Using Depth information in the Faster R-CNN architecture for 2D object detection

Student Name: Mohammad Bilal Chaudhry

Supervisor Name: Khalid Ismail

Submitted as part of the degree of MEng Computer Science to the

Board of Examiners in the Department of Computer Science, Durham University.

Abstract –

Context: Object detection is one of the key tasks in computer vision. As autonomous vehicles and robots become more popular, we need to research, implement and adopt better methods for object detection, which can be used by autonomous vehicles and robots to understand their surroundings. One of the best object detection frameworks being used today is Faster R-CNN [1]. Literature [15][16] on object detection algorithms suggests that we can increase accuracy of our detectors by using additional information like depth. A question arises: "*Is using Depth information in Faster R-CNN worth the extra computational cost for 2D object detection?*"

Aims: The main aim of the project is to answer the question above by implementing different methods through which we can incorporate depth information in the Faster R-CNN architecture and then comparing them with each other and with the baseline RGB Faster R-CNN. An overarching aim of the project is to contribute to the research area by making it easier to carry out further research on Faster R-CNN by using and proposing updates to an excellent existing GitHub repository implementing Faster R-CNN architecture to support multi-modal fusion.

Method: We use the KITTI2012 2D object detection dataset [2] for the RGB images and we produce depth images using Pyramid Stereo Matching Network (PSMNet) [3]. We use ResNet-50 [4] and VGG-16 [5] Base Networks to implement, evaluate and compare 4-Channel fusion, Early-Layer Fusion and Mid-Layer Fusion approaches using 'Addition', 'Averaging' and 'Maximum' fusion layers.

Results: We show that incorporating depth information improves the accuracy of Faster R-CNN and we provide a comparison, comprised of accuracy, speed and resource utilization, of the different ways depth information can be incorporated in Faster R-CNN. This comparison can be used as a reference guideline for researchers interested in using Faster R-CNN in their systems. We show that different set-ups have different effects on the performance of Faster R-CNN. We also hypothesize that the concatenation of middle layer features leads to higher accuracy in VGG-16 based Mid-Layer fusion approach.

Conclusion: Based on our results, we conclude that Faster R-CNN does perform better and achieves higher accuracy when depth information is incorporated in it. We observe that each fusion approach and fusion layer has varying effects on different Base Networks. Faster R-CNN is one of the best object detection frameworks in use today and in this project, we contribute to the research area by providing a detailed comparison of some of the best methods of incorporating depth information in Faster R-CNN.

Keywords – Machine Learning, Deep Learning, Computer Vision, RGB-D, KITTI, ResNet, VGG, Faster R-CNN, object detection.

I. INTRODUCTION

Our eyes and brains can detect, locate and identify objects present in our surroundings

with ease and can quickly and accurately perform complex tasks such as detecting obstacles with little conscious thought.

Object detection is one of the main fundamental tasks in computer vision. In simple terms, the goal of 2-Dimensional (2D) object detection is to detect all instances of objects from a known class, like people, animals, vehicles or faces in an image. Note that the detection of objects in an image corresponds to not only classification but also the localization of objects in an image. This is important as we not only want to know what objects are in the image but also exactly where they are in the image.

Object detection is applied in several areas of computer vision, including security, autonomous vehicles, robotics, machine inspection, surveillance and image retrieval. Our object detection algorithms need to be able to explore the images and find all objects we might be interested in at all possible locations and scales in which they might appear in an image to create robust object detectors. A few years ago, the 2D object detection problem was a hard problem to solve as coming up with robust and accurate algorithms to identify objects in an image and to classify them which allow real-time, accurate and fast object detection was difficult. But with the advent of Neural Networks (NN), specifically the Convolutional Neural Networks (CNN) as well as faster and more powerful computers, things have changed drastically for the better. For example, now not only can we identify a vehicle in an image, but we can also identify what kind of a vehicle it is. This in-depth semantic understanding of objects in images/frames is important for computer vision-based systems to have a thorough understanding of images and/or their surroundings.

In general, object detection algorithms work by computing a model for an object class from the provided training examples and then checking different locations in an image to find whether an object is present at a particular location using the computed models. In primitive object detection techniques (e.g., SIFT, HOG, bag-of-words), these models/filters had to be hand-engineered. More recently, 2D object detection has benefited from data-driven representation learning, where massive image datasets (e.g. ImageNet [6] and Places [7]) provide the necessary data to effectively train complex CNNs, which can learn models/filters/features of object classes by themselves by continuously tuning their millions of parameters. The features extracted from models pre-trained with those datasets are generic and powerful enough to obtain state-of-the-art performance in 2D object detection. Therefore, the best methods to solve the 2D object detection problem today involve deep learning. There are several deep learning object detection frameworks in use today. Some of the most prominent deep learning object detection frameworks in use today are either based on the Region-based Convolutional Neural Network (R-CNN), proposed by Ross Girshick et al. [8], or You-Only-Look-Once (YOLO), proposed by Joseph Redmon et al. [9].

A. Project Aims

In this project we will be focusing on one of the most important and popular object detection frameworks called Faster R-CNN which is a member of R-CNN family and is based on the original R-CNN architecture developed by Ross Girshick et al. Usually object detection frameworks are evaluated and compared after being trained on images which have 3 channels (RGB). A research question arises:
"*Is using Depth information in Faster R-CNN worth the extra computational cost for 2D object detection?*"

In this project, we attempt to answer this question by doing the following:

- We use an excellent existing Faster R-CNN implementation and update it to not only support fusion of multiple channels together but also the different ways in which fusion can take place.
- We use two Base Networks in our experiments which are ResNet-50 and VGG-16.
- We use KITTI2012 2D object detection dataset and we use PSMNet to create high quality depth images.
- We focus on detecting 3 different classes in our experiments; Pedestrian, Cyclist and Vehicle.
- We implement the 4-channel, Early-Layer and Mid-Layer fusion approaches.
- We implement 3 fusion layers which are Summing, Averaging and Maximum.
- We then provide a thorough comparison of the different methods of incorporating depth information in Faster R-CNN which involves comparing accuracy, speed and resource utilization.

B. Motivation

Object detection is one of the main focuses in computer vision. Facial recognition, which is a subset of object detection, alone is predicted to be worth a $7.76 Billion industry by 2022 [10]. This shows how important object detection really is. We concluded from our literature survey that there was a lack of research papers focusing on multi-modal fusion in Faster R-CNN. We attempt to make further research into Faster R-CNN easier by proposing updates to a Faster R-CNN repository on GitHub to support most important Base Networks (e.g., ResNet-50, ResNet-100, VGG-16, ResNet-34, and SqueezeNet [11]) and different methods of multi-modal fusion. Using the updated repository, one can also use other sources of information like Infrared Radiation and X-Rays instead of depth information to train and evaluate Faster R-CNN. We contribute to the research area by providing results that researchers can not only take advantage of but also use to carry out further research.

II. RELATED WORK

A. 2D Object Detection using Machine Learning

There are two important and popular methods of object detection which use machine learning. The first method involves standard non-deep learning methods such as a sliding window and image pyramids, this method is typically used in the HOG + Linear Support Vector Machine (SVM)-based object detectors. The problem with this method is that it is not a pure end-to-end deep learning object detector and involves sliding a fixed size window from left-to-right and top-to-bottom, across an image, to localize objects at different locations. It uses image pyramid to detect objects at varying scales. At each stop of the sliding window and the image pyramid, we extract the Region of Interest (ROI), feed it to a pre-trained CNN to detect an object if there is any. Feeding large number of ROIs to CNNs makes this method slow. We need an intelligent way of choosing ROIs.

The second method involves taking a pre-trained CNN and using it as a Base Network in a deep learning object detection framework. This method is regarded as the best method to perform 2D object detection in images and is also used in state-of-the-art scene understanding algorithms used in autonomous vehicles and robots. It is faster and outperforms the previous method as it involves deep learning. Instead of a sliding

window and image pyramids, we have a Region Proposal Network (RPN) which intelligently selects interesting areas in an image where the objects that we are looking for might be. Then we use these ROIs and feed them to the Base Network for object classification and bounding box refinement. This method reduces the number of times we have to feed the ROIs to the CNNs; therefore, it outperforms the previous method. Faster R-CNN is one of the many architectures which uses this approach.

The speed of evaluation is important in 2D object detection as we need to be able process large number of Frames per Second (FPS) for fast travelling autonomous vehicles to make the semantic understanding of the surroundings accurate and lag-free.

B. The R-CNN object detection framework family

The R-CNN architecture, which stands for Region-based Convolutional Neural Network, is the first and the simplest architecture in the R-CNN family. It involved using the Selective Search algorithm to compute 2000 region proposals and using a pre-trained CNN on each of these region proposals to extract useful features. These features from the CNN are fed into a Support-Vector Machine (SVM) to classify the region and a linear regressor is used to tighten the bounding box around the object, if an object we want to detect exists in that ROI. This architecture is intuitive, accurate but slow. It is the predecessor of several object detection architectures which form the R-CNN family. Figure 1 outlines steps involved in the R-CNN architecture.

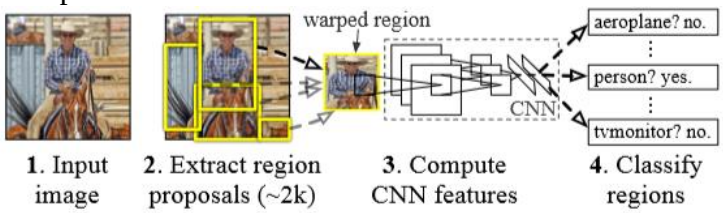

Figure 1: The R-CNN model [8]

Ross Girshick, realizing the performance issues with R-CNN, soon proposed the Fast R-CNN architecture [12]. Fast R-CNN improved on its detection speed compared to the R-CNN architecture. Girshick made this possible by:

1. Performing feature extraction over the image *before* proposing regions.
2. Extending the architecture for detecting objects by replacing the SVM with a SoftMax classification layer.

Now, the Fast R-CNN architecture involved inputting an image and multiple ROIs into a fully convolutional neural network after which each ROI is pooled into a fixed-size feature map and then mapped to a feature vector by the Fully Connected Layers (FCs). The network, then produces two output vectors per ROI, SoftMax probabilities and per-class bounding-box regression offsets. This architecture is faster as now we are generating region proposals based on the last feature map of the network; therefore, now, unlike in the R-CNN, we just need to train *one* CNN for the entire image. Moreover, now we don't need to train several SVMs to classify each object as we can simply get the class probabilities from the SoftMax layer. Figure 2 outlines the steps required in Fast R-CNN.

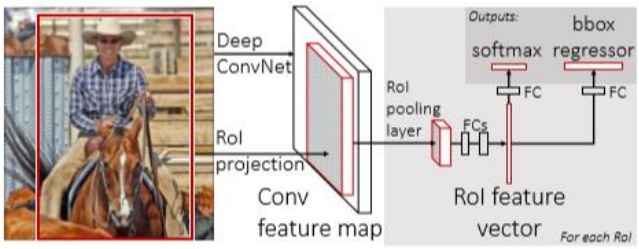

Figure 2: Fast R-CNN architecture [12]

The Fast R-CNN outperformed R-CNN, but it was still too slow to allow real-time object detection. Ren at al. proposed the Faster R-CNN architecture which solved this problem by replacing the slow selective search algorithm with a Region Proposal Network (RPN). We describe the architecture of Faster R-CNN in the Solutions section. The Faster R-CNN outperforms its predecessors and is among the most accurate 2D object detection architectures in use today. Other member of the R-CNN family include R-CFN [13] and Mask R-CNN [14] but these are out of scope of this project.

C. Related work on 2D object detection involving Depth information

Some work has been carried out to study the effects of using depth information alongside RGB channel information for 2D object detection using Convolutional Neural Networks. Martinson and Yalla [15] showed how CNNs can outperform traditional approaches for detecting people in images by incorporating depth information to compute base features from which region proposals can be calculated. However, in their research, depth information has not been used in classification of ROIs, a significant disadvantage.

Mees et al. [16] developed a mixture of CNNs to extract features independently from different modalities including color, depth and motion, and used a gating network to fuse them for further classification. However, they use the traditional sliding window approach to generate region proposals, which ignores the potential of depth information for ROI selection.

Zhou et al. [17], recognizing the limitations of these works, proposed their own CNN-based architecture which uses RGB and depth information for ROI estimation and classification of ROIs. However, they do not use any popular object detection framework and just like the previous two works, they only focus on detecting a single class. Zhou et al. also don't evaluate and compare different forms of multimodal fusion approaches and layers and only implement a Late-Layer fusion approach like the one shown in Figure 3. Their architecture involves two independent CNNs forming separate RGB and depth pipelines. So, we end up fusing classification probabilities rather than RGB and depth modalities. This does not work well in practice, as the fusion of classification probabilities received from the depth pipeline and RGB pipeline ends up making the object detection accuracy worse while also using *double* the number of parameters compared to using a single RGB pipeline, which in turn reduces the speed at which training and evaluation of architecture takes place.

We get the inspiration for this project from the work of Ertler et al. [18]. They studied the effects of different methods of fusing depth information in the RGB pipeline in the Faster R-CNN architecture. They implemented Mid-Layer and Late-Layer fusion approaches along with 5 different fusion layers. However, no code was made public to allow further research to be carried out with ease. They also only focus on detecting a single class in images and do not evaluate and compare fusion methods on multiple Base Networks. Our work can be thought of as an extension to the work of Ertler et al. as we implement Mid-Layer and Early-Layer fusion approaches along with 3 most important

fusion layers proposed by Ertler et al. Our work also involves a more detailed analysis of these models.

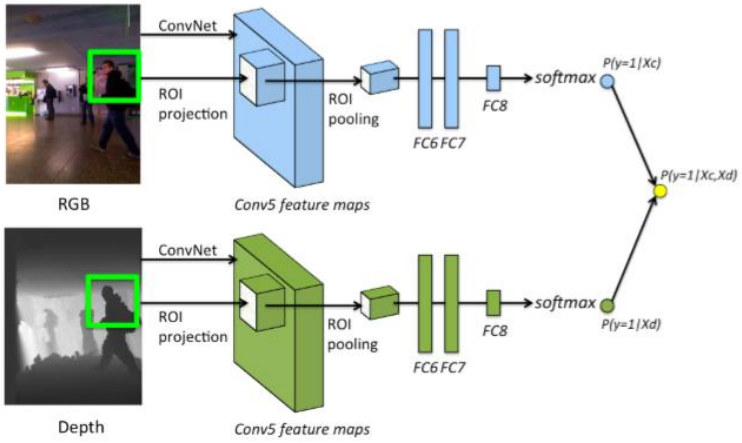

Figure 3: Late-Layer Fusion CNN-based object detection architecture proposed by Zhou et al. [17]

III. SOLUTION

A. Dataset Preparation

We use the KITTI2012 2D object detection dataset [2], which has been compiled and prepared by Andreas Geiger, Philip Lenz and Raquel Urtasun, for this project. The dataset consists of 7481 training RGB images and 7518 RGB testing images. It also contains a text file for every training image detailing information about the objects in the image. For each object, the text file contains the following information:

- Object Class (e.g. 'Car')
- 1 truncated Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
- 1 occluded Integer (0,1,2,3) indicating occlusion state:
  - 0 = fully visible, 1 = partly occluded, 2 = largely occluded, 3 = unknown
- 1 alpha Observation angle of object, ranging [-pi..pi]
- 4 floats for 2D Bounding Box coordinates of object in the image (0-based index):
  - contains left, top, right, bottom pixel coordinates
- 3 floats for 3D object dimensions: height, width, length (in meters)
- 3 floats for location of 3D object location x, y, z in camera coordinates (in meters)
- 1 Rotation angle around Y-axis in camera coordinates [-pi..pi]
- 1 score, indicating confidence in detection, higher is better.

An example text file for annotations is shown in Figure 4.

```
Car 0.00 0 -1.56 564.62 174.59 616.43 224.74 1.61 1.66 3.20 -0.69 1.69 25.01 -1.59
Car 0.00 0 1.71 481.59 180.09 512.55 202.42 1.40 1.51 3.70 -7.43 1.88 47.55 1.55
Car 0.00 0 1.64 542.05 175.55 565.27 193.79 1.46 1.66 4.05 -4.71 1.71 60.52 1.56
Cyclist 0.00 0 1.89 330.60 176.09 355.61 213.60 1.72 0.50 1.95 -12.63 1.88 34.09 1.54
DontCare -1 -1 -10 753.33 164.32 798.00 186.74 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 738.50 171.32 753.27 184.42 -1 -1 -1 -1000 -1000 -1000 -10
```

Figure 4: Example Object annotation text file for a single image in KITTI2012 2D

Although the annotations of this dataset have been created using the PASCAL criteria [2], the format of the annotation files is different. We changed the format of annotation

text files of this dataset to the format of PASCAL's xml-based annotation files. We needed to do this to make the dataloaders of our Faster R-CNN repository work with KITTI2012 2D dataset. This needed to be done as the repository was built only to work with the PASCAL format. An example of this change in format is shown in Figure 5.

Figure 5: Our transformed annotation file in PASCAL's xml format for a single image

As mentioned earlier, our dataset contains the 7481 training and 7518 testing images. However, the dataset only contains annotation text files for the training images. Since we needed the annotation text files for training and evaluating our Faster R-CNN architecture, we had to divide the provided 7481 *training* RGB images into training and evaluation groups of images. We train our architecture using the first 6000 training RGB images and use the remaining 1481 training images for evaluating it.

B. Depth Images

The KITTI2012 2D object detection dataset does not contain depth images but it contains left and right camera RGB images from which depth information can be computed. We first tried calculating depth information by creating disparity maps using simple OpenCV functions, but the results were unsatisfactory to be used for our Faster R-CNN architecture. So, we used the Pyramid Stereo Matching Network (PSMNet) proposed by Jia-Ren Chang and Yong-Sheng Chen [3]. PSMNet takes advantage of recent findings which suggests that depth estimation from stereo pair of images can be formulated as a supervised learning task which can be resolved with CNNs. Previously proposed architectures which use machine learning for depth estimation rely on patch-based Siamese networks, lacking the means to exploit context information for finding correspondences in badly-posed images. This leads to resulting depth maps having regions with abnormally large or small values of depth. PSMNet gets around this by using two main modules called the Spatial Pyramid Pooling Module and 3D CNN. The architecture uses multiscale context aggregation to form a 'cost volume' which is then fed into the 3D CNN to create the disparity map [3]. Figure 6 shows the architecture of the PSMNet used.

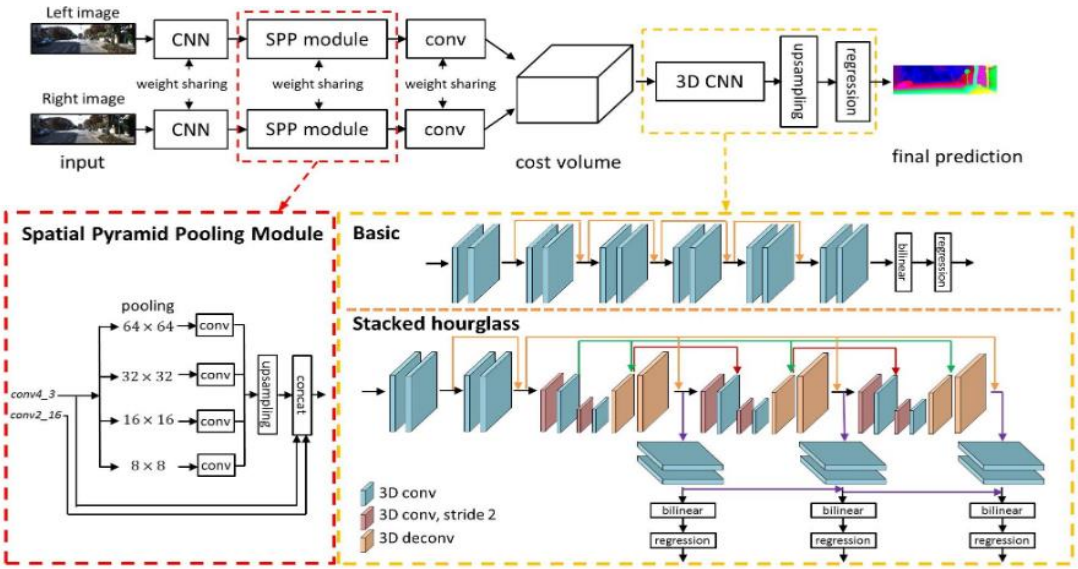

Figure 6: Pyramid Stereo Matching Network proposed by Jia-Ren Chang and Yong-Sheng Chen [3]

We use the implementation of PSMNet created and kindly shared by the authors of the research paper. Please refer to the paper and the code provided by authors for further information. PSMNet, however, has a generalization problem and needs to be heavily trained to produce good depth maps for a particular dataset. We get around this problem by using the pre-trained weights provided by the authors, specifically for KITTI2012 2D object detection dataset, and training it further, on our entire dataset, until the validation loss was minimal. Some resulting depth maps and their corresponding RGB images, which are used in our project, calculated from PSMNet are shown in Figure 7. The depth maps are better than results of primitive methods of calculating depth images.

Figure 7: RGB images (left) and corresponding PSMNet-generated depth images (right)

C. Faster R-CNN

This subsection provides an overview of the Faster R-CNN architecture. We choose to not explain the Faster R-CNN architecture used in detail here, as we use the same set up initially proposed by Ross Girshick et al. We mainly focus on making changes to the Head Network and the Classifier, which together form the Base Network, of the Faster R-CNN architecture. For further information about the Faster R-CNN architecture used, please refer to the original paper [1]. The overall steps carried out in Faster R-CNN are shown in Figure 8.

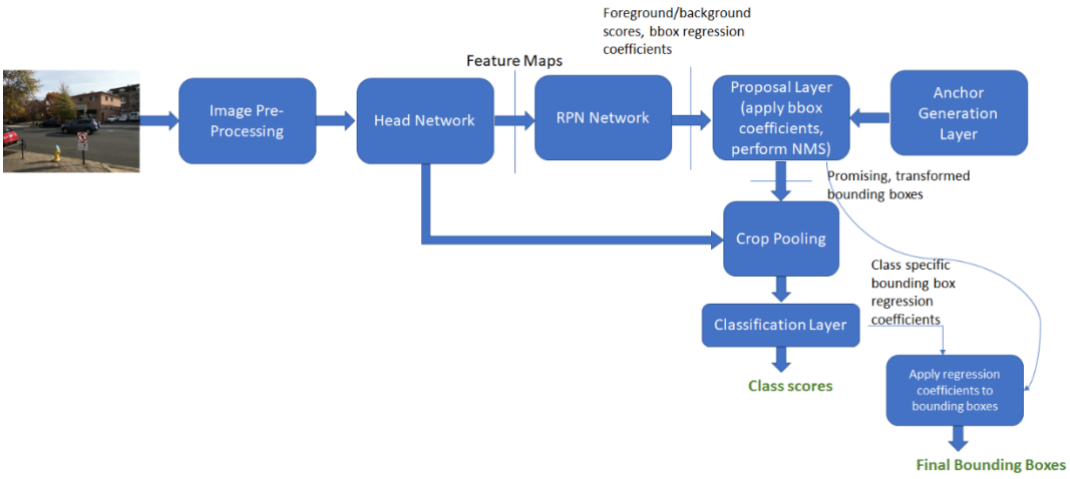

Figure 8: Faster R-CNN architecture [19]

The architecture consists of 3 key stages. The first stage is the Head Network which is a CNN pretrained for the task of object classification. The purpose of the Head Network is to train convolutional layers/blocks to extract the appropriate features from pre-processed images and to learn, through training, the shapes and colors that only exist in the objects that we are interested in detecting. There is no consensus on which Base Network is the best for Faster R-CNN as the choice of the Base Network is application dependent. Different Base Networks provide different accuracies and runtime performances. The original Faster R-CNN used VGG pretrained on ImageNet database but since then, several new networks have been proposed with varying number of weights. We focus on two Base Networks in this project; ResNet-50 which has 25.6 million parameters and VGG-16 which has 138 million parameters.

The next two stages use the *feature map* generated by the Head Network. The second key stage involves the Region Proposal Network (RPN). The purpose of the RPN is to check which locations contain an object in the image using the feature map generated by the Head Network and generate a set of region proposals.

In the third stage, a Region of Interest (ROI) Pooling layer pools the features inside the various region proposal boxes to fixed-size (i.e. size of the original feature map created by the Base Network) feature maps. These features are then passed through the fully-connected classification layers (CLS layer) and Bounding Box regression layers to perform object classification and bounding box refinement on each sub-window.

D. Base Networks

As mentioned in the last sub-section, we focus on the Base Network (Head Network and

Classifier) of Faster R-CNN. The two Base Networks we focus on are the ResNet-50 and the VGG-16. Since, these Base Networks were originally designed to *only* classify objects, we needed to modify their structures to divide each network into a Head Network and a Classifier to use them in our Faster R-CNN architecture. In this subsection, we describe the structure of the Base Networks used.

ResNet-50

We need to use deep CNNs to avoid overfitting the network. If we just stack convolutional layers together, the network suffers from vanishing gradient problem which means that as the gradient is back-propagated to earlier layers, repeated calculation makes the gradient too small to be of any use. This leads to performance of deep networks getting saturated and degrading rapidly. He et al. introduced the Residual Network (ResNet) [4] which solves the vanishing gradient problem by introducing 'identity shortcut connections' that skips one or more layers. We use the ResNet-50 Base Network, as proposed by He et al., by removing the last two layers (i.e. the Average Pooling layer and the Fully Connected layer) and then dividing it into a Head Network and a Classifier as shown in Figure 9.

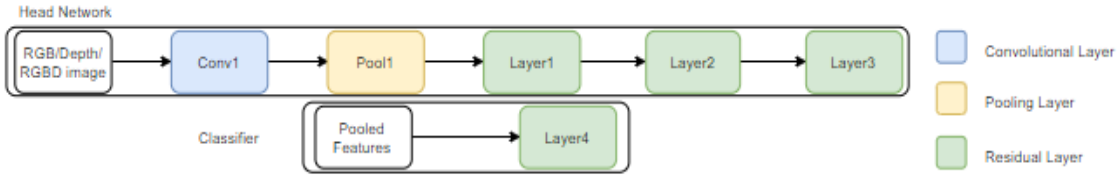

Figure 9: ResNet-50 Head Network and Classifier

The Head Network pipeline involves a convolutional layer (Conv1) and a maxpooling layer (Pool1) in the beginning, followed by 3 Residual Layers (Layer1, Layer2 and Layer3). The output of the Layer3 is then used as the feature map which is fed into the RPN and the Feature Pooling layer. Note that there is another Layer4 which is shown in the Figure 9. This Layer4 is used as the classifier to classify the pooled features. The outputs of the Layer4 are then fed into the CLS and Bounding Box Regression layers in the Faster R-CNN architecture. Each Residual Layer has a number of Residual Blocks in it. The structure of a Residual Block is shown in Figure 10. Here, for Conv[(k; k), s, *c], (k; k) means the spatial size of the convolution kernel. Parameter s is the stride of the convolution, and c is the increase or decrease factor of the output feature channel.

Table 1 shows the network configuration of ResNet-50 Base Network, here m denotes the number of input feature channels, n denote the number of output feature channels, and l_{unit} denote the number of residual blocks in that layer. Each Residual Layer starts with a downsample residual block and followed by several residual blocks.

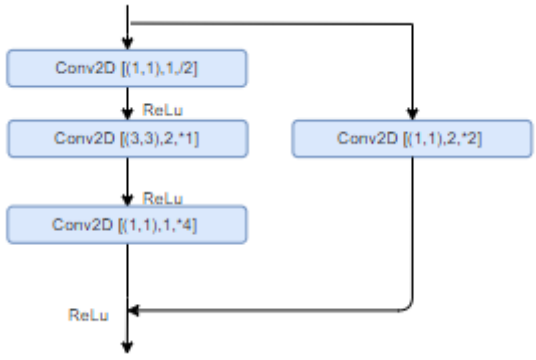

Figure 10: The structure of a single Residual Block with ReLu activation layers

Block	m	n	l_{unit}
Conv1	1 (if only Depth used) |3 (if only RGB used) | 4 (if RGBD channels used)	64	-
Layer1	64	256	3
Layer2	256	512	4
Layer3	512	1024	6
Layer4	1024	2048	3

Table 1: ResNet-50 network configuration

VGG-16

VGG-16 is a popular alternative CNN which can be used as a Base Network in the Faster R-CNN architecture. It was one of the most famous models submitted to ILSVRC-2014 [5]. The structure of a VGG-16 Base Network pipeline is shown in Figure 11.

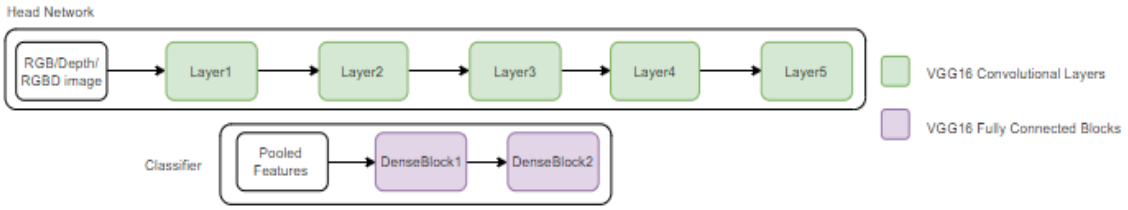

Figure 11: VGG-16 Head Network and Classifier

The structure of each VGG-16 Convolutional Layer is shown in Figure 12, the only exception here is Layer1 which does not have the Pooling Layer in its structure. The structure of the Fully Connected Blocks is shown in Figure 13. Each Fully Connected Block consists of a fully connected layer followed by a DropOut unit with probability of 0.5. DenseBlock1 employs a Fully Connected layer of the size [input: 25088, output: 4096] and DenseBlock2 employs a Fully Connected layer of the size [input: 4096, output: 4096]. Table 2 shows the network configuration of VGG-16 Base Network. The Table 2 has the same format as Table 1 except in Table 2, l_{unit} denote the number of Conv2D units in the layers.

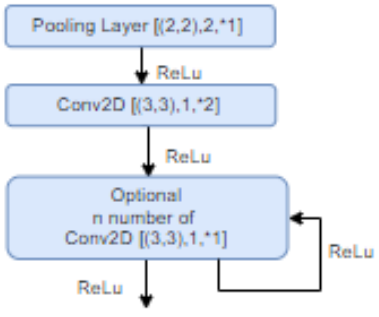

Figure 12: VGG-16 Convolutional Layer with ReLu activation units

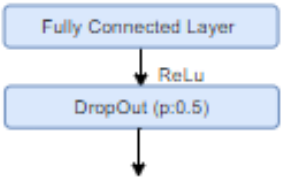

Figure 13: VGG-16 Fully Connected Block with ReLu activation unit

Block	m	n	l_{unit}
Layer1	1 (if only Depth used) |3 (if only RGB used) | 4 (if RGBD channels used)	64	2
Layer2	64	128	2
Layer3	128	256	3
Layer4	256	512	3
Layer5	512	1024	3
DenseBlock1	25088	4096	-
DenseBlock2	4096	4096	-

Table 2: VGG-16 network configuration

E. Implemented Fusion Approaches

This subsection details the fusion approaches implemented.

4-Channel Depth Fusion

This is the most straightforward approach to incorporate the depth information in the Faster R-CNN architecture. It involves loading the RGB (3 channels) and Depth (1 channel) images and fusing the channels together to create RGBD (4 channels) images which can then be fed to the ResNet-50 and VGG-16 Base Networks. To make this work, we modified the first Conv2D units in ResNet-50 and VGG-16 pipelines to support 4 channels rather 3 channels, which is the default setting.

We also modified the dimension of the pre-trained weights corresponding to the first Conv2D units in ResNet-50 and VGG-16 pipelines from 3 to 4 to make the pretrained weights compatible with the modified Base Networks to take advantage of transfer learning. We did this by taking an average of the weights corresponding to RGB channels and concatenating the average values as the depth dimension to the pre-trained weights corresponding to the first Conv2D units in ResNet-50 and VGG-16.

Early-Layer Fusion

This and the Mid-Layer fusion approach involve having two independent pipelines, one for RGB channels and the second for the Depth channel and combining these pipelines together at some point. We implemented this approach using ResNet-50 and VGG-16. In the case of ResNet-50, we carry out the fusion after the Layer1 as shown in Figure 14. For VGG16, we carry out the fusion after Layer2 in the same way as ResNet-50. Note also that the classifiers of ResNet-50 and VGG-16 are not changed. Since, the fusion does not involve any form of tensor concatenation, the network configurations, for ResNet-50 and VGG-16, remain the same as described in Table 1 and Table 2 respectively.

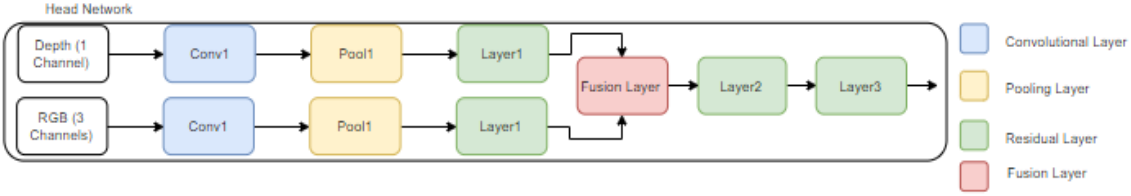

Figure 14: ResNet-50 Early Fusion Head Network

Mid-Layer Fusion

This approach involves fusion of *outputs* (feature maps) of two copies of the original pipeline Head Networks, one for RGB and the second for depth. We implemented this

approach using ResNet-50 and VGG-16 too. In the case of ResNet-50, we carry out the fusion after the Layer3 as shown in Figure 15. For VGG-16, we carry out the fusion after the Layer5 in the same way as ResNet-50. The network configurations and the classifiers remain same for ResNet-50 and VGG-16 in the same way as in Early-Layer fusion.

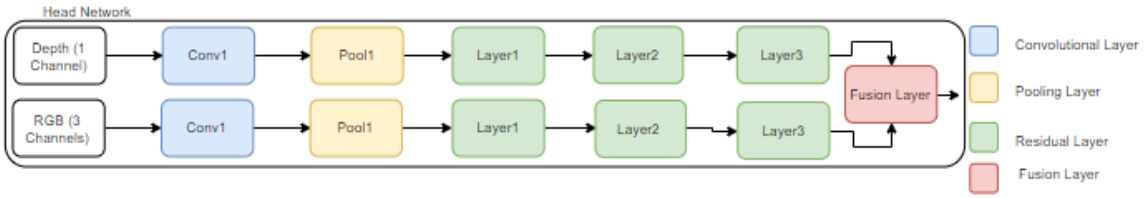

Figure 15: ResNet-50 Mid-Layer Fusion Head Network

Just like in the 4-Channel Depth Fusion, we use pretrained weights to initialize the layers in Early-Layer and Mid-Layer fusion approaches. We use the same pretrained weights as ones used in baseline RGB Faster R-CNN. We had to change the dimension of the pre-trained weights corresponding to the Conv1 layer in ResNet-50's depth-only pipeline and the first Conv2D unit in Layer1 in VGG-16's depth-only pipeline to support 1 channel input instead of the default 3 channel input. We implement this reduction in dimension by taking an average of the weights corresponding to the 3 R, G and B channels.

F. Implemented Fusion Layers

In this subsection we outline *how* the feature maps are fused together in the Early-Layer fusion and Mid-layer fusion approaches. Given two feature maps M^{rgb}, $M^{depth} \in R^{H*W*C}$, the output of a fusion must be another feature map $M^{fused} \in R^{H*W*C}$, where H, W *and* C denote the height, width and the number of channels of the feature maps, respectively. It is necessary for the fused feature map to be of the same dimensionality as the RGB and Depth feature maps to avoid having the need to change the underlying Faster R-CNN architecture as if we change the dimension of fused feature maps, then we will need to change the RPN, ROI pooling along with CLS and bounding box regression layers. Another reason for the preservation of dimensions of fused feature maps is to be able to exploit the benefits of transfer learning using pre-trained weights.

We implement 3 different fusion layers:

1. **Average**: computes average value at every spatial location and channel.

$$M_{h,w,c}^{fused} = \frac{M_{h,w,c}^{rgb}}{2} + \frac{M_{h,w,c}^{depth}}{2}$$

2. **Sum**: computes sum at every spatial location and channel.

$$M_{h,w,c}^{fused} = M_{h,w,c}^{rgb} + M_{h,w,c}^{depth}$$

3. **Maximum**: computes maximum value at every spatial location and channel.

$$M_{h,w,c}^{fused} = \max\left(M_{h,w,c}^{rgb}, M_{h,w,c}^{depth}\right)$$

G. Modifications made to the Repository and Implementation issues

We have previously specified that we used an existing Faster R-CNN repository. We use and modify the Faster R-CNN implementation by Yang et al. [19] which is based on the original implementation by Ross Girshick et al [20]. We use this repository because of two reasons. The first reason is that it is faster, more efficient and based on the more popular Python library Pytorch unlike the original implementation which is based on

Pycaffe. The second reason is that this repository uses Object Oriented Programming extensively which results in a cleaner and understandable code. As our modifications to the repository are aimed to aid researchers to do further research on Faster R-CNN, this feature of Yang et al.'s Faster R-CNN implementation is important.

The structure of the class hierarchy in the repository resembles the Faster R-CNN architecture itself as shown in Figure 8. Our biggest modifications to the repository involve creation of separate classes to support multi-modal fusion approaches and layers. This involved creating separate classes for each fusion approach and layer described in the previous subsections. Another contribution to the repository involves creation of new classes and modifications to existing classes to support KITTI2012 2D dataset. We provide functions to first convert the format of KITTI2012 2D dataset to the format of PASCAL and we then provide classes which allow the modified KITTI2012 2D dataset to be used to train and evaluate Faster R-CNN architecture. To incorporate depth information, the classes corresponding to 'image preprocessing', as shown in Figure 8, had to be modified. For example, one of the protocols employed in image preprocessing involves subtracting each pixel value of each channel (R, G and B) in an image by the mean channel values in the entire dataset. This protocol had to be modified to support proper mean subtraction for depth images (1 Channel) and RGBD (4 Channel) images. The modifications made to the repository make it simple and trivial for a researcher to evaluate different fusion approaches and layers on a dataset and use different Base Networks in the Faster R-CNN architecture.

H. Verification, Validation and Testing

As this project is based on using and modifying an existing GitHub repository, rigorous testing was carried out. This included White-box testing, Unit testing and Integration testing the software. The focus of these tests was functionality and performance. One of the main reasons for rigorous testing was to make sure that all the parts which make up the Faster R-CNN architecture were working correctly together as even a small mistake in the code can result in tens of hours of computational effort being wasted. Especially, a lot of focus was given to dataloader classes which had to be created to support our dataset and depth images. The performance of the system was also considered as each experiment carried out training for 50 epochs with each epoch requiring an hour on average. Unit and Integration testing worked brilliantly in solving errors in image pre-processing classes, dataloaders and classes created to implement the various fusion approaches and layers. After completion of training a model during an experiment, we would carry out an Alpha test by evaluating the entire architecture on 1481 testing images and checking the mean Average Precision (mAP). If the accuracy is what we expected it to be (i.e. from theory), using the existing baseline RGB Faster R-CNN as the benchmark, we would deem the implementation to be correct. The success of these tests proved that the implementation of the software is correct as if it was not correct, the experiments would result in accuracies which are nowhere close to the benchmark baseline RGB Faster R-CNN accuracies.

IV. RESULTS

In this section we first describe the evaluation method and then we compare the different fusion approaches and the fusion layers described in the Solutions section.

A. Evaluation Method

Set Up

To initialize all the layers, we use the publicly available Caffe-based pre-trained weights [19] which are trained on ImageNet dataset. These weights outperformed Pytorch based pre-trained weights in our findings. Note that we also use these weights for the depth-only pipeline layers as explained in the Solutions E subsection.

We train all the models with Stochastic Gradient Descent (SGD) optimizer and train each model on 12000 images (i.e. 6000 training images and their horizontally flipped versions) for 50 epochs and with varying starting learning rates which we reduce by half after every 10 epochs. For each model, we repeat the training experiments using starting learning rates of 0.001, 0.0005 and 0.0001, to get the highest accuracy achievable for each model. Momentum is kept at 0.9 and weight decay to 5×10^{-4} for all experiments. We also perform gradient clipping for all VGG-16 based experiments. During training, we save checkpoints of the model every epoch which gives us 50 checkpoints for every experiment. We then evaluate each checkpoint on 1481 test images and choose the one with highest validation accuracy (i.e. mAP).

Hardware Used

All experiments were carried out using workstations equipped with Quad 3.60 GHz Intel Xeon (R) CPU chips and 8GB 1080 GPUs.

Performance Metrics

We adhere to the PASCAL's evaluation metrics by computing the mean Average Precision (mAP) as the mean interpolated precision at eleven equally spaced recall levels. So, an object detection will only be considered positive, if its bounding box overlaps more than 50% with a ground truth bounding box.

B. Accuracies Achieved

Table 3 shows the accuracies achieved by Faster R-CNN architecture using different combinations of Base Networks, fusion approaches and fusion layer.

We observe from our findings that different set-ups provide varying accuracies. We observe that each fusion approach and fusion layer has a different effect on the Base Networks. For VGG-16, the highest accuracy we achieved is 87.3 mAP using Mid-Layer fusion approach and Maximum fusion layer. Meanwhile for ResNet-50, the highest accuracy we achieved is 85.6 mAP using the 4-Channel fusion approach. The reason for the difference in accuracies lies in underlying structure of these Base Networks. We can observe from Table 3 that the only findings which are unexpected are the ones corresponding to ResNet-50's Early-Layer and Mid-Layer fusion approaches. Their accuracies are lower than expected. We describe the reasons for these unexpected observations in the Evaluation section.

In Table 3, we also specify the number of parameters used by the entire Faster R-CNN architecture (including RPN) for a particular set-up. This is important to take into account as the number of parameters used by the architecture effects the computational power and graphics card's Video-RAM (VRAM) required to use it for object detection.

Base Network	Fusion Approach	Number of Parameters (million)	Fusion Layer	Accuracy (mAP)
ResNet-50	*Baseline RGB*	*30.2*	-	*85.1*
	RGBD (4-Channel Depth Fusion)	30.2	-	**85.6**
	Early Layer Fusion	30.4	Summing	83.6
			Averaging	80.1
			Maximum	82.3
	Mid-Layer Fusion	50.3	Summing	82.2
			Averaging	79.9
			Maximum	80.1
VGG-16	*Baseline RGB*	*136.7*	-	*84.5*
	RGBD (4-Channel Depth Fusion)	136.7	-	85.2
	Early Layer Fusion	136.8	Summing	86.6
			Averaging	85.9
			Maximum	86.9
	Mid-Layer Fusion	151.4	Summing	86.8
			Averaging	86.3
			Maximum	**87.3**

Table 3: Findings produced using experiments

The findings in Table 3 can be used as a reference guideline for researchers who wish to incorporate depth information in Faster R-CNN.

C. Run-time Performance

As a part of this project, we also compare the run-time performances of fusion approaches on ResNet-50 and VGG-16 as shown in Figure 16. Fusion layers are not compared here as they do not affect performance. For each fusion approach and Base Network, Figure 16 specifies speed in seconds, required to evaluate a single image, and Frames per Second (FPS) achieved.

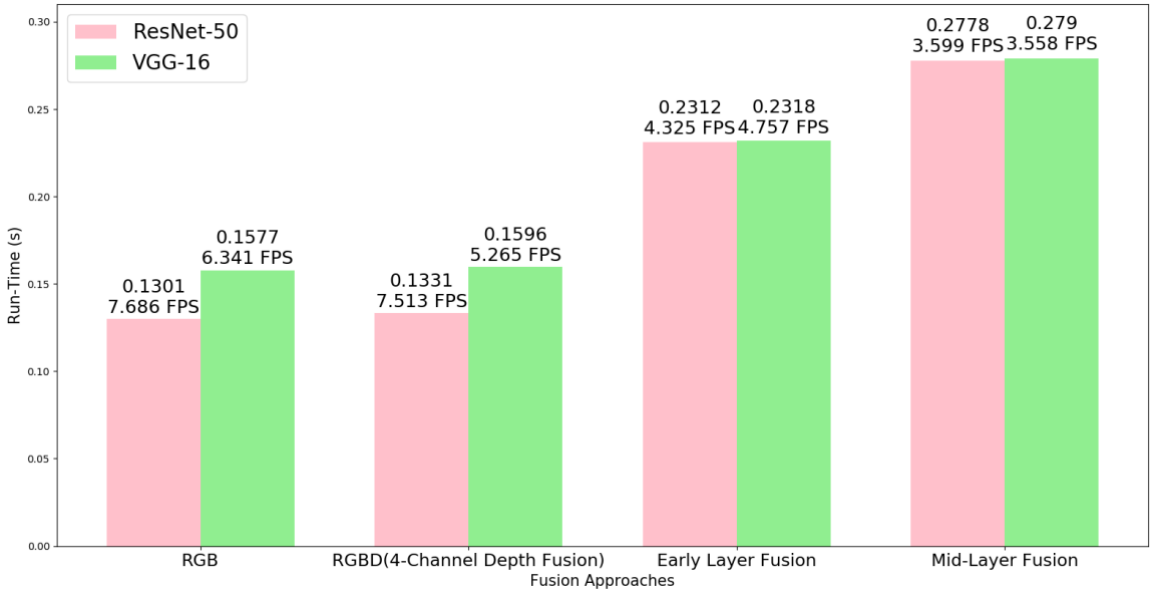

Figure 16: Run-time performance achieved by experiments

This performance measurement is important because autonomous vehicles and robots usually require higher FPS processing for accurate and lag-free scene understanding. We carry out this evaluation using the same approach as our previous evaluation. We evaluate the trained models on 1481 test images and measure the total amount of time required to carry out the evaluation and then we divide the total time measured by 1481 to compute the average time (in seconds) the architecture requires to evaluate a single image, from which we then compute the Frames Per Second performance achieved.

D. Object Detection Examples

Figure 17 shows some outputs of evaluation of a Faster R-CNN model employing VGG-16 Mid-Layer fusion approach with Summing fusion layer.

Figure 17: 2D object detection using Faster R-CNN (left: RGB image, middle: Depth image from PSMNet, right: RGB image with bounding boxes)

V. EVALUATION

In this section, we provide an evaluation of our work using the results presented in the previous section. The evaluation is done in terms of the research question proposed in the first section of this report (i.e. Introduction), providing explanations of the weaknesses and successes of the solution specified.

A. Strengths of the Solution and Suitability of Approach

We chose to work with ResNet-50 and VGG-16 because these are among the most popular Base Networks in use today for image classification and object detection. This was important for this project to give researchers results which they can compare with

findings from other research papers to propose useful conclusions and aid further research. We implement 3 out of 4 approaches, according to our literature survey, which can be used to incorporate depth information in Faster R-CNN. The only approach that is missing in this project is the Late-Layer fusion approach, where we fuse the two modalities after the ROI pooling layer, due to time and resource constraints for this project. We tried to implement the Late-Layer fusion approach but found it not possible due to the large number of parameters involved to implement this approach for the chosen Base Networks which resulted in models requiring more graphics card Video-RAM than we had available for this project. According to our literature survey, there are 5 fusion layers which can be implemented in Faster R-CNN as shown by the work of Ertler et al. in [18]. We chose to implement Summing, Averaging and Maximum fusion layers as [18] proved that these are the best fusion layers for Faster R-CNN and outperform the other 2 (namely Conv fusion and Inception fusion layers) by a significant margin.

In the evaluation of the models, we employ thorough investigation by repeating experiments with different learning rates (0.001,0.0005 and 0.0001) to find the highest accuracy which can be achieved for every model. Table 3 shows the variation of highest accuracies we achieved for the models. Table 3 shows that for VGG-16, all models involving different fusion approaches and fusion layers outperform the baseline VGG-16 RGB model by $\approx$ 0.7 to 2.8 mAP scores. Table 3 also shows that for ResNet-50, the 4-Channel RGBD approach outperforms the baseline RGB ResNet-50 model. This is a proof that the depth information helps to improve the detection results of a Faster R-CNN architecture. The reason why ResNet-50's Mid-Layer and Early-Layer approaches did not perform as expected is explained in the next sub-section. Furthermore, the results show that for VGG-16, the Mid-Layer fusion outperforms the other fusion approaches and provides a good tradeoff between accuracy and speed. We hypothesize for a Faster R-CNN model, with VGG-16 Base Network using Mid-Layer fusion approach, a mixture of semantic understanding and visual and spatial information in these layers provide more complementary features than the early layers, which encode the lower-level representations. Thus, the network is able to perform better by gaining more information from the combination of both modalities in later layers and then training the rest of the layers using the *combination* of modalities.

B. Weaknesses and limitations of the Solution

Although the solution proves using the results above, that incorporating depth information increases the accuracy of Faster R-CNN using VGG-16 Base Network, the solution did not work as expected for ResNet-50. This observation, however, is not a result of wrong implementation, but it is because of the dataset and the pre-trained weights used for training the models. As we mentioned earlier, that the all the layers (including layers corresponding to depth-only pipeline) in all models are initialized using pre-trained weights, initially trained on RGB images on ImageNet database. So, the pre-trained weights used for depth pipelines are actually tuned for RGB images rather than depth images. This coupled with the fact that our own dataset only contains 6000 images for training, mean that ResNet-50 struggles to train itself as big networks require larger datasets to achieve high accuracy. So, in theory ResNet-50 employing Early-Layer fusion and Mid-Layer fusion approaches should outperform the baseline RGB ResNet-50. But in this implementation, only the ResNet-50 employing 4-Channel RGBD fusion approach outperforms the baseline model. On the other hand, VGG-16 does not suffer from the same problem as although it has higher number of parameters which need to be

trained, it has lower number of layers which need to be trained compared to ResNet-50. This results in VGG-16 being able to train quicker and require smaller datasets than ones required for ResNet-50 to achieve high accuracy.

The project does a good job at answering the research question and proving that incorporating depth information increases accuracy of Faster R-CNN architecture. The only other weakness is that we only used 3 types of fusion layers. Although these layers are more than enough to answer the research question, more types of fusion layers could have been compared. For a comparison of different types of fusion layers in Faster R-CNN, please refer to [18].

C. Appraisal of Project Organization

One positive aspect of the project organization is that agile method was used to develop the solution. This enabled us to implement the changes to the Faster R-CNN repository is stages and aid in rapid prototyping for the project meetings. Initially, a lot of time was spent to fix the Pytorch dataloaders to work with the KITTI2012 2D dataset. This is also one of the main contributions of the project as we provide classes and Python scripts which allow one to convert KITTI2012 2D dataset format into the more popular PASCAL dataset format. These scripts and classes can be used for any project. Then, we implemented each fusion approach and fusion layers in stages for each Base Network to evaluate the models. We also performed rigorous testing, validation and verification of software and the evaluation outcomes produced by the Faster R-CNN architecture.

VI. CONCLUSION

In this project, we aimed to show the reader how depth can be incorporated in Faster R-CNN architecture and how it effects the accuracy and performance of the system. We chose to focus on the incorporation of depth information, as opposed to other sources of information like X-Rays and Infrared Radiation (IR) maps, because it is easy to use to improve computer vision systems for autonomous vehicles and robots using instruments like LIDAR. To incorporate depth information in Faster R-CNN, we implemented VGG-16 and ResNet-50 based RGBD 4-Channel fusion, Early-Layer fusion and Mid-layer fusion approaches along with Summing, Averaging and Maximum fusion layers. Our aim was to incorporate depth information in the Faster R-CNN architecture using a variety of different ways to give researchers a reference guideline to how this architecture performs using these set-ups and experimental settings.

Using the results, we conclude that it is indeed worth the computational effort to incorporate depth information in Faster R-CNN. As we saw from the findings in Table 3, the accuracy of baseline RGB VGG-16 is 84.5 mAP. This can be improved using the Mid-Layer fusion approach and Maximum fusion layer to 87.3 mAP. The increase in accuracy may not seem significant but in practice where faulty object detection can result in autonomous vehicles and robots failing to work as expected causing malfunctions which may even lead to loss of life, this increase in accuracy matters. We saw how fast the fusion approaches perform compared to baseline RGB models in Figure 16. For VGG-16, the baseline RGB model has the FPS performance of 7.686 and the Mid-Layer fusion model has the FPS performance of 3.558. By using Mid-Layer fusion approach, we decrease the FPS performance by 4.128. This decrease in FPS performance may seem daunting but it is acceptable for small autonomous vehicles and robots using similar

hardware. The FPS performance can be improved by using purpose-built computers for object detection. An example of this is Tesla's Hardware 3 (HW 3.0) which is a purpose-built computer consisting of CPU, GPU and motherboard to power Tesla's car autopilot. Hardware 3 is currently used in Tesla's Model X and has the computational capability to process 2000 FPS [22].

Future work involves implementation and evaluation of the Late-Layer fusion approach. To create a complete reference guideline for helping researchers choose which methods to use to implement accurate and efficient object detectors, other object detection frameworks need to be considered too. This includes other members of the R-CNN family [13][14] and a powerful new object detection framework YOLO [9].

REFERENCES

[1]: Ren, S., He, K., Girshick, R. and Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

[2]: Geiger, A., Lenz, P. and Urtasun, R. (2012). Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. *Conference on Computer Vision and Pattern Recognition (CVPR).*

[3]: Chang, J. and Chen, Y. (2018). Pyramid Stereo Matching Network.

[4]: He, K., Zhang, X., Ren, S. and Sun, J. (2015). Deep Residual Learning for Image Recognition.

[5]: Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

[6]: Deng, J., Dong, W., Socher, R., Li, L., Li, K. and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database.

[7]: Zhou, B., Lapedriza, A., Khosla, A., Oliva, A. and Torralba, A. (2016). Places: A 10 million Image Database for Scene Recognition.

[8] Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Tech report*, 5.

[9] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection.

[10]: Marketsandmarkets.com. (2019). Facial Recognition Market worth 7.76 Billion USD by 2022. [online] Available at: https://www.marketsandmarkets.com/PressReleases/facial-recognition.asp [Accessed 1 May 2019].

[11]: Iandola, F., Ashraf, K., Dally, W., Keutzer, K., Moskewicz, M. and Han, S. (2017). SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE.

[12]: Girshick, R. (2015). Fast R-CNN.

[13]: Dai, J., He, K., Sun, J. and Li, Y. (2016). R-FCN: Object Detection via Region-based Fully Convolutional Networks.

[14]: He, K., Gkioxari, G., Dollár, P. and Girshick, R. (2017). Mask R-CNN.

[15]: Martinson, E. and Yalla, V. (2016). Real-Time Human Detection for Robots using CNN with a Feature-Based Layered Pre-filter. 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN).

[16]: Mees, O., Eitel, A. and Burgard, W. (2016). Choosing Smartly: Adaptive Multimodal Fusion for Object Detection in Changing Environments. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[17]: Zhou, K. and Paiement, A. (2017). Detecting Humans in RGB-D Data with CNNs.

[18]: Ertler, C., Opitz, M., Bischof, H. and Possegger, H. (2017). Pedestrian Detection in RGB-D Images from an Elevated Viewpoint.

[19]: Telesens.co. (2019). Object Detection and Classification using R-CNNs – Telesens. [online] Available at: http://www.telesens.co/2018/03/11/object-detection-and-classification-using-r-cnns/ [Accessed 2 May 2019].

[20]: GitHub. (2019). jwyang/faster-rcnn.pytorch. [online] Available at: https://github.com/jwyang/faster-rcnn.pytorch [Accessed 2 May 2019].

[21]: GitHub. (2019). rbgirshick/py-faster-rcnn. [online] Available at: https://github.com/rbgirshick/py-faster-rcnn [Accessed 2 May 2019].

[22]: Lambert, F. and Lambert, F. (2019). Tesla claims to have 'world's most advanced computer for autonomous driving' with Autopilot 3.0 update coming next year. [online] Electrek. Available at: https://electrek.co/2018/08/01/tesla-chip-most-advanced-computer-autonomous-driving-autopilot-hardware-3-update/ [Accessed 2 May 2019].