs to a certain class.

2. Object detection

Task where the system uses image classification to identify certain classes from the image and extend the work further to detect and tabulate their appearance in image or video. One of the examples is detecting white and red blood cells in the bloodstream.

3. Object tracking

Task where the system needs to follow the movement and tracks of the detected object. This task is often executed using a real time camera, where data needs to be refreshed

every time. Most common example is autonomous vehicles. The system not only needs to classify and detect objects around and ahead of the vehicle, also it has to track them in motion as the vehicle is moving to avoid collisions and adhere to traffic laws. These algorithms have to be quick, accurate and instant.

4. Content-based image retrieval

Task where the application needs to use computer vision to browse, search and retrieve images from large databases, based on the content of the images rather than metadata tags associated with them. This task is important to automate the system, replacing manual labour.

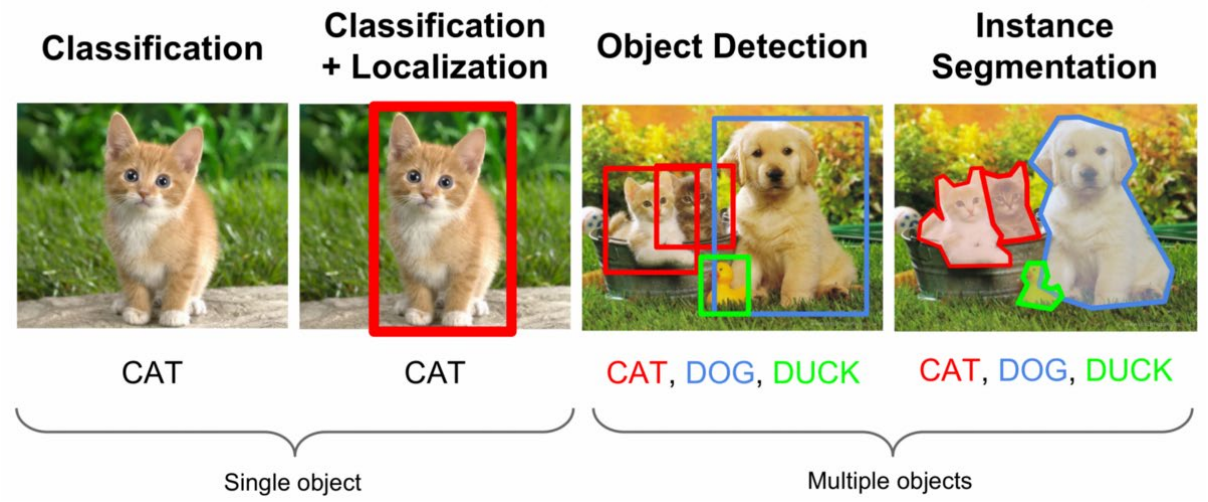


Figure 4 Object detection and Image Segmentation algorithm

The main objective of this research is to develop a program that can possibly automate parking space detection. While utilising computer vision and convolutional neural networks, the program will try to replace sensors and solely relying on cameras and computers only.

4. Literature Review

4.1. Convolutional Neural Network (CNN)

Over the years, AI starts to develop and grow in minimising the gap between the capabilities of humans and machines. Researchers have been diving deep in areas like computer vision to make this happen. The main goal of this field is to let machines view the world as humans do. If the machine can perceive the information similarly as human, with the use of knowledge into the system, they can carry out a multitude of tasks such as image recognition, image analysis and classification and other powerful tasks.

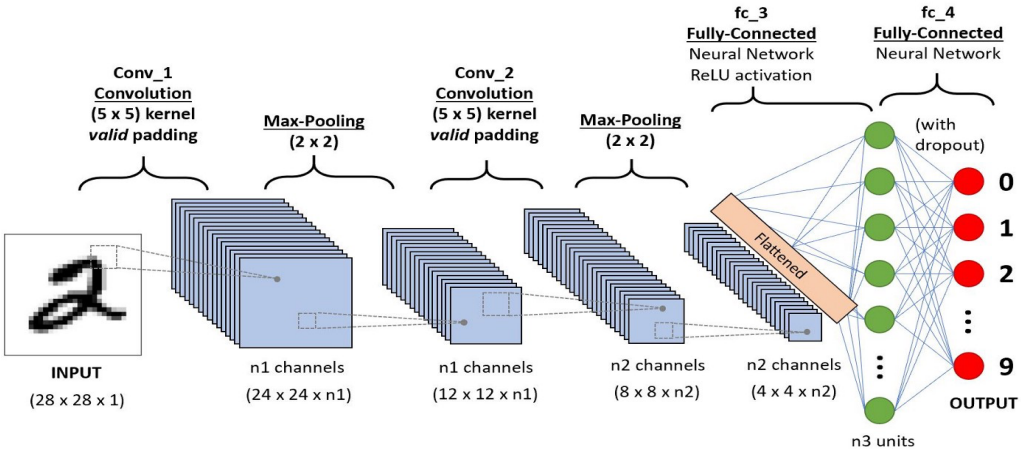


Figure 5 Convolutional Neural Network Architecture

A CNN is a Deep Learning algorithm which can be fed with input image, assign importance, this importance represents its learnable weights and biases, to the object in the image to let the system differentiate the features of one from the other.

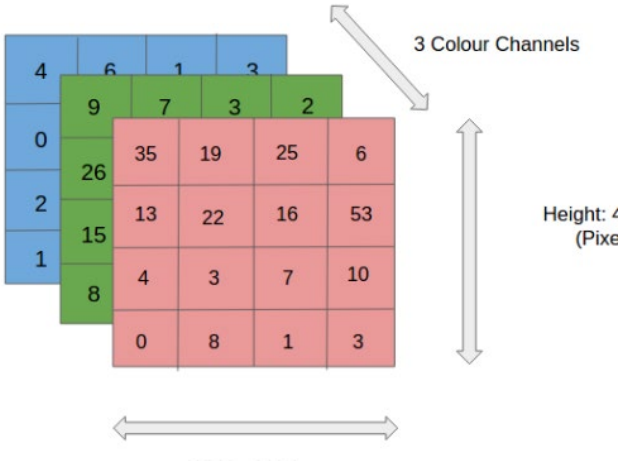


Figure 6 Matrices of a RGB image

Machines take images as matrices of pixel values, or numbers. The role of CNN is to reduce the images into an easier form to process, retaining the important features to get good predictions afterwards. This allows machines to learn massive datasets.

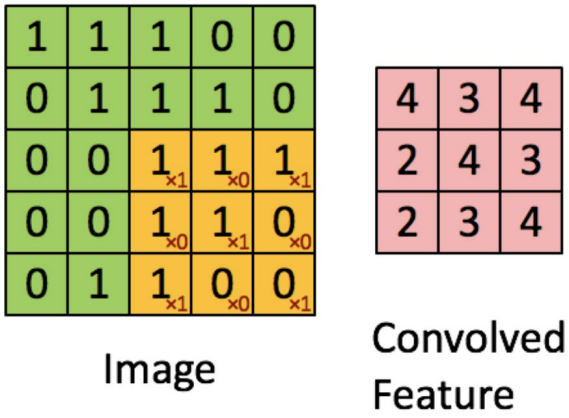


Figure 7 Kernel is sliced across the input image

The kernel is also known as a filter. This convolution operation is the first part of the CNN. This filter will hover and slice on top of the image to extract features, from left to right then hops down until the entire image is traversed. The results are summed with the bias to get the squashed one-depth channel convoluted feature output. As mentioned before, the main objective of this operation is to extract high level features from the input image. There are few layers in the CNN depending on configurations, the first layer is usually responsible for capturing low level features, then the second layer to extract higher level features and so on. These added layers allow the network to understand the images in the dataset.

After the convolution, the pooling layer is responsible for reducing spatial size of the convolved feature. The data is going to be too big for the machine to process without a pooling layer. The computational power required is too massive to handle. Convolutional layer and pooling layer are always together in CNN. Number of these layers can be increased depending on the complexities of the images, but it will definitely cost more computational power.

This final output is then flattened and fed to a regular neural network for classification purposes. An activation function is needed to nonlinear the linear output. They are fed into the fully connected layer to learn the nonlinear combinations of the high level features. For every iteration of training, the flattened output is fed to feed-forward neural network and backpropagation. The models will eventually be able to distinguish low level features in

images and undergo classification via Softmax Classification technique. A softmax function normalises the output from the network to a probability distribution over predicted output classes.

4.2. VGGNet

VGG is also known as Visual Geometry Group, a standard deep CNN architecture with multiple layers. VGG-16 consists of 16 convolutional layers and VGG-19 consists of 19 layers. VGG is the basis of developing object detection models. It is one of the most popular image recognition architectures out there.

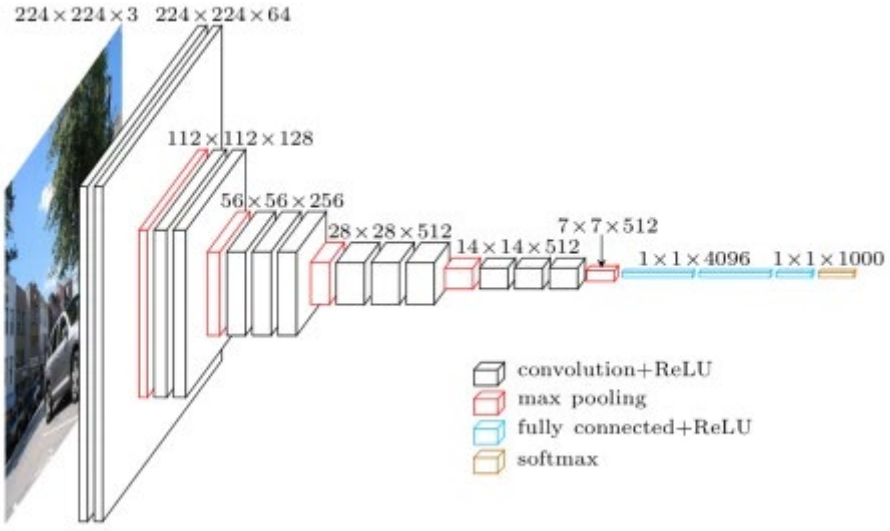


Figure 8 VGG architecture

According to the research, VGG-16 has achieved almost 93% test accuracy in ImageNet, where ImageNet is a massive dataset consisting of more than 14 million images from around 1000 different classes. Image input size to the model is 224 by 224.

From the 16 layers of VGG-16, 13 of them are convolutional layers and three of them are fully connected layers. Instead of using Softmax function in the hidden layers, VGG uses ReLU, also known as Rectified Linear Unit activation function. The output of this linear function is always positive or otherwise zero. It has three fully connected layers, the first two have 4096 channels each while the third has 1000 channels, each channel for each class.

From the architecture, the number of filters doubles every stack of the convolutional layer, from 128 filters to 512 filters. This makes VGG16 a huge network, training its parameters will take more time.

4.3. YOLOv5

Figure 9 YOLOv5 by Ultralytics

YOLO is an acronym for 'You only look once'. It is an object detection algorithm in computer vision to identify and localise objects in photos or video. YOLOv5 is the latest version, released on 18 May 2020, the first version, YOLOv1 was released on 8 Jun 2015. YOLOv5 is built with the Pytorch framework. Due to the speed and accuracy, YOLO became one of the most famous object detection algorithms today.

The YOLOv5 model is trained on the COCO (Common Objects in Context) dataset. It is a large-scale dataset specialising for object detection, segmentation, key-point detection and captioning dataset. It consists of up to 328k images. The dataset has annotations for each specific task, making training easier as the programmers do not have to make these annotations from scratch. For object detection, each image comes with bounding boxes and per instance segmentation masks with up to 80 classes. It contains more than 200k images and 250k person instances labelled with key points to indicate figures such as eyes, nose, hip, ankle etc for keypoints detection purpose. Since the format of the COCO dataset is automatically interpreted by advanced neural network libraries, this dataset is often used to benchmark algorithms to compare performances from real-time object detections.

4.4. JetPack 4.6.1

NVIDIA has released several JetPack SDKs for building end-to-end accelerated AI applications. All NVIDIA Jetson modules and developer kits are supported by JetPack SDK, such asJetson AGX Xavier, Xavier NX, TX2 and Nano. This SDK includes Jetson Linux Driver Package (L4T), equipped with Linux operating system and CUDA-X accelerated libraries and APIs for Deep Learning and Computer Vision.

JetPack 4.6.1 is the latest version released prior to JetPack 4.6. It is running the Linux Ubuntu 18.04 with Linux kernel 4.9. One of the key features in JetPack is TensorRT. It is built on CUDA, which enables programmers to further customise inference for deep learning frameworks. It is a high-performance inference runtime for object detection, image classification neural networks that delivers low latency in inference applications. The version of TensorRT equipped in JetPack 4.6.1 is TensorRT 8.2.1.

4.5. TensorFlow

Figure 10 TensorFlow

Google Brain first built DistBelief as a proprietary machine learning system based on deep learning neural networks in 2011. To simplify and refactor the codebase of DistBelief into a faster and more robust library named TensorFlow. TensorFlow is Google Brain's second-generation system released in 2017. It can run on several CPUs and GPUs with CUDA and SYCL extensions for general purpose computing on graphics processing units.

TensorFlow computations are expressed as stateful dataflow graphs, where the context and history will be stored, programmers can pick up where they left off without starting all over again. TensorFlow's market share among research papers shows a decline to the advantage of PyTorch, TensorFlow team released a new version of library in September 2019, TensorFlow 2.0. They change the automatic differentiation scheme from static computational graph to the more popular PyTorch, "Define-by-Run" scheme. This allows programmers to define conditions and loops into the network definitions easily, no need to predefine the network beforehand. They included cross-compatibility between other trained models on different versions of TensorFlow, making importing models more feasible, and improving the performance of TensorFlow on GPUs.

5. Methodology

The methodology will consist of 3 parts, the first part is to develop the TensorFlow model, second part is to develop the YOLOv5 model and lastly to combine these two models for inference in Jetson Nano.

5.1. TensorFlow model

5.1.1. Preparation of Dataset

i. Parking Lot Database

This database contains 12,417 images with resolutions of 1280x720 pixels, captured from two different parking lots, parking lots of the Federation University of Parana (UFPR) and the Pontifical Catholic University of Parana (PUCPR) in Curitiba, Brazil. In the folder, it also contains 695,899 images of segmented parking lots.

The images were taken with a low cost full HD camera, Microsoft LifeCam, positioned at the top of a building to have a great view of the parking lot. A five minutes time lapse interval for more than 30 days to capture images in different weather conditions including overcast, sunny and rainy periods.

To train the YOLOv5 model, each sample must have Extensible Markup Language (XML). This XML format describes the position and situation of each parking space in the image. Fortunately, this dataset has created this XML format, it can be utilised later when training the YOLOv5 model. The link for the dataset is as follow: http://www.inf.ufpr.br/vri/databases/PKLot.tar.gz

ii. Create Dataset

The ultimate goal of the TensorFlow model is to differentiate between occupied and empty spaces. Therefore, segmented images will be used to train this model. From the segmented dataset, all the images from UFPR04 and UFPR05 folders will be used for training, and the remaining PUC folders will be used for validation.

To ease the separation, the list of images will be randomised and recorded in a csv file. There are 81,406 images for training and 42,384 for validation. To avoid biases in training, the training set must have a balanced number of samples from each class. There are

43,027 of occupied samples and 38,379 of empty samples, roughly a 53% to 47% ratio, which is considerably adequate for training.

With the csv file created earlier, the samples are copied to another directory with separate training and validation folders accordingly.

5.1.2. Prepare and Train VGG16 CNN model

The VGG16 base model can be instantiated with the TensorFlow library. We will declare the include_top function to false, which means the 3 fully connected layers at the top of the network will be excluded, input_shape to height 49 and width 37. Before feeding input training samples to the model, these samples need to carry out VGG16 image processing, apply a rotation range of 90, horizontal flip and vertical flip randomly.

We will apply transfer learning techniques to train our model. The model summary generated shows 14,714,688 parameters to be trained. Therefore, the trainable layer will be frozen. We will then finally define the top and final layers, with the modified model as top, flatten, dense, 50% dropout and lastly dense layer. The final model summary should be like this, with 131,585 parameters to be trained:

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 49, 37, 3)]       0

block1_conv1 (Conv2D)        (None, 49, 37, 64)        1792

block1_conv2 (Conv2D)        (None, 49, 37, 64)        36928

block1_pool (MaxPooling2D)   (None, 24, 18, 64)        0

block2_conv1 (Conv2D)        (None, 24, 18, 128)       73856

block2_conv2 (Conv2D)        (None, 24, 18, 128)       147584

block2_pool (MaxPooling2D)   (None, 12, 9, 128)        0

block3_conv1 (Conv2D)        (None, 12, 9, 256)        295168
```

```
block3_conv2 (Conv2D)       (None, 12, 9, 256)        590080

block3_conv3 (Conv2D)       (None, 12, 9, 256)        590080

block3_pool (MaxPooling2D)  (None, 6, 4, 256)         0

block4_conv1 (Conv2D)       (None, 6, 4, 512)         1180160

block4_conv2 (Conv2D)       (None, 6, 4, 512)         2359808

block4_conv3 (Conv2D)       (None, 6, 4, 512)         2359808

block4_pool (MaxPooling2D)  (None, 3, 2, 512)         0

block5_conv1 (Conv2D)       (None, 3, 2, 512)         2359808

block5_conv2 (Conv2D)       (None, 3, 2, 512)         2359808

block5_conv3 (Conv2D)       (None, 3, 2, 512)         2359808

block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

flatten (Flatten)           (None, 512)               0

dense (Dense)               (None, 256)               131328

dropout (Dropout)           (None, 256)               0

dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 14,846,273
Trainable params: 131,585
Non-trainable params: 14,714,688
_________________________________________________________________
None
```

The model is then compiled with binary cross entropy loss, SGD optimizer with learning rate of 0.0001. The training is set to run for 10 epochs, and a callback is initiated if the result is not improving. Before running training, Weights & Biases are used to collect and record all the graphs and results of the training.

5.1.3. Validate model

After getting the saved model with trained weights, we have to validate our model. Before feeding the validation samples, we have to resize them to the same size as the input sample, which is dimension of height 49 by width 37, then apply VGG16 image processing. A ground truth table of prediction results is created.

5.2. YOLOv5 model for parking space detections

5.2.1. Preparation of Dataset

Instead of using segmented samples to train the YOLOv5 model, full parking lot samples are used. All the images in the dataset come with their xml files.

Same method as before, all samples will be recorded into a csv file, with their corresponding xml files. There are a total of 2338 images from UFPR folders and 430 images from PUCPR folders. All these images will be copied to one directory to ease us uploading to Roboflow. Roboflow is a computer vision platform that eases users to create annotations of their samples to a variety of formats for machine learning models. Roboflow allows users to create custom datasets and draw annotations. With these annotations, users can easily modify image orientations, resize and also perform data augmentation. Once uploaded to Roboflow, 80% of the samples will be used for training and 20% for validation. We first try with no preprocessing and augmentations, then with 90% rotation augmentations. Export the samples to YOLOv5 PyTorch format.

Figure 11 Roboflow platform

5.2.2. Prepare and Train YOLOv5

In the exported folder, there will be a data.yaml file and 2 folders, train and valid. The data.yaml contains the directory of the training folder and validation folder, we have to modify the directories for each folder. This file also shows the number of classes and names of each class, for our case, it is empty and occupied.

The YOLOv5 repositories are cloned and all the required libraries for YOLOv5 are installed. The data.yaml file is then copied into the data file in the YOLOv5 folder and renamed to 'parking.yaml'.

The model is now ready for training. Weights and Biases are logged on again to track and record all the results for the training. Finally, the model is trained with the parking.yaml file, run for at least 300 epochs. YOLOv5 can be resumed if the training is interrupted as all epochs will be recorded in the folder.

5.2.3. Detect with YOLOv5

After training, all the weights from each epoch and also the best weights of the training will be saved. The best weights will be used to run detection. Before running detection, the input sample should be resized to image size 640x640, same as the training image size.

5.3. Combine VGG16 with YOLOv5 model

5.3.1. Prepare YOLOv5 model

YOLOv5 object detection will be used to locate the coordinates of the bounding boxes for each car. Therefore, vehicle detection YOLOv5 model is needed for our case. Maryam Boneh has created the dataset required for training the YOLOv5 model in her github source. This dataset has a number of images including 5 classes, car, motorcycle, truck, bus and bicycle. The model is trained for at least 300 epochs as well. The weights of the model will be saved and be used later in the program.

5.3.2. Prepare Jetson Nano environment

Before doing anything to the machine, it is necessary to create a virtual environment. Jetson Nano is a rather small machine, it is better to use python virtual environment function instead of installing anaconda-like third party software. The machine should be kept as

minimum condition to make sure it runs in the optimum condition. Jetson Nano comes with Python 3.6.9. Some guidelines can be followed from this link when setting up Jetson Nano for the first time. Creating a virtual environment is crucial when deploying or creating any program as it allows developers and programmers to destroy the folder if the file is corrupted at some point in the process. It is also easier for python to locate all the libraries.

After creating a virtual environment, it is time to install all the libraries. Some of the required libraries are TensorFlow, Matplotlib, Open Cv, numpy, pandas, torch, Pillow, PyYAML, requests, scipy, torchvision and tqdm. From the YOLOv5 official github page, the requirement.txt file includes all the dependencies required, but it is only compatible with Python 3.7 and above. However, it is tested to work if you install the latest version of each library to Jetson Nano despite the Python version.

5.3.3. Create Code

The general idea is to use the YOLOv5 model to first detect the location of parking spaces, then crop every parking space to the model for prediction.

At first, the vehicle detection YOLOv5 model is loaded with the previous weights developed before. The model class is set to 0, corresponding to car class, as detection of the car is only required for this case. However, other classes can be added into the model if the targeted parking lot has different parking spaces such as trucks, motorcycles or even bicycles.

Assuming that the camera will be placed at the same position, the first picture of the parking lot is assumed to be full as well. The input image will be resized to 1280 by 720, then fed into the YOLOv5 model for vehicle detection. The results are printed in tensor format, it includes starting and ending of the bounding boxes coordinates, confidence and also class. The results are converted to pandas then to numpy arrays, saved in a coordinates.txt file.

Once the coordinates are obtained, the VGG16 model developed before is loaded. Before feeding the next image from the same camera, the image has to be resized to the same dimension as the first input image. If the dimension is different, the coordinates will not correspond to the input image. The coordinates.txt file is loaded and ready line by line to extract the coordinates. These coordinates will be used to crop areas from the new image to feed into the VGG16 model. If prediction is greater than 0.8, it will display a red box and be classified as occupied. If prediction is lower than 0.8, it will display a green box and be

classified as empty. A counter is added in each class to record the number of occupied and empty spaces. They will be displayed on the top left corner of the result.

5.3.4. Run inference on the Jetson Nano

The main goal is to run this program on Jetson Nano, utilising the camera module to detect parking spaces in real time. It is easy to install the camera module. For our case, a Raspberry Pi camera is used. There are two slots ready to be installed on Jetson Nano. The plastic cover holding the ribbon cable is lifted up. The ribbon cable of the camera is aligned to the slot and the cover is pressed down to hold the module in place.

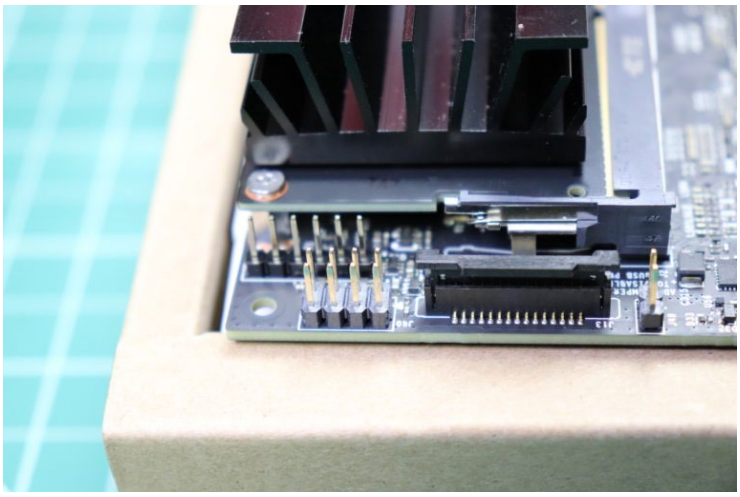


Figure 12 Slot for CSI camera module

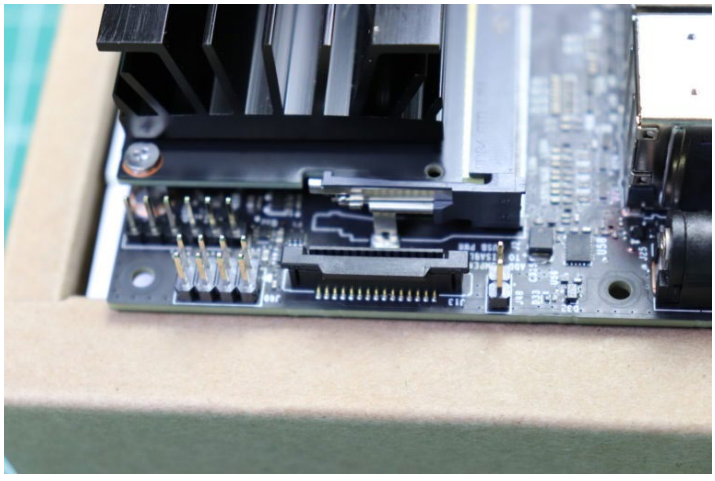


Figure 13 Lift up the cover of the slot

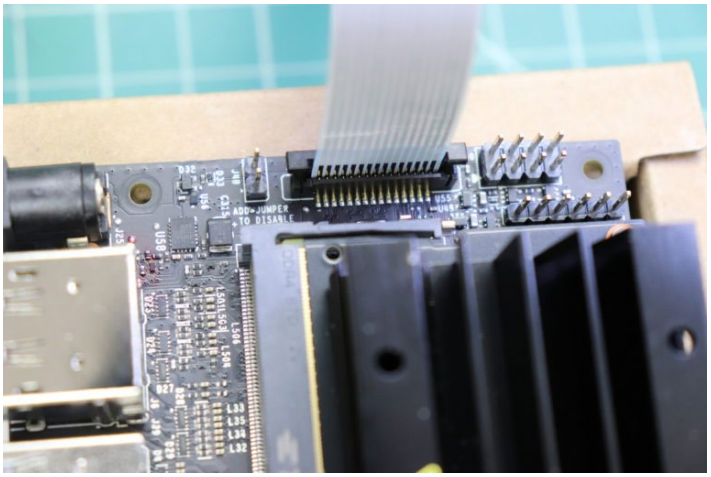


Figure 14 Align ribbon cable the module with the slot and close the cover

The code developed is modified to activate the camera. It is different from activating a camera from a laptop. The camera in Jetson Nano should be defined before calling the camera. The code can be copied from below:

```
def gstreamer_pipeline(
    capture_width=3280,
    capture_height=2464,
    display_width=820,
    display_height=616,
    framerate=21,
    flip_method=2,
):
    return (
        "nvarguscamerasrc ! "
        "video/x-raw(memory:NVMM), "
        "width=(int)%d, height=(int)%d, "
        "format=(string)NV12, framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
        "video/x-raw, format=(string)BGR ! appsink"
        % (
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
```

```
            display_height,
        )
    )
```

Instead of feeding images to the program, the user will be asked to take the photo by pressing a key. A while loop is created for the user to take the photo until the user is satisfied. The photo will be saved, resized and fed into the YOLOv5 model for first vehicle detection. Once the coordinates are recorded and saved in the text file, another loop will be created. Camera is activated again, a photo will be taken every 30 seconds, this photo will be fed into the VGG16 model for prediction. The results will be refreshed every iteration until the user decides to exit the loop and close the program.

5.4. Tools

This section will introduce some of the tools used in developing the program.

5.4.1. Google Colab

Figure 15 Google Colab

Google Colab is a free platform for students, data scientists or AI researchers to write and execute Python in a browser with no configuration needed and also comes with free GPUs. One of the advantages of Google Colab is the ability to link with a personal Google Drive account, making sharing to other users very easy.

5.4.2. JupyterLab

Figure 16 JupyterLab

JupyterLab is a free platform for users to work with documents and activities such as Jupyter notebooks, text editors, and terminals in an integrated and flexible environment. Users can integrate documents and activities side by side, enhancing workflows for interactive computing. For example, it comes with code consoles where users can run code interactively and get rich output. It can be linked to a notebook kernel to retrieve a computation log from the notebook. It is also compatible with Kernel-backed documents like Markdown, Python, R, LaTex and other text files.

JupyterLab is equipped with the Jupyter Notebook App. It is a server-client application that allows users to edit and run notebook documents via a web browser. It is executed on a local desktop, without the need of internet access. It can be installed on a remote server and accessed through the internet. A notebook kernel uses a “computational engine” to execute python code. Therefore, it is highly reliant on the local CPU and RAM. The more complicated the computations, the more significant the kernel may consume CPU and RAM, the more powerful the machine required. Unlike Google Colab, the kernel will be shut down if users idle for a period of time, the kernel of Jupyter will not be released unless the kernel is shutdown.

5.4.3. Jetson Nano

Figure 17 NVDIDIA Jetson Nano Developer Kit

Jetson Nano is a small, powerful computer developed by NVIDIA, that allows users to run multiple neural networks in parallel for image classification, object detection, segmentation and other applications. To boot the kit, a microSD card with JetPack SDK system image is required. JetPack is compatible with the NVIDIA AI platform, both for training and deploying purposes.

Jetson Nano has a 128-core Maxwell GPU, a Quad-core ARM A57 CPU, 4GB 64-bit LPDDR4 RAM, 2 MIPI CSI-2 DPHY lanes for camera modules, a Gigabit Ethernet for internet connection, HDMI and display port, 4 USB 3.0 ports and a USB 2.0 Micro-B port. A monitor, a mouse, and a keyboard are needed to boot the computer.

5.4.4. Raspberry Pi Camera Module 2

Figure 18 Raspberry Pi camera Module V2

This Raspberry Pi camera module has a Sony IMX219 8 megapixels sensor. It supports 1080p30, 720p60 and VGA90 video modes and also still capture. A 15cm ribbon cable to the CSI port is attached to this module.

5.5. Gantt Chart

UNIVERSITI TEKNOLOGI PETRONAS

SECTION A: SIP TRAINING SCHEDULE

Student's Name: Hong Jowell **Student no:** 18002394 **Programme:** EE

Place of Training : Telecom SudParis **Period of Training:** 7 months

Department	Training activities	Week No/ Date													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Research	VGG16 CNN Training	✓	✓	✓											
	YOLOv5 Training			✓	✓	✓	✓								
	Discussion	✓		✓			✓								
	Code Writing							✓	✓	✓					
	Report								✓	✓	✓	✓			
	Presentation												✓		

(Please return this form to SIIU within three weeks after student's registration) *(Make copies if necessary)*

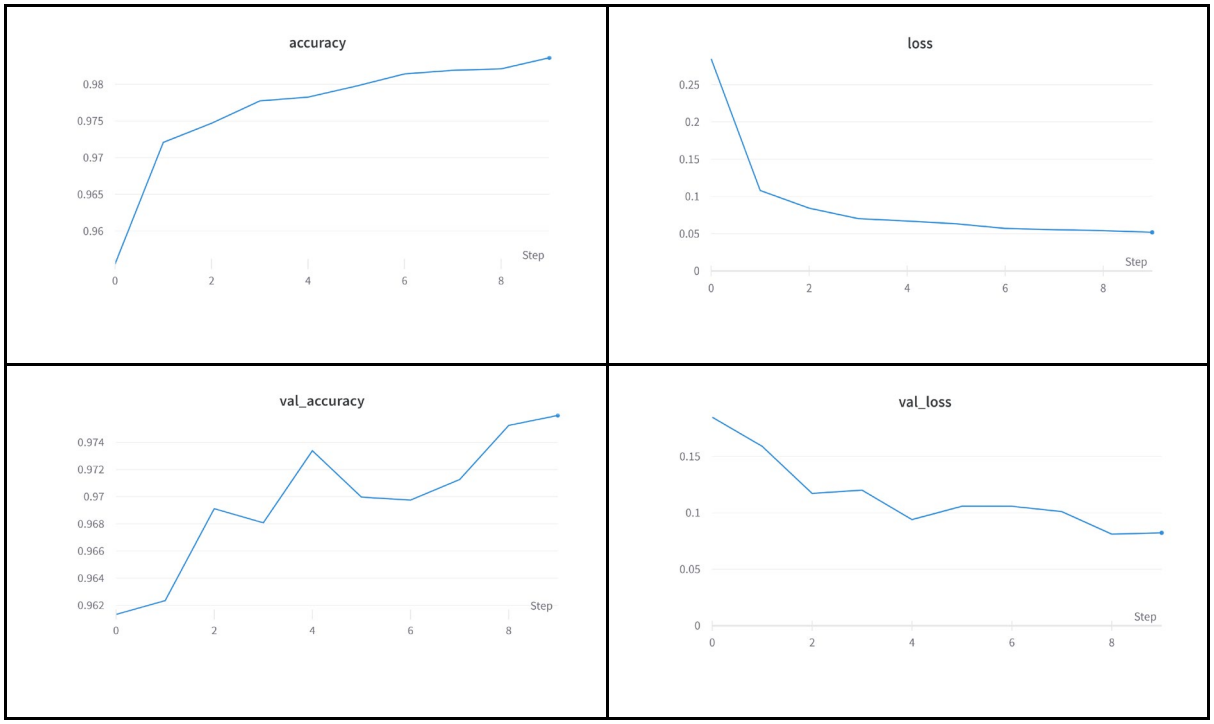
Host Company Supervisor Signature &stamp:
Name:
Designation:
Date:

6. Results and Discussion

6.1. VGG16 model

The model is trained for 10 epochs using a 4 core CPU only machine from Microsoft Azure. The training using this machine consumed 11 days 17hours 22 minutes and 43 seconds without any GPUs.

Overall results:



With every epoch increasing, loss decreased and accuracy increased. Validation loss decreased and validation accuracy increased, this depicted that the model built was learning and working fine. In other scenarios, if validation loss starts increasing and validation accuracy decreases, the model is possibly not learning and cramming values. If validation loss starts increasing and validation accuracy also starts increasing, the model is probably overfitting.

From the graph, the best epoch is at 8th epoch, with accuracy of 0.9836 and validation loss of 0.08108. The model is tested on this sample picture:

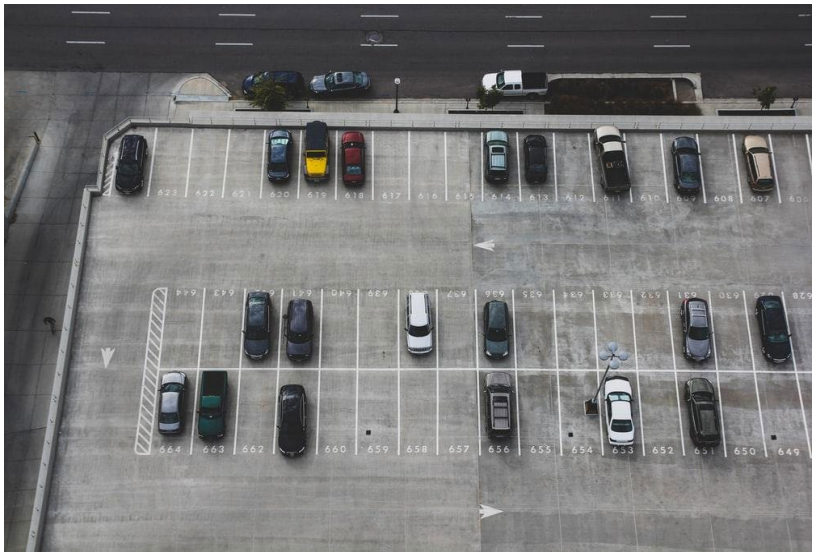


Figure 19 Parking Lot 1

To determine these parking spaces manually, a software called GIMP-2.10 is used to find out the coordinates of each parking space. I tried to find the pattern, space width of each parking space, using simple computations to locate the next parking space. With these coordinates, this area of image is cropped and resized to height 49 by width 37, then fed into the model for prediction. If prediction exceeds 0.8, a red box will be shown and vice versa a green box will show if prediction is lower than 0.8.

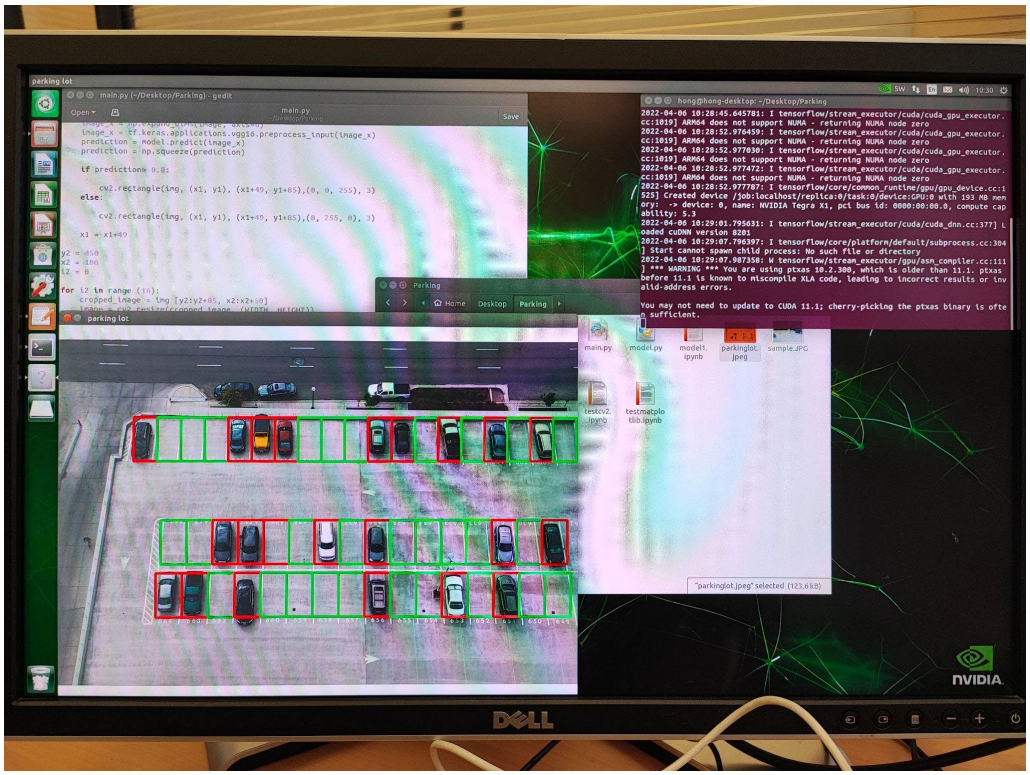
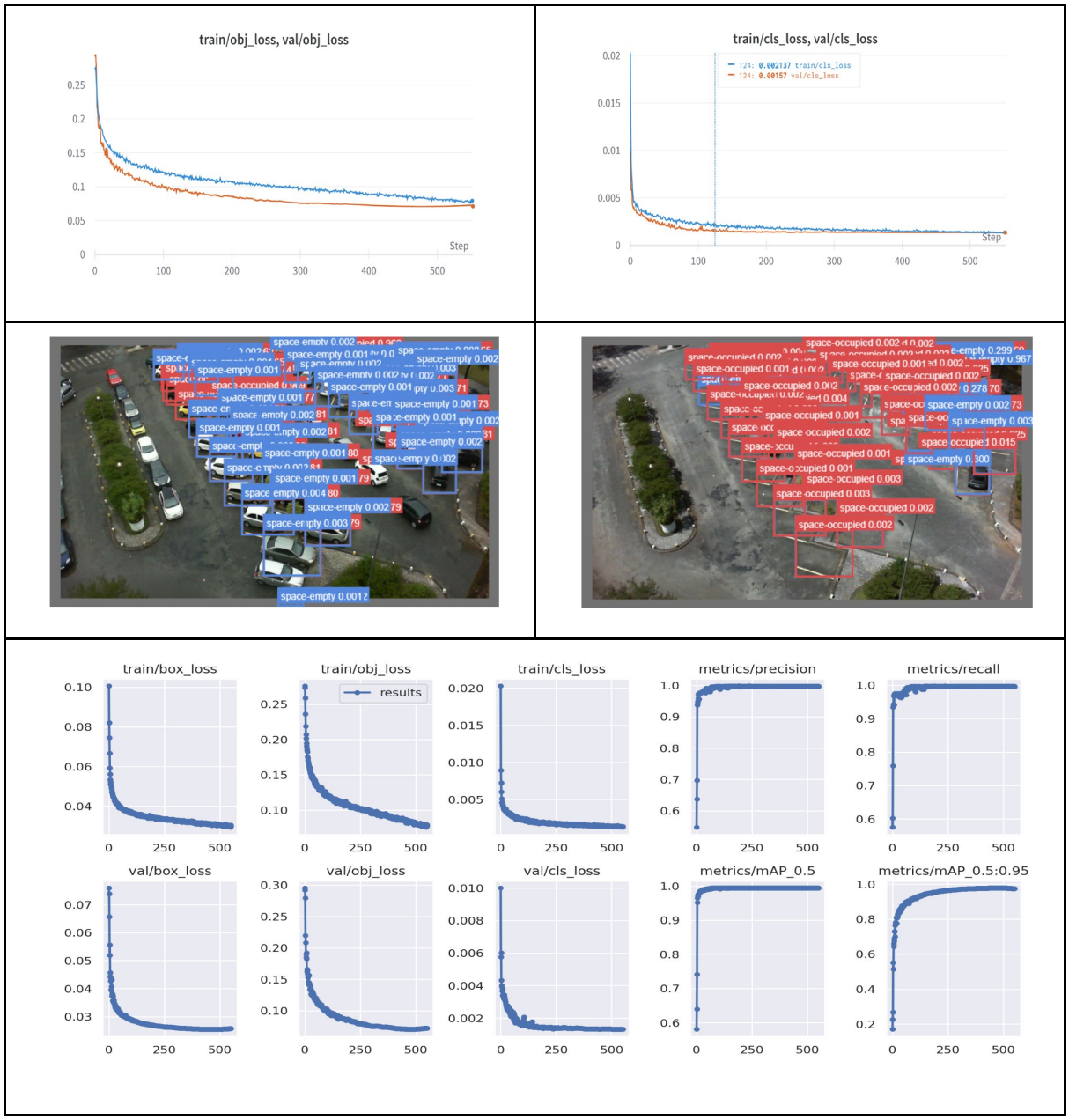


Figure 20 Parking lot 1 detection result

6.2. YOLOv5 model

The amount of manual labour to locate every parking space is expensive and not ideal. An idea came up, which is to use an object detection algorithm to detect parking spaces. There are 2 classes to be detected, “Occupied” and “Empty”. The training was run with a 64-core CPU and a GPU NVIDIA RTX A5000. The first training was run with no augmentations and modification to the dataset, it took 16 hours 20 minutes and 29 seconds. The training was stopped at 539th epoch as there was no improvement to the training, early stopping was initiated.

Training results:



From the graph, it showed decent results where ideally the model should behave properly. The best epoch was selected at 451st epoch, with overall precision with 0.997. The model with the best weight is then used for detection.

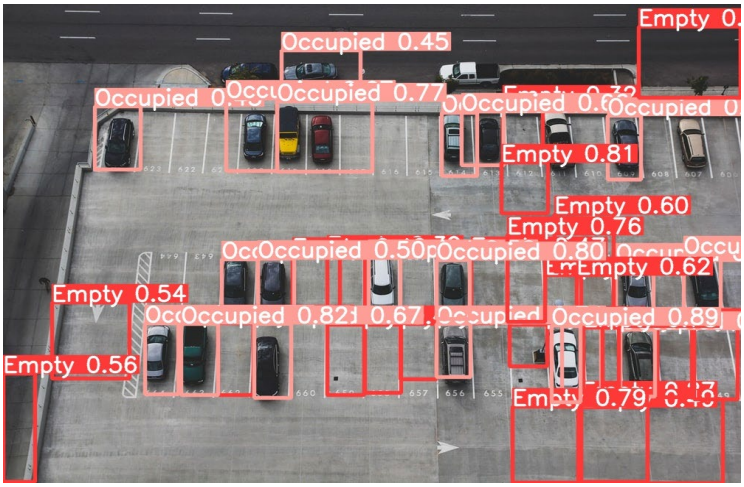
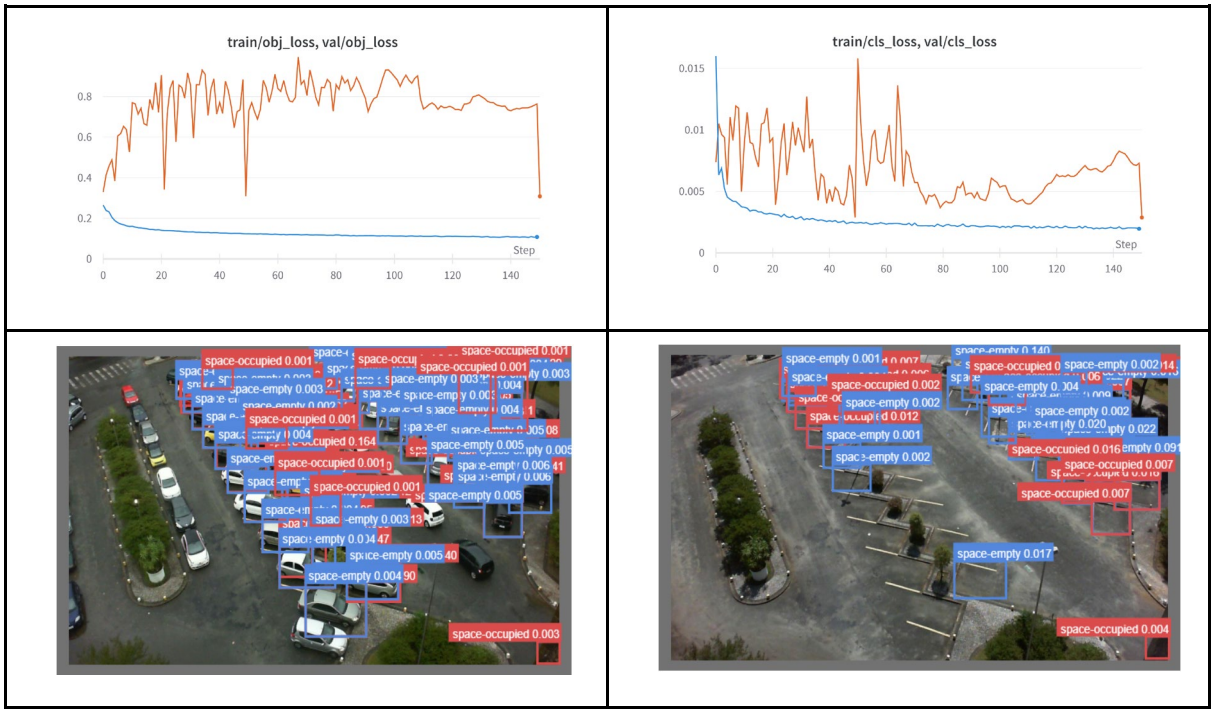
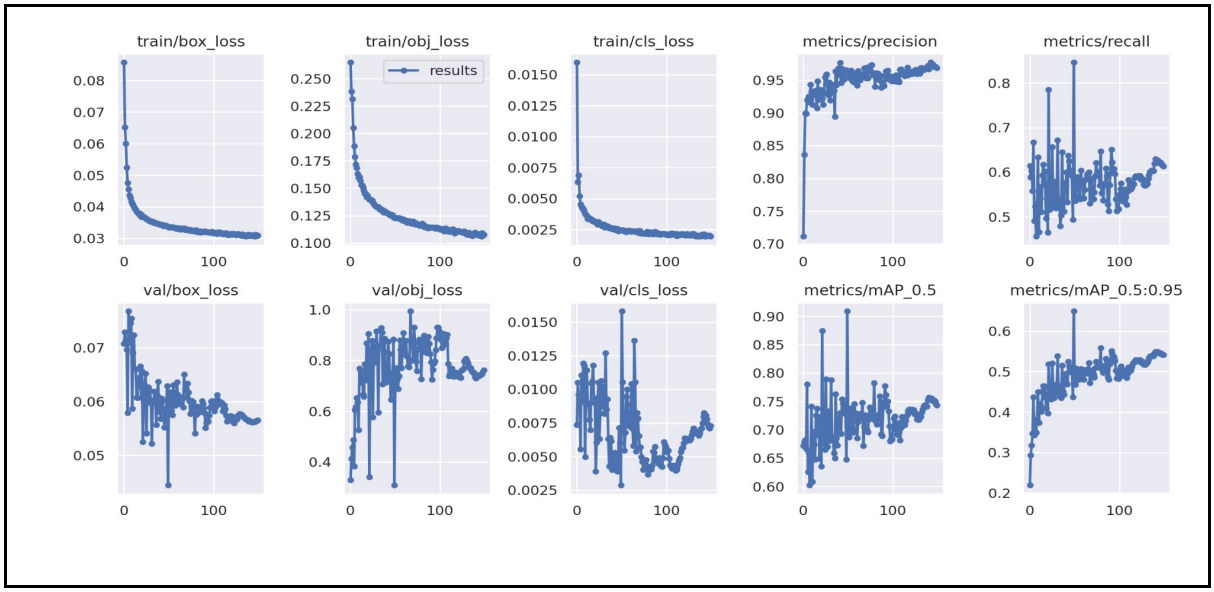


Figure 21 First YOLOv5 detection result

It can be seen that the model struggles to differentiate the empty spaces and the street due to the similarity in colour and shape. To help the model to learn better, I did 90 degrees augmentations randomly to the parking spaces to diversify the dataset. The second training was done with the same machine in 13 hours 4 minutes and 55 seconds. The training stopped at the 151st epoch since no improvement was seen.

Training result:





From the graph, it can be seen that the validation obj_loss and cls_loss were noisy, they both fluctuated a lot. This could mean that the model is not learning. However, the algorithm depicted the best weight at 49th epoch, with overall precision of 0.9597. The best weight is used for detection test.

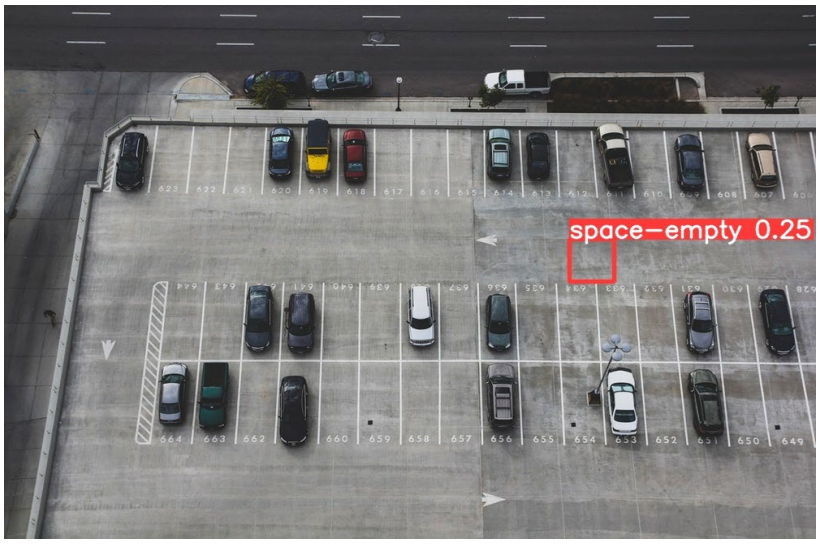


Figure 22 Second YOLOv5 detection result

It can be concluded that the model is not learning as the model struggles to detect occupied spaces as well. It could also because of the size of the dataset. The dataset only includes 2 parking lots, taking from several angles. The model can be improved by increasing a few more parking lots, increasing the varieties, decreasing the biases of the model. However, YOLOv5 is a relatively fast model for object detection tasks. We can utilise these bias properties for the case, but it requires a lot of work before deployment. For instance, if

the program is to detect only this particular parking lot, we can use this parking lot to create a dataset, labelling all the parking spaces. The model is then trained with this parking lot, without the need for augmentations. We know that the model is biased to this parking lot only, the parking spaces are fixed all the time, the prediction will always be accurate.

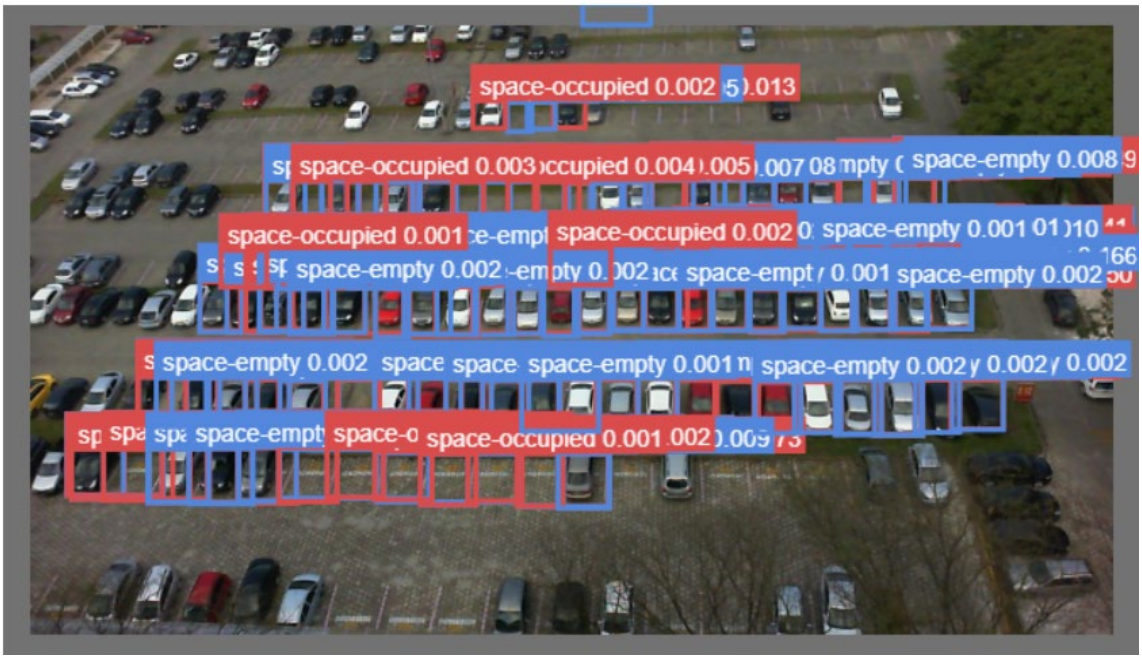
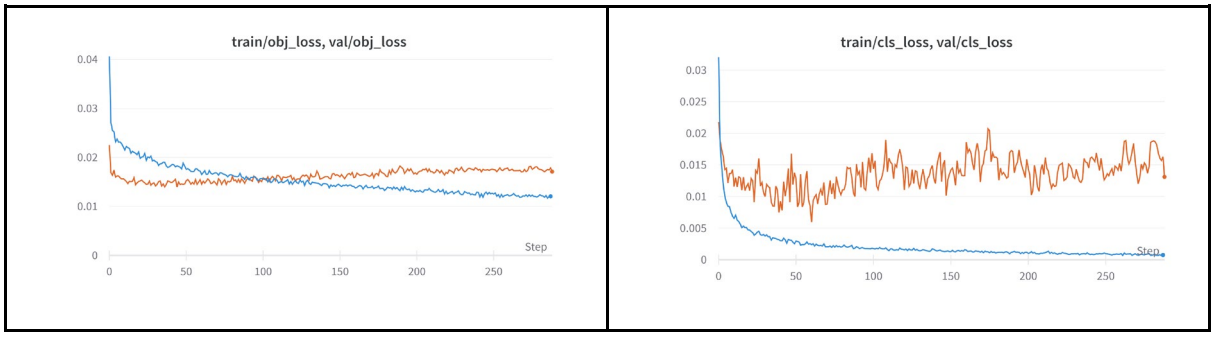


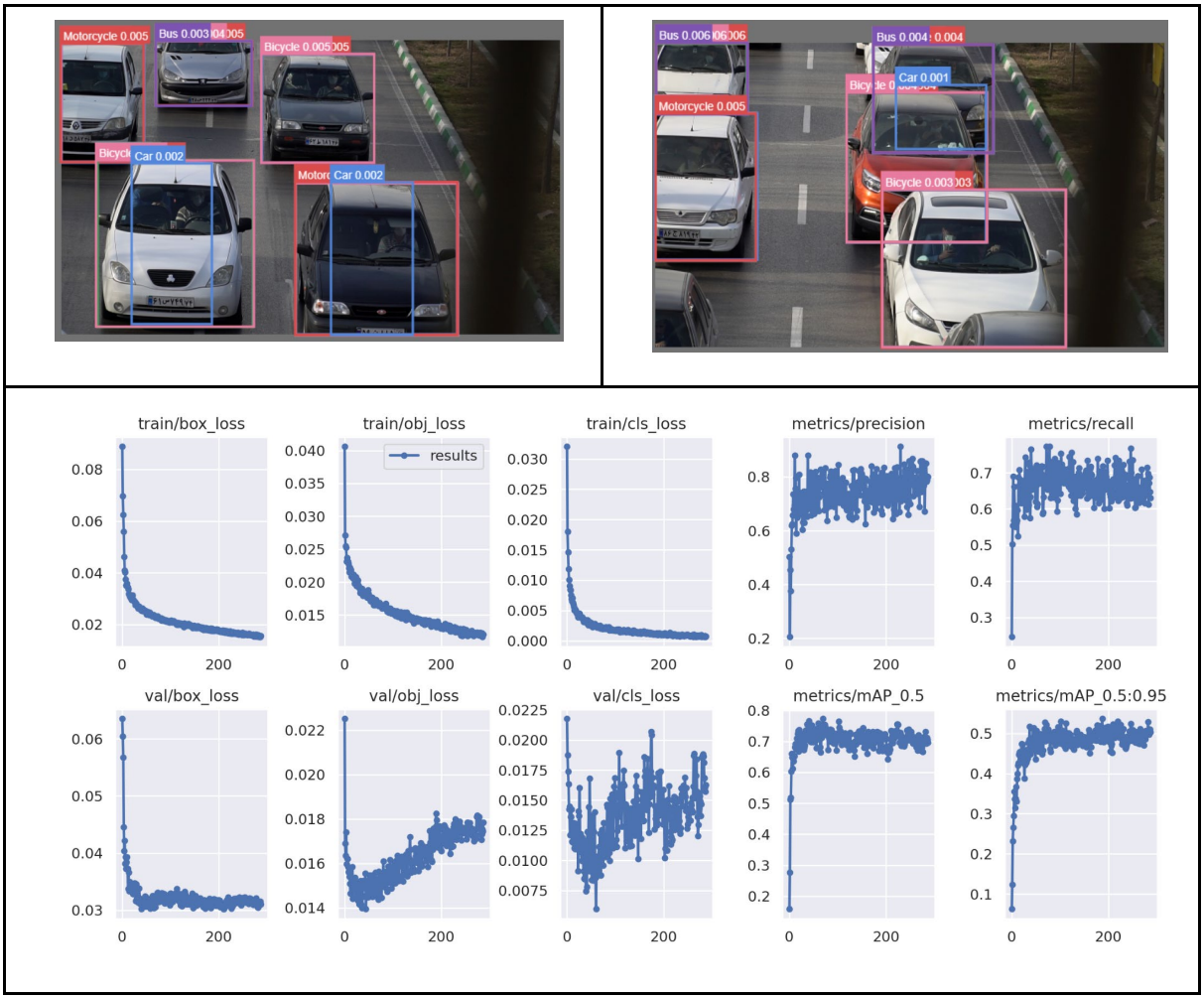
Figure 23 Detection of YOLOv5 on sample from dataset

6.3. YOLOv5 and VGG16 model

To semi automate the program, we decided to combine the object detection algorithm with the occupancy detection algorithm. We first use object detection to locate the parking spaces, with all these known locations, the occupancy detection algorithm can predict the occupancy. Assuming the parking lot is full during the first deployment, YOLOv5 is used first for vehicle detection. The training was run on the same machine, it stopped at 289th epoch. It took 2 hours 3 minutes and 37 seconds.

Training result:





The best weight is obtained at the 187th epoch, with overall precision of 0.7711. The model is tested for vehicle detection.

Figure 24 YOLOv5 vehicle detection result

Since the model is working perfectly, we began to develop the program. The program is tested with this sample picture:

Figure 25 Parking lot 2

The program is deployed for vehicle detection with YOLOv5, the coordinates are saved to a text file.

The output:

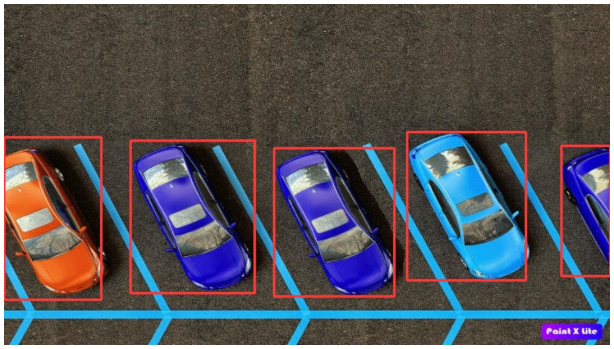


Figure 26 Parking lot 2 detection result

coordinates.txt

```
851.7301635742188 270.29473876953125 1102.6671142578125 583.3836059570312 0.9312772154808044 0 Car
1177.21533203125 299.32281494140625 1280.0 574.0947265625 0.8254418969154358 0 Car
570.0693969726562 304.0845947265625 824.6968383789062 617.4840087890625 0.7984285354614258 0 Car
267.2314453125 288.02410888671875 529.2528686523438 610.4918212890625 0.7849224209785461 0 Car
0.12139892578125 281.11065673828125 205.53707885742188 624.5782470703125 0.3456573784351349 0 Car
```

Figure 27 Coordinates.txt file

The YOLOv5 model successfully detected 5 cars in the image and all the coordinates are recorded in the “coordinates.txt” file. Each row represents each detection result, it shows its

x1, y1, x2, y2, confidence and class accordingly. We then removed one of the cars from the image for the parking space detection test.

Sample:

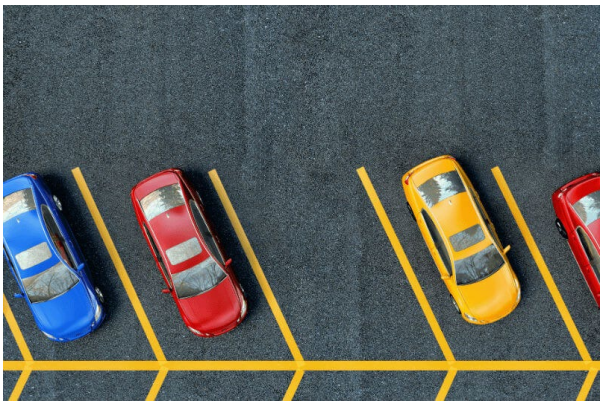


Figure 28 Parking lot 2 with one car removed

The output:

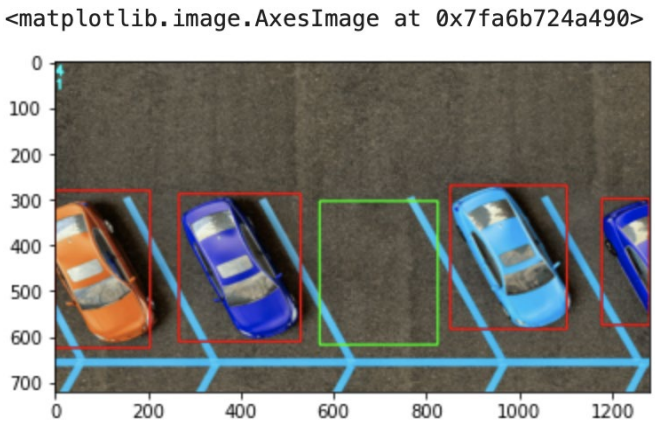


Figure 29 Result from the developed program, top left corner indicates the number of occupied and empty spaces

The program used the coordinates from the text file, cropped them from the image and fed into the VGG16 model for predictions. The result of the image is resized and displayed. The number of occupied and empty spaces are displayed on the top left corner of the image.

This program is very easy to execute even for small machines like Jetson Nano. It only requires simple computation to execute the program, which is ideal for Jetson Nano. The advantage of this program is it is not restricted by the variety of parking lots, it can work in every parking lot. It can even work in parking lots with no parking boxes at all. The only downside of this program is it has to be deployed when the parking lot is full. The parking lot has to be full to record the locations of the parking spaces.

7. Discussion on Sustainability

7.1. Environmental

The impact of electrical gadgets like sensors has been huge to the environment, both good and bad. As the number of vehicles increases over time, the demand of building more parking lots increases, more sensors are built these days. When the sensors are beyond maintained or repaired, they ended up become useless, these wastes are also known as E-waste. As technology advances, E-waste grows rapidly due to old and outdated electronics devices. These wastes contain extremely toxic chemicals, such as lead, cadmium, arsenic, mercury, vinyl chloride and other elements that could harm the planet if they are not handled carefully. Therefore, by implementing computer vision in applications could reduce the demand of building new electronics, by using the current equipment, deploying new algorithm, making our environment more sustainable, in conjunction with the Sustainable Development Goals (SDGs) 11, “Sustainable Cities and Communities”.

7.2. Social

In correspondent to Industrial Revolution 4.0, global production has been shifting from traditional manual labour to modern automation by joining the technologies such as artificial intelligence. This integration results in better efficiency, self-monitoring, without the need of human intervention to diagnose issues. This modernisation could bring a lot of benefits to the society, especially computer vision. For example, self-driving cars utilise computer vision to make sense of their surroundings, such as traffic signs, obstructions and pedestrians, enhancing the safety of the driver due to fatigue from long driving. Other than that, people use computer vision in healthcare, helping the automation of detecting cancer in x-ray and MRI scans. Computer vision has always been a great impact to the society. Although this field has been years in development, to create a machine that can see like human is difficult task, but people can always use creativity to invent innovations to improve daily lives.

7.3. Economy

What is circular economy model? Circular economy is an economic growth that is substantial not only for the planet, but it also aims to eliminate waste, pollution, encourages keeping products and materials in use without intertwining consumption of the finite

resource. This can spur more innovations, to resolve global challenges and also create job opportunities. Artificial intelligence plays an important role in this circular economic shift. It enhances and facilitates innovation across industries in three ways, design, operate and optimise. New jobs are generated across the global economy in this shift, more openings for skilled professionals are created.

8. Reference & citation

1. Acharya, Debaditya, et al. “Real-Time Parking Lot Occupancy Detection Using Deep Learning.”
2. Almeida, P., Oliveira, L. S., Silva Jr, E., Britto Jr, A., Koerich, A., PKLot – A robust dataset for parking lot classification, Expert Systems with Applications, 42(11):4937-4949, 2015.
3. Bhattacharyya, Jayita. “Step by Step Guide to Object Detection Using Roboflow.” Analytics India Magazine, 2 Dec. 2020, https://analyticsindiamag.com/step-by-step-guide-to-object-detection-using-roboflow/.
4. Birchenko, Y. (n.d.). Smart parking solutions - IOT sensors space race. Farnell. Retrieved June 28, 2022, from https://il.farnell.com/smart-parking-solutions-the-iot-sensors-space-race
5. Cho, J. (2020, January 11). Define-and-run vs. define-by-run. Medium. Retrieved June 28, 2022, from https://medium.com/@zzemb6/define-and-run-vs-define-by-run-b527d127e13a
6. “Fourth Industrial Revolution.” Wikipedia, Wikimedia Foundation, 24 June 2022, https://en.wikipedia.org/wiki/Fourth_Industrial_Revolution.
7. Google Colab, Google, https://colab.research.google.com/?utm_source=scs-index#scrollTo=5fCEDCU_qrC0.
8. Helaman. “Programmer Group.” Jetson Nano Deployment Process Record: yolov5s+TensorRT+Deepstream Detects Usb Camera, https://programmer.group/yolov5s-tensorrt-deepstream-detects-usb-camera.html.
9. JetPack SDK 4.6.1 | NVIDIA Developer. Retrieved June 28, 2022, from https://developer.nvidia.com/embedded/jetpack-sdk-461

10. “Jetson Nano Developer Kit.” NVIDIA Developer, 14 Apr. 2021, https://developer.nvidia.com/embedded/jetson-nano-developer-kit.
11. Kangalow, et al. “Jetson Nano + Raspberry Pi Camera.” JetsonHacks, 29 Feb. 2020, https://jetsonhacks.com/2019/04/02/jetson-nano-raspberry-pi-camera/.
12. MaryamBoneh. “MaryamBoneh/Vehicle-Detection: Vehicle Detection Using Deep Learning and Yolo Algorithm.” GitHub, https://github.com/MaryamBoneh/Vehicle-Detection.
13. Overview - JupyterLab 3.4.3 Documentation, https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html.
14. Raspberry Pi. Raspberry Pi, https://www.raspberrypi.com/products/camera-module-v2/.
15. Saha, S. A Comprehensive Guide to Convolutional Neural Networks — the . Retrieved June 28, 2022, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
16. Sciforce, Sciforce. “Artificial Intelligence and the Shift to the Circular Economy.” Medium, Sciforce, 18 Mar. 2021, https://medium.com/sciforce/artificial-intelligence-and-the-shift-to-the-circular-economy-d439c9c82e3c.
17. TensorFlow - Wikipedia. Retrieved June 28, 2022, from https://en.wikipedia.org/wiki/TensorFlow
18. The Jupyter Notebook - Jupyter Notebook 6.4.12 Documentation, https://jupyter-notebook.readthedocs.io/en/stable/notebook.html.
19. The World Counts, https://www.theworldcounts.com/challenges/planet-earth/waste/electronic-waste-facts.
20. VGG Net | Build VGG Net from Scratch with Python. Retrieved June 28, 2022, from https://www.analyticsvidhya.com/blog/2021/06/build-vgg-net-from-scratch-with-python/
21. What is Computer Vision? | IBM. Retrieved June 28, 2022, from https://www.ibm.com/topics/computer-vision
22. YOLOv5 Documentation. Retrieved June 28, 2022, from https://docs.ultralytics.com/