3D Object Localisation with Convolutional Neural Networks

A thesis submitted in partial fulfilment of requirements for the degree of
Bachelor of Engineering

Oscar McNulty
School of Aerospace, Mechanical and Mechatronic Engineering
The University of Sydney
November 2017

Abstract

Interpreting the real world from images and other sensor measurements is critical to advancing the abilities and use-cases of robotics. Recent advances in hardware and convolutional neural network (CNN) structures have lead to breakthroughs in object recognition in 2D and 3D. A synthetic data approach to training is investigated using the Grand Theft Auto 5 game engine with which a set of synthetic road scene data is created. A practical investigation into object localisation in 3D using CNNs is conducted with experiments using both pure 2D visual techniques and techniques utilising synthetic point cloud data. A combination of predictors using visual and depth data shows promising results, reaching a mAP of 49.6% with a 0.3 3D IoU threshold on the challenging synthetic dataset. A discussion is presented into the viability of CNNs for higher order tasks such as 3D localisation and how future work might improve on the results presented.

Dedication

To Mum for getting me thus far.

Declaration

To the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

As part of my research I have contributed the following,

- I conducted a review of previous work in the area.
- I wrote software to create synthetic training data using the Grand Theft Auto game engine. A number of external libraries were utilised and are referenced.
- I used Amazon Web Services to generate a large 3D dataset that is available publicly at `https://github.com/oscarmcnulty/gta-3d-dataset`
- I implemented and trained a number of networks using the MXNet framework for deep learning.
- I conducted experiments to evaluate the performance of different network structures and techniques on the 3D dataset generated.
- I carried out analysis of the results. The conclusions drawn are my own, influenced by discussion with my supervisor.

Acknowledgements

I want to thank Dr. Ian Manchester for letting me pursue a unique topic and giving me very useful advice at the most opportune moments.

Contents

Chapter 1

Introduction

1.1 Background

The increasing power of computer vision has broad impact across many industries. Tasks that were previously only possible with human interaction are increasingly becoming possible with machine driven techniques. While the shift away from human labour will have substantial social impact, the rise of artificial computer vision has the potential to improve the lives of people around the world.

Recent work in the area often focuses on 3D localisation in the context of autonomous vehicles where large investments have been made into academic research by industry. Features from lane markings to pedestrians are all items of interest in a road scene and techniques for detection vary significantly for different elements although many rely on neural network classifiers and regressors.

More broadly, computer vision is a critical component of general purpose, responsive robotics that can interact with humans in everyday life.

1.2 Research Goals

The goal of this investigation is twofold, to investigate synthetic scene data as a mechanism for cheap and efficient training of neural networks; and to evaluate different methods for 3D object localisation using 2D visual data and 3D point cloud data.

The goal of using synthetic computer generated data for training neural networks is to create generalisable neural networks that can be trained virtually but perform well on real world data. The lack of physical constraints on data collection or testing

means that vastly larger datasets can be created with accurate labelling. The ability to create data that represents unlikely or dangerous situations presents exciting opportunities to rigorously validate models safely and move beyond expensive real world testing.

Extracting 3D information from a scene using 2D imagery data and depth information is key to robotic interaction with the physical environment. A vast range of techniques are presented in previous work that cover detection of different scene elements using convolutional neural networks and other techniques. This work focuses on using convolutional neural networks for detection of vehicles within a road scene using a variety of approaches and combination of data sources.

Chapter 2

Literature Review

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are built on a selection of simple mathematical building blocks. While inspired by the biological of brain neurons and the electrical signals with which they interact, modern artificial networks have deviated significantly from true real-world neurological behaviour.

ANNs are a type of generalisable regressor or classifier and are fitted to a dataset using a modified version of gradient descent. The universal approximation theorem states that a network with a single hidden layer of sufficient size can approximate any continuous function. A form of the theorem was first proven by Cybenko in 1989[1]. The theorem implies that ANNs can represent a wide range functions but notably does not provide assertions on the learnability of a function which is a critical component of the effectiveness of ANNs.

An example ANN implementation is a fully connected network with a single hidden layer. At each node the sum of products of edge weights and inputs is passed to an activation function. The activation function is typically a non-linear function as linear functions reduce the network to a polynomial that cannot solve non-trivial problems with a small number of nodes. The output of the activation is fed to further nodes in the network.

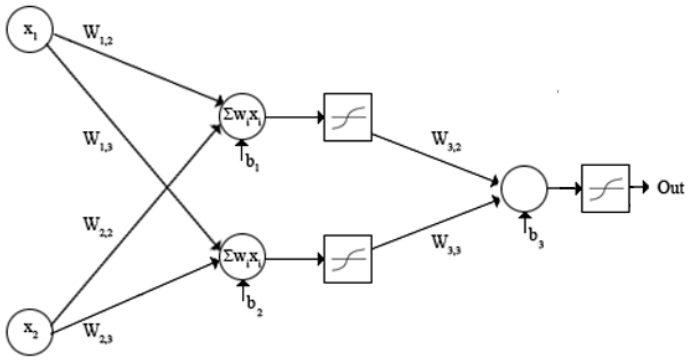


Figure 2.1: An simple feed-forward ANN[2].

Training an ANN can be thought of as an optimisation problem of the system $\arg\min_\theta L(\theta)$ where θ is the set of all network weights and $L(\theta)$ is the sum of a loss function calculated across all data samples. The loss function is a metric that represents how far a predicted value is from the true label. Standard gradient descent would use the iterative sequence $\theta_{i+1} = \theta_i - \gamma \nabla L(\theta_i)$ where $L(\theta_i)$ is calculated across all data and γ is the learning rate. Modern deep learning uses such large volumes of data that to calculate the Jacobean across the whole dataset is infeasible. To make the problem computationally feasible, the gradient is only calculated across a rotating subset of the total dataset and the learning rate γ is tuned accordingly. Optimising in this manner is called stochastic gradient descent (SGD). The learning loss is back propagated through the network to update weights in the network based on the loss function calculated across the current rotating data subset.

Like the structure of ANNs, the activation functions that networks are built on have diverged from their biological truth. Rectified linear units (ReLu) are the most prevalent activation function used in modern research. The ReLu neuron is typically implemented as $f(x) = \max(0, x)$ but variants exist that, for example, give a small non-zero gradient when $x < 0$ which are described as 'leaky' variants. Sigmoid type functions are truer to the biological inspiration of neural networks but have limitations compared to ReLu units. Sigmoids suffer from the problem of a vanishing gradient where the gradient of the neuron can effectively become 0 due to the two asymptotes after a substantial amount of training. Additionally, where computational limits impose restrictions on the size and speed of networks, sigmoid units are much more costly than ReLU units.

2.2 2D Image Classification

Image classification is the task of assigning a label to an image the describes the content of an image. Current state of the art 2D image classification is driven by Convolutional Neural Networks (CNN).

The fully connected nature of a standard feed-forward network requires a number of connections that increases polynomially with larger layer sizes. The computation required to apply a fully connected network to pixel level image data is simply infeasible without vastly reducing the input size or internal size of the network, both of which reduce predictive power. Additionally the large number of connection weights takes a long time and has to fit more parameters which makes the training less stable.

CNNs are not a recent concept, they were first proposed by LeCun et. al [3] in 1989. Extracting local features from an image and then combining to create higher order features is the underlying principle of this network type. The CNN shares layer connection weights between regions of each layer and vastly reduces the connectedness of the network compared to a fully connected net by only linking local regions between layers. Applying these constraints (in the sense that a number of weights are constrained to 0 compared to a fully connected network) is the key to the success of the CNN as it reduces the entropy of the learning task and enforces implicit regularisation of the network.

The 2000's were dominated by techniques such as Scale Invariant Feature Transform (SIFT) proposed by Lowe [4] and Histogram of Oriented Gradients as proposed by Dalal and Triggs [5]. It wasn't until 2012 that Krizhevsky et al. [6] brought CNNs back to the forefront of image classification by substantially beating other models in the ImageNet Large Scale Visual Recognition Challenge [7] (ILSVRC) with their model, AlexNet. Their approach was not dissimilar to the approach described by LeCun 23 years prior but used a much larger network and was applied to a much larger dataset using modern GPU hardware.

Networks much larger than AlexNet now dominate classification competitions such as ILSVRC. Network types such as ResNet[8], VGG[9] and Inception[10] take weeks to train on modern hardware but have significantly improved over the initial results of AlexNet and are used as the basis for most modern object detectors.

Initially CNNs were designed to resemble biological neurons in the visual cortex but as research progressed, modifications not true to biology were developed. CNNs utilise three broad mechanisms to improve training and predictive performance on

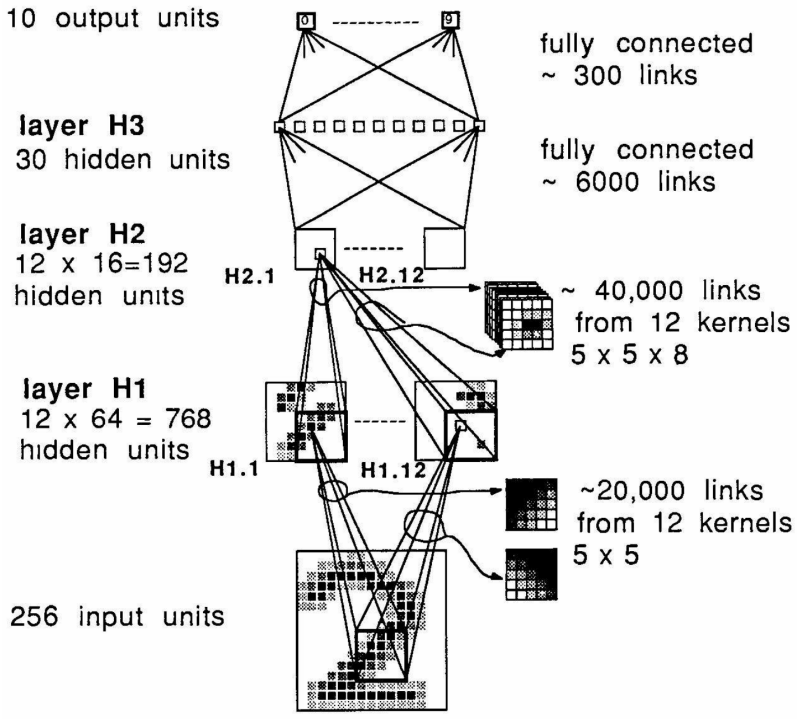


Figure 2.2: LeCun's initial CNN for prediction of handwritten numerals [3].

images.

- Local receptive fields introduce the idea of restricting the connectedness of layers to only link nearby areas in adjacent layers.
- Intermediate layers of the network are subsampled to reduce the intermediate feature set to include only the features with the most confidence or highest probability.
- Weights are shared between edges such that the degrees of freedom in the network is substantially reduced while maintaining accuracy.

The initial motivation to apply these modifications is again inspired in biology. LeCun et. al cite the discovery of locally-selective, orientation-selective neurons in cats and monkeys by Hubel and Wiesel in 1962 as inspiration for the their initial networks[11].

2.3 2D Object Localisation

While CNNs provide high accuracy image classification they only provide a single output per input. Additionally they are unable to process non-standard image sizes

without arbitrary scaling or cropping and do not adapt well to scene changes. To locate many objects within an image of varying shape and size requires targeted modifications. These modification typically utilise CNNs as low level feature extractors then apply unique methods to create many localised predictions for each image.

The Regional-CNN (R-CNN) was proposed by Girshick at el. in 2013 and is widely recognised as the first notable application of CNNs to object detection within an image[12]. It uses a region proposal network (RPN) based on selective search which groups similar pixels based on colour and intensity to generate around 2000 object proposals for each image. These regions are sampled from the image and scaled to be the same size. A modified version of AlexNet is then applied to each one of these proposals to generate class confidences per proposal.

The convolution section of a classification CNN does not specify image size, kernels can be applied to any arbitrarily sized image. It is the fully connected component for classifying high level features that must have a set size and aspect ratio. He et. al propose a method, spatial pyramid pooling (SPP), that connects an arbitrarily sized convolution output to a fixed size feature classification input [13]. This is achieved by splitting the final convolutional feature layer into $n \times n$ regions (whose individual size will depend on the input) then apply maxpooling to each region. Multiple levels of $n \times n$ tilings are used, He et. al proposed tilings of $\{4 \times 4$, 3×3, 2×2, $1 \times 1\}$ for images around 224×224 pixels. Using a variant of SPP trained for object detection, a speedup of $24 - 102\times$ over R-CNN is claimed with similar accuracy[13]. The key to this speedup is the shared use of the convolutional features instead of producing thousands of individual confidences of cropped areas.

Fast R-CNN improves on the speed of R-CNN while increasing accuracy by reducing the number of full end-to-end network predictions required and swapping AlexNet for VGG[14]. This is achieved with Region of Interest (RoI) pooling which applies varied size maxpooling operations similar to SPP. The region to pool over is determined by the same selective search RPN but because intermediate convolutional features are shared the network makes predictions $213\times$ faster than the initial R-CNN.

Faster R-CNN further improves speed and accuracy over Fast R-CNN by further increasing the sharing of convolutional features[15]. The selective search RPN is replaced with a convolutional predictor that uses the same VGG network as the class prediction. This sharing allows the RPN and class predictor to be jointly optimised by training end-to-end. The convolutional RPN is also faster than the

selective search RPN.

Redmon et. al propose further increasing the prediction speed of a network by combining the RPN and class predictor into a single network using a network structure "You Only Look Once" (YOLO)[16]. A CNN is used to reduce the input to a 7×7 feature map. This single feature map is used to predict a class likelihood at each cell of the feature map and in addition is used to predict a number of bounding boxes and objectness scores per cell. With these simplifications the network can create inferences as 45Hz but at a substantial accuracy reduction of 8.4ppt of mean average precision compared to Fast R-CNN on the VOC2007 dataset.

The single feature map of YOLO reduces the computation required compared to other CNNs but at the cost of worse object prediction due to its limited ability to predict objects of varying sizes. The Single Shot Predictor (SSD) proposed by Liu et. al introduces modifications to the single network approach that increase the accuracy to a level comparable with R-CNN methods while maintaining faster run speed[17]. Instead of using a single feature map, multiple levels of feature extraction are used to extract different sized features from multiple layers. Additionally, instead of predicting bounding boxes directly from features, multiple hardcoded proposal shapes ('anchors') are used and the network is trained to make adjustments to these based on feature map input. These modifications lead to similar accuracy as Faster R-CNN but with a quicker run speed.

2.4 3D information from 2D images

All the approaches to image recognition explored thus far are purely for 2D identification of objects in a scene. The task of extracting 3D information from a scene with one or more objects of interest is often approached by extending existing 2D methods.

For multiple images with offset views of the same scene it is common to match features between images and estimate depth in the scene using these matchings. A typical "local" stereo algorithm calculates a matching cost that measures the similarity of a local patch in one image to another slightly offset patch for many combinations to determine where a 3D point appears in different images. Patches of an image are only matched within a limited range.

Zbontar and LeCun have together presented various methods for matching cost computation and cost aggregation using CNNs [19, 20]. These approaches build on the concept of the "siamese network" proposed by Bromley et al. [21] where features

are extracted from two input cases using identical networks and the results are then concatenated and used as input for a final classification function which may be a CNN.

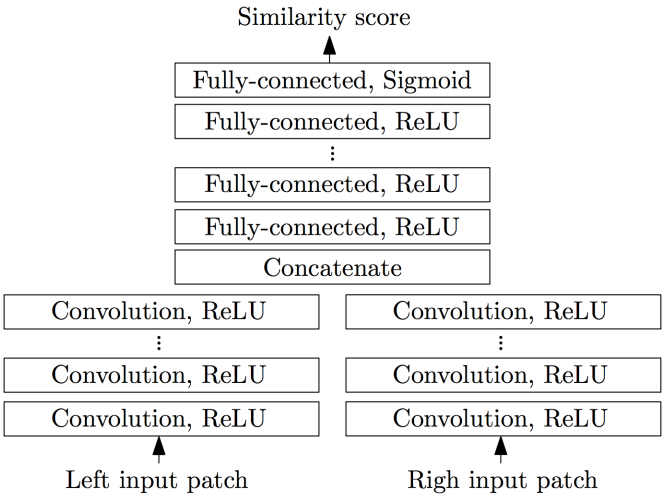


Figure 2.3: A variant of Zbontar and LeCun's siamese network for matching features in stereo imagery [3].

Due to the prevalence of single cameras in many systems, techniques have been developed that are able to accurately detect 3D features from image context in single image scenes. Chen et. al present a method of extracting 3D semantics from an image using a combination of deep learning decompositions of a road scene image. Hand crafted features are used to reason about the geometry of the scene which allows reasonable depth estimation with only visual data[22].

Wang et. al use monocular image data augmented with road map data from Open Street Map to improve the predictive power of a monocular predictor with geographical context. The road map data which includes birds-eye building shapes is used to generate geographic priors of the scene before passing this data to a combination of predictors that reason jointly about the scene[23].

While full depth information about the scene is difficult to obtain purely from 2D imagery, Zhang et al. propose a method for instance segmentation that provides instances with relative depth ordering. This ordering is then fed to an algorithm based on a Markov random field which provides a single coherent explanation of the input[24]. This approach is particularly relevant when a depth ordering of objects rather than absolute 3D position is required and may be useful as an intermediate feature step in other predictors.

In [25], a pose estimation technique is demonstrated that separates localisation

and orientation when generating 3D bounding boxes using existing 2D CNN techniques. Given a tight 2D bounding box and the 3D orientation of an object it is possible to generate a 3D bounding box. An existing CNN for 2D object detection is extended to estimate an objects orientation and size as well as class. This is then combined with a typical 2D bounding box prediction to geometrically determine a 3D bounding box.

2.5 Combining Imagery and Depth Data

In the context of autonomous driving, point cloud data is typically a critical input to predictive algorithms. The sparsity of the point cloud data creates challenges that are not typically faced with 2D imagery.

Attempts have been made to extend 2D convolution to 3D by simply adding another dimension to the data, network, and predictions. Song and Xiao present a method of 3D convolution using a 3D RPN similar to Faster-RCNN [26]. Their method "Deep Sliding Shapes" represents the point cloud as a grid of 3D voxels. They utilise deep learning to learn the 3D structures of common objects using modern convolutional techniques applied to 3 dimensions. This method, however, is computationally costly. Performing convolutions in 3D requires the network be an order of magnitude larger and consequently requires substantially more data and computation to train than 2D networks. In their research a typical 2D CNN was used to augment the expensive 3D voxel convolution.

A dense 3D voxel representation of data is an inefficient way of projecting the sparse data. More efficient approaches entail projecting the data to 2D planes. 2D projections of data also enable the use of existing 2D classification methods. Chen et. al use this approach in their 3D vehicle detecter MV3D [27]. They opt to use two projections of point cloud data, a birds-eye view and a front view. The point cloud is flattened to 2D representations by representing the 3rd dimension as colour. MV3D creates axis aligned birds-eye bounding box proposals similar to Faster-CNN which are then projected to the front view so that classification can be done in both projections. They find that, while the front view point cloud projection and 2D visual data provide a small accuracy increase, the birds-eye projection is key to the accuracy of the predictor.

2.6 Practical Overfitting and Regularisation

From the universal approximation theorem, a single hidden layer feed forward network with a finite number of neurons can fully represent any function on $\mathbb{R}^n$. Similarly, multi-layer networks excel at fitting complex functions. This leads to the risk that training neural networks using gradient descent results in overfitting.

Just like in less complex regression problems, the generalisability of a neural network is measured by segmenting the data into training and validation sets where the validation data can be used to measure how well the model generalises. With ANNs this validation set is also used to stop training once validation accuracy is no longer increasing. After this point training accuracy would continue to increase without increasing validation accuracy which indicates overfitting and is not useful for creating a good model.

Weight decay is one technique to used to regularise the model created during training. Weights are regularised after each SGD batch in a manner similar to linear ridge regression by decaying each weight based on its $L2$ norm. By preferring smaller weight values a less complex model fit is preferred which regularises the network and encourages the problem to be fitted in a minimalistic manner.

Model ensembles are often used to reduce the effects of overfitting a certain type of model. This approach is not as feasible with neural networks since training many networks for an ensemble with varying architectures requires training that is computationally expensive. Srivasta et al. propose a method called dropout that prevents specific neurons from co-adapting too much and effectively creates an ensemble of networks[28]. From a full network of n weights to be trained, up to 2^n "thinned" networks can be generated by probabilistically removing edges for a single training batch. Edges are removed with a probability p and back propagation is applied to the thinned network. This results in an exponential number of different neural networks being trained. It isn't feasible to train an exponential number of different networks but by averaging during training an large ensemble is effectively built. Once the network is trained the dropout component is removed and all weights are used for prediction.

Batch normalisation is a method that both improves training speed and acts as a regulariser [29]. It adds normalisation between network layers to ensure that a layer feature input has constant mean and constant variance. This is distinct from standard normalisation as the mean and variance to normalise to are learnt parameters for each layer and are trained concurrently with layer weights during

back propagation. The authors claim that in a test case, batch normalisation allows equivalent accuracy to be achieved in 14× fewer training steps in an example network. [30] suggests that in practice, batch normalisation helps spread feature learning over many layers in a very deep network such as ResNet and substantially reduces the requirement for careful weight initialisation in large networks.

2.7 Generalisability of Neural Networks

The strength of neural networks is their ability to learn complex non-linear functions. Despite the large number of practical mechanisms for regularisation of networks, the reason as to why specific networks generalise well is still a somewhat open question. The ability of a neural network to generalise well relies on it's ability to learn intermediate features that can be observed outside of the training set. If a network cannot learn intermediate features on a training set then the network is not truly learning to understand the input but is simply memorising the training input labels. While regularisation techniques such as dropout and weight decay are effective, they only provide incremental benefit and networks are still able to generalise well without these training modifications in place. In experimental investigation conducted by Zhang et. al [31], the conclusion is drawn that while explicit regularisation techniques may improve generalisation it is not necessary to control overfitting and itself is not sufficient to ensure good generalisation. This result suggests that the key to good generalisability lies not within the regularisation techniques applied, but with the structure of the network model.

In the case of CNNs, the sharing of weights and locally connected features forces intermediate features to be learnt which is perhaps the key reason why the networks tend to generalise well in practice. This still does not preclude this family of network from poor generalisation. In [31] an experiment is presented that demonstrates the ability of the Inception network to learn correct image labels in a CIFAR10 training set to 100% accuracy but also to learn completely randomised labels to 100% accuracy on the same CIFAR10 dataset. This result is significant as it shows that while intermediate features can be learnt, modern neural networks still do posses the parameter size to completely memorise labels without consistent features (as labels are randomly assigned) hence no useful intermediate features are learnt.

These findings imply that deep neural networks, even with regularisation, can simply memorise inputs without learning any useful intermediate features that would allow the network to generalise well beyond the training dataset. With meaning-

ful image labels instead of random labels a carefully crafted network will tend to learn useful representations but this is appears to be highly dependant on the network structure being suitable for the input and label types rather than on explicit regularisation (although explicit regularisation *can* marginally improve validation accuracy).

Generalisability also depends on how a model is fitted. The nature of the loss surface in a deep neural network is complex and non-convex yet with the right training constraints, SGD is able to fit a useful models as measured with a validation set. It is unlikely that in a network of massive scale the global minima is determined by SGD. It seems that in practice, finding a good local minima is sufficient for good validation performance[32] and that finding a global minima beyond this will tend towards completely fitting a training set as observed in [31] without improving validation results.

2.8 Non-Neural Approaches

The key benefit of CNNs over alternate models for image recognition tasks is that intermediate features and representations can be learnt with end-to-end training instead of being handcrafted. This is not to say that, with unbounded computational resources, neural networks will be able to learn any problem. In fact the handcraftedness of explicit intermediate features is replaced by handcrafted network structure. In many ways handcrafting features can be done with more rigour and conceptual understanding than the structuring of neural networks.

Scale invariant feature transforms (SIFT) were proposed by Lowe as a method for extracting distinctive features from images in a robust manner[4]. Distinctive colour patches of different size are identified as keypoints by comparing different scale gaussians of an image. These keypoints are given rotation and type descriptions based on pixel level functions applied in the region of the keypoint. This seemingly crude method for extracting features can be used by higher order matching functions to match objects and scenes based on similarity of these keypoints.

Another flexible feature extractor is the Histogram of Oriented Gradients (HOG) as proposed by Dalal and Triggs[5]. A grid of higher level features can be extracted from the image by sampling the colour gradient between adjacent pixels in a cell of a grid over the image and constructing a histogram of the direction and magnitude of the gradient per cell.

Both SIFT and HOG still rely on a secondary classifier such as an SVM to

classify and interpret the features they extract. The key is that the dimensionality of the image is reduced prior to the second classifier and that the features generated are robust to transformations like translation, scale and rotation.

Hand crafted feature extractors also exist to handle 3D point cloud data. Models exist to both compress point cloud data into denser representations such as polygon surfaces and to extract semantic information from point cloud data.

[33] presents an approach to detecting objects in a kitchen scene using hand crafted intermediate features. A 3D point cloud is transformed into a surface representation. The surface is fitted to prefer planar surfaces as these represent the true shape of many real world scenes. Cuboid objects are fitted to the surface geometry by generating discrete planes in the surface based on predicted edges. The cuboids are extruded to intersect with estimates of walls, floor or ceiling. The cuboid shapes are classified into objects by distinctive 3D features such as handles or knobs. A key limitation of this method is that it relies on cuboid models being a good estimation of shape and that the majority of 3D features in the scene are planar surfaces.

Another useful intermediate representation, particularly for object detection, is to use 3D keypoints. An approach outlined in [34] generates surface features of objects which are simplified with PCA and matched with input scenes. The method uses an offline training stage with hand labelled objects to determine high level features for different objects.

While hand designed algorithms still perform well on a number of tasks compared to neural networks, there is a great deal of uncertainty as to whether manually created features can be competitive with machine learnt features in the future given the continuous expansion of computational power. In particular, the high complexity of the leading hand crafted approaches mean they take substantial human time to develop.

2.9 Datasets for 3D Object Detection

There are many widely used academic data sets which allow researchers to investigate new network structures, algorithms and training procedures. Widely used sets such as ImageNet[7], KITTI [35], Pascal [36] and MS Coco[37] cover a broad range of computer vision tasks including image classification, object recognition, semantic segmentation and instance segmentation in both 2D and 3D. Image classification is the task of assigning a label to the whole image while object recognition is producing axis aligned bounding boxes for multiple objects within an image. Semantic segmen-

tation is pixel level labelling of the type of object that a certain pixel displays and instance segmentation is semantic segmentation where instances of a single object type are differentiated. All these datasets are generated using real world imagery and 3D scans and then hand labelled using methods such as crowdsourcing.

The KITTI dataset focuses on the inferencing challenges presented in the autonomous vehicle context. It was generated in Karlsruhe, Germany using a test vehicle at the technological institute of Karlsruhe. The KITTI dataset has become a baseline for many 3D recognition tasks as they pertain to self driving vehicles. The KITTI Dataset consists of six hours of driving data which equates to 15,000 individual image and point cloud samples that can be used for 3-D image recognition [35]. To put into perspective the different scales of data being used for deep learning in a commercial setting, Google's self driving car subsidiary, Waymo, reached 2.3 million total miles driven in November 2016 which is equivalent to over 350 million samples at 1Hz [38]. To generate a dataset of this scale has substantial cost that precludes most groups from being able to conduct research on deep learning tasks that don't have substantial public datasets.

Video games are an ideal mechanism to generate large datasets quickly and cheaply. The computing resources required to generate a synthetic dataset are cheap and readily available. Video games have full scene knowledge which can be dumped in an efficient manner to avoid manual labelling. Modern games also have photo-realistic rendering that can closely replicate the real world. Previous results using Grand Theft Auto 5 (GTA5) to generate data for training CNNs suggest that the rendering is similar enough to real world data that the domain shift from synthetic data to real world inferencing is small enough that the technique is feasible to train models for the real world[39].

Beyond the cost and speed benefits of synthetic training data, scenes can be generated to effectively handle a long tail of real world edge cases or to focus on only a subset of scenarios such as night time or poor weather. An example in the autonomous driving context is the prediction of pedestrians where accurate prediction of a pedestrian located on a footpath is inconsequential in most cases but vital if the pedestrian shows signs of entering the road space. If physical data were collected the data would be vastly skewed towards safe pedestrian behaviour but using a synthetic rendering system, scenarios can be generated to simulate important or rare situations in great quantity.

Using synthetic data to replace or augment real data is currently done in industry. Waymo uses an elaborate simulation system *Carcraft* to procedurally generate many

variants of real world data captures that engineers can use to backtest or train new algorithms. In 2016 alone over 2.5 billion simulated miles were driven, more than 1000× the number of real miles driven[40].

Chapter 3

Extensible Techniques for Evidence Generation

3.1 Design Choices

Training deep CNNs for image recognition requires large amounts of data to produce accurate results. This presents a significant challenge as collecting datasets of the scale necessary to inference highly complex scenes and labels is both costly and time-consuming.

There are a wide variety of game engines that could be used to generate synthetic data. Grand Theft Auto 5 (GTA5) has a number of advantages compared to alternatives. There is an active modding community that has developed tools to enable external software to interact with the game engine in realtime. For transferring trained CNNs to the real world, the models in GTA5 are detailed and varied and the game world is designed to replicate real suburbs of Los Angeles.

With the advent of cloud computing it is now possible to quickly and cost efficiently generate large sets using rented online computing power. Data collection was completed on the Amazon Web Services (AWS) platform. A g2.2xlarge instance type with a NVIDIA GRID K520 graphics card was utilised which has sufficient power to run a single instance of GTA5. Using AWS infrastructure allows a virtual server image to be snapshotted and subsequently used to start many server instances that can concurrently generate data. AWS S3 was used as the persistent data store for evidence after it was generated. This combination allows the collection method to scale horizontally with the only limitation being cost.

3.2 Collection Method

Within the game engine a rendering camera was attached to an in game vehicle and orientated to face forward relative to the vehicle. The camera was configured with a 90° field of view (FOV) in the horizontal direction which is equivalent to a 18mm focal length in 35mm film. An 18mm focal length is around the widest angle lens that commodity cameras use. Beyond this field of view, consumer grade cameras often have significant image distortion. The in game rendering camera is completely rectilinear, applying no distortion to the image. By comparison, consumer grade wide angle lenses would typically apply at least some level of barrel distortion to the final image. The in-game camera renders at the native screen resolution which was set to 1680×1050. This resolution was selected to closely mimic collection parameters of similar real world datasets. The KITTI dataset utilised Flea2 video cameras with a resolution of 1392×512 pixels and FOV of $90° \times 35°$.

Each scene was procedurally generated using the in-game driving AI and traffic generation. Software was written in C# to interface directly with the game process at runtime and make the vehicle to which the camera was attached drive like an AI car in-game. The existing in-game driving behaviour was used rather than create driving behaviour from scratch. This task was made substantial easier by building on existing game extension libraries ScriptHookV[41] and ScriptHookVDotNet[42] which together provide C# bindings to the functions exposed by GTA5 engine.

3.3 Rendering Process

The dataset was rendered and captured on a Windows server platform hosted on AWS. 3D rendering in a game engine is typically done through Microsoft's proprietary DirectX APIs which allow abstraction away from physical hardware to graphics orientated primitives that allow software to be agnostic to the type of graphics hardware. The effective use of these APIs is critical for collection of depth data in addition to visual data.

Render and scene data was sampled at 1Hz with the game running at normal speed. This frequency represents a trade-off between data generation taking longer and maximising the variety of the dataset. Increasing the sample rate would increase the data rate at the cost of reducing the variance of the dataset since sequential samples have higher similarity.

A simplified version of the Direct3D rendering pipeline can be thought of as a

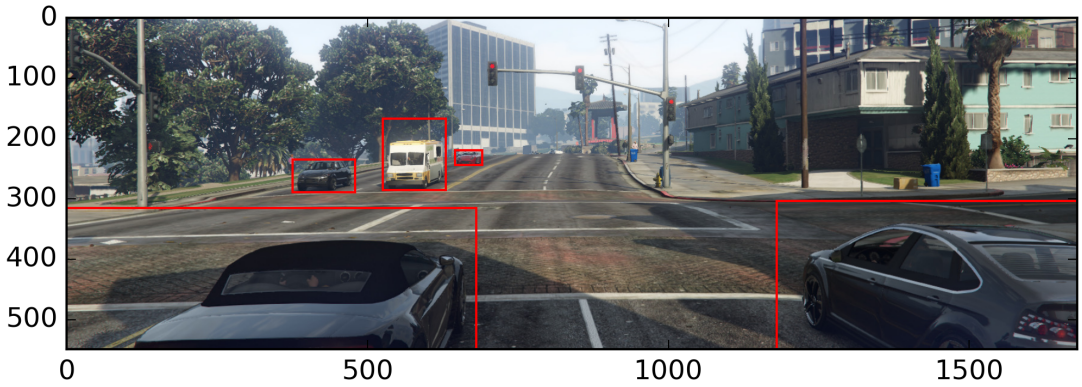


Figure 3.1: An example image capture after cropping to 1680×550 resolution with 2D bounding boxes overlaid. The bounding boxes were generated automatically as part of the data collection process.

sequence of linear transforms that take objects and project them to a 2D image plane.

1. Primitive model information is read that describes geometry, colour and effects to be applied. A model is typically a discrete object such as a vehicle, bin, lamp post or similar that is placed somewhere in the virtual world. Elements like the ground are typically built into a single large model representing the whole world. All the model information is transferred to the graphics processing unit (GPU).

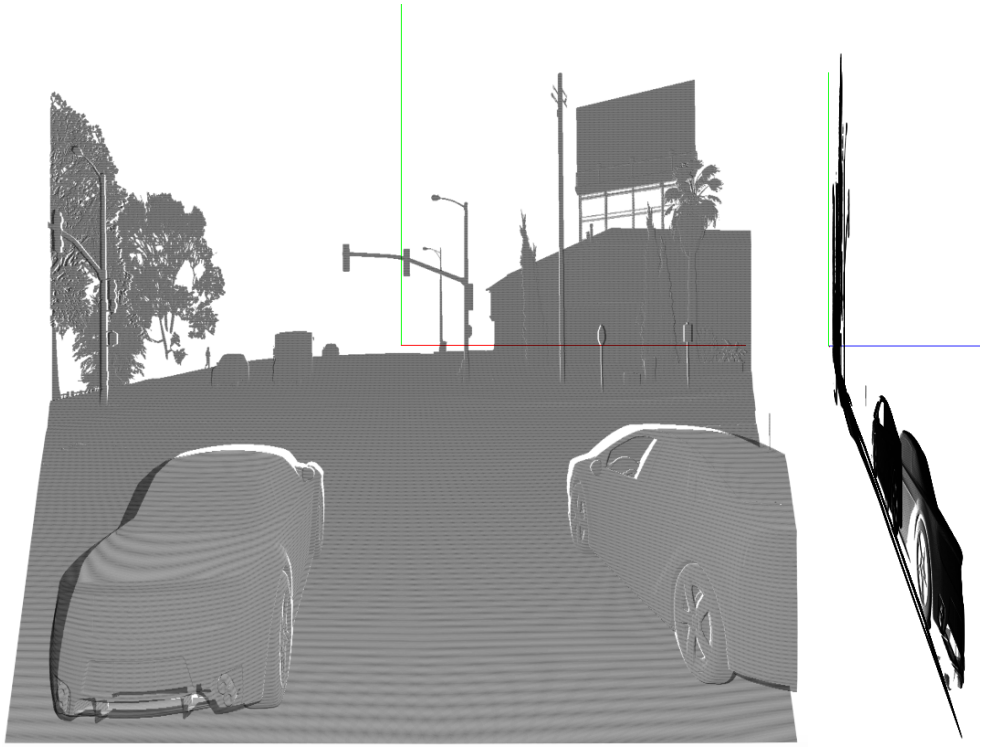
2. Each model geometry is placed into world coordinates then camera coordinates through two sequential affine transforms. In the camera space, the camera is located at $(0, 0, 0)$ and faces in the negative z-direction with coordinate system relative to the camera.

3. Geometry is projected to the clip space or viewport using a non-affine transform. This transformation captures the camera parameters like field of view and distortion. The matrix can also be used to apply effects such as an orthographic rendering instead of perspective.

4. Any geometry in the clip space outside of $([-1, 1], [-1, 1], [0, 1])$ is "clipped" from the frame.

5. The geometry is then rasterised into a 2D image, by flattening geometry in z-direction.

6. The rasterised geometry is transferred to the frame buffer which is read by the display.



(a) Front View (b) Side View

Figure 3.2: The point cloud projected into the clip space. The z-axis is scaled by factor of 10 to emphasise depth

Depth information is encoded into the clip space representation of the image as a *depth map* (the output of Step 4 in the rendering process and highlighted in Figure 3.2b). Typically this is used for depth ordering of different render layers (a common technique is to render objects separately then combine the final pixels based on which pixel is closer to the camera). The depth map of different render layers is used to determine which pixel in the clip space is nearest the camera point.

The normal DirectX3D rendering pipeline allows features to be extracted by dumping specific memory regions. To access critical areas of memory it is necessary to attach to the main rendering thread. This is done using the MinHook[43] and ScriptHookV[41] libraries in a manner similar to [39]. MinHook is a library that allows arbitrary code to be hooked onto existing functions within a process. The code is injected before the Direct3D method IDXGISwapChain::Present. This method is called to send a final rendered frame to the display memory which is read by the computer display. The injected code dumps depth, stencil and display buffers into memory readable by external code used to write the buffers to disk.

The depth information generated for the purpose of combining objects in the rendering pipeline is normally discarded once each frame has been generated. How-

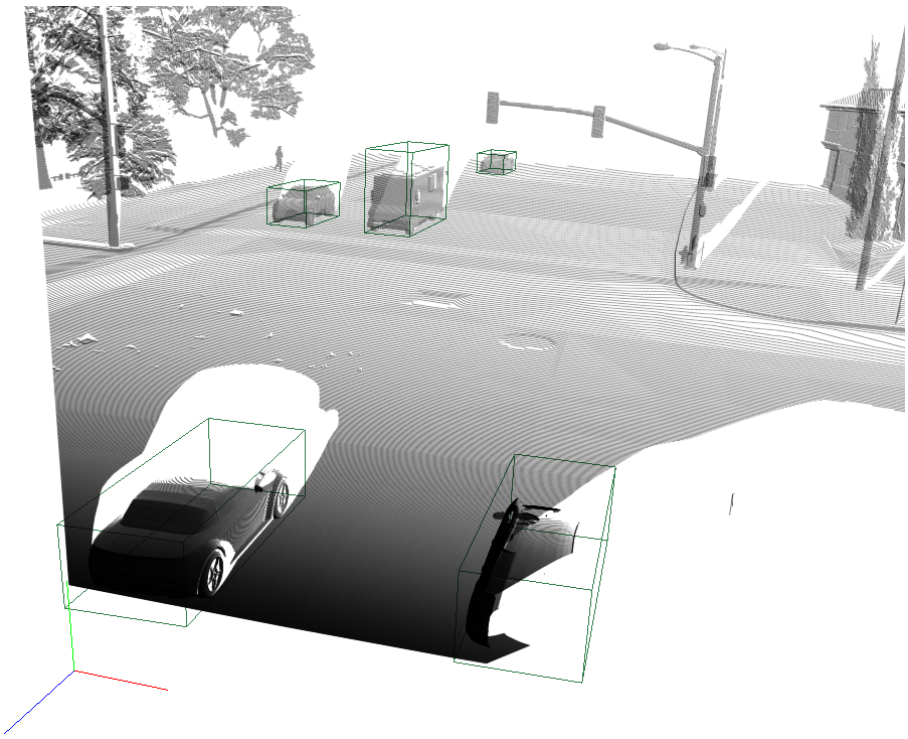


Figure 3.3: Point cloud in the view space with computer generated 3D bounding boxes overlaid. The camera axes are also drawn using standard RGB colouring representing X,Y and Z axes respectively in the view space. Note that the camera points in the negative Z direction.

ever this depth information is invaluable for the purpose of simulating real world non-visual sensor data. In order to capture this pixel level data the depth buffer is dumped concurrently with the display buffer. The display buffer contains 3 channels of 8bit colour data whereas the depth buffer contains a single channel of 32bit floating point values.

3.4 Data Generated and Discussion

9756 scenes samples were captured using the automated cloud-based pipeline. In the raw format, this totals 55GB of data. The data is written to disk in the local instance in realtime during capture and pushed to S3 at the end of each capture run. RGB data and depth data are stored in a single multipage TIFF per sample. The TIFF is compressed losslessly at original capture resolution of 1680x1050. Scene information is written to a JSON file per sample. The scene information contains the transformation matrices used in the rendering pipeline and information about what has been rendered including object bounding box sizes, positions and rotations. The

combination of coordinate transformation matrices and scene information allows 2D and 3D bounding boxes to be projected into the same projection as the visual data for use as labels for training.

The generated dataset has notable differences compared to typical academic datasets in the field such as KITTI. The synthetic data has 100% complete labelling where any object within the FOV is included in the label set no matter how occluded it is or how far it is from the camera. This artificially deflates accuracy scores when comparing performance on the synthetic dataset to datasets such as KITTI. Using a more difficult dataset has little bearing when computing 3D localisations in a disparate manner but these partially or occluded labels are critical if data is shared temporally. By maintaining all labels, predictions such as trajectory estimation can be better trained and evaluated.

The computer generated labels are based on the ground truth as it was rendered rather than an interpretation of the data after capture of the data as is common in academic datasets. This ensures that every label is perfectly positioned and categorised. Small variations in the positioning of labels and interpretation of object types lead to lower predictive accuracy after the same amount of training.

The full dataset as well as boilerplate python code are available publicly at `https://github.com/oscarmcnulty/gta-3d-dataset`.

Chapter 4

2D Methods for 3D Object Localisation

4.1 Introduction

Convolutional Neural Networks (CNNs) are the leading type of predictor for 2D image recognition tasks. The strength of existing 2D predictors is extended to create a 3D predictor that can operate on solely 2D visual data. Semantic label information is used to train a network to interpret rotation and knowledge of priors in the synthetic road scenes are utilised to fully determine 3D oriented predictions.

4.2 Network Implementation

4.2.1 Similar networks

The network constructed is similar to the Single Shot Detector proposed by Liu et. al [17]. It is trained on non-square images with a single object type where predictive classes are overloaded as rotations of the single vehicle object type. The early layers of the VGG-16 network [9] are used to initialise the network with weights pre-trained on ImageNet. These VGG layers act as a low level feature extractor for 2D bounding box generation and class prediction. The high level network structure can be seen in Figure 4.1 and the full structure in Appendix A.

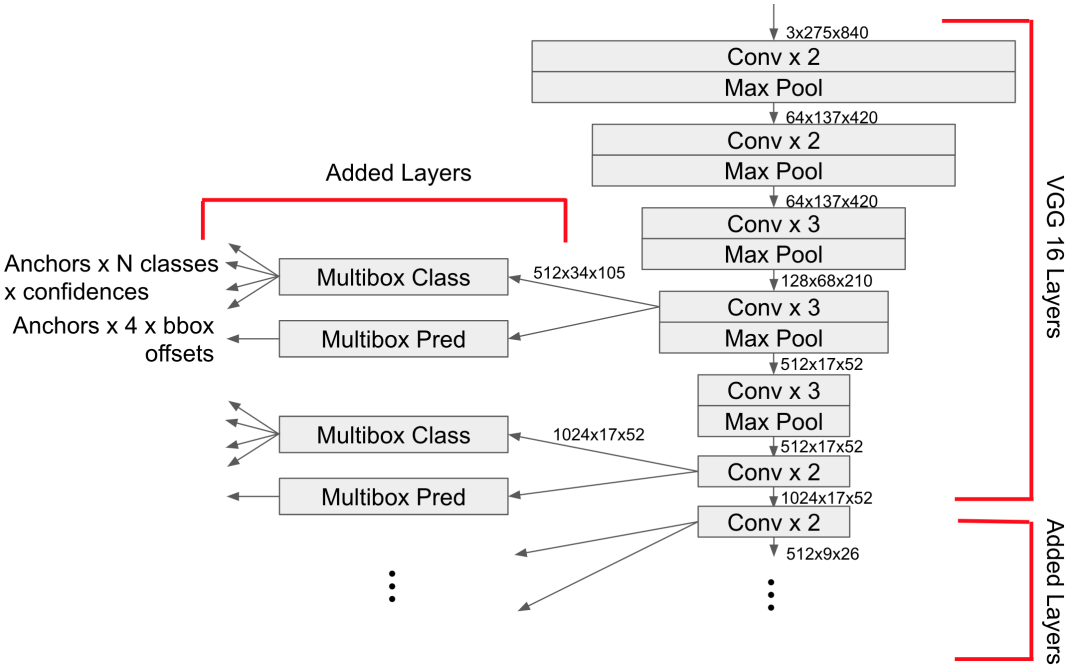


Figure 4.1: The basic structure of the network used.

4.2.2 Data Input

The network uses a data input of size $3 \times 275 \times 840$ which encompasses three colour channels of a 275×840 $(h \times w)$ image. The data generated in Chapter 3 is at a resolution of 1050×1680 which is cropped to 550×1680 by removing 300 pixels from the top of the image and 200 pixels from the bottom and then linearly down-sampling the image $2\times$ to reach the data used for network input. Using the full image data may increase accuracy slightly but would represent a 764% increase in input size which would result in a substantial increase in network size and training time. Additionally this would reduce the batch size to around 2 given the memory limit of the GPU used which would have a material negative impact on accuracy after a set number of training iterations.

4.2.3 Transfer Learning

The large number of parameters used in high performance CNN structures mean that training any network from scratch requires a large amount of data and compute time. One mechanism to minimise this is to utilise transfer learning where weights from an existing trained network are used as a starting point for a new network. Most high performing object detectors utilise the low level layers of a net pre-trained on Image Net data [7]. These low-level layers act as feature extractors, reducing pixel

Figure 4.2: An example input to the network.

data to semantic image features.

Training the VGG-16 detector network from scratch would be prohibitively expensive using available resources. When Simonyan and Zisserman trained VGG initially, it took 2-3 weeks using a machine with 4 NVIDIA Titan Black GPUs depending on the network variant. Training VGG-16 from scratch using an equivalent 4-GPU instance on AWS like g3.16xlarge would cost as much as $2300 at standard on-demand rates. Additionally, training from scratch requires complex multilayer training as random initialisation can stall learning because of the instability of gradients in deep network architectures. To avoid this when initially training VGG the network was trained with only its early convolutional layers and late fully connected layers. The intermediate section is then added to the partially-trained network[9].

To increase training speed and stability, the weights of the first four convolutional layers of VGG are frozen in the 3D object detector implemented here. Freezing the weights reduces the number of gradients that need to be calculated during each backwards pass which increases the training speed. The stability of training is also increased since there are substantially fewer free parameters. This potentially reduces the ability of the network to generalise the ImageNet training to synthetic data but this is unlikely to cause issues given the photorealistic nature of the data.

4.2.4 Using Anchor Boxes as Proposals

The network does not utilise a region proposal section as Faster-RCNN does. Instead a single predictive network is used with hardcoded proposals. 15568 hardcoded anchor boxes of varying size and aspect ratio are evaluated during each training pass or prediction. Anchor boxes are 2D, axis-aligned rectangles that represent all object predictions that can be made, every detection by the trained network can be attributed to a single anchor box. These hardcoded proposals are linked to multiple

layers of the VGG16 network structure as shown in Figure 4.1. In order to detect objects of various sizes the anchor boxes are linked to layers that roughly correspond with their size. Larger anchor boxes are connected to late layers in the network and small anchors to earlier layers. Since the smaller anchor boxes cover less area, more are required to sufficiently cover the image thus there are disproportionately more anchors connected to early layers than late layers. The 6 layers which have anchor connections have $(10710, 3536, 936, 260, 199, 27)$ anchors connected respectively from earliest layer to latest.

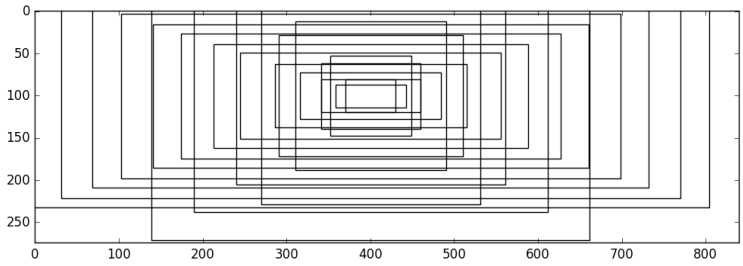


Figure 4.3: The range of anchor sizes and ratios utilised. These boxes are then tiled to cover the whole image at each anchor size level.

The same anchor box layer connections as proposed in the initial SSD network were used [17]. The height-to-width ratios of the anchor boxes were determined empirically using k-means clustering on 2D training labels to determine the 3 ratios which best covered all 2D labels.

4.2.5 Encoding rotation information

In order to extract 3D semantic information from 2D data the orientation of the vehicles is one-hot encoded into classes for training. A 45° granularity was used to segment the orientation according to their y-axis rotation. The object angles used for one-hot encodings are measured relative to the camera position instead of using their absolute rotation measured from view space axes in order to make the object class visual appearances invariant to their position in the view space.

4.2.6 Technical Implementation

The network was implemented using the MXNet framework [44]. MXNet is a deep learning framework that provides python bindings to efficient symbolic computation on multiple platforms spanning CPU, single GPU and multi-GPU based. The wide

range of platforms allows cross-platform development and experimentation where a single network graph can be trained on powerful workstation hardware and evaluation and adhoc testing can be conducted locally on low power hardware. The framework offers automatic differentiation to determine gradients based on the symbolic graph across a broad range of symbols.

The network is regressing both a class and adjustment to predetermined bounding boxes thus requires optimisation of some combination of both these factors in training. When optimising with a gradient descent type algorithm only a single loss function can be used thus some loss function needs to be created that is a combination of both. Cross entropy was used to measure loss of rotation angle classes and L1 loss was used to measure the inaccuracy of the 2D bounding box predictions. Prior to the loss function being evaluated, positive overlapping bounding box predictions are culled so that no two boxes have Intersection over Union (IoU) greater than 0.45.

4.2.7 Training

SGD was used to train the network on the generated GTAV data with a single Nvidia Tesla M60 graphics card on AWS. A learning rate of 0.001 was used and the network was trained for 104 epochs. A batch size of 16 was used which is the largest batch size that can fit in the GPU memory. Freezing the first two VGG layers substantially reduces the memory required which allows a larger batch size to be used.

A momentum parameter of 0.9 was used during training. Momentum combines gradients across multiple batches producing a similar effect to if larger batch sizes were used. It both reduces the training examples required and increases the stability of the learning process. Weight decay was applied across all weights during each training batch with a coefficient of 0.0005.

The data samples generated in Chapter 3 were split into a training and validation set. The validation set was excluded during the training process to provide an objective way of measuring the accuracy of the predictor and overfitting. The training-validation split used was 90/10.

Combining the different aspects of the SGD optimiser, each parameter in the

network is updated as follows after each batch.

$$\delta_i \leftarrow \text{momentum} \times \delta_{i-1} - \text{learning rate} \times \text{gradient} - w_d \times \text{weight}_{i-1}^2$$
$$\text{weight}_i \leftarrow \text{weight}_{i-1} + \delta_i$$

The parameter change δ is persisted for each parameter between batches since momentum depends on the previous step in the optimisation. The network for was trained for 100 epochs or 61k batch iterations prior to the validation accuracy plateauing as shown in Figure 4.4.

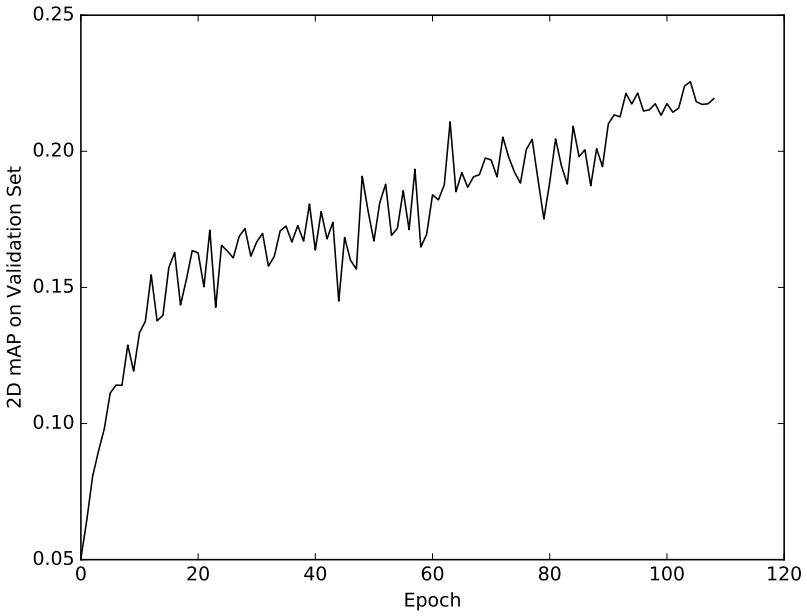


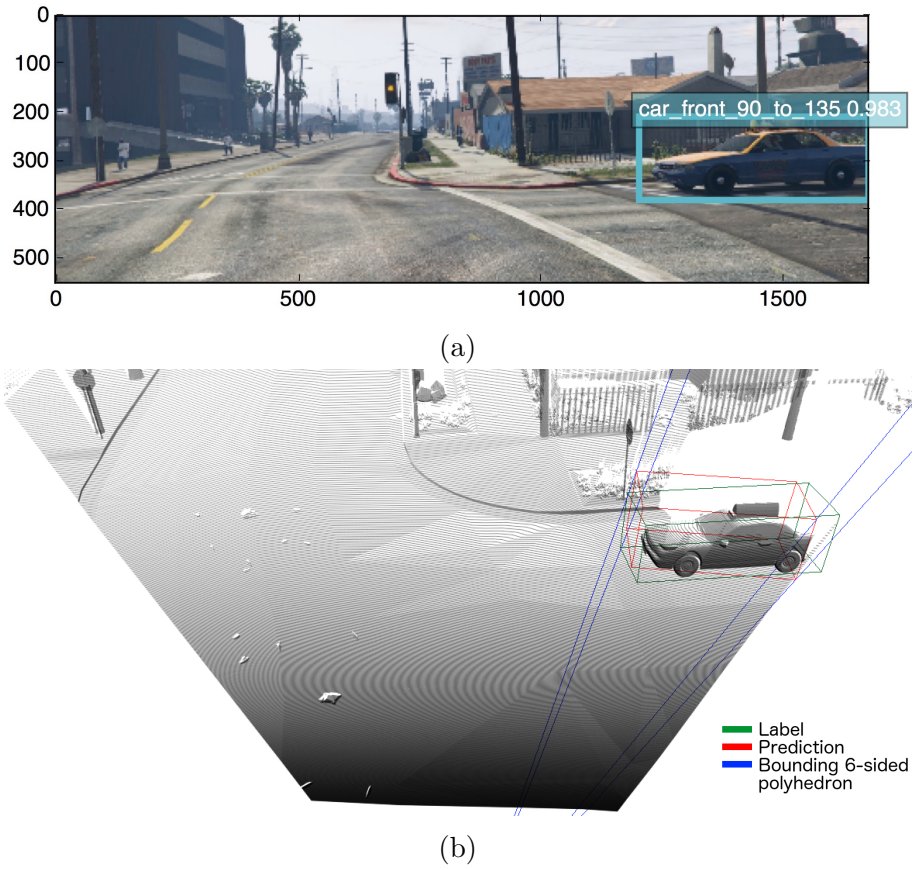
Figure 4.4: Accuracy of net during training.

4.3 Creating 3D predictions from 2D augmented predictions

There are two outputs from the network for each anchor box. The first is the class likelihood which is overloaded to one-hot represent rotation and whether the anchor box contains a vehicle. The second is the 2D bounding box adjustment which adjusts the anchor box edges for a tight 2D fit. Together these can be used to infer 3D position and rotation.

Given a 2D bounding box with 4 corners in format $((x_1, y_1), \ldots (x_4, y_4))$, a 3D bound on the object location can be derived. A 2D image is in the clip space

(as shown in Figure 3.2) which has a linear mapping to 3D view space. Recall that points in the clip space are represented with (x, y, z) where z is the depth map. Thus the possible locations of a vehicle in clip space are bounded by the rectangular prism from a 2D bounding with the 8 corners $((x_1, y_1, 0), (x_1, y_1, 1), \ldots (x_4, y_4, 0), (x_4, y_4, 1))$. This rectangular prism can be projected into view space using the inverse of the projection matrix (which is captured for each sample in the data generation process). This transformation creates a 6-sided polyhedron as shown in Figure 4.5b.



(a)

(b)

Figure 4.5: (a) A 2D bounding box prediction. (b) The bounding box in (a) projected to 3D with corresponding 3D prediction made using the ground plane assumption.

Once a tight 6-sided polyhedron has been ascertained in 3D space an oriented 3D bounding box is fitted to that space. The 3D bounding box is taken to be axis aligned in the y-axis (upwards axis in the view space). Rotation about the y-axis is determined by the rotation class prediction adjusted for the relative angle of the 2D bounding box centre since the rotation classes were initially measured relative to view angle. The midpoint of the angle class is taken such that for the class representing the rotation range $[90°, 135°)$, the relative angle is $112.5°$.

To remove an unknown from the system, the vehicle length-to-width ratio for predictions is fixed at 2.8 : 1. This ratio is fitted empirically using median ratio from the training dataset. Additionally the 3D bounding box is assumed to be axis aligned with the y-axis in the view space.

With these assumptions the problem now has a single degree of freedom, the depth of the object within the 6-sided polyhedron. Two approaches to solving this unknown are tested.

The first approach assumes the bottom of each vehicle is resting on a constant ground plane. The intersection of the 6-sided polyhedron of allowable space with the theoretical ground plane provides the depth of the nearest vertical edge of the bounding box which is sufficient to solve the system. The ground plane offset from the camera is measured from the exact placement of the camera in the virtual world which is 1.98m from the ground.

The second approach assumes a constant vehicle height such that the height of the 2D bounding box can be mapped to the depth of the nearest vertical edge of the bounding box. The median height of the vehicles in the training set is empirically determined to be 1.9m. The vertical FoV of the camera without any cropping is 56°. The depth of the vehicle is them mapped to the 2D height through the FoV which allows depth the be predicted.

The python code for fitting using the two different assumptions is presented in Appendices B.1 and B.2.

4.4 Results

The predictor creates 3D oriented bounding box predictions with associated confidences. The accuracy of these predictions is evaluated using the intersection over union (IoU) metric which is equivalent to a volumetric Jaccard similarity metric in 3D. An IoU threshold for a true positive prediction is set to 0.3 which is equivalent to 46% of the prediction intersecting with the true 3D label if the label and prediction have equal volume. Predictions that overlap less than the IoU threshold are treated as false positives. This metric provides a joint evaluation of the localisation, pose estimation and classification (in the sense that bounding box sizes are not fixed) performance of the network.

The overall performance of the predictor on the validation dataset is measured using mean average precision (mAP). The mAP metric is conceptually similar to the area under the ROC curve for a binary classifier but accounts for the lack of

a 1:1 relationship between a prediction and label and that . Assume a set of m labels with n predictions made across the whole validation data set with descending confidences $c_1, \ldots, c_n$ and a variable x_i as described in Equation 4.1. The mAP for the predictor across the whole dataset is then determined by Equation 4.2. This calculation for mAP is equivalent to the area under curve in Figure 4.6.

$$x_i = \begin{cases} 1, & \text{if prediction } i \text{ exceeds IoU threshold with a label not} \\ & \text{previously identified by a higher confidence prediction} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

$$\text{mAP}\% = \sum_{i=1}^{n} \frac{\sum_{j=1}^{i} x_j}{i} \times \frac{x_i}{m} \tag{4.2}$$

Inferences using the ground assumption achieve a mAP of 3.6% and inferences using the constant vehicle size assumption achieve a mAP of 1.8%. Assuming a constant ground plane offset substantially outperformed the vehicle size based method for depth prediction.

Analysis of the results show that while 2D predictions are highly accurate the depth estimation performs poorly. The inaccurate depth leads to a large number of correct 2D predictions not reaching the IoU threshold in 3D. The predictor demonstrates much higher accuracy on closer vehicles where both the depth estimation is more robust and the 2D predictions are more reliable.

Vehicle rotation classes are predicted accurately in almost all cases when a positive prediction is made. While the granularity of the rotation encoding causes some loss in precision, the poor depth estimation causes a much larger reduction in performance.

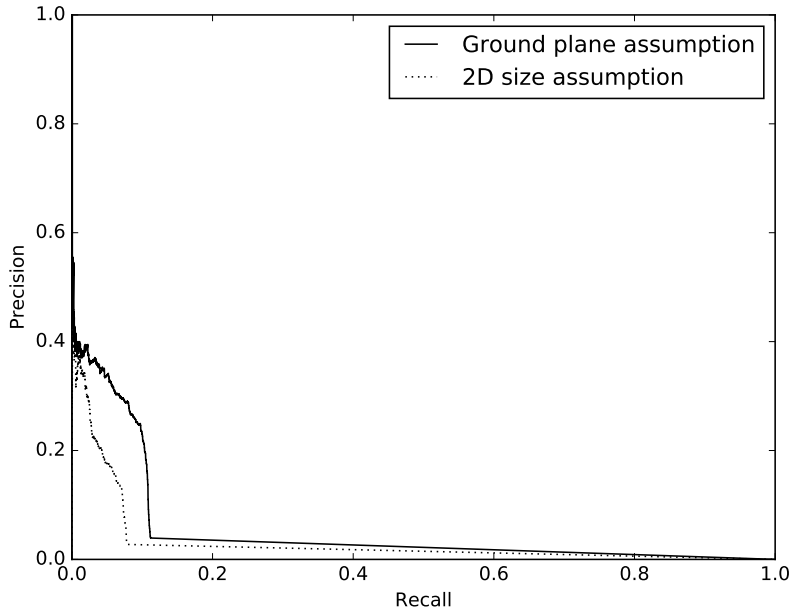


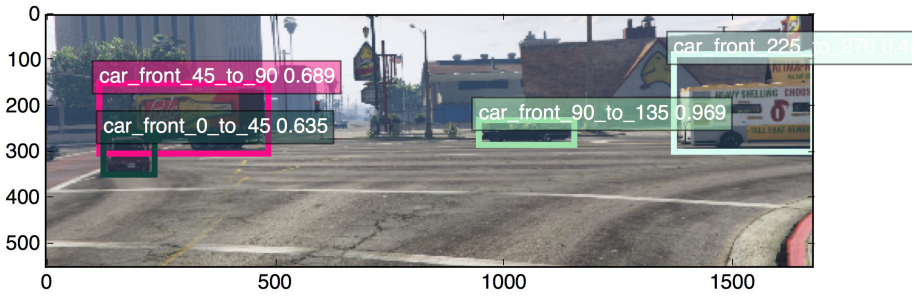
Figure 4.6: Precision vs recall as the confidence threshold for discarding a prediction is decreased. Measured at 3D IoU threshold of 0.3.

4.5 Discussion

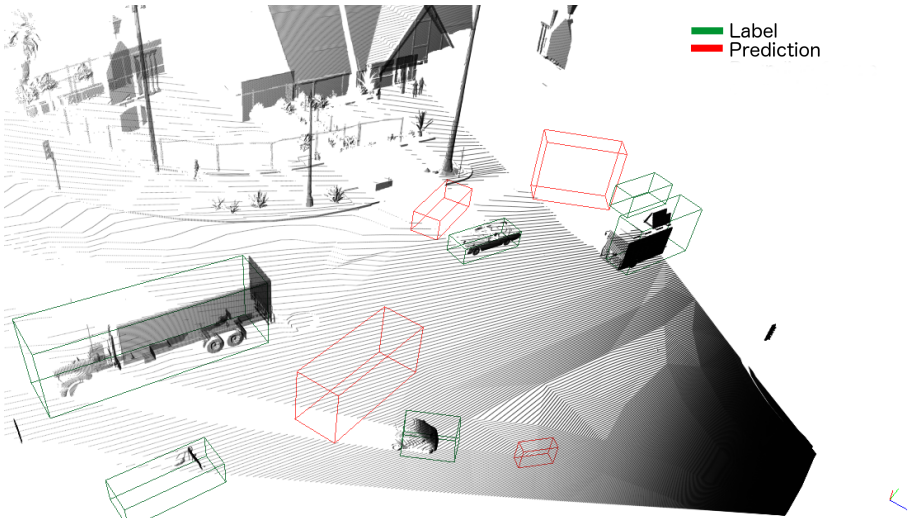
4.5.1 Limitations of 2D predictions and future changes

This novel 2D method for object detection has inherent limitations. The two assumptions to predict object depth are weak in both real world and simulated data.

The problem of depth estimation for distant objects is poorly conditioned for both assumptions. Small variations in the 2D bounding box estimation result in large depth changes for both the ground plane and vehicle height assumptions (as seen in Figure 4.7b). With both assumptions the sensitivity of the depth is related to bounding box inaccuracy by a $\frac{1}{\sin\theta}$ relationship. θ is the angle between the horizon and bottom of 2D bounding box for the ground plane assumption and is the estimated field of view angle of the bounding box for the vehicle size assumption. This results in depth being highly sensitive to bounding box inaccuracy prediction at high vehicle distance.



(a)



(b)

Figure 4.7: (a) Accurate prediction in the image plane. (b) Depth inference is highly sensitive to inaccurate 2D bounding box prediction, particularly further from camera.

In practice, the use of a constant vehicle length-to-width ratio has varied results. The median ratio is accurate for the majority of vehicles which comprise of sedan and SUV type vehicles. The 2.8 ratio is too small for larger commercial vehicles which have a higher length-to-width ratio. Compact cars are also poorly represented as the length-to-width ratio is smaller than the true ratio. A future iteration on the predictor might fit these ratios more intelligently as an additional set of class outputs.

The constant ground plane assumption varies in its truth. Particularly for near objects, the physical ground plane does not vary substantially and results in accurate predictions such as in Figure 4.5b. As distance from camera increases the variance in ground plane height increases which reduces the accuracy of the assumption. This effect is compounded by the issue of high depth sensitivity at large distances.

A constant vehicle height assumption performs worse than the ground plane assumption. This is likely due to the large variance in height across vehicles. The height method could be improved by regressing vehicle height from the CNN instead

of measuring a single height across all vehicles.

While the vehicle rotation class encodings are accurate the 45° granularity could be reduced to increase the precision of the rotation predictions. This would be achieved by increasing the number of one-hot encoding classes which would lead to a longer training time and possibly more data required to reach the same level of class prediction accuracy.

An edge case that is not handled well is when a vehicle is occluded by the edge of a frame. The network is trained by cropping any labels that fall outside the frame boundary. This is necessary to avoid the training instability introduced if an anchor of a given size needs to be optimised for predictions with a full feature map but also a partial feature map with some padding feature. If this inaccuracy needs to be improved the most effective solution would be to increase the FOV of the visual data to fully encompass all training labels even though this will increase data size and decrease training speed.

4.5.2 Network computational performance

MXNet can be run on CPU or GPU based platforms. Utilising the highly parallel nature of GPU computing greatly increases the speed of training. On a 2.9GHz Intel Core i5 a forward and backwards pass takes 4s whereas a forward and backward pass takes 0.1s on the NVIDIA Tesla M60 using the network structure outlined above.

Chapter 5

Detection with Point Cloud Data

5.1 Introduction

Sensors that capture 3D information are commonly used in real world robotics to augment visual sensory data. These sensors often provide data that can be represented as an unstructured 3D point cloud. Alternatively stereo imagery is sometimes used to generate 3D information for use in further predictors. Extracting high level features from point cloud data presents a number of issues when compared to 2D methods. The sparse encoding of the point cloud makes it computationally expensive to process this data using typical methods that operate on dense representations.

The strengths of both point cloud and visual methods complement each other. Visual CNNs excel at extracting semantic information while point clouds allow accurate position determination. By combining the semantic class (rotation) information and 2D positioning generated by visual predictions with depth information derived from other sources the large uncertainty in depth in the 2D visual predictor can be reduced.

5.2 Data Used

Depth data was captured concurrently with image data in Chapter 3. The depth map was dumped synchronously with image data so that for each image there is a virtual 3D point cloud similar to what would be captured with high resolution LIDAR or radar with a FOV of 90° forward. The depth map is captured in the clip space on the interval $((-1, 1), (-1, 1), (0, 1))$ at the same pixel granularity as the image data. The depth map in the clip space is shown in Figure 3.2.

This depth information can be transformed to simulate different sensors. For example, to simulate a 2D radar array the data could clipped to only cover a reasonable 2D plane and then downsampled to match the resolution of the radar. This allows a full array of sensors to be simulated without the cost of constructing a test rig and capturing data with many different sensors.

For this experiment the depth data was projected to match the cropped field of view of the 275×840 image input to the predictive network in Chapter 4.

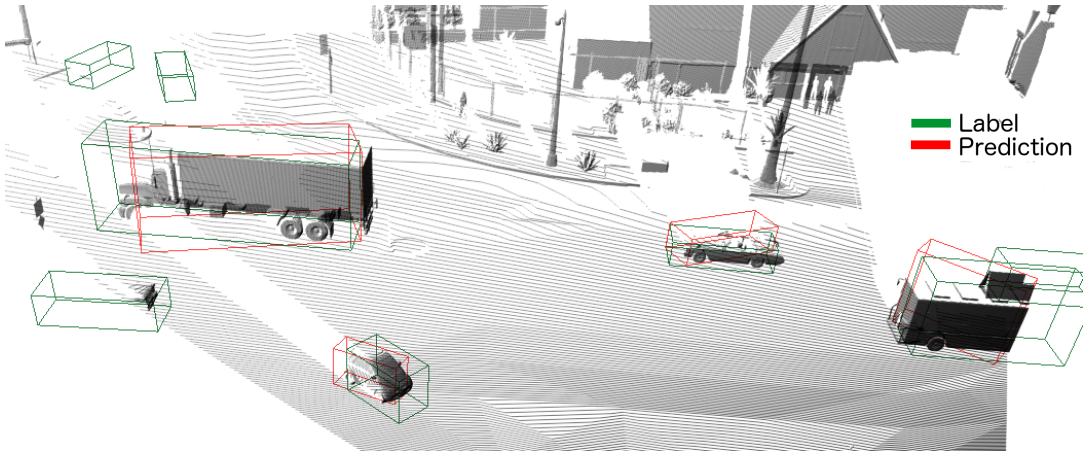


Figure 5.1: 3D predictions created by augmenting the 2D predictions with 3D depth data.

5.3 Fitting Algorithm

Similar to Chapter 4, a 3D bounding box is fitted to a tight 6-sided polyhedron created from a 2D image prediction. For each 2D prediction from the image-based network, the depth data corresponding to the area projected by the 2D bounding box is cropped from the total point cloud. In the 3D view space, the crop boundary is the same 6-sided polyhedron as used in the fitting algorithm in Chapter 4.

The addition of depth distribution as an input to the bounding box fitting algorithm removes the need to make weak assumptions about depth. The distance to the camera is calculated for each 3D point contained within the 6-sided bounding polyhedron. This gives a distribution of depth for each prediction area which is used to estimate the depth of the nearest vertical bounding box edge. It was found that taking a percentile of the depth distribution and adding a constant offset outperformed other methods. The 35th depth percentile and an offset of 0.9m were used based on empirical investigation using the training set.

5.4 Results

The 3D performance of the predictor is measured using the same methodology as described in Section 4.4. Augmenting the 2D image predictions with depth information using the method described yields a mAP of 11.6% which is a substantial improvement on the 3.6% mAP using only visual data with the ground plane assumption.

Inaccurate depth estimate from visual data is an inherent weakness of image based methods. By augmenting the image predictions with depth data the 3D predictive accuracy measured through IoU is greatly improved compared to the image only methods in Chapter 4.

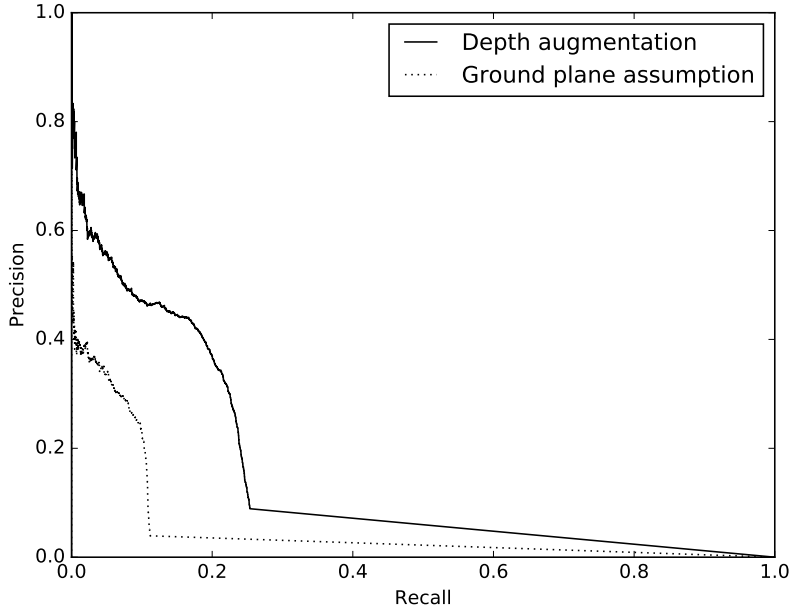


Figure 5.2: Precision vs recall as the confidence threshold for discarding a prediction is decreased. Measured at IoU threshold of 0.3.

5.5 Discussion

The addition of 3D physical data significantly increases the accuracy of the depth prediction compared to visual only methods by removing the largest error source. Analysis of the results suggests that low recall of the 2D visual CNN predictor is the most significant sources of missed predictions. The true positives are biased towards labels near the camera due to the higher accuracy of the of 2D CNN at larger 2D

bounding box sizes.

The precision and recall of the 2D predictor are dependant on a number of factors. The dataset used does not prune 'hard' labels like in KITTI or other similar datasets. Thus much more difficult object detections remain in the dataset that are partially or fully occluded or distant from the camera. Increasing performance on these hard labels is increasingly difficult. Increasing the size of the training dataset and training for longer would improve the accuracy on occluded labels. The extent of this improvement would need to be measured empirically by undertaking the larger scale training which is limited by the available computational resources.

Increasing accuracy on distant objects might be achieved through solely more data and training but there are structural limitations in the network as to the size of predictions. Downsampling the 2D input image directly reduces the amount of information available for distant vehicles. Removing this downsampling would substantially improve distant vehicle prediction but at the cost of a significantly larger network.

While the early layer connections of SSD type networks improve small object recognition over single feature layer connections like in YOLOv1, the predictor is still not truly scale invariant. The earliest connection to the VGG layers is made after the feature map is $8\times$ downsampled (the features represent 8×8 pixel segments). This puts a structural limit on the size of predictions. Adding an earlier layer connection to create smaller predictions would increase the network size with a huge number of new anchors but may result in increased accuracy without increasing the training size input.

The granularity of the rotation classes leads to a small precision loss that reduces the IoU of predictions. While this is not a large source of error, it is expected that reducing the granularity of the classes would improve predictions at the cost of longer training and possibly a larger amount of data required.

The combined percentile and offset model used to extract depth from a sparse point cloud has limitations. It is tuned to represent the majority of predictions but for partially occluded predictions this naive method does not correctly interpret depth. In works such as MV3D[27] this problem is overcome by integrating a front view of depth into the 2D network. By training this end-to-end the network learns to predict depth based on semantic information from the visual input which increases accuracy.

Chapter 6

Birds-Eye Representation of Depth Information

6.1 Introduction

One approach to object recognition with 3D data is to project the data into a 2D plane then use 2D convolutional techniques to classify and localise objects. Ideally, an efficient 2D representation should be able to fully represent the 3D information. The birds-eye plane is of particular interest in the area of vehicle detection since vehicles are typically axis-aligned with the y-axis in view space (they should have 4 wheels on the ground). Additionally, the knowledge of a ground plane prior means that vehicles can also be accurately located on the y-axis in view space. The use of an orthographic projection also vastly reduces the size variance of the labels compared to the perspective front view projection. A suitable 2D projection such as a birds-eye view allows the problem to be reduced to a 2D localisation problem.

6.2 Encoding sparse point cloud data to 2D visual data

Previous attempts have been made to re-project 3D point cloud data to different views and efficiently encode the data such as by Chen et. al in [27]. In their network, MV3D, they utilise a 2D encoding of sparse 3D LIDAR data as input for an object predictor. In combining this birds-eye approach with standard 2D approaches they achieve state-of-the-art performance on the 3D recognition task in the KITTI dataset. They utilise a combination of maximum height, density and

laser reflectance measures to compress the sparse 3D data into full 2D data on a grid with a 0.1m resolution.

Three metrics were selected to encode the synthetic 3D point cloud: maximum point height, point density, and the range of point heights. These three measures were encoded into a 512×512 pixel grid of 24bit colours. The point cloud was clipped prior to encoding such that only the 70×70m area immediately forward of the camera was used. Thus the pixel grid is at a physical resolution of 0.137m. The density of points decreases greatly as the distance from the camera increases. To minimise the effect of this all the density measurements were scaled by d^2 where d is the distance of the grid space from the camera neglecting y-offset in the view space.

The 3D bounding box labels were one-hot encoded into classes based on their rotation around the view space y-axis. Unlike the encoding used in Chapter 4, the front of the vehicle is not considered, rather the angle of the vehicle is considered on a 180° interval neglecting the way the vehicle is facing.

The data set contains 3D bounding boxes that are axis-aligned with the y-axis thus generating orientated 2D bounding boxes is straightforward. From 2D orientated bounding boxes, 2D axis-aligned bounding boxes are created that fully contain the orientated boxes. The axis-aligned boxes as shown in Figure 6.1 are used as the labels for training.

6.3 Network Implementation and Training

6.3.1 Network Structure

A single shot network similar to Chapter 4 is constructed for creating detections. The same loss function and optimisation strategy are utilised but a number of critical changes are made to improve the accuracy of birds-eye predictions.

The early layers of the network are still structured based on VGG and initialised using weights pre-trained on ImageNet. The use of the existing weights of VGG will not provide the same benefit as it does for image inferencing, the domain shift is much larger to the encoded birds-eye representation. For this reason the first four convolutional layers are not frozen to the VGG initialisation values during training. Unfreezing the early weights increases training time and reduces training stability since there are more parameters being jointly optimised but this is done since the low level image features in the birds-eye encoding differ substantially to the features in the ImageNet dataset. Initialising using the VGG16 weights should still provide

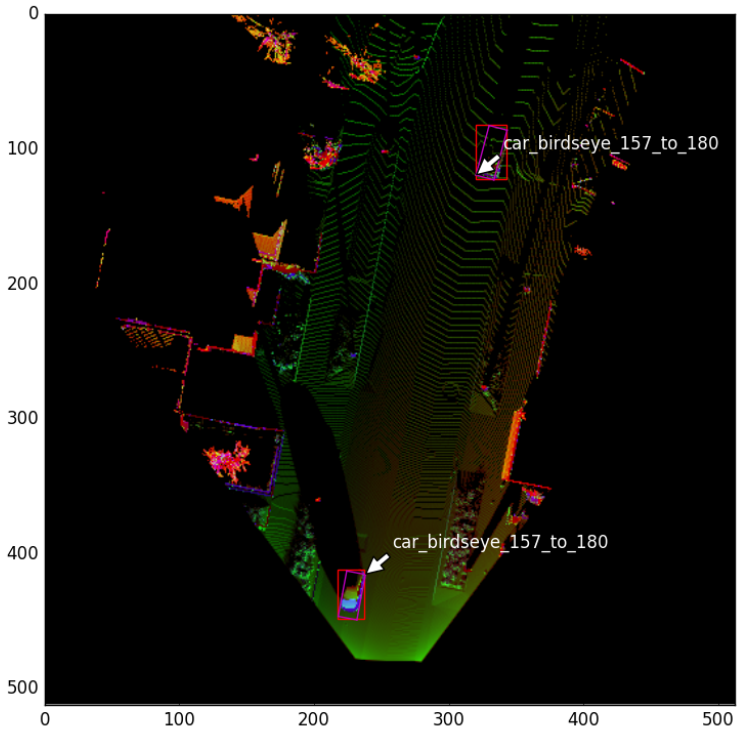


Figure 6.1: Birdseye encoding with training labels overlaid. Both the axis-aligned and oriented bounding box labels are shown.

benefit over random initialisation of weights but further training is necessary to fully capture features unique to the birds-eye encoding.

The network is trained with a batch size of 16 which is the largest batch that can fit in the GPU memory. While the input data size is 13% larger than the image detector in Chapter 4 and more gradients are calculated for the unfrozen layers, the reduction in anchor boxes and removal of the late layers of VGG substantially reduce the memory required which increases the allowable batch size. A larger batch size both increases the training speed and should improve final accuracy over training with smaller batch sizes.

6.3.2 Selection of anchor boxes

The birdseye encoding projects vehicles such that they will be a constant scale no matter their position in the birdseye plane. This enables substantial simplification of the anchor boxes used compared to the perspective projection of the front view. Recall in Chapter 4 that a vast number of anchor boxes were required to cover

potential object positions and sizes within the image. In the birds eye representation only a small number of shapes are required. In order to determine the optimal sizes and ratios of the anchor boxes a hierarchal k-means clustering was used. Firstly, 3 size groups were clustered on rectangle area. Secondly, the ratios were clustered into 6 groups segmented on size cluster. To avoid biasing towards vehicles facing in the y-direction as shown in Figure 6.2 (which represents vehicles with the same rotation as the collection vehicle) a rotational symmetry in the anchors is enforced. This effectively reduces the clustering to 3 ratios per size level.

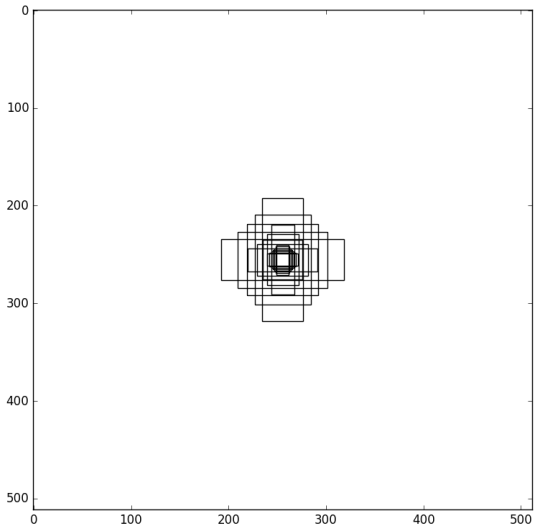


Figure 6.2: The range of anchor sizes and ratios utilised for the birdseye encoding.

6.3.3 Training

The network for was trained for 104 epochs or 63k batch iterations on the dataset generated in Chapter 3 encoded into the birdseye view. After this point the validation accuracy appeared to have plateaued. The same 90/10 training-validation split as Chapter 4 was utilised such that validation set performance can be directly compared.

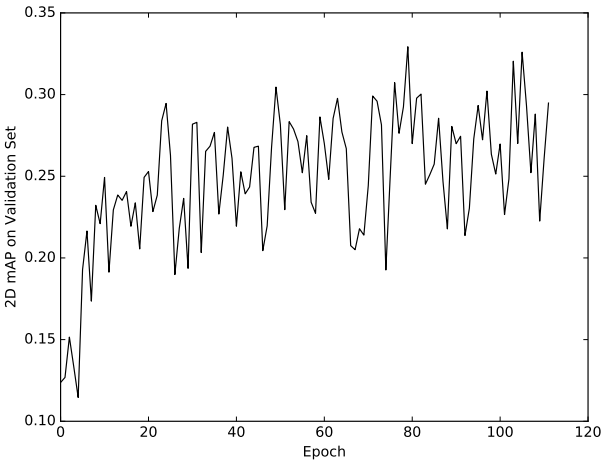


Figure 6.3: Accuracy of net during training.

6.4 Prediction Methodology

A 3D bounding box is fitted to a 2D axis-aligned bounding box similar to previous chapters. An axis-aligned 2D bounding box along with an assumed vehicle width-to-length ratio is sufficient to fully define the 2D oriented bounding box in the birds-eye view. To obtain a 3D box the 2D box is extruded to a height determined by the 3D point cloud data.

The rotation of the 2D oriented box is determined from the class of the prediction. The midpoint of each 22.5° class is used such that a prediction of rotation between 45° and 67.5° will use a rotation of 56.25° for bounding box construction. Once a bounding box of correct rotation and ratio is constructed it is scaled to the largest size possible while remaining contained in the axis-aligned 2D prediction. The algorithm for this can be found in Appendix B.3

The height of each prediction is not extracted directly from the network or birds-eye encoding. Instead a slice of the synthetic 3D point cloud is taken for each prediction. From this slice the road height and vehicle height are predicted by taking the 2nd and 98th percentile points on the y-axis respectively.

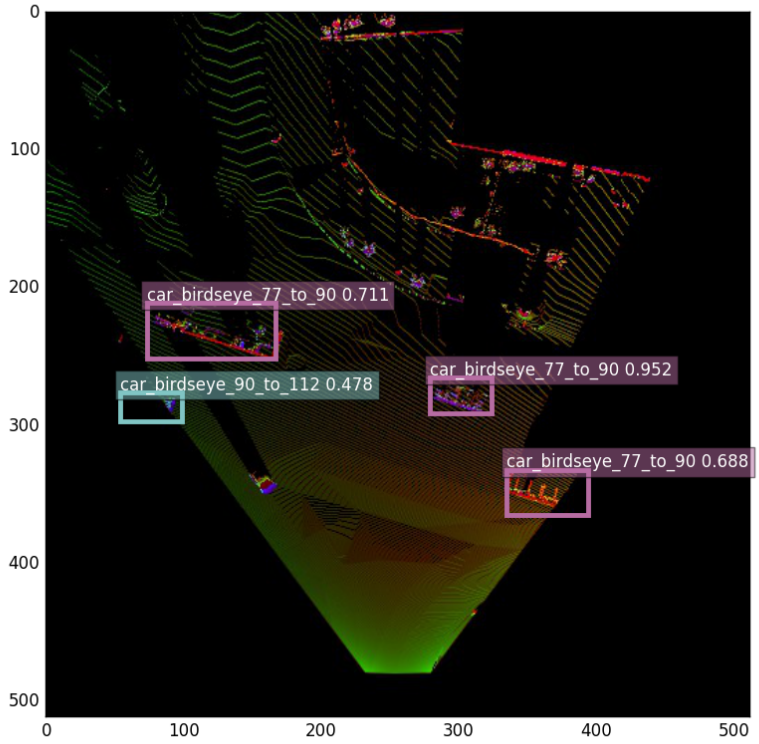


Figure 6.4: The same scene as Figure 4.7b & 4.7a encoded in the alternate birds-eye representation. 2D detections using the trained birds-eye network are overlaid.

6.5 Results

The 3D performance of the predictor is measured using the same methodology as described in Section 4.4. Using the birds-eye predictor a mAP of 48.3% is achieved. This represents a significant improvement over the best performing method explored previously, the combination of vision and depth in the front view, which achieved an 11.6% mAP.

Analysis suggests that the primary driver of imperfect recall is 'hard' labels that are partially or fully occluded. The predictor accurately predicts almost all fully visible examples correctly with precise bounding boxes. The largest source of IoU reduction is inaccurate height estimation where the bottom of the vehicle is not accurately determined due to the point cloud being occluded.

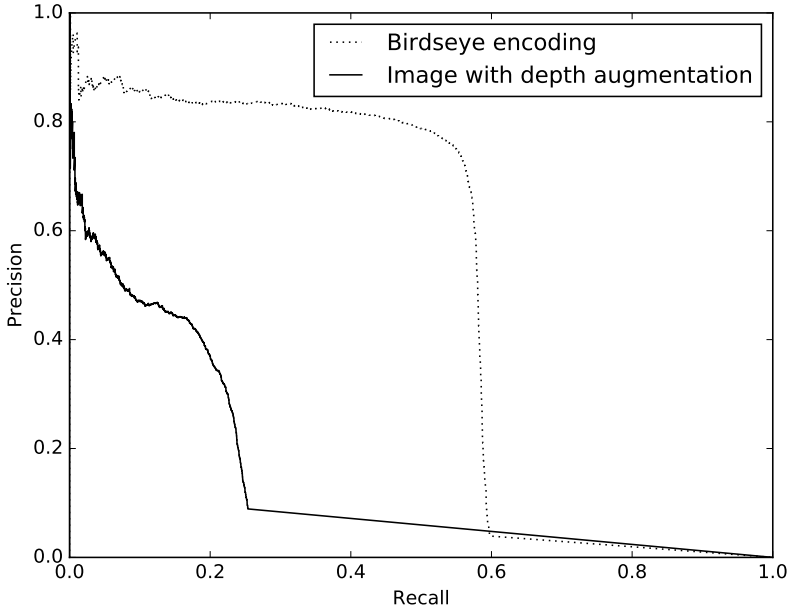


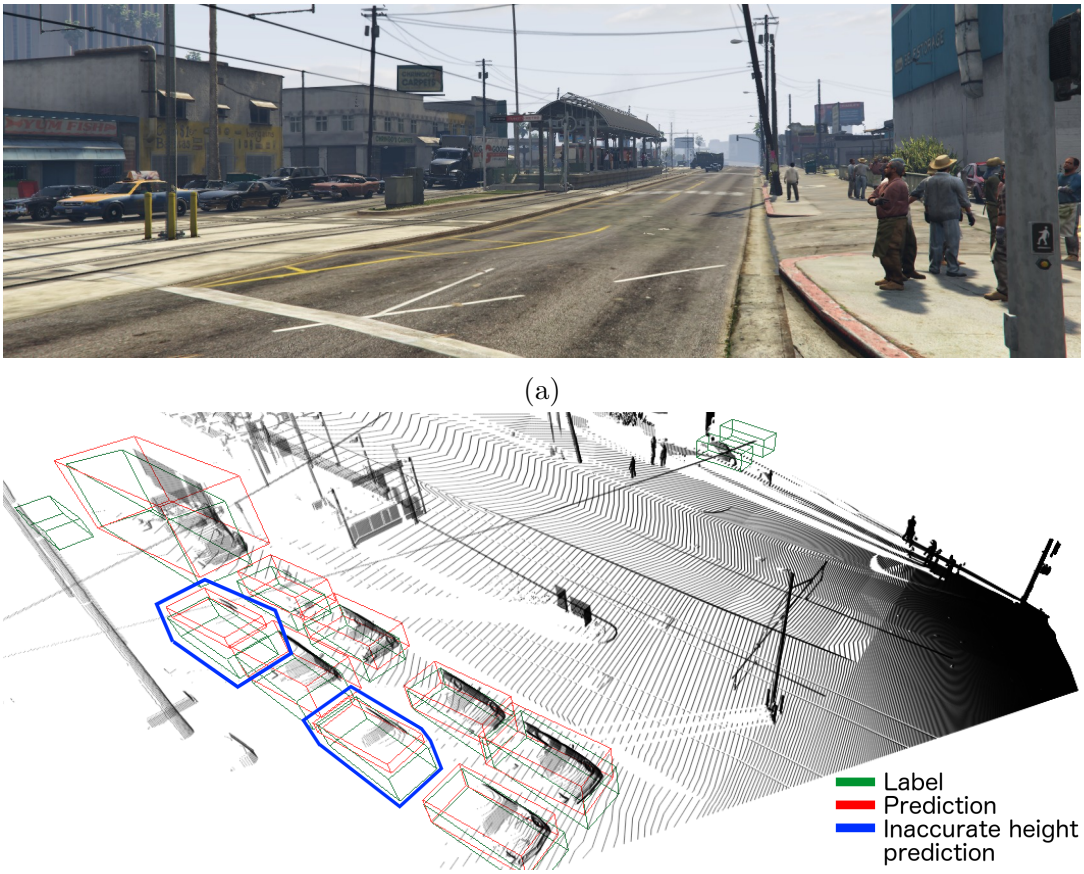
Figure 6.5: Precision vs recall of birds-eye predictor as the confidence threshold for discarding a prediction is decreased. Measured at 3D IoU threshold of 0.3.

6.6 Discussion

The substantial increase in accuracy compared to the visual front view predictor can be explained by a number of factors. The orthographic birds-eye encoding regularises the size of the 2D vehicle representations compared to the perspective projection encoding of the visual data. This improves accuracy since the network can be trained to optimise for a single feature size instead of requiring that intermediate features provide scale invariant information. The accuracy improvement may also suggest that the birds-eye encoding is a more efficient representation of the scene data than the visual front view. A corollary to this is that the visual front view could potentially reach the same accuracy but would require more data and training.

The largest source of inaccuracy in the birds-eye predictor is from false negatives on difficult labels. Improving performance on these hard labels is not straight forward. It likely would require more training data and longer training to improve prediction of these labels and even then, an improvement is not guaranteed.

More complex changes could be made to leverage the data differently, particularly the use of semantic information from visual data to augment the purely physical data contained in the point cloud similar to the technique presented in Chapter 5. By only considering one source of evidence the maximum accuracy of the predictor is



(a)

(b)

Figure 6.6: (a) Visual scene representation. (b) 3D predictions and labels created by the birdseye predictor.

likely capped.

The percentile based height estimation performs well when vehicles are not occluded but when a vehicle is partially occluded the accuracy is limited. Primarily the bottom of the vehicle is obscured from the camera point which results in the minimum height not being correctly estimated using the point cloud data as shown in Figure 6.6b. This can likely be improved by moving height estimation into the CNN instead of predicting it separately using the predicted bounding box. Priors relating vehicle size in the birds-eye view to height can likely be learnt accurately by the network without significant additional training.

It is unclear how much benefit is gained by initialising the network with the pre-trained VGG layers. The birds-eye network reaches a higher accuracy level than the visual front view network after the same training volume but this is not necessarily indicative of VGG benefit. It is likely that the initial VGG weights provide a 'good' initialisation in that the network starts with a set of varied filters at each level.

Further testing could be done to compare the different in performance without the use of transfer learning.

Chapter 7

Combining multiple predictors

7.1 Introduction

Different approaches to 3D object localisation have been presented that achieve varying levels of accuracy. One common approach to improve predictive performance in machine learning tasks is to combine many predictors into an ensemble that can outperform any one of the single predictors alone. Some predictor types lend themselves to this strategy more easily than others. The computational requirements of training neural networks mean that dropout during training is the only feasible method to create a large ensemble of classifiers. Instead of generating many similar predictors a small ensemble can created by combining structurally different predictors operating on different input.

Conventional approaches to combining classifiers using ensembles do not comprise methods that can combine the front view predictor in Chapters 4 & 5 with the birds-eye encoding approach in Chapter 6. A 1:1 mapping does not exist between the representations and there are infinitely many predictions that can occur in both representations since the bounding box adjustments are continuous.

7.2 Naive combination of methods

A method of combining predictions from both the front view and the birds-eye view was developed. The method matches predictions from the two different predictors and combines confidences to create a single unified prediction output that is shown to outperform either approach alone.

Since each prediction is continuous in the 3D cartesian space the predictions from

the two methods are matched using IoU. For each birds-eye prediction the front view prediction with the highest IoU is joined. If there is no matching prediction that has IoU > 0.1 then no prediction is joined. This method may leave some predictions from the front view network or birds-eye network unmatched.

New confidence values are created by taking the product of the two confidences from each joined pair. If predictions from either predictor remain without a match their confidence is updated to 0.05 of their initial confidence. The weight 0.05 is chosen since predictions with confidence below 0.05 are pruned prior to matching in both networks thus this is equivalent to weighting unmatched confidences below all matched confidences.

The bounding boxes are not combined in the matching process. Instead the birds-eye bounding box is used for a matched pair. This is done because the birds-eye predictor has higher average IoU while having higher mAP when testing the predictors independently. Unmatched predictions are included in the output with their bounding boxes unmodified.

7.3 Results

The 3D performance of the predictor is measured using the same methodology as described in Section 4.4. The combination of predictors resulted in a mAP of 49.6% with an IoU threshold of 0.3. This represents a slight improvement over the single birdseye predictor in mAP. Investigation of the results show that the predictions are still dominated by the same issues as the birds-eye predictor, namely height estimation and imperfect recall of occluded labels.

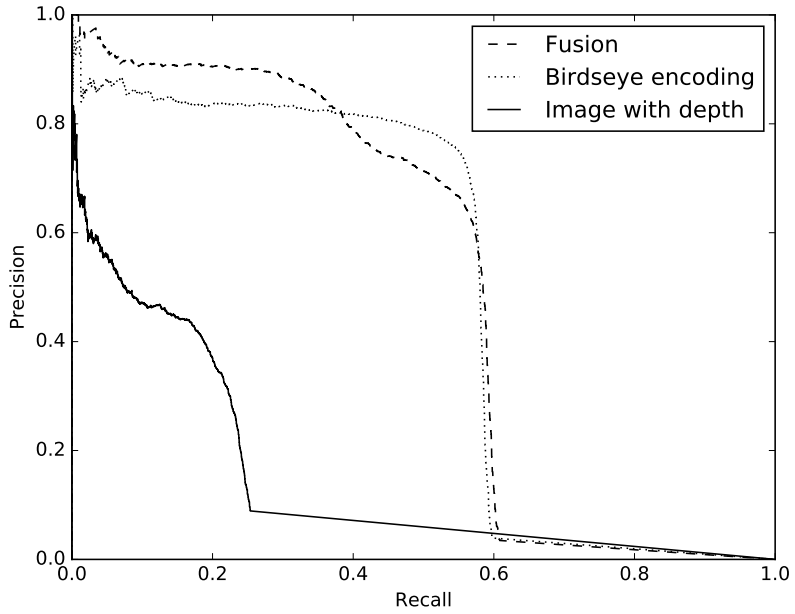


Figure 7.1: Precision vs recall of the predictors explored as the confidence threshold for discarding a prediction is decreased. Measured at 3D IoU threshold of 0.3.

7.4 Discussion

The performance of the fusion predictor is dominated by the birds-eye predictor and similar issues and trends are present in the fusion predictions as in the birds-eye predictions.

The basic process for fusing the birds-eye and front view predictors together limits the benefits of using multiple predictors. The predictors are trained separately without knowledge of the strength or weaknesses of the other predictor. The fusion predictor could be improved if the birds-eye network and front view network could be combined in a manner that allowed end-to-end training and created 3D predictions that utilised the accurate components of each predictor.

The fusion predictor has a significantly higher precision at low recall than the birds-eye predictor. This is likely the result of the different characteristics of the 2 networks used. The front-view predictor has substantially higher recall on objects closer to the camera than in the distance. The birds-eye network does not predict height well when the object is occluded. By using the product of confidences the nearer objects are given higher confidence on average due to the front-view network. Nearby objects also tend to be less occluded which means the less occluded predictions with more accurate height estimation are typically rated with higher

confidence.

Maximum recall of the fusion predictor is slightly higher than the birds-eye predictor which indicates there are at least a small number of samples that are detected by the front-view visual network but not by the birds-eye network. This corroborates findings that the different projections do have at least slightly different strengths and that a fusion of viewpoints will result in the optimal predictor.

Chapter 8

Conclusion

8.1 Findings

Generating synthetic data allows experimentation and training on a large dataset that targets a single type of lighting and weather condition while providing a large range of variable scenes with 100% accurate labelling. The scalability of such a collection pipeline is demonstrated as having potential for removing the high barrier to entry for creating new datasets where physical collection is necessary.

The predictive results demonstrate that combining different convolutional neural networks provides a high accuracy predictor that achieves a mAP of 49.6% on a dataset that contains many fully or partially occluded labels.

Image only techniques accurately place objects within a bounded region but the size of the bounded region in the direction normal to the image plane is difficult to tightly bound using 2D convolutional techniques. Approaches to depth estimation without augmenting the visual data result in differing success but none improve the mAP to a useful level. The prediction of a rotation class from visual data is accurate at the granularity tested but further approaches might perform better by directly regressing a rotation angle or creating a secondary regressor to predict a continuous rotation from a number of discrete classes.

Augmenting 2D image predictions with 3D point cloud data results in substantial accuracy improvement over image only techniques. The bounded region can be tightly constrained in the direction normal to the image plane using the point cloud depth distribution which reduces the largest source of IoU error in the visual only predictor.

The birds-eye encoding of the point cloud performs substantially better than any other individual technique. By flattening the point cloud data to a 2D representation

using multiple metrics the sparsity of the point cloud is reduced while, based on validation accuracy, important information appears to be retained. The inaccuracy in height estimation is driven by occlusion of useful features in the birds-eye view and would likely be improved if height was a trained output of the birds-eye network given the inherent relationship between vehicle base size and height.

The poor performance of the front view predictor, using combined 2D and 3D data, compared to the birds-eye predictor might be explained by a number of factors. The birds-eye representation is chosen because it represents vehicles at a constant scale and with constant appearance. In contrast, the front view represents vehicles with varying scale and appearance. The difference in accuracy suggests that the front view predictor might benefit from a larger dataset and longer training. Intuitively, the front view predictor should require more training as it must learn separate rotation appearances at many scales whereas the birds-eye predictor need only learn a single scale representation with vastly less variance between the rotation appearances.

Combining predictions from the different viewpoints results in the highest mAP of all methods. Both the precision at low recall and maximum recall of the predictor are improved with the fusion technique. The fusion occurs after bounding boxes have been predicted from both viewpoints which limits the ability to intelligently combine the spatial predictions but even by naively preferring the birds-eye boxes when combining predictions, the mAP is increased. The late fusion of the predictions is not optimal as the strengths of each projection are not leveraged in full and the limitations from the preferenced network are still present.

8.2 Future Work

8.2.1 Generalisability of Predictors Presented

The generalisability of CNNs relies on their ability to learn useful intermediate representations that can be effectively transferred to unseen data. For 2D image recognition the learning of useful pixel features is enforced through the layer and connectivity structure. The intuition behind the structure is obvious in the 2D case, each layer extracts higher and higher level features in the local image region which it is connected to. When training a CNN to recognise 3D features like rotation it is unclear how this is learnt and if any useful 3D intermediate features are extracted or if the information is simply brute force learnt as described in Section 2.7.

The techniques presented for inferring depth from 2D predictions demonstrated the weakness of 2D localisation techniques in 3D. Other monocular methods such as [22, 24, 23] separately train intermediate features such as pixel level class and instance segmentation which is not utilised in this investigation. By training intermediate features, semantic information about the scene can be utilised in the final prediction improving predictions with scene context such as depth estimation.

Future research on relative 3D performance of variants in the network structure would further knowledge on if and how 3D intermediate features can be learnt. A network structure that is highly generalisable on 3D object localisation and pose estimation tasks would have broad applications across many industries. Can an ANN learn to extract surfaces or match 3D keypoints as done in handcrafted 3D methods? Perhaps end-to-end training is not optimal for this and the solution is to force handcrafted intermediate representations using an intermediate auxiliary loss?

8.2.2 Fusion Techniques

The basic fusion technique presented in Chapter 7 improves on the individual predictors but it is conjectured that the method presented could be improved with joint end-to-end training and inferencing instead of combining predictions from separately trained networks.

The deep fusion technique described in [27] uses end-to-end training of multiple projections and could likely be extended to a single shot detection type network. While a 1:1 mapping between different projections does not exist, by assuming ranges for ground plane position and vehicle height, a one-to-many mapping can be established from birds-eye to front view which means anchor boxes could be jointly trained to produce merged bounding boxes. Jointly training the networks should increase the accuracy of the final predictions as the most accurate predictive part of each projection will be utilised if the right network structure and training parameters are used.

8.2.3 Validation of Domain Shift

All the investigation conducted focuses on a validation set of synthetic data. Training with synthetic data is only useful if predictive accuracy achieved on a synthetic validation set can be transferred to real world data.

Directly evaluating with a physical dataset such as KITTI using predictors trained on GTA5 data will determine whether the synthetic data is similar enough

to real world data for full training on virtual input to be useful. It is expected that the average precision will decrease since there is at least some level of domain shift between the tasks but ideally the difference is minimal. An alternate method could use the KITTI dataset to train the predictors both from VGG initialisation and from weights pre-trained on GTA5. Even if the predictor is not directly transferable, there may be benefit to pre-training a network on synthetic data before fine tuning on real world data.

8.2.4 Temporal Information

A commonality between many different deep learning tasks is that data is presented to the network in an ordered manner based on the data collection time. Deep learning tasks that are less data intensive than image recognition such as speech recognition have used recurrent neural networks (RNNs) to interpret time series data with great success. RNNs have not been used substantially for image recognition tasks to date but research such as [45] suggests that RNNs can successfully model complex temporal dynamics while still being trained with backpropagation. The dimensionality of image data presents unique challenges when expanding to RNNs across time as multi-GPU processing will be required to effectively train such large networks and a massive dataset will be required.

Bibliography

[1] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[2] (2011) Neural networks - a multilayer perceptron in matlab. Accessed: 2017-10-21. [Online]. Available: http://matlabgeeks.com/tips-tutorials/neural-networks-a-multilayer-perceptron-in-matlab/

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[4] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

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

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *European Conference on Computer Vision*. Springer, 2014, pp. 346–361.

[14] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

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

[16] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *ECCV*, 2016.

[18] D. Scharstein, R. Szeliski, and R. Zabih, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," in *Stereo and Multi-Baseline Vision, 2001.(SMBV 2001). Proceedings. IEEE Workshop on*. IEEE, 2001, pp. 131–140.

[19] Y. Zbontar, Jure and LeCun, "Computing the stereo matching cost with a con-

volutional neural network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1592–1599.

[20] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *Journal of Machine Learning Research*, vol. 17, no. 1-32, p. 2, 2016.

[21] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *IJPRAI*, vol. 7, no. 4, pp. 669–688, 1993.

[22] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2147–2156.

[23] S. Wang, S. Fidler, and R. Urtasun, "Holistic 3d scene understanding from a single geo-tagged image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3964–3972.

[24] Z. Zhang, A. G. Schwing, S. Fidler, and R. Urtasun, "Monocular object instance segmentation and depth ordering with cnns," *CoRR*, vol. abs/1505.03159, 2015. [Online]. Available: http://arxiv.org/abs/1505.03159

[25] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," *arXiv preprint arXiv:1612.00496*, 2016.

[26] S. Song and J. Xiao, "Deep Sliding Shapes for amodal 3D object detection in RGB-D images," in *CVPR*, 2016.

[27] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," *arXiv preprint arXiv:1611.07759*, 2016.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[29] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[30] W. Shang, J. Chiu, and K. Sohn, "Exploring normalization in deep residual networks with concatenated rectified linear units." in *AAAI*, 2017, pp. 1509–1516.

[31] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *CoRR*, vol. abs/1611.03530, 2016. [Online]. Available: http://arxiv.org/abs/1611.03530

[32] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Artificial Intelligence and Statistics*, 2015, pp. 192–204.

[33] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008.

[34] J. Bielicki and R. Sitnik, "A method of 3d object recognition and localization in a cloud of points," *EURASIP Journal on Advances in Signal Processing*, vol. 2013, no. 1, p. 29, 2013.

[35] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[36] Y. Xiang, R. Mottaghi, and S. Savarese, "Beyond pascal: A benchmark for 3d object detection in the wild," in *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*. IEEE, 2014, pp. 75–82.

[37] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[38] Waymo. (2016, dec) Disengagement report. [Online]. Available: https://www.dmv.ca.gov/portal/wcm/connect/946b3502-c959-4e3b-b119-91319c27788f/GoogleAutoWaymo_disengage_report_2016.pdf?MOD=AJPERES

[39] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, "Driving in the matrix: Can virtual worlds replace human-

generated annotations for real world tasks?" in *IEEE International Conference on Robotics and Automation*, 2017, pp. 1–8.

[40] Waymo. (2017, sep) How simulation turns one flashing yellow light into thousands of hours of experience. [Online]. Available: https://medium.com/waymo/simulation-how-one-flashing-yellow-light-turns-into-thousands-of-hours-of-experience-a7a1cb475565

[41] "ScriptHookV." [Online]. Available: http://www.dev-c.com/gtav/scripthookv/

[42] "ScriptHookVDotNet." [Online]. Available: https://github.com/crosire/scripthookvdotnet

[43] "MinHook." [Online]. Available: https://github.com/TsudaKageyu/minhook

[44] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[45] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Appendix A

Network structures

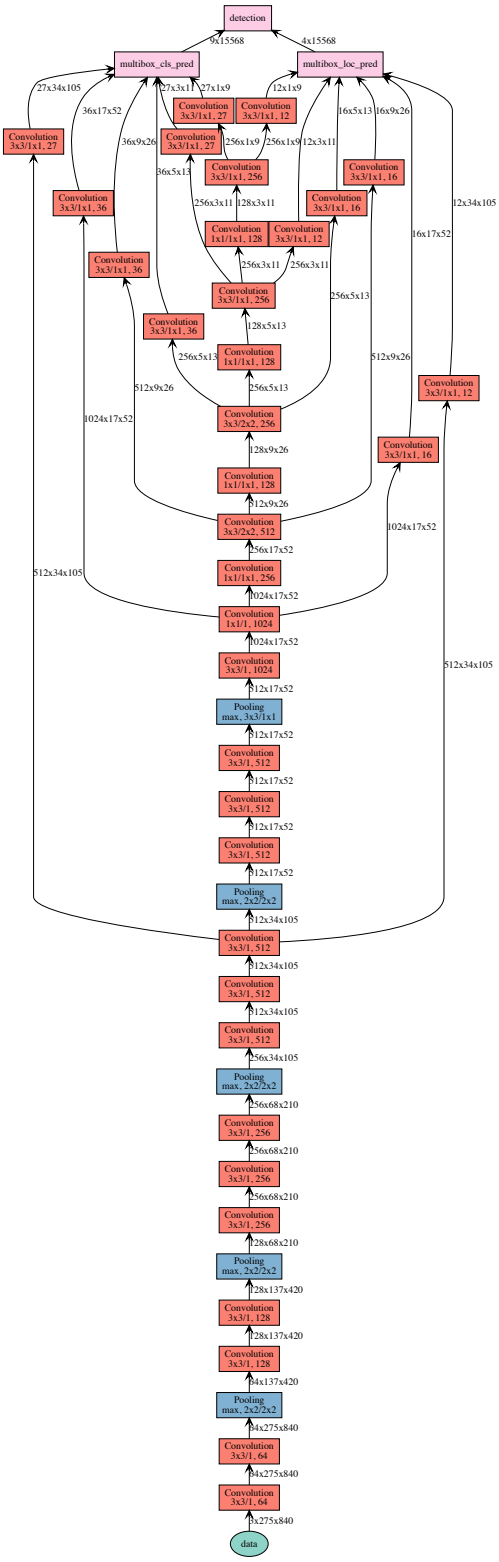


Figure A.1: Structure of network for 2D image inference.

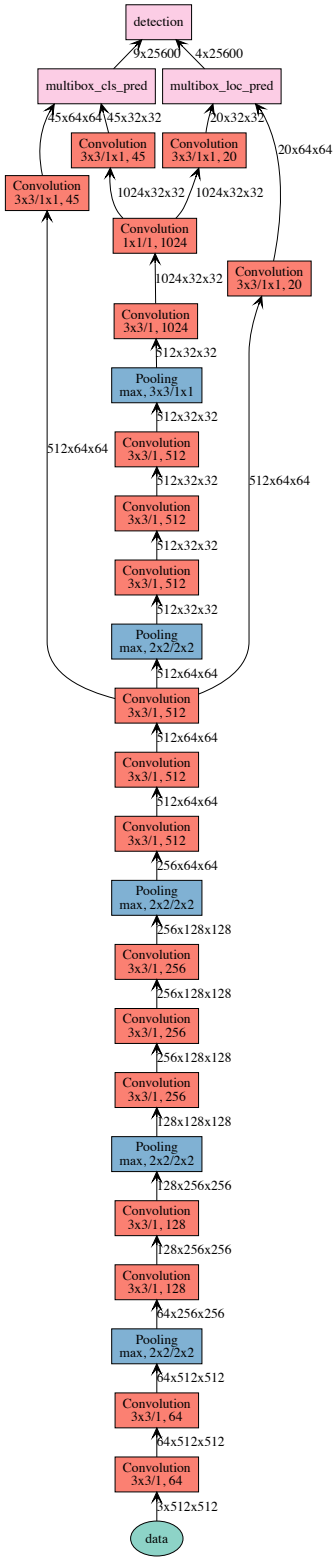


Figure A.2: Structure of network for inference of 2D birds-eye encoding.

Appendix B

Algorithm for fitting 3D bounding box

B.1 Image prediction only

```
def fit_bbox_to_pyramid(e, cls_id):
    """
    Solve 3D  bbox for [x offset, z offset, scale, height]

    Parameters
    e - 8x2 ndarray of corners that define the 6 sided polyhedron in which the
    bounding box must be contained
    cls_id - the rotation class of the prediction
    """
    # Determine centroid of the polyhedron
    centroid = np.mean(e[:, :], axis=0)

    # Determine the planes that define the main face of the polyhedron that
    # bound the allowable area from the edges of the polyhedron
    bottom_plane = get_plane(e[2, :], e[3, :], e[4, :], e[5, :])
    left_plane =   get_plane(e[0, :], e[1, :], e[2, :], e[3, :])
    right_plane =  get_plane(e[4, :], e[5, :], e[6, :], e[7, :])
    top_plane =    get_plane(e[0, :], e[1, :], e[6, :], e[7, :])

    # Create 4 points that represent the base rectangle of the vehicle bounding
    # box
    car_base = np.array([[-1, 0, -VEHICLE_RATIO],
                         [-1, 0,  VEHICLE_RATIO],
                         [1,  0,  VEHICLE_RATIO],
                         [1,  0, -VEHICLE_RATIO]])

    # Rotate the car base (that is initially axis aligned) by the angle of the
    # class prediction
    centroid = centroid / np.linalg.norm(centroid, ord=2)
    centroid_angle = math.atan2(-centroid[0], -centroid[2])
    class_to_angle = np.linspace(0, 2*np.pi, num=8, endpoint=False) + np.pi/8
    view_angle = -(class_to_angle[cls_id] + centroid_angle)
    rot = np.array([[math.cos(view_angle), 0, -math.sin(view_angle)],
                    [0, 1, 0],
                    [math.sin(view_angle), 0, math.cos(view_angle)]])
    car_base = np.dot(rot, car_base.T).T

    def build_A(plane, car_base):
        """
        Determine which corner of the car_base will create the tightest
        constraint given all points must be above the plane. Then create a row
        of the A matrix for an x = [x_position, z_position, vehicle_scale]
```

```
        """
        idx = np.argmin(np.dot(plane[0:3], car_base.T)) # find tightest constraint
        return np.concatenate((
            [plane[(0,2),]],
            [[np.dot(plane[0:3,], car_base[idx,].T)]]
        ), axis=1)

    A = np.concatenate((
        build_A(bottom_plane, car_base),
        build_A(left_plane, car_base),
        build_A(right_plane, car_base)), axis=0)
    b = np.concatenate((
        [[-bottom_plane[3,] - bottom_plane[1,] * GROUND_PLANE_OFFSET]],
        [[-left_plane[3,] - left_plane[1,] * GROUND_PLANE_OFFSET]],
        [[-right_plane[3,] - right_plane[1,] * GROUND_PLANE_OFFSET]]), axis=0)

    # Solve linear system of Ax=b
    x, z, scale = np.dot(np.linalg.inv(A), b).T[0]

    # Translate and scale the car base
    car_base = car_base * scale + [x, GROUND_PLANE_OFFSET, z]

    # Determine which point of the car base will apply the tightest constraint
    # on the vehicle height
    idx = np.argmin(np.dot(bottom_plane[0:3], car_base.T))

    # Solve for height given the point car_base[idx,] will apply the tightest
    # constraint
    height = (-top_plane[3] - np.dot(top_plane[0:3,], car_base[idx,].T) ) / top_plane[1]

    if height < 0:
        height = -height

    return car_base, height
```

B.2 Image prediction with depth augmentation

```
def fit_bbox_to_pyramid_with_depth(e, cls_id, depth):
    """
    Solve 3D  bbox for [x offset, y offset, z offset, scale, height]

    Parameters
    e - 8x2 ndarray of corners that define the 6 sided polyhedron in which the
    bounding box must be contained
    cls_id - the rotation class of the prediction
    depth - metres from camera to closest point of bounding box
    """
    # Determine centroid of the polyhedron
    centroid = np.mean(e, axis=0)
    # Determine the planes that define the main face of the polyhedron that
    # bound the allowable area from the edges of the polyhedron
    bottom_plane = get_plane(e[2, :], e[3, :], e[4, :], e[5, :])
    left_plane =   get_plane(e[0, :], e[1, :], e[2, :], e[3, :])
    right_plane =  get_plane(e[4, :], e[5, :], e[6, :], e[7, :])
    top_plane =    get_plane(e[0, :], e[1, :], e[6, :], e[7, :])

    # Create 4 points that represent the base rectangle of the vehicle bounding
    # box
    car_base = np.array([[-1, 0, -VEHICLE_RATIO],
                         [-1, 0,  VEHICLE_RATIO],
                         [1,  0,  VEHICLE_RATIO],
                         [1,  0, -VEHICLE_RATIO]])

    # Rotate the car base (that is initially axis aligned) by the angle of the
    # class prediction
    centroid = centroid / np.linalg.norm(centroid, ord=2)
    centroid_angle = math.atan2(-centroid[0], -centroid[2])
    class_to_angle = np.linspace(0, 2*np.pi, num=8, endpoint=False) + np.pi/8
    view_angle = -(class_to_angle[cls_id] + centroid_angle)
    rot = np.array([[math.cos(view_angle), 0, -math.sin(view_angle)],
                    [0, 1, 0],
```

```
                    [math.sin(view_angle), 0, math.cos(view_angle)]])
    car_base = np.dot(rot, car_base.T).T

    # Build the front plane that is positioned at depth m from the camera in
    # direction of the centroid with normal vector the position vector of the
    # centroid.
    d = -(np.dot(centroid, centroid*depth))
    front_plane = np.concatenate((centroid, [d]))

    def build_A(plane, car_base):
        """
        Determine which corner of the car_base will create the tightest
        constraint given all points must be above the plane. Then create a row
        of the A matrix for an x = [x_position, z_position, vehicle_scale]
        """
        idx = np.argmin(np.dot(plane[0:3], car_base.T)) # find tightest constraint
        return np.concatenate((
            [plane[0:3,]],
            [[np.dot(plane[0:3,], car_base[idx,].T)]]
        ), axis=1)

    A = np.concatenate((
        build_A(bottom_plane, car_base),
        build_A(left_plane, car_base),
        build_A(right_plane, car_base),
        build_A(front_plane, car_base)), axis=0)

    b = np.concatenate((
        [[-bottom_plane[3,]]],
        [[-left_plane[3,]]],
        [[-right_plane[3,]]],
        [[-front_plane[3,]]]), axis=0)

    # Solve Ax=b
    x, y, z, scale = np.dot(np.linalg.inv(A), b).T[0]

    # Translate and scale the car base
    car_base = car_base * scale + [x, y, z]

    # Determine which point of the car base will apply the tightest constraint
    # on the vehicle height
    idx = np.argmin(np.dot(bottom_plane[0:3], car_base.T))

    # Solve for height given the point car_base[idx,] will apply the tightest
    # constraint
    height = (-top_plane[3] - np.dot(top_plane[0:3,], car_base[idx,].T) ) / top_plane[1]

    return car_base, height
```

B.3 Birdseye Encoding

```
def fit_bbox_to_birdseye(bbox, cls_id, base_y):
    class_to_angle = np.linspace(0, np.pi, num=8, endpoint=False) + np.pi/16

    # Create 4 points that represent the base rectangle of the vehicle bounding
    # box
    car_base = np.array([[-1, 0, -VEHICLE_RATIO],
                         [-1, 0,  VEHICLE_RATIO],
                         [1,  0,  VEHICLE_RATIO],
                         [1,  0, -VEHICLE_RATIO]])

    view_angle = -class_to_angle[cls_id]
    rot = np.array([[math.cos(view_angle), 0, -math.sin(view_angle)],
                    [0, 1, 0],
                    [math.sin(view_angle), 0, math.cos(view_angle)]])
    car_base = np.dot(rot, car_base.T).T

    x_size = np.max(car_base[:,0]) - np.min(car_base[:,0])
    y_size = np.max(car_base[:,2]) - np.min(car_base[:,2])

    scale = min(
```

```
        (bbox[2] - bbox[0]) / x_size,
        (bbox[3] - bbox[1]) / y_size,
    )

    return car_base * scale + [(bbox[2] + bbox[0]) / 2, base_y, (bbox[3] + bbox[1]) / 2]
```