Multimodal Learning for the Joint Understanding of 2D and 3D Data

Nathaniel J. McAleese
Gonville & Caius College

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements of Part III of the Computer Science Tripos

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: nm583@cl.cam.ac.uk

November 5, 2018

Declaration

I Nathaniel J. McAleese of Gonville & Caius College, being a candidate for Part III of the computer science tripos, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 11, 940

Signed:

Date: November 5, 2018

Abstract

Deep learning has improved classification, segmentation and pose estimation results on both 2D and 3D data. However limited work has been done in instances where both 2D and 3D features are available. The recent release of large labelled and aligned 2D/3D datasets means that deep multi-modal learning approaches can now be applied, and this project will aim to determine whether jointly learning on multi-modal 2D/3D data can improve performance and data efficiency when compared to systems that treat each modality individually.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer systems are frequently presented with multiple data sources that describe the same underlying facts. Consider a video of someone speaking. Usually, in this case, the same underlying data source (the person's speech) is recorded in two modalities – as audio and video. In this particular case, substantial success has been had in combining the two modalities to improve performance on speech recognition and source separation tasks [1, 2]

More advanced computer systems, however, often record even more types of data. Autonomous vehicles and medical equipment both often capture the same scene simultaneously in 2D and 3D via depth-sensing cameras or Light Detection and Ranging (LIDAR). Given these data sources, there are numerous *perception* tasks that we would like to perform. It is extremely useful to be able to identify objects and their locations, for example, in order to build autonomous systems that can interact with the 3D world. The output of such a perception system can then be passed to higher level systems, such as a planner, to determine how to achieve goals such as collision avoidance [3].

However, combining data from different modalities is not always straightforward. They often share distinct representations, with distinct or even incomparable statistical properties.

This dissertation explores how two particular modalities, 2D and 3D data in the form of natural images and point clouds, can be combined to improve performance on joint perception tasks. It proposes a novel combination of approaches taken both from recent work on graph neural networks and image recognition that exploits the advantages of both. Whilst the ideas are generally applicable, particular attention is paid to 3D object detection in the context of autonomous driving to provide a concrete example of the utility of the proposed techniques.

1.1 Structure

The rest of this document proceeds as follows:

Chapter 2 describes at a high level the tasks in which we are interested and the tools we will apply to solving them.

Chapter 3 outlines both competing techniques and the details of state of the art approaches that share components with the proposed method.

Chapter 4 outlines the work done to implement both graph and convolutional neural networks, and the considerations required to combine them effectively.

Chapter 5 evaluates the proposed system on the ModelNet40 object classification task [4] and the standardised KITTI [5] 3D object detection datasets.

Chapter 6 presents a summary and directions for future work.

Chapter 2

Background

2.1 Scene Understanding Tasks

Presented with sensor data from a scene, such as an image or a LIDAR scan, there are numerous "scene understanding" tasks that humans may accomplish trivially. For example, we easily recognise the relative location of objects and can produce natural language descriptions of visual scenes.

Whilst such tasks are straightforward for humans, they are non-trivial for computer systems. Indeed, computer vision is sometimes summarised by experts as simply determining "what is where" [6].

In order to emulate human performance at scene understanding, the problem has been broken down into several subtasks that are better specified. A brief taxonomy of these scene understanding tasks includes:

- Object Classification – given an image is known apriori to contain one object in a predetermined set of classes, what object does the image depict? The ImageNet task is a canonical example [7].
- Scene Classification – given an image from a class of scene, such as "Restaurant" from a predetermined set of classes, what type of scene does the image depict? A classic example is Places2 [8].

- Object Localization & Detection – given an image containing zero or more objects from a predetermined set of classes; what objects are depicted and where are they in the scene?
- Semantic Segmentation – assign a class label from a predetermined set to each point or pixel in an image, for example indicating what pixels are road vs pavement.
- Instance Segmentation – assign each point in an image a class label from a predetermined set, and indicate which instance of that class it is; disambiguating, between different objects of the same class.

These tasks are not straightforward; however, a great deal of progress has been made in recent years through the application of convolutional neural networks and other deep learning approaches. We also note here that whilst the guiding motivation of this description of scene classification is in terms of the canonical example of autonomous vehicles; these tasks are extremely useful in other domains. For example, many medical imaging tasks directly apply semantic and instance segmentation tasks to the detection of tumours or lesions.

2.2 A Toy Example

By the end of this thesis, we will be accurately identifying the 3D location of vehicles, cycles and pedestrians. However it is useful to have a running example whilst explaining the core ideas, and for this purpose, we introduce the example of classifying whether a point cloud depicts a depicts a chair or an aeroplane. These two cases are illustrated in Figure 2.1.

2.3 Projective Geometry

Because we have both two and three-dimensional data, it is useful to understand the ways in which the two can be related.

Figure 2.1: For our illustrative example, we consider classifying whether a point cloud depicts a chair or an aeroplane. These two dimensional images are produced via perspective projection, as described in Section 2.3.

Projective geometry is one area of mathematics that can be applied to modelling the imaging process of a camera. In particular, it can tell us how to transform from the three-dimensional space of world coordinates into the two-dimensional space of pixel coordinates. This is how the images in Figure 2.1 were produced.

In order to describe the process, it is useful to define several coordinate frames. Let *world coordinates* be an arbitrary 3D coordinate system in which our camera is positioned. *Camera coordinates* then describes the basis in which the camera is at the origin, x is to the right, y is down and z is into the image plane.

Image coordinates are a 2D coordinate system sharing the x and y direction of camera coordinates, with the origin at the focal centre of the image. *Pixel coordinates* then describe the coordinate system given by the array of pixels, with the origin at the top left. These coordinate systems are illustrated in Figure 2.2.

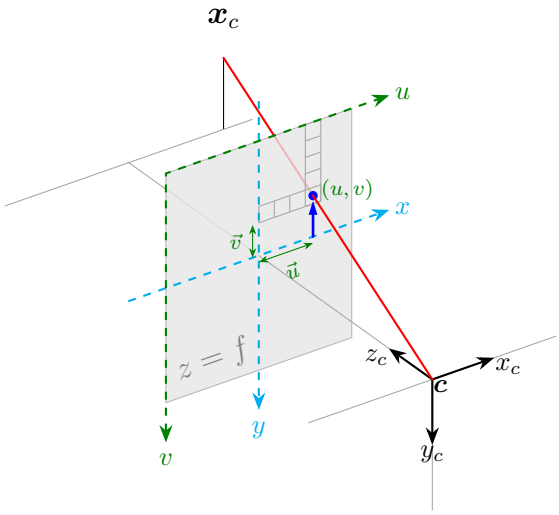

Figure 2.2: The relationship between camera, image and pixel coordinates. $\boldsymbol{x}_c$ is a point in camera coordinates; with $\boldsymbol{c}$, the focal origin, illustrating the theoretical camera position. The imaging plane $Z = f$ is illustrated, at the focal distance f from the focal origin. $(\bar{u}, \bar{v})$ then gives the position of the projection of x_c in image coordinates, and (u, v) the position of the projection in pixel coordinates. Diagram modified from [9].

2.3.1 Homogenous Coordinates

In order to succinctly describe perspective projection, it is very useful to introduce *homogenous coordinates*. The transformations to and from homogenous coordinates can then be described as

$$\boldsymbol{x} = (x, y, z)^\top \implies \tilde{\boldsymbol{x}} = (x, y, z, 1)^\top \tag{2.1}$$

$$\tilde{\boldsymbol{x}} = (x, y, z, w)^\top \implies \boldsymbol{x} = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)^\top \tag{2.2}$$

An analogous definition is omitted for two-dimensional coordinates, in which we introduce a third component. We can then describe the transformation from world coordinates to pixel coordinates as a sequence of matrix multiplies (i.e. linear transformation) in homogeneous coordinates. In particular, the *extrinsic* parameters of a camera are defined to be the translation and rotation that map from world coordinates to camera coordinates. If $\boldsymbol{c}$ gives the

location of the camera, and R the rotation that maps into camera coordinates we note that this may be represented as a single matrix multiply.

$$M_{\text{extrinsic}} = \left[\begin{array}{c|c} R & -R\boldsymbol{c} \\ \hline \mathbf{0}_3^\top & 1 \end{array}\right] \tag{2.3}$$

$$\tilde{\boldsymbol{x}}_c = M_{\text{extrinsic}}\ \tilde{\boldsymbol{x}}_w \tag{2.4}$$

Perspective projection is also a linear operation in homogeneous coordinates, and can be represented by disregarding the last component, or equivalently the matrix:

$$M_{\text{proj}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.5}$$

We may then use the intrinsic parameters of the camera K to translate from image coordinates to pixel coordinates. A linear transformation may account for non-square or skewed pixels, amongst other anomalies. However, if there are nonlinearities in the imaging process such as pin-cushion or barrel distortion, they may be accounted for at this step with an appropriate nonlinear transformation. Alternatively, an "undistort" transformation may be applied to the original image, which alters it such that the imaging process is linear. The latter approach is standard and removes the need to consider such distortions from this work. For a more detailed reference, the reader is referred to [10].

This results in a final transformation from world coordinates to pixel coordinates:

$$\tilde{\boldsymbol{x}}_{\text{pixel}} = K M_{\text{proj}} M_{\text{extrinsic}}\ \tilde{\boldsymbol{x}}_w \tag{2.6}$$

2.4 Machine Learning & Optimization

The dominant paradigm in scene understanding currently centres around *supervised learning*. In this scheme, a set of labelled training data is provided

with ground truth annotations for the task of interest.

For example, in our toy example, we would be presented with a set point clouds annotated with a label of either chair or aeroplane.

These data may also be more complex – for the 3D object detection task that we will eventually address, the training data consists of LIDAR scans taken by a vehicle mounted scanner; images captured by a front-mounted camera, and a set of objects that occur in each scene, including their location, 3D pose and class (such as "Car", or "Pedestrian").

We then define a loss function $\mathcal{L}$ which tells us about the quality of a particular model output, given the input and ground truth. These are usually motivated by probability theory, but can also include heuristic or task-specific metrics.

For example, given real-valued vectors for our input $\boldsymbol{x}$ and target $\boldsymbol{y}$, we might seek to minimise the squared Euclidian distance between our model's prediction and the ground truth. Supposing our model is given by f, this would result in a loss function:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = \|f(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 \tag{2.7}$$

And would thus penalize predictions that were far away from the ground truth.

This is an example of a *regression* loss and is applied to cases in which we seek to predict a real-valued vector from our input data. In the classification case, we instead seek to predict a categorical variable from our input data, such as whether it depicts a chair or an aeroplane. In both the two-class case and the k distinct class categorical case, we seek to minimise the expected surprise[1] of the model under the empirical distribution. This is equivalent to minimising the Kullback-Leibler divergence between the predicted distribution and the empirical distribution, or similarly the cross-entropy[2]. All result in equivalent

[1] In the information theoretic sense; wherein the surprisal of an event is precisely the logarithm of the probability of it occurring.

[2] The cross entropy and the KL divergence differ by a constant additive term, thus

loss functions. Assuming $y_i = 1$ if x is in class i, and zero otherwise, we have:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = -\sum_i y_i \log(f(\boldsymbol{x})_i) \tag{2.8}$$

Assuming that our model output is normalised to be non-negative and sum to 1, so we may treat it as a probability distribution.

In almost every case, our model will be parameterised by some set of parameters $\boldsymbol{\theta}$, and it is thus useful to denote the loss for a given set of parameters. Assuming the output of our model is given by $f(\boldsymbol{x}, \boldsymbol{\theta})$, we have:

$$\mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{L}(f(\boldsymbol{x}, \boldsymbol{\theta}),\ \boldsymbol{y}) \tag{2.9}$$

For any particular modelling task there is always some ground truth distribution of examples and their annotations $\mathcal{R}$. We would ideally like to find the ideal paramaters $\boldsymbol{\theta}^\star$ that minimises the loss for this true distribution:

$$\boldsymbol{\theta}^\star = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{R}} \mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}) \tag{2.10}$$

Unfortunately we very rarely have access to the entire true distribution $\mathcal{R}$, and instead have only a finite training set sampled from it, $\mathcal{X}$. The goal of supervised learning is then to approximate from this finite training set a set of parameters that will generalise well to the true distribution. Given that we have such a training set it is useful to denote the total loss for a given set of parameters as

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{X}} \mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}) \tag{2.11}$$

Minimising this value then turns learning into an optimisation task. If $\mathcal{L}$ is differentiable with respect to $\boldsymbol{\theta}$ then we may use *gradient descent*. In this approach, we initialise $\boldsymbol{\theta}$ randomly, and then repeatedly take small steps in the direction of steepest local descent:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \nabla_{\boldsymbol{\theta}_i} \mathcal{L}(\boldsymbol{\theta}_i) \tag{2.12}$$

making them equivalent for the purposes of optimization.

Mini-batch stochastic gradient descent (SGD) is often used in practice, where the gradient is computed not with respect to the entire training set, but instead with respect to a small set of randomly chosen examples from it.

This optimisation procedure and its variants have been applied with great success to the training of differentiable models on large datasets [11, 12, 13].

We should note, however, that finding the parameters that minimise $\mathcal{L}(\boldsymbol{\theta})$ for the training set is not guaranteed to find us a set of parameters that generalises well to $\mathcal{R}$. This phenomenon is known as overfitting and is another motivator for SGD. The stochasticity introduced by randomly selecting mini-batches acts as a *regulariser*, which prevents the optimisation finding a minimum which only performs well on the training data.

Because SGD is an iterative procedure, we must be able to decide when to stop (i.e. what $\boldsymbol{\theta}_n$ to select). Cross-validation and early stopping are two popular approaches [11]. Both rely on the core idea of holding out some portion of the training set as unseen *validation* data. This allows us to approximate how well the model will generalise to the true distribution by computing the validation loss. Early stopping is the approach that we use in the training of our models. We train the model with SGD and, periodically, we test the loss on this validation set, and when the model has converged we select the set of parameters that achieved the best validation loss throughout training.

This is an example of hyperparameter optimisation. Hyperparameters are the components of a learning system that are not directly trained by the core learning algorithm. The term is often interpreted very liberally and can include the structure of the model or any other design choice not derived from the data.

2.5 Learning from Vectors (MLPs)

Our framework for supervised learning has, thus far, said nothing about the form of our model, aside from the requirement imposed by SGD that it be differentiable with respect to $\boldsymbol{\theta}$.

Numerous models are available, but one of the canonical examples is the *multi-layer perceptron*.

An MLP with n hidden layers can be defined as follows:

$$f_i(\boldsymbol{x}) = \sigma(M_i\boldsymbol{x} + \boldsymbol{b}_i) \tag{2.13}$$

$$f(\boldsymbol{x}) = f_0(f_1(...f_n(\boldsymbol{x})...)) \tag{2.14}$$

$$\boldsymbol{\theta} = \{M_0, M_1, ..., \boldsymbol{b}_0, \boldsymbol{b}_1, ...\} \tag{2.15}$$

Where σ is an elementwise nonlinearity, such as ReLU, LeakyReLU, tanh or similar [14], and $\boldsymbol{b}$ is referred to as the *bias*.

$$\text{ReLU}(\boldsymbol{x})_i = x_i \text{ if } x_i > 0 \text{ else } 0 \tag{2.16}$$

$$\text{LeakyReLU}_\alpha(\boldsymbol{x})_i = x_i \text{ if } x_i > 0 \text{ else } \alpha x_i \tag{2.17}$$

In the case of an MLP with one hidden layer, we can see that the shape of M_0 and M_1 are partially constrained by the shape of $\boldsymbol{x}$ and $\boldsymbol{y}$, but that we may select the number of *hidden units* h arbitrarily. In particular, if $\boldsymbol{x}$ and $\boldsymbol{y}$ are of length a, and b respectively, the shape of M_0 and M_1 will be (a, h) and (h, b) respectively. Similarly, we can completely specify an MLP, if the input and output dimensions are known, by specifying the number of hidden units in each layer.

An important result is that as we tend h to infinity, and MLP can approximate an arbitrary smooth function arbitrarily well [11]. Thus they are referred to as *universal function approximators*, despite the fact that h is always finite.

We can see that our composite f is differentiable with respect to $\boldsymbol{\theta}$, and is

thus trainable via SGD. One layer of an MLP (f_i in the above equations) is often referred to as a dense layer.

2.6 Learning from Images (CNNs)

A convolutional neural network, or CNN, is another form of differentiable function. It exploits redundancy in the type of ground truth distributions that we care about by encoding invariances into its structure. For example, many aspects of visual understanding are translation invariant. The exact location of our chair or aeroplane does not alter its label. By encoding translation invariance into the structure of our model, we encode a bias that will hopefully reduce the generalisation error for learning tasks where this assumption is true.

A CNN "bakes in" this assumption of translation invariance by using banks of small convolutional filters that are applied sequentially in layers.

If we consider an image as a rank-3 object with two spatial dimensions (height and width) and a number of colour channels (three, in the case of RGB images), then given an input image I and a "kernel" K, discrete convolution is defined as:

$$(I \star K)[i,j] = \sum_m \sum_n \sum_c I[i-m, j-n, c] K[m,n,c] \tag{2.18}$$

A layer in a CNN is defined as multiple such convolutions, with a "filter bank", or collection of K's.

$$f(I)[i,j,c] = (I \star K_c)[i,j] \tag{2.19}$$

$$g(I)[i,j,c] = f[i,j,c] + b_c \tag{2.20}$$

$$h(I)[i,j,c] = \sigma(g(I)[i,j,c]) \tag{2.21}$$

Where $\boldsymbol{b}$ is again called the bias and σ is an elementwise nonlinearity. These operations are differentiable with respect to the elements of K and b, which

are used as the learnable parameters $\boldsymbol{\theta}$ of the layer. Note that they are deeply analagous to the definition of an MLP - indeed a convolution where K has shape $(1, 1, c)$ is exactly equivalent to applying an MLP at every spatial location of the input independently.

Because the application of a layer transforms a rank-3 object into another rank-3 object, these functions may be composed. It has been empirically demonstrated that "deep" networks consisting of the composition of many such layers perform very well on a number of tasks. The intuition suggested for this is by analogy with the visual system of the brain, in which low-level features such as corners or edges are assembled hierarchically into higher level concepts such as a nose or eye [15].

This transformation between rank-3 objects does not offer a direct way to solve a classification task. Suppose we are seeking to solve our "Chair or Aeroplane" classification problem with the rendered images presented in Figure 2.1. The application of several convolutional layers will give us a set of features for each spatial (eg pixel) location in the image, not a length two vector of Chair/Aeroplane probabilities.

In early CNNs a flattening approach was used to solve this problem, but this has been mostly superseded with a "fully convolutional" approach [16, 17, 18]. In this approach "Global Pooling" is used, where a maximum or average is taken over the spatial dimensions of a feature bank -

$$\text{GlobalMaxPool}(I)[c] = \max_{(i,j)} I[i, j, c] \tag{2.22}$$

$$\text{GlobalMeanPool}(I)[c] = \operatorname*{mean}_{(i,j)} I[i, j, c] \tag{2.23}$$

This removes the spatial component of our function, and so may be applied to obtain a classification output.

A similar operation can also be applied to reduce the size of an input filter bank. This is simply referred to as pooling. The most frequently used form

is 2×2 max pooling:

$$\text{MaxPool}(I)[i,j,c] = \max_{(i,j)\in(\{2i,2i+1\}\times\{2j,2j+1\})} I[i,j,c] \tag{2.24}$$

Where $\times$ denotes cartesian product. This reduces the spatial resolution by half.

It is straightforward to note that a CNN composed of such layers and using only global pooling is, ignoring boundary effects at the edge of the image, fully translation invariant.

Having arrived at a vector representation of the image, we may now apply normalisation to ensure that the model outputs are non-negative and sum to one. By far the most popular approach is the Softmax function, defined as:

$$\text{Softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.25}$$

Thus far we have achieved one recipe to solve our toy problem. Given a point cloud in one of our two classes, we may use projective geometry to compute a two-dimensional image of the object, and then pass this image through several convolutional layers. Global pooling and softmax can be applied to convert these feature maps to a vector of two elements, these may then be converted to a scalar value by using the ground truth labels and a categorical loss. We can then minimise this loss via SGD on the training set, and use a validation set to stop early and prevent overfitting.

2.7 Learning from Point Clouds

An alternative approach would be to operate on the points directly.

Whereas we considered an image as a rank-3 object; a point cloud in nD can be represented as a rank-2 object. Given a point cloud P, of we might consider the jth point of the ith feature; $P[i,j]$. We will construct a model that operates on this representation directly.

Just as in the case of images; there are invariances that we may want to encode into the structure of our model in order to improve generalisation. In particular; we almost always want to be invariant to the ordering of the points in the cloud. In order to describe the structure of the model we will use, it is useful to introduce several core concepts.

A symmetric function of n variables is one that is equal under all permutations of its inputs. We may define a symmetric function on n vectors as a function such that

$$\forall \text{ permutations } \pi.\ f(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) = f(\boldsymbol{x}_{\pi(1)}, ..., \boldsymbol{x}_{\pi(n)}) \tag{2.26}$$

Given a commutative and associative operation $\circ$ on two vectors, we can note that aggregating a sequence of vectors with $\circ$ is a symmetric function:

$$\forall \text{ permutations } \pi.\ \boldsymbol{x}_1 \circ \boldsymbol{x}_2 \circ ... \circ \boldsymbol{x}_n = \boldsymbol{x}_{\pi(1)} \circ \boldsymbol{x}_{\pi(2)} \circ ... \circ \boldsymbol{x}_{\pi(n)} \tag{2.27}$$

Both + and elmentwise maximum are often used as the operator $\circ$. Henceforth this operation, used to combine the transformed features of the set in an order-invariant way, shall be referred to as the aggregation function.

Having noted this, we can then define a learned order-invariant function on a point set by a symmetric function of a transformation of that set. For example; using addition as our commutative and associative operator we might define

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) = g_{\boldsymbol{\theta}}\left(\sum_i h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right) \tag{2.28}$$

Where g and h are MLPs.

We will denote such learned set aggregation functions as $f_{\boldsymbol{\theta}}(\{\boldsymbol{x}_i\})$.

These ideas of learnable set aggregation were first applied to point cloud classification in the development of PointNet [19]. This work also proved that using max aggregation, such a construction can approximate any smooth set function given arbitrarily large dimensionality of $h_{\boldsymbol{\theta}}$, and further that this approach is effective for the classification and segmentation of point clouds.

However, despite these theoretical guarantees, implementations of the approach have shown some limitations. In particular, the approximation theorem says nothing about how easy such a function will be to learn, and intuitively we can see why it might be non-trivial – there is no direct notion of locality, as all points are considered simultaneously. Most classical point-processing techniques strongly exploit the notion of locality, or neighbourhoods, to directly capture things like corners.

2.8 Learning from Graphs

Figure 2.3: A 5 nearest neighbour graph of one of our point clouds. This could be used, along with a graph classification technique, to determine what what sort of object it was.

Graph processing techniques offer ways to counter the limitations of the PointNet approach by introducing further bias into the model. We can note that several recent techniques, including Graph Attention Networks, Relational Networks and Dynamic Graph Convolutional Networks all fall into the

same general paradigm [20, 21, 22]. Indeed, they simply extend the set function learning techniques described in the previous section by incorporating the local neighbourhoods $\mathcal{N}_i$ defined by a graph.

More specifically, given a graph $G = (\mathcal{V}, \mathcal{E})$ of nodes and edges, we can define the neighbourhood of node $\boldsymbol{x}_i$ as the set $\mathcal{N}_i = \{\boldsymbol{x}_j \mid (i,j) \in \mathcal{E}\}$

We can then learn a description of node $\boldsymbol{x}_i$ as some smooth function of it and it's neighbourhood, before aggregating the descriptions of all nodes in the graph. Producing a learned function of a set and an element is straightforward given our framework for set aggregation – supposing we wish to learn a function of a set and a vector, we may simply pick an aggregation function such as addition or elementwise max (intentionally left here as $\boxplus$) and define

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}', \{\boldsymbol{x}_i\}) = g_{\boldsymbol{\theta}}\left(\bigboxplus_i h_{\boldsymbol{\theta}}(\boldsymbol{x}', \boldsymbol{x}_i)\right) \tag{2.29}$$

Much existing work is an instantiation of this general description. For example, in relational networks, the graph is taken to be fully connected, and summation is used as the aggregation operator. We can therefore consider the node features as being extracted as:

$$f_{\boldsymbol{\theta}}(x_i, \mathcal{N}_i) = \sum_{j \in \mathcal{N}_i} h_{\boldsymbol{\theta}}(x_i, x_j) \tag{2.30}$$

In Graph Attention Networks, a new aggregation technique is employed, "self attention". Here we have

$$w_i = a_{\boldsymbol{\theta}}(\boldsymbol{x}', \boldsymbol{x}_i) \tag{2.31}$$

Where $a_{\boldsymbol{\theta}}$ is a scalar valued function of a vector.

$$\alpha_i = \text{Softmax}(\boldsymbol{w})_i \tag{2.32}$$

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}', \{\boldsymbol{x}_i\}) = g_{\boldsymbol{\theta}}\left(\sum_i \alpha_i \; h_{\boldsymbol{\theta}}(\boldsymbol{x}', \boldsymbol{x}_i)\right) \tag{2.33}$$

By contrast, in dynamic graph convolutional networks (work done in parallel with this thesis), a similar framework is proposed, and the max operation

used for neighbour aggregation.

These approaches can then be layered. At each layer ℓ, the representation of node $\boldsymbol{x}_i^{(\ell)}$ is computed as:

$$\mathcal{N}_i^{(\ell)} = \{\boldsymbol{x}_j^{(\ell)} \mid (i,j) \in \mathcal{E}\} \tag{2.34}$$

$$\boldsymbol{x}_i^{(\ell)} = f_{\boldsymbol{\theta}}^{(\ell)}\left(\boldsymbol{x}_i^{(\ell-1)},\ \mathcal{N}_i^{(\ell-1)}\right) \tag{2.35}$$

As we move to higher layers, the receptive field of each node's representation grows larger. In the first layer, it may only have been influenced by its immediate neighbours. In the second, it's neighbour's neighbours and so on.

This gives us a third approach to our toy problem of Aeroplane/Chair classification. Given a point cloud, we may compute a nearest neighbours graph of the points. Such a graph is illustrated in Figure 2.3. We can then compute successively higher level representations using a graph-based approach, before finally aggregating these vectors with learned set aggregation. Again, because such a model is differentiable with respect to its parameters, it may be trained by SGD.

2.9 Multimodal Learning

Thus far, the approaches described have operated on only a single modality. *Multi-modal* approaches combine the information from multiple modalities with the aim of improving the performance of the prediction task. This might be in terms of the overall accuracy of the predictive model, the training or inference speed, the interpretability of the results, or some combination of all three.

Traditional approaches consider either *early fusion* in which the modalities are combined in their raw form (for example, through a concatenation scheme) or *late fusion* in which two independent models are trained and their final outputs combined.

Late fusion, or ensembling, is the most flexible of these approaches. Almost by definition, if we can produce a model that operates on one modality we will be able to average its results with a model that operates on any other. However, this is also the approach which most seriously limits information sharing between the two modalities.

More recent approaches, however, have developed *cross-modal learning* [23], in which a network is trained to incorporate cross-modal connections at multiple levels in a deep neural network. This approach has been successfully applied to audio-visual speech recognition [1], and improves image classification performance on sparse datasets by decomposing images into their various perceptual channels [23].

One particular advantage of multi-modal learning is in the benefits provided by *transfer learning*. The term applies broadly to the idea of using data available for one task to improve performance on another, but there is one approach in particular that has become increasingly popular. In this scheme, a network is first trained on a task for which data is abundant (such as object classification, due to ImageNet[7]). These weights are then used as the initialisation for training on the target task, often leading to greatly improved performance [24, 25, 11]

However, in order for cross-modal learning to work effectively, we must overcome the differences between the modalities. The effectiveness of each model we have described so far is largely due to the encoded biases that we have highlighted. However important assumptions about one modality do not apply to the other. Our point cloud methods are carefully considered to ensure that they are invariant to ordering – yet such an assumption is not appropriate for the pixels of an image!

The next section of this thesis will expand upon the basic ideas presented here, detailing how they are applied in the state of the art, and what competing methods exist.

Chapter 3

Related Work

3.1 Classical Point Cloud Processing

A great deal of work has been done on point cloud segmentation and classification that does not use large differentiable models or "deep" learning. It is worth noting, however, that whilst these approaches do have some advantages, the accuracy leaderboards of the most challenging point cloud tasks KITTI, ModelNet40, Semantic3D, do not contain any classical approaches within the top 10%. It could be argued that this is because there has been insufficient time invested in such techniques, but it may also be because end-to-end ML approaches are very powerful when large amounts of data are available.

The dominant classical approach proceeds in two stages. First local or global features are extracted, and then a learning approach is applied to solve the target classification or segmentation problem.

By far the most popular features are functions of the local eigenvalues and eigenvectors. If we consider a small neighbourhood $\mathcal{N}_i$, then we can compute these values locally at manageable computational cost. The last eigenvalue approximates the surface normal if the points were sampled from a plane, and the eigenvalue ratios can be informative of how planar the local neigh-

bourhood is.

Numerous other feature extraction techniques have been developed. They are often developed based on physical intuitions, using wave models [26] or diffusion based approaches [27]. These are characterised as "intrinsic", as opposed to "extrinsic" approaches that more closely resemble the histogram based feature extraction techniques we are familiar with from computer vision [28]. As with classical visual feature extractors, there is a large design space with many reasonable proposals. Popular methods include [29, 30, 31, 32].

The wide variety of reasonable features available makes it a non-trivial task to determine what approach will work best for a given task without substantial hand tuning. Removing this source of complexity is one of the goals of the end-to-end approach.

After feature extraction, there are also a variety of applicable learning algorithms. Random forests have been used, alongside scale pyramid feature extraction for the segmentation of very large static outdoor LIDAR scans of millions of points [33].

Conditional random fields (CRFs) are also particularly applicable because they operate directly on graphs. Combined with substantial feature engineering, they have been successfully applied to road classification from aerial LIDAR scans [34]

Ad-hoc methods driven by clustering and ground estimation using RANSAC variants have also achieved some success [35].

These approaches share the same broad set of advantages – they are often less computationally intensive than neural networks at training time. However, they are just as expensive, if not more so, at inference time, and do not achieve the same final level of accuracy [5, 4, 36].

3.2 Classical Fusion Approaches

The classical approach of extracting manually designed features to feed into a learning system has also been applied the multimodal fusion problem. Colour features from back-projected images have been included in random forest-based point cloud segmentation systems [37]. Work such as [38, 39] uses classical point cloud feature extraction along with a superpixel segmentation of the image to establish a joint graph of 2D and 3D features. [38] goes to great lengths to ensure that predictions can be made on the spatial union of the two modalities, whereas [39] focuses on allowing for inference when they are captured from different locations and at different sampling rates. Both use CRFs to make segmentation predictions on the final multimodal graph. Markov random fields and random forests have also been combined to extract dense 3D information from video whilst simultaneously segmenting [40].

Similarly to the raw point cloud processing case, these solutions are thoughtful and well designed. They provide valuable insights that can motivate the direction of future work. However, they do not achieve comparable performance to large differentiable models in any of the cases where the two approaches have been directly compared [5, 41, 42].

3.3 Bounding Box Prediction

Because several of the following techniques have focused on bounding box prediction, it is useful to introduce the two competing state of the art approaches. Proposal-based methods such as faster-RCNN use *region proposals* and *region of interest pooling* (ROI pooling) in order to achieve object classification and localisation [13]. This can be contrasted with single shot detectors, which were pioneered by YOLO [12].

In a proposal-based method, an initial pass through a CNN is used to indicate candidate regions of an image that may contain an object. The features of these regions are then combined and fed into a secondary classifier that

determines whether or not a region contains an object (objectness), what type of object it contains and a more precise bounding box. Varying and large numbers of regions are extracted per image, which complicates parallelisation and increases processing time.

In the single shot design, we instead predict all objectness scores, classifications and bounding boxes simultaneously with one pass through the network. This is done by predicting the objectness, class and bounding boxes for predetermined anchors in the image. This can be around 10 to 100 times faster [12, 43, 44], but results in a difficult class imbalance problem – whereas the secondary classifier in a proposal method may be fed regions individually that in which objects are present at some known up-sampling rate, objectness prediction in a single shot detector must happen for an entire image at a time. Most anchors in the image are not associated with objects, and so "no object" is the heavily dominant class. Despite this core challenge, single shot designs have achieved increasingly compelling performance. The focal loss [44] is a recent approach to tackling this class imbalance problem. It is introduced fully in Section 4.8.

3.4 Volumetric CNNs

Volumetric CNNs are the most straightforward adaptation of CNNs to 3D. Given a point cloud, it is first voxelised at some resolution into a fourth rank object with three spatial dimensions and an arbitrary number of features per spatial location. Convolutions can be extended into 3 dimensions, "sliding" over each of the spatial dimensions. These models can be very accurate [45], but they have one large drawback – resource usage. Every quadratic property of a CNN is cubic in the volumetric case, resulting in more parameters, more memory usage and more compute load. This can be prohibitive. Several approaches have been taken to mitigating this issue, including sparse voting schemes [46] and learning how to query the space [47].

3.5 Multiview Models

Another popular approach to handling 3D point clouds is to fuse information from multiple 2D views of the same scene. This can leverage highly tuned image detection architectures and is an extension to the rendering approach described in the introduction. Multi-view CNNs are the canonical example [48] and apply to object detection where a full circle of captured views are available.

One difficult component of this approach is determining from what point and direction to capture the image. RotationNet [49] solves this problem by jointly estimating object pose and class from captured images. This can be used to predict informative views after an initial random image is captured.

Whilst these approaches are effective at leveraging transfer learning from large object classification datasets, several approaches make unusual choices in terms of how data are fused. For example [48] arbitrarily layers features captured from multiple cameras into a single multichannel "image" that is then processed. This is not supported by any natural data invariance that we might expect.

3.6 Other PointNet Extensions

The current state of the art for 3D object detection from LIDAR are works derived from PointNet. FrustumPointNet approaches the KITTI task by using an off-the-shelf object detection and localisation framework, Faster-RCNN [13], to produce two-dimensional bounding box predictions. This defines a frustum (square cone) in three dimensions, and LIDAR points from that frustum are then sampled and fed into PointNet based networks that predict object position and orientation. This scheme is multi-modal, in that it exploits both the 2D and 3D data in order to derive the result, but there is no shared learning and the model has not successfully been end-to-end trained. The scheme also requires numerous evaluations of the PointNet model (one

for each proposed box), and is consequentially quite slow, running at around five frames per second on a 2017 workstation GPU.

VoxelNet is another PointNet extension that achieves good performance on 3D object detection. It divides space into comparatively large voxels – the local structure of these voxels is then captured by a PointNet, before being fed into several dense volumetric convolutional layers that generate region proposals [45]. This results in accurate predictions, but the dense convolutions use a great deal of GPU memory.

3.7 Multimodal Deep Learning

Approaches to exploiting multimodal data using end-to-end trainable models must determine how to combine the disjoint representations of the modalities. This can be done via *late fusion* in which models compress the inputs into a fixed size vector representation before combining these representations through a merging operation and an MLP. This approach has been effective on lip reading tasks [41] but limits the amount of information that can be shared between the modalities. Fusion can also occur earlier in the model, as in "Looking to listen at the cocktail party", in which high-quality speech separation is achieved by exploiting audiovisual data. In this instance, a fixed size representation is computed for each timestep of each modality before these fixed length representations are merged and fed into a deep sequence processing model [2].

As opposed to creating a single point of communication, "cross-modal" approaches seek to combine the modalities at multiple different layers throughout the network. Intuitively this allows each modality to influence the interpretation of the other at multiple different levels of the increasingly generall feature hierarchy. These approaches have also been shown to improve performance in the case where training data is limited [23]. Cross-modal ideas have been applied to the 1D/2D fusion case for audiovisual classification [1], but the cross-modal idea does not provide a general recipe for how to fuse

arbitrary data – instead, it illustrates how fusion throughout the network can be beneficial for performance.

A similar idea has been applied to 2D/3D fusion for object detection in road scenes, using renderings of KITTI data along with captured images [50, 51]. These methods operate by using region-of-interest (ROI) pooling to combine the information from each format after a proposal has been generated. In ROI pooling, a bounding box predicted in one modality is projected into the other, and the features in that region then combined using, for example, summation. As a result, the proposal generation phase itself is unimodal, and this approach suffers from the same computational overhead of all RPN based methods

Chapter 4

Design and Implementation

The complete system has a number of components and relies heavily on a number of core underlying technologies. This section describes the implementation of and relationships between the various components. Figure 4.1 gives an illustrative overview of how the system works, to give context to the various components.

4.1 Automatic Differentiation

Much of the recent success of large differentiable models has been driven by readily available automatic differentiation libraries. Most follow the approach pioneered by Theano [52], in which a directed acyclic computation graph (DAG) is specified in a high-level language such as Python, and then compiled to a DAG of fast kernels written in low-level languages. Tensorflow is now the most popular of these tools and provides all of the basic capabilities needed for this project [53]. It uses reverse mode automatic differentiation to compute the derivatives required for the implementation of SGD and provides GPU kernels for fast evaluation.

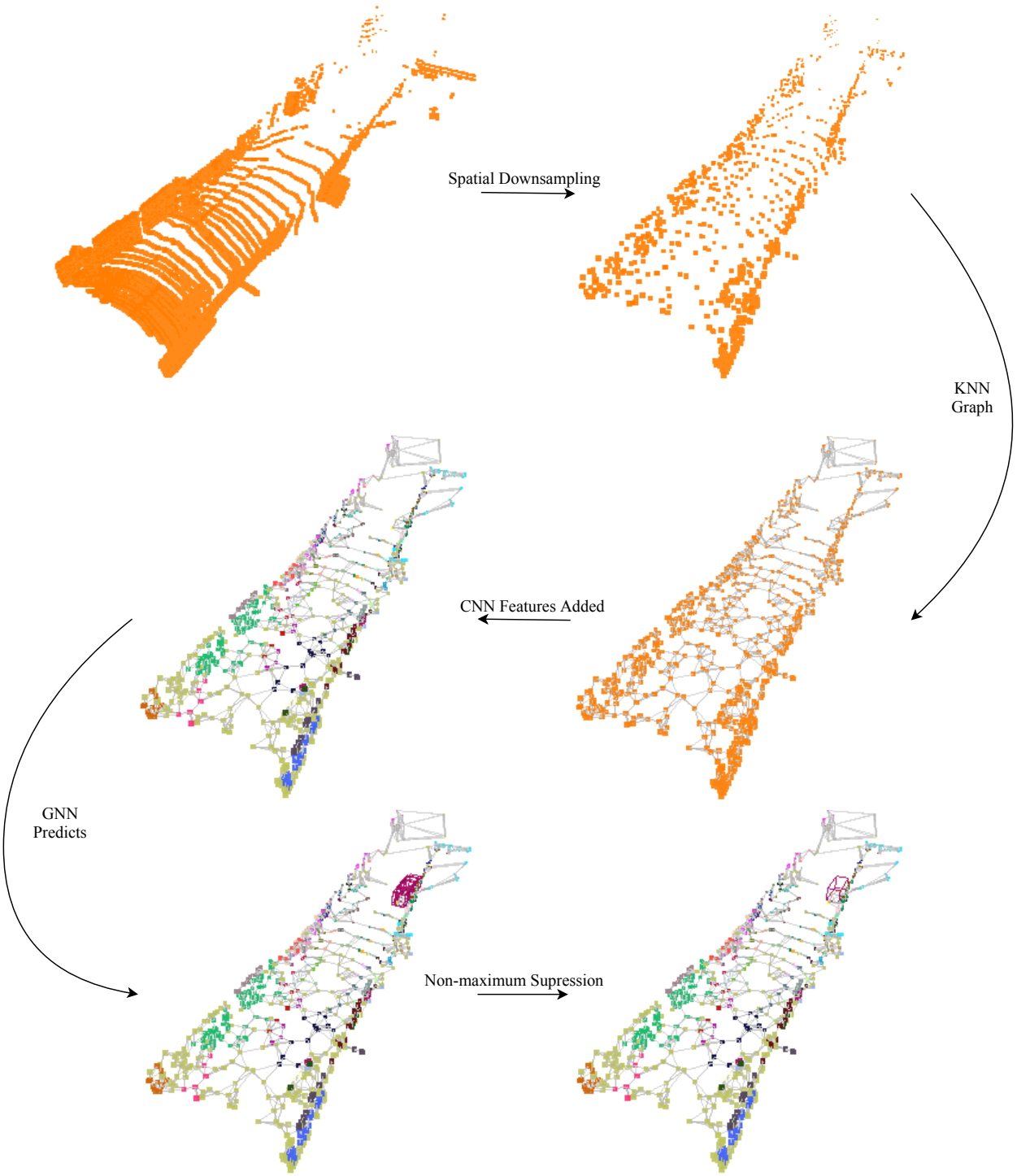

Figure 4.1: An illustration of the 3D object detection pipeline used to evaluate the multimodal learning scheme. This shows the stages of the pipeline that will be outlined in the next section. It also illustrates network performance on an unseen example, with a close-up presented in Figure 4.2. Colouring of the points is via a clustering of the CNN features but is not relevant to the operation of the system. The number of neighbours shown here is 5, as opposed to 20 as used in the system at test time.

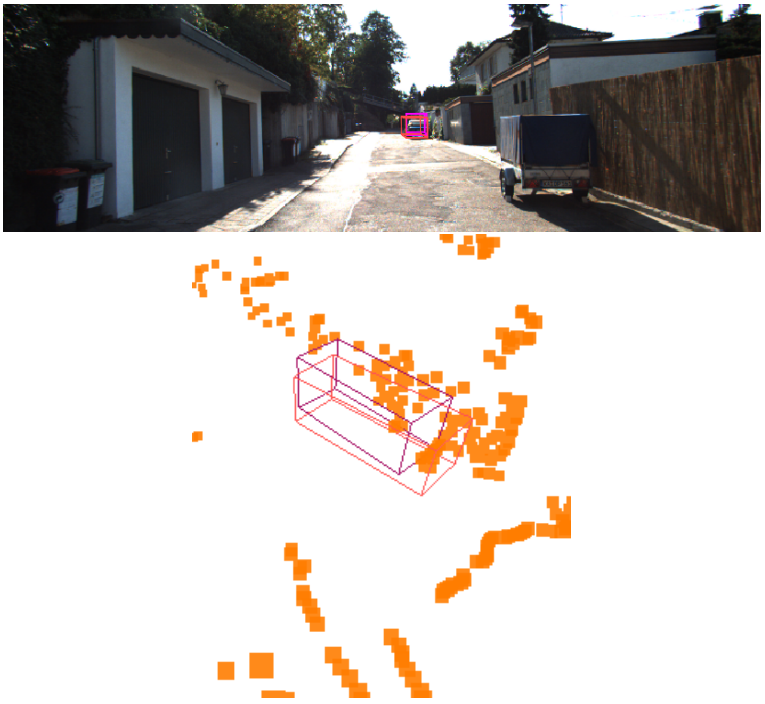

Figure 4.2: Close-up of the unseen example used as an illustration of the pipeline in Figure 4.1. Model output is in purple, ground truth in pink.

4.2 Project Architecture

Keras is a popular front-end that simplifies the use of Tensorflow [54]. In order to maximise the utility of this work to future researchers, and also to prevent unnecessary concern over minor API choices, several core components of this project were implemented as low-level functions in Tensorflow before being wrapped into simple Keras layers. This makes them compatible with a huge amount of existing work. The project was divided into two libraries, `DeepGraph` and `Multimodal2D3D`, which will be publicly released after submission to foster future progress on these problems.

4.3 DeepGraph

In the background section of this thesis, we developed a general framework for using deep neural networks to extract features from graphs. DeepGraph is a small library that exposes the fundamental operations of this framework. DeepGraph offers three set aggregation layers (Mean, Max and Attention) and one core graph operation, the GraphPairingLayer. Their operations are described at the level of an individual graph, but the scheme targets mini-batch gradient descent and thus under-the-hood operates on higher rank objects to allow for efficient batching of multiple graphs at training time.

4.3.1 GraphPairingLayer

This layer gathers the set $\mathcal{N}_i$ for each node in the graph. It is initialised with a maximum degree, K, and takes two arguments at runtime – a rank-2 tensor of features to be gathered, and a rank-2 tensor of up to K node indices for each input node. It returns two rank-three tensors. The first axis of each now indicates the neighbourhood, $\mathcal{N}_i$, the second the neighbour and the third the neighbour feature. The first of the two returned tensors simply contains the original features, $\boldsymbol{x}_i$, repeated $\#\mathcal{N}_i$ times, whereas the second returned tensor

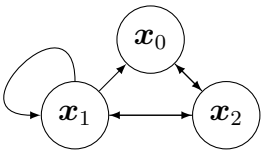

Figure 4.3: This is the graph used in Equation 4.1 as an example of the PairingLayer and MeanAggregationLayer APIs

contains the gathered features of each neighbour. This process is illustrated for an example graph in Equation 4.1. If the repeated central nodes are not needed, the first returned value can be ignored and will be compiled out of the graph at no additional overhead. Gradient masking is applied to ensure that only those indexes that are provided are used in the computation of the gradient, and the layer is fully compatible with the existing Keras masking API.

The main drawback of this approach is that the memory and compute usage scales as $O(d \times \#\mathcal{V} \times \max_i \ \#\mathcal{N}_i)$, where d is the number of features of each node. This is as opposed to the optimal $O(d \times \#\mathcal{V} \times \text{mean}_i \#\mathcal{N}_i)$. Thus it is very sensitive to the presence of nodes with a particularly high order. However, the principle advantage is that it is very amenable to parallelisation on modern GPUs. In the case of our point cloud approaches, where we will use constant-order nearest neighbour graphs, it works very well. Small numbers of large-order nodes, and in particular nodes that are connected to all others, can also be efficiently special cased. Further work could implement downsampling schemes such as those from [55] if efficient learning on graphs with a highly variable degree is needed.

4.3.2 Set Aggregation Layers

The three provided aggregation layers take rank-3 tensors as input. They aggregate over the penultimate dimension and treat their input masks appropriately. They compute, respectively, the three set aggregation functions discussed in the background section, namely mean, max and attention. The

attention set aggregation layer also takes an arbitrary tensor-to-scalar function a, used to compute the attention coefficients (from Equation 2.31). An example of how these layers can be used with the GraphPairingLayer is shown in Equation 4.1.

This API makes it simple to implement state of the art deep graph processing algorithms and allows for rapid experimentation via compatibility with a widely used framework.

4.4 Multimodal Fusion with Projection in the Network

The question remains of how best to combine the point cloud and image features that we may extract. There are several plausible schemes, but the one proposed here relies on the perspective geometry presented in the introduction. Given a calibrated camera, we know appropriate transformations to map from world coordinates to pixel coordinates. Combined with a depth buffer, this allows us to do both *projection in the network* and *back projection in the network*.

4.4.1 Projection

This is the points-to-image case. Given a point cloud, image and known calibration matrix, we may produce a projected feature map. Supposing we have a 2D feature map of $h \times w$ and n points with d dimensional features; we can project these features directly into a $h \times w \times d$ feature map. This map may be sparse, populated at n of hw spatial locations. This could potentially cause issues – at high resolution, we do not want to accidentally see through walls, for example. However, this problem can be solved by modelling points as spheres with some appropriate radius. Further, given the calibration matrix of the original image, it is straightforward to calculate

$$A = \text{GraphPairLayer(K=4)} \left(\underbrace{\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}}_{\text{Features}}, \underbrace{\begin{bmatrix} 2 & \perp & \perp & \perp \\ 0 & 1 & 2 & \perp \\ 0 & 1 & \perp & \perp \end{bmatrix}}_{\text{Neighbours}} \right)$$

$$= \left(\begin{bmatrix} \mathbf{x}_0 & \perp & \perp & \perp \\ \mathbf{x}_1 & \mathbf{x}_1 & \mathbf{x}_1 & \perp \\ \mathbf{x}_2 & \mathbf{x}_2 & \perp & \perp \end{bmatrix}, \begin{bmatrix} \mathbf{x}_2 & \perp & \perp & \perp \\ \mathbf{x}_0 & \mathbf{x}_1 & \mathbf{x}_2 & \perp \\ \mathbf{x}_0 & \mathbf{x}_1 & \perp & \perp \end{bmatrix} \right) \tag{4.1}$$

$$B = \text{Concatenate}(-1)(A) = \begin{bmatrix} (\mathbf{x}_0 \parallel \mathbf{x}_2) & \perp & \perp & \perp \\ (\mathbf{x}_1 \parallel \mathbf{x}_0) & (\mathbf{x}_1 \parallel \mathbf{x}_1) & (\mathbf{x}_1 \parallel \mathbf{x}_2) & \perp \\ (\mathbf{x}_2 \parallel \mathbf{x}_0) & (\mathbf{x}_2 \parallel \mathbf{x}_1) & \perp & \perp \end{bmatrix} \tag{4.2}$$

$$C = \text{Dense}(n)(B) = \begin{bmatrix} h_{\boldsymbol{\theta}}(\mathbf{x}_0 \parallel \mathbf{x}_2) & \perp & \perp & \perp \\ h_{\boldsymbol{\theta}}(\mathbf{x}_1 \parallel \mathbf{x}_0) & h_{\boldsymbol{\theta}}(\mathbf{x}_1 \parallel \mathbf{x}_1) & h_{\boldsymbol{\theta}}(\mathbf{x}_1 \parallel \mathbf{x}_2) & \perp \\ h_{\boldsymbol{\theta}}(\mathbf{x}_2 \parallel \mathbf{x}_0) & h_{\boldsymbol{\theta}}(\mathbf{x}_2 \parallel \mathbf{x}_1) & \perp & \perp \end{bmatrix} \tag{4.3}$$

$$D = \text{MeanAggregation}()(C) = \begin{bmatrix} h_{\boldsymbol{\theta}}(\mathbf{x}_0 \parallel \mathbf{x}_2) \\ \frac{1}{3}(h_{\boldsymbol{\theta}}(\mathbf{x}_1 \parallel \mathbf{x}_0) + h_{\boldsymbol{\theta}}(\mathbf{x}_1 \parallel \mathbf{x}_1) + h_{\boldsymbol{\theta}}(\mathbf{x}_1 \parallel \mathbf{x}_2)) \\ \frac{1}{2}(h_{\boldsymbol{\theta}}(\mathbf{x}_2 \parallel \mathbf{x}_0) + h_{\boldsymbol{\theta}}(\mathbf{x}_2 \parallel \mathbf{x}_1)) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\#\ \mathcal{N}_0} & \sum_{i \in \mathcal{N}_0} & h_{\boldsymbol{\theta}}(\mathbf{x}_0 \parallel \mathbf{x}_i) \\ \frac{1}{\#\ \mathcal{N}_1} & \sum_{i \in \mathcal{N}_1} & h_{\boldsymbol{\theta}}(\mathbf{x}_1 \parallel \mathbf{x}_i) \\ \frac{1}{\#\ \mathcal{N}_2} & \sum_{i \in \mathcal{N}_2} & h_{\boldsymbol{\theta}}(\mathbf{x}_2 \parallel \mathbf{x}_i) \end{bmatrix} \tag{4.4}$$

Figure 4.4: An example of the GraphPairingLayer and aggregation layers being applied to a small graph. Masked values are denoted by $\perp$. Concatenation is denoted by "$\parallel$". The graph is illustrated in Figure 4.3

.

appropriate transformations to project into any layer of the network, even if the spatial dimensions have changed due to pooling.

We can see this as an operation of two steps. First, we use a depth buffer and projective geometry to render a two-dimensional array of indices from the point cloud. A spatial location in this array is equal to the index of a point if and only if that point renders to that location in the image. Then, we gather the point features indexed by this array of indices into a rank-3 spatial feature map. If X is a point cloud; $\boldsymbol{x}_i$ gives the 3D spatial location of the point with features $X[i]$; `render` encapsulates the rendering process, transforming from 3D to 2D coordinates and returning $(\perp, \perp)$ if a point does not render to the image; and Pad denotes some a vector with which to fill the unoccupied 2D location, then

$$\text{Projection}(X)[a, b, c] = \begin{cases} X[i, c] & \text{if render}(\boldsymbol{x}_i) = (a, b) \\ \text{Pad}[c] & \text{otherwise} \end{cases} \tag{4.5}$$

We can learn Pad or fix it to a constant.

4.4.2 Back-Projection

This is the image-to-points case. Similarly to projection, we may calculate the 2D spatial coordinates of any point. We may then use bilinear interpolation, or a local Gaussian kernel to blend the image features at this calculated spatial coordinate into a vector that may be combined with the point. Again, this may occur at any level in the feature hierarchy.

To illustrate the simplest approach (with no feature interpolation), consider a feature map I. We may then define

$$\text{BackProjection}(I)[i, c] = \begin{cases} I[a, b, c] & \text{if render}(\boldsymbol{x}_i) = (a, b) \\ \text{Pad}'[c] & \text{otherwise} \end{cases} \tag{4.6}$$

These two operations are fully differentiable with respect to the parameters of each network. In Multimodal2D3D, they are implemented on the CPU; but this was only made because these operations were not the bottleneck in the experiments that were run. Because the projection operations used are almost identical to those in a standard graphics pipeline, they could be implemented very efficiently on the GPU. The random access gathering of features might pose more of a performance hot point for very large point clouds, but the operation is asymptotically $O(n)$ assuming constant time memory access.

4.4.3 Advantages & Limitations

This *projection in the network* approach is fully differentiable with respect to both the parameters of the CNN and the graph network. It allows for both spatial and image features to be captured effectively.

Projection in the network has a number of theoretical advantages over alternative approaches, but one principal drawback. The limitation is the requirement of camera calibration information – without this mapping, the approach is not viable. However in autonomous systems, calibrated camera systems are usually assumed, and numerous schemes are available for reconstructing camera calibration parameters on-the-fly [56, 57, 58].

The principal advantage is the preservation of the relative spatial information. If our graph network can confidently indicate that a point is on a planar surface based on local geometry, then we can project this into our CNN's view of the world and indicate exactly where that information is salient.

This approach is also directly applicable to the multiview case, even if the cameras may move independently. This is because there are no intrinsic ordering assumptions to be violated by wildly different camera positions.

By using both mappings, we may also employ cross-modal approaches to further increase the flexibility of communication between the modalities. Features could be exchanged via projection and back projection at multiple

points in the network, and combined with powerful methods for cross-modal architecture search which automatically tune where to allow communication [59].

4.5 Single Shot 3D Object Detection

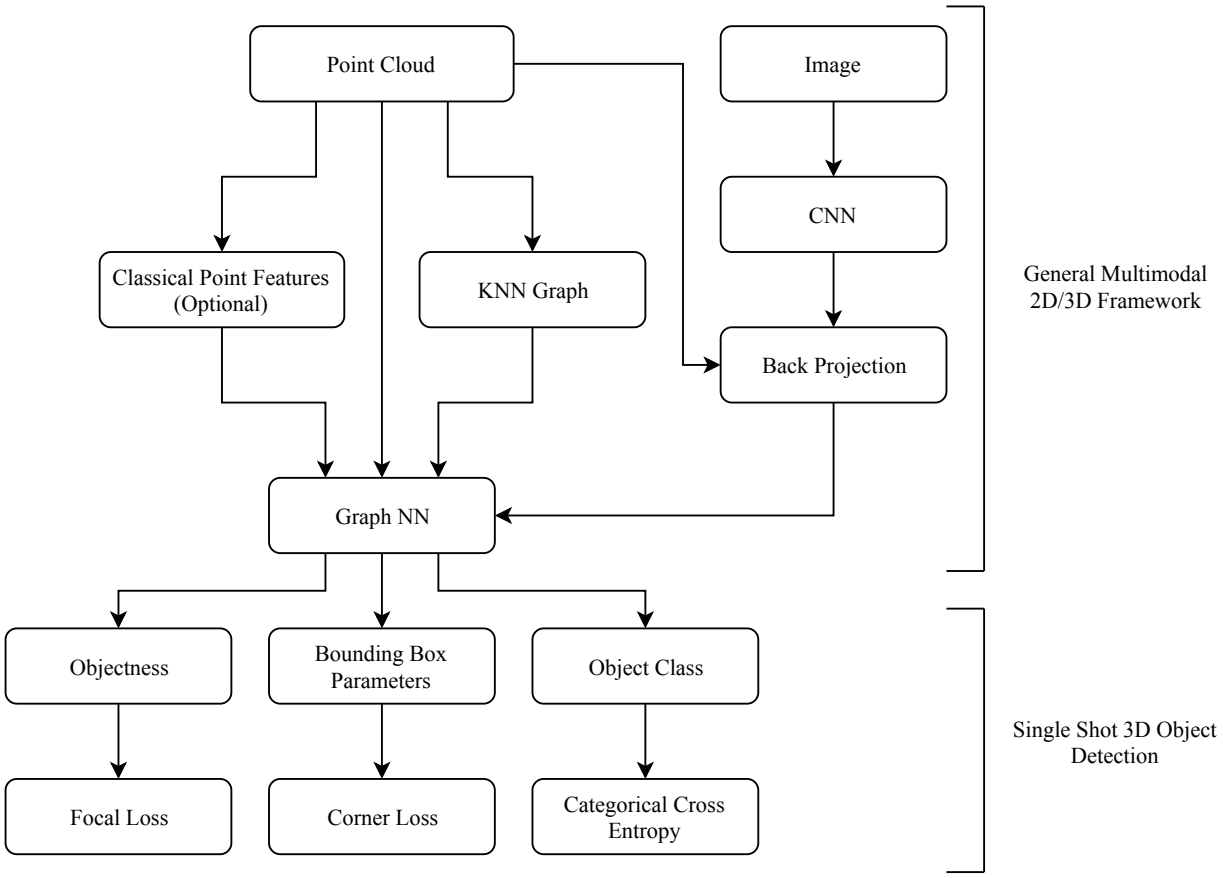

Figure 4.5: This diagram outlines the two core components of this work. The upper section shows an instantiation of the general 2D/3D fusion approach. The bottom section indicates how this network is used to predict 3D bounding boxes via a multi-task objective.

Having proposed a general framework for handling 2D/3D multimodal fusion, we now apply it to a difficult problem. 3D object detection was chosen because it represents a scenario that benefits greatly from the combination

of the two modalities – however, this particular task poses more challenges beyond simple segmentation tasks (where the implementation is simpler but the benefits less pronounced). The overall architecture is illustrated in Figure 4.5. This shows how the scheme combines the 2D and 3D features to make a prediction and provides an overview of the process that is now described.

For 3D bounding box prediction, we need to decide an appropriate scheme for mapping from our graph features to bounding boxes. The approach used was based on single shot detection (SSD). This is because SSD approaches are faster and more conceptually straightforward than their region-proposal-based counterparts.

3D bounding boxes are usually sufficiently tight for each detected LIDAR point to fall into exactly zero or one bounding box. Thus we can predict a binary "objectness" property for each sampled point that is 1 if that point is within the bounding box of any annotated object in the scene, and 0 otherwise. These 3D points, which fall into bounding boxes, are referred to as object points.

Each bounding box has exactly one label from a predefined set. Thus we may annotate all object points with a categorical label, referred to as the box label. Each bounding box also has a location, rotation and size. We annotate each object point with the rotation and size, as well as the vector from the point to the object centre. We use the term "bounding box parameters" to refer to these variables.

3D object detection then achieved by jointly optimising these outputs as a *multi-task objective*, with three distinct loss terms. In some previous works on 2D single-shot detectors, the authors (quite surprisingly) have jointly optimised all the outputs with a mean squared regression error [12]. This did not converge in our case.

4.6 Categorical Cross Entropy

The simplest of the three loss terms is the categorical label. This output from the model is standardised with `Softmax` and the categorical cross entropy loss introduced in Equation 2.8 is used. The KITTI dataset contains 3 labels for 3D object classification – Cars, Pedestrians and Cyclists.

4.7 Corner Loss

The bounding box parameters are optimised via regression. Previous work has established that ignoring geometry and simply regressing the orientation terms with MSE is inappropriate [60, 61]. Thus I elected to replicate the "corner loss" of [61] that reconstructs the corners from the bounding box parameters and uses a total absolute error in distance between these reconstructed corners.

Given that $(\boldsymbol{l}, \boldsymbol{d}, \boldsymbol{\phi})$ are the true location, dimensions and orientation of the box; that $(\hat{\boldsymbol{l}}, \hat{\boldsymbol{d}}, \hat{\boldsymbol{\phi}})$ give our model's estimates of these parameters and that $\boldsymbol{c}$ gives the points of the unit cube, then the reconstructed corners and the loss term are given by:

$$\text{BBox}(\boldsymbol{l}, \boldsymbol{d}, \boldsymbol{\phi})_{ij} = \sum_k [R_{\boldsymbol{\phi}}]_{ik} \boldsymbol{d}_k \boldsymbol{c}_{kj} + \boldsymbol{l}_i \tag{4.7}$$

$$\text{CornerDistance}((\boldsymbol{l}, \boldsymbol{d}, \boldsymbol{\phi}), (\hat{\boldsymbol{l}}, \hat{\boldsymbol{d}}, \hat{\boldsymbol{\phi}})) = \sum_{i,j} \text{abs}([\text{BBox}(\boldsymbol{l}, \boldsymbol{d}, \boldsymbol{\phi}) - \text{BBox}(\hat{\boldsymbol{l}}, \hat{\boldsymbol{d}}, \hat{\boldsymbol{\phi}})]_{ij}) \tag{4.8}$$

The advantage of this approach is that it automatically scales the contribution of the angle, dimension and location parameters appropriately. The absolute value is used (as opposed to the squared error) to reduce the effect of outliers on the dataset. Some care must also be taken in regressing the orientation parameters. Directly regressing the angle is unstable because of discontinuities about 2π, and so instead we regress two parameters for each

angle ϕ, namely $\sin\phi$ and $\cos\phi$. These functions of ϕ may then be used to reconstruct $R_{\boldsymbol{\phi}}$.

4.8 Focal Loss

As mentioned at their introduction, single shot detectors suffer from a core problem of class imbalance. Most of the anchors in most images are not associated with objects. Classical approaches to solving class imbalance problems often artificially upsample the smaller classes, but this cannot easily be done in an SSD because the anchor predictions are fundamentally associated with each other at the image level.

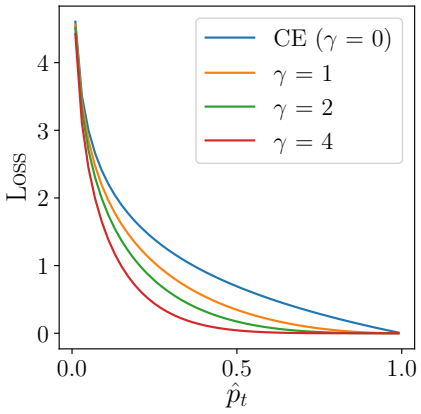

Figure 4.6: The focal loss is a heuristic approach to solving the issues caused by the variety in example difficulty and the class imbalance. Setting $\gamma > 0$ reduces the relative loss for well-classified examples, focusing the model on difficult cases.

Another related issue is the variety of example difficulty. Some objects are large, close to the camera and well represented in the dataset. The model quickly learns to classify these easy examples, but the loss can still be quickly reduced by pushing the predicted probability of the true class ever closer to

1.

These two problems can be remedied by altering the loss function. The focal loss [44] is one recent and effective approach. In this case, we depart from the purely probabilistic formulation of minimising the KL divergence (or equivalently the cross-entropy), and instead define the following heuristically constructed loss function. Given a predicted probability from the model, $\hat{p}$, we define the predicted probability of the true label as $\hat{p}_t$ –

$$\hat{p}_t = \begin{cases} \hat{p} & \text{if } y = 1 \\ 1 - \hat{p} & \text{if } y = 0 \end{cases} \tag{4.9}$$

We can then note that the cross-entropy loss and focal loss are respectively defined as

$$\mathrm{CE}(p_t) = -\log(p_t) \tag{4.10}$$

$$\mathrm{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t) \tag{4.11}$$

Where γ and α_t are parameters of the focal loss that may be tuned to influence to what extent easy examples are discounted (γ) and the recall/precision tradeoff (α). The effect of altering γ is illustrated in Figure 4.6.

This class imbalance between the object and non-object classes is severe. On average, around 5% of captured points are from labelled objects. If binary cross entropy is used, the model collapses to predicting all points as non-object and does not converge.

Ideally, a hyperparameter search would be carried out using cross-validation to optimise for the parameters of the focal loss; however with each model taking a little under 24 hours to train, this was not viable, and so standard parameters from a similar task were used [44]. Fortunately, this alteration was the difference between non-convergence and a working model.

4.9 Combining the Losses

Having determined the three loss functions of the network, these must then be combined, or *scalarised* into a single loss parameter for the network. This task is not trivial, as the difficulty of each task varies. The most successful approach for multi-task scalarization in a similar domain uses strong assumptions about the probabilistic nature of the classification and regression losses used [62] that are not applicable to the focal and corner loss. Such multi-task reweighting strategies would also add more complexity. Amongst the state of the art for 3D object detection, a simpler approach is widely used, in which a simple linear reweighting is used based on heuristic expert assumptions about relative task difficulty [61, 45, 60, 50, 51]. This approach is both plainly sub-optimal and almost ubiquitous. Optimal dynamic reweighting of arbitrary multi-task losses would be a valuable research contribution. In this instance, however, we emulate the existing state of the art and use a weighted sum of the three loss functions.

4.10 CNN Features

The multimodal architecture can use features extracted by an arbitrary CNN. The overview of CNNs given in the background section neglected several key insights that have been gained over several years of empirical experimentation, which have resulted in a large design space to choose from. This thesis does not aim to contribute the field of low-level CNN design, and thus an "off the shelf" architecture was used that provides very desirable properties for the task at hand.

4.10.1 MobileNetV2

The architecture selected was "MobileNetV2", a state of the art convolutional neural network that makes extensive use of depthwise separable convolutions

and residual connections to improve training performance and reduce memory load [63]. This network is not the most accurate available, but it has a light GPU memory footprint, allowing the simultaneous training of all components of the multimodal learning system on a single GPU with reasonable batch size. The architecture was initialised with weights from pre-training on the ImageNet classification task [7], as a form of transfer learning.

4.10.2 Hypercolumns

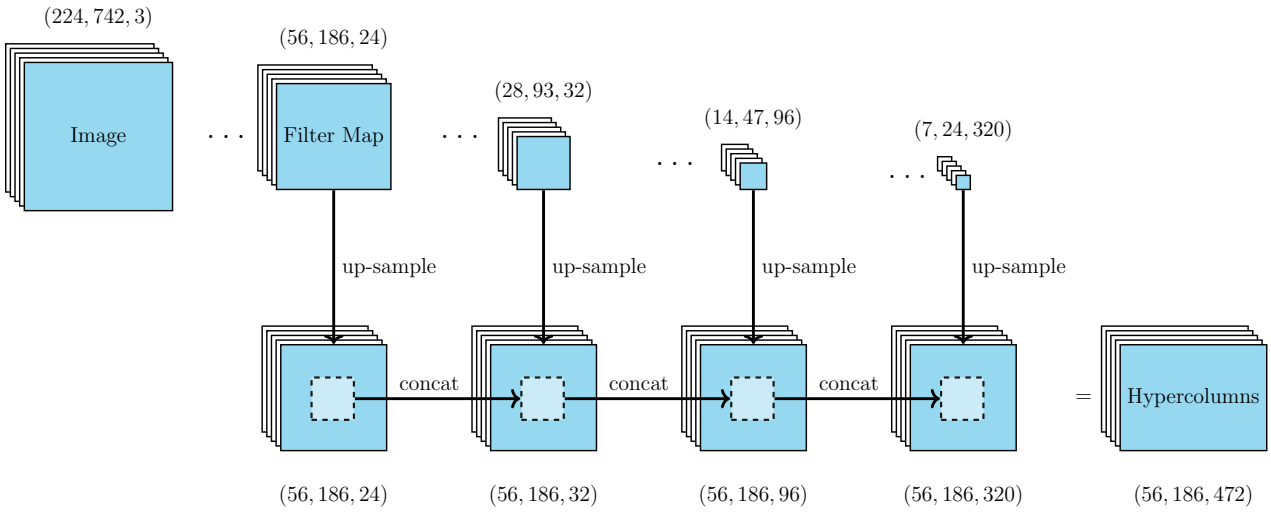

Figure 4.7: Hypercolumns. Layer shadow effect used with permission from [64].

MobileNetV2 extracts feature maps at a number of scales throughout the network. This is accomplished by increasing the *stride* of some convolutional layers to 2, which halves the spatial dimension. This has a number of advantages, increasing the size of the later kernel's receptive fields whilst reducing the memory and compute requirements of the network.

However, there are also a number of disadvantages. The final output of the network is at only 3% of the resolution of the input, and such a low-resolution cannot provide the fine-grained information needed to solve the bounding box prediction task for our sampled points. In order to solve this problem, we can

use bi-linear interpolation to increase the resolution of the later feature maps before concatenating them with earlier ones. This process is differentiable, and so the weights of the CNN may still be trained with SGD.

This process is illustrated for some layers of MobileNetV2 in Figure 4.7. It shows the layers extracted, along with their sizes for the KITTI object detection task.

There are alternative techniques available for this upsampling process, including approaches designed for pixel-wise semantic segmentation [65, 66]. The hypercolumn approach was selected because it has shown good previous results for transfer learning [67] and introduces the smallest amount of additional complexity. Experimenting with different CNNs and upsampling approaches is an interesting avenue for future work.

4.11 Graph Network Architecture

The graph network hyperparameters were selected based on a design that achieves good object classification scores [22]. Two levels of neighbour aggregation were used, with concatenation to combine centre and neighbouring node features. MLPs perform the non-linear transformation before max aggregation is used to combine neighbourhood features. A hidden layer size of 64 was used for each MLP. The network is then split into three heads that predict objectness, box label and bounding box parameters. This design is fully illustrated in Figure 4.8.

4.12 Other Components

Thus far, we have described the differentiable parts of the object detection system. However, there are two more necessary parts to complete the scheme.

4.12.1 Pointcloud Downsampling

Point clouds that are collected by modern sensor systems often contain very large numbers of points. This can slow down training; potentially unnecessarily. One approach to surmounting this issue is to downsample the point cloud before passing it to the training system. This also confers the additional advantage of acting like a data augmentation scheme, which can potentially improve the performance of the system in terms of accuracy, not just speed.

However, the dynamics of how the points are captured does matter. Unfortunately, whilst interesting obstacles are distributed quite uniformly over the road plane, the pattern of LIDAR capture is not uniform over space. instead, we end up with two effects. Because the sensor emanates light radially from the centre of the scene, the density of sampled points per unit volume decays as we move outward from the vehicle. Further, the likelihood of successfully detecting the returned light also decreases. As a result, we end up with many more points captured quite close to the vehicle than further away. If naive random downsampling is used, then very important information about distant objects is lost, and a great deal of redundant information about close objects (and indeed the road surface itself) is retained.

In order to prevent this loss, we can artificially alter the sampling strategy to be more uniform over space. There are several ways to accomplish this; some of which are very slow [68]. The technique used was based on the jitter approach to approximating Poisson disk sampling [68]. For each point $\boldsymbol{x}_i$ we quantise the value $\boldsymbol{x}_i + \boldsymbol{\epsilon}_i + \boldsymbol{u}$ where $\boldsymbol{\epsilon}_i$ is sampled from a scaled diagonal Gaussian distribution for each point, and $\boldsymbol{u}$ is sampled for each point cloud from a uniform distribution. This divides the point cloud into a number of voxel bins but mitigates any local boundary artefacts that might occur due to quantisation. The importance of these jitter terms is illustrated in Figure 4.9. Points are then sampled with probability inversely proportional to the number of voxels in their bin. This spatial sampling approach is compared to uniform sampling in Section 5.2.1

4.12.2 Non-maximum Supression

The model itself estimates an objectness score and bounding box for every sampled point. This gives many bounding boxes for the same object; but fortunately, there is a standard approach from 2D object detection that is directly applicable, known as greedy non-maximum suppression (greedy NMS). In this instance, we take all predicted bounding boxes and set an intersection over union threshold. First, we compute the box predicted for each predicted object point. Starting with the box that has the highest objectness score, we greedily consider all boxes that overlap with that box to be predicting the same object. This continues, clustering the predicted boxes. Each cluster can then be combined into a single prediction of the model, traditionally by considering the prediction of the highest scoring box.

Because the objects of interest in the KITTI dataset all lie on the road plane, this was done with a top-down projection of the predicted boxes. Because no publicly available code could be found that performs NMS for non-axis aligned bounding boxes, a small C++ library with Python bindings was written and released to perform only this task.

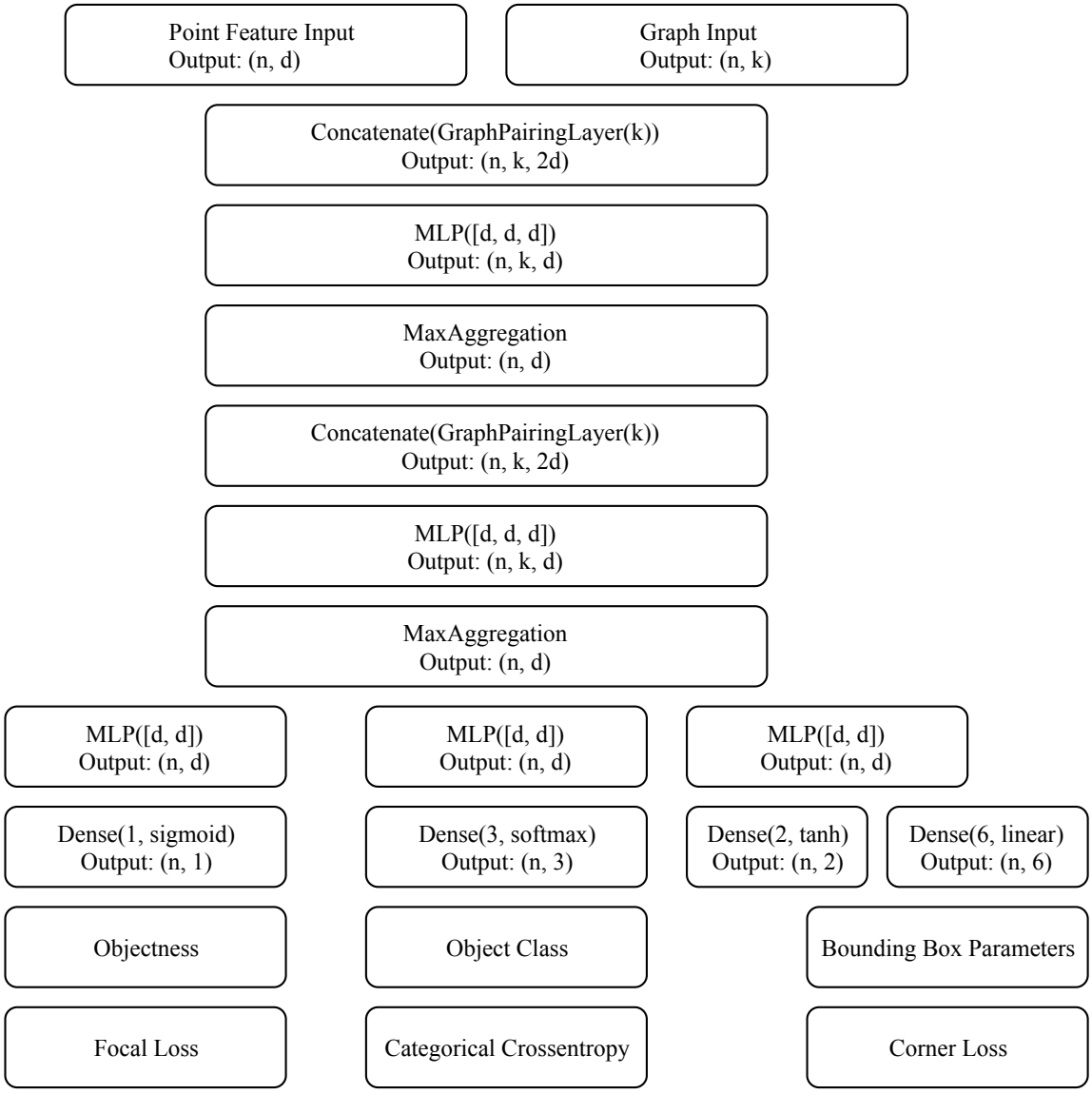

Figure 4.8: The structure of the graph network. The hyperparameters used for the KITTI experiments were – $d = 64, K = 20, n = 2048$

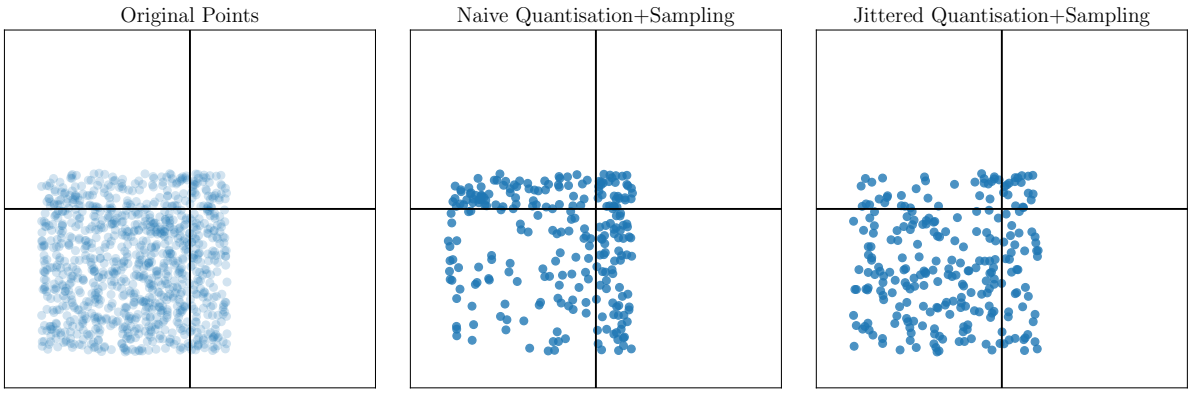

Figure 4.9: It is important to include the jitter terms in the spatial sampling process to prevent artefacts at voxel boundaries. This can be illustrated by sampling a constructed distribution with and without jitter. Black lines are voxel boundaries.

Chapter 5

Evaluation

5.1 ModelNet40

The Stanford ModelNet40 dataset is a standard dataset of labelled object models. [4]. It can be treated as a 40-way point cloud classification problem by sampling points from the faces of the models, as in [19, 22]. This was the source of the Chair and Aeroplane images in the introduction. The dataset provides a standard training/test split that was used for comparisons to other work. In addition to this, 10% of the data was separated as a validation set to allow for early stopping.

This dataset was used as a benchmark to ensure that the DeepGraph API was functioning as intended. It shows that a small amount of code using the

	Top 1 Test Accuracy (%)
3D ShapeNets	77
VoxNet	83
PointNet++	**92**
DeepGraph Benchmark	83

Table 5.1: The DeepGraph API allows for the quick construction of models that provide reasonable performance.

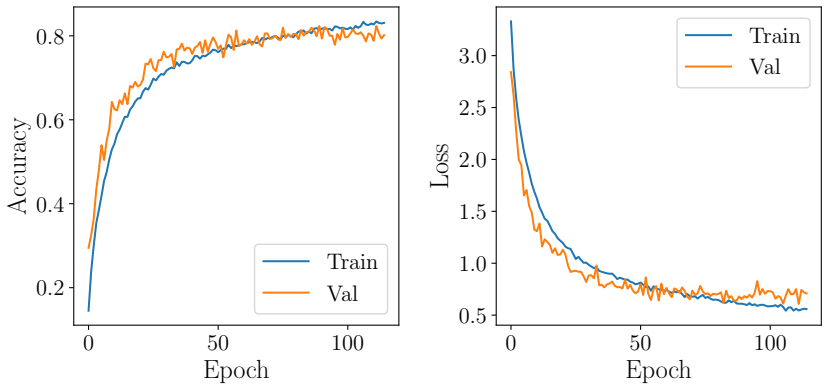

Figure 5.1: The DeepGraph API can be used to quickly develop models that compare well to highly tuned object classification schemes.

API can quickly approach the performance of the state of the art. Appendix A provides the full specification of the model used.

Table 5.1 compares the performance to canonical work on 3D object classification, and Figure 5.1 illustrates the training curves.

5.2 KITTI

The KITTI dataset was used to test the multimodal 3D object classification scheme [5]. The dataset consists of 7481 annotated images and associated LIDAR scans. The standard evaluation procedure from [51, 69, 70, 45] was used, in which this set of images is divided into two, giving 3,712 data samples for training and 3,769 data samples for validation. The split ensures that no images from the same driving sequence are included in both the training and the validation set. Detections are evaluated according to three difficulty levels (Hard, Moderate, Easy) based on object size in the image, truncation and occlusion. Typical KITTI images, along with the predictions from the multimodal model, are illustrated in Figure 5.2.

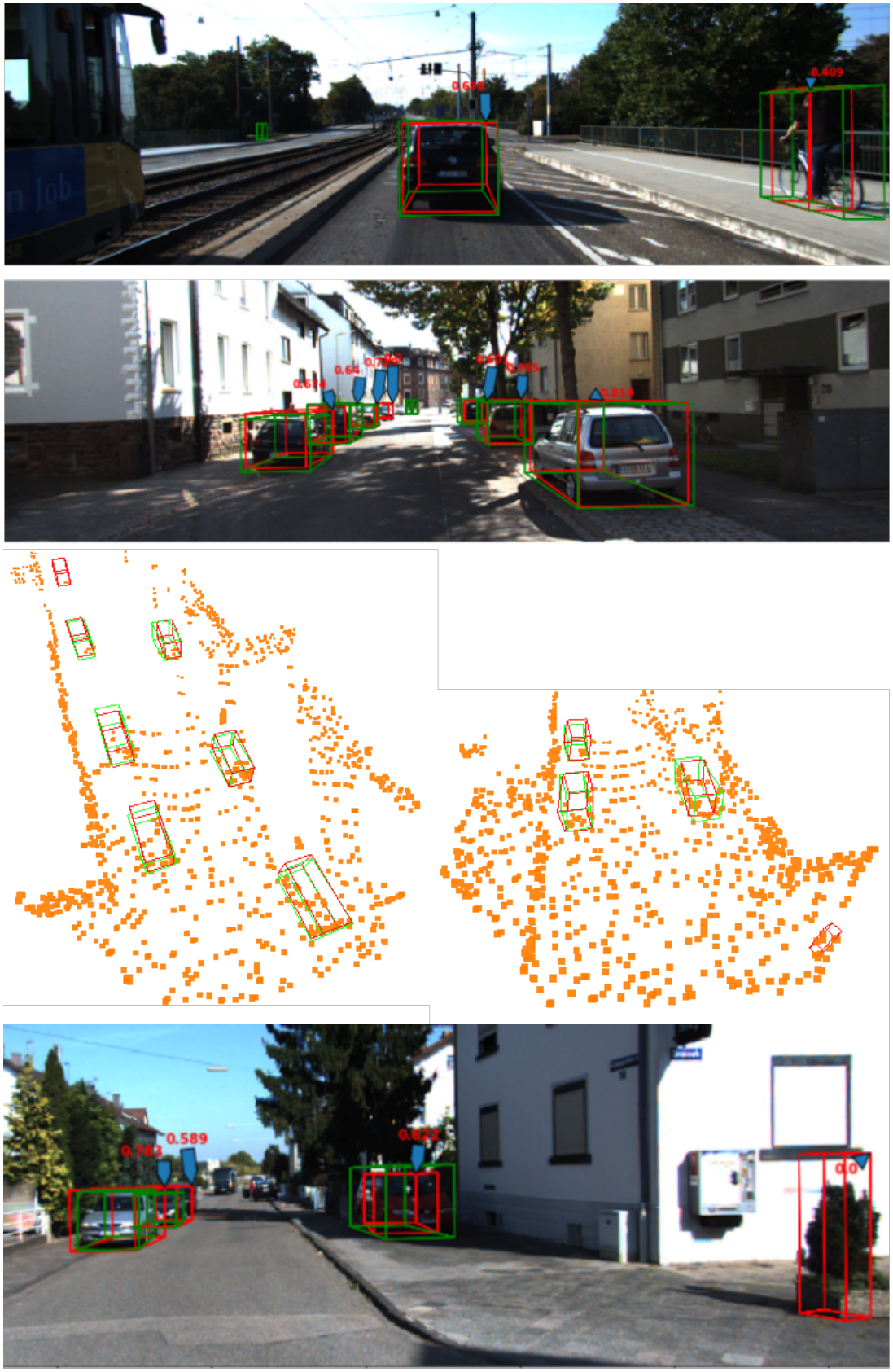

Figure 5.2: Typical KITTI images and object detection results from the multimodal system. Ground truth in green, predicions in red. Numbers indicate volume IoU, no point cloud is shown for top image.

The mean average precision (MAP) is the primary evaluation metric for KITTI. In this scheme, object detection performance is reduced to a single number by specifying a required intersection over union (IoU) threshold. This threshold is in terms of the area of overlap of the predicted box and the ground truth in the 2D case. In the 3D case, it is defined by volume. It is defined as:

$$\text{IoU} = \frac{\text{Overlapping volume of prediction and ground truth}}{\text{Total volume of prediction and ground truth}} \tag{5.1}$$

Given an IoU threshold, a set of predictions for a scene may then be classified as true positives, false positives or false negatives. Given these binary scores, precision and recall curves may be drawn, and the mean average precision calculated for the classifier at that IoU threshold.

The required area overlap for 2D predictions is 50% for pedestrians and 70% for vehicles. The standard volume IoU thresholds are the same. This is perhaps one reason why KITTI is frequently referred to in the literature as "the challenging KITTI dataset" [71, 72, 73] – a volume overlap of 70% is equivalent to an overlap of 88% in each dimension. Comparisons between the proposed systems and the unimodal baselines were difficult to interpret at this very high threshold because results were dominated by predictions close to the camera. Thus comparisons between the baselines are reported at a reduced volume overlap for cars – 50%, as in [51, 74, 75].

In order to determine how the multimodal fusion technique influenced the performance of the system, two baselines were used for comparison. These follow an identical architecture to the multimodal system, with features from the unused modality stripped out. The architecture and hyperparameters used are specified in Figure 4.8. In the case of the image-features-only prediction system, there were several approaches that could be taken to constructing predictions – recall that the bounding box parameters in the multimodal system design are defined in relation to the object points that they capture. The most favourable approach was used (eg the one that achieved the best image-only performance). In this scheme, the image model has implicit ac-

cess to the point cloud, because predictions are made relative to sampled points. This prevents the model from needing to learn to fully infer depth. This is the closest possible model for comparison to the multimodal approach but is not truly unimodal because it cannot be applied without access to the point cloud. The graph-only scheme, by contrast, can be applied to LIDAR data without any knowledge of a paired image.

Optimisation used a standard SGD variant, AMSGrad, with hyperparameters at the author's reccomended settings [76]. The batch size was 12 for all models. The multimodal model was trained until the validation loss did not improve for 25 epochs. The unimodal baselines were trained for 200 epochs (despite overfitting), to ensure that the results of the experiment were not due to slower convergence. The evaluation then took the best performing model on the validation set in every case.

5.2.1 Sampling Recall

In the implementation section (Section 4.12.1), we asserted that naively randomly downsampling points from the point cloud would over-sample close to the vehicle and reduce the number of object points captured. To verify this, we can inspect the per-instance recall of the two sampling schemes over the training set. For each point cloud, we apply the sampling procedure and determine the proportion of annotated objects for which points were successfully captured in the sample. These results, shown in Table 5.2 shows that there is a large advantage to the proposed spatial downsampling procedure.

	Instance Recall
Naive Sampling	0.80
Spatial Sampling	0.91

Table 5.2: The spatial sampling procedure improves recall. Because objects that are not captured in the downsampling process cannot be detected, this is an important aspect of upper-bound performance.

5.2.2 Training

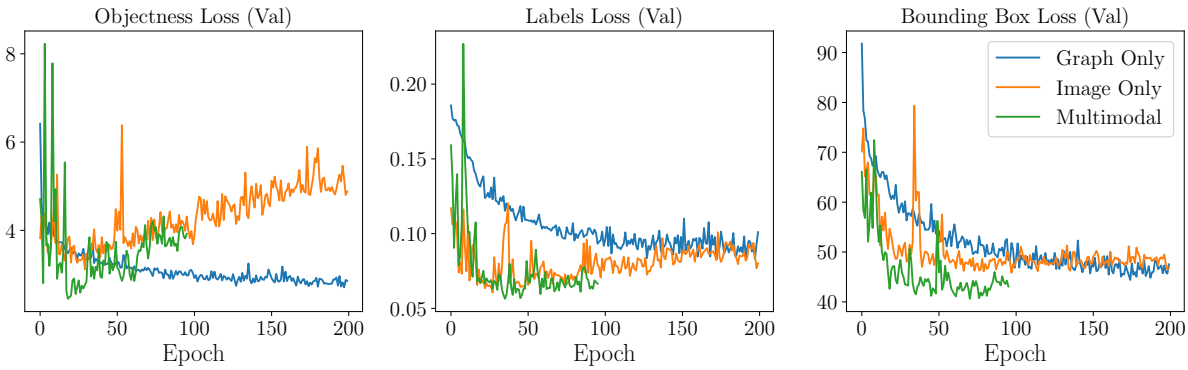

Figure 5.3: Inspecting the individual validation losses shows how some objectives begin to overfit before others. Note also that the image-only system achieves better object classification performance, whilst the graph-only approach is better at identrifying object location, size and orientation by optimising the bounding box loss.

Figure 5.3 illustrates the training losses for each of the three loss components over the course of training the model. It highlights one of the more challenging issues presented in this project – varying task difficulty in the multi-task loss. As mentioned when multi-task learning was introduced (Section 4.9), there is no broadly accepted approach to selecting optimal task weights other than cross-validation. Because training times for the detection networks approached 24 hours, it was not possible to determine an optimal weighting. As a result, we can see that the validation losses do not reach a minimum simultaneously, reducing final performance. Developing a general approach to balancing multi-task objectives that can accommodate heuristic losses such as the focal loss would be an exciting research direction that could improve performance on numerous tasks, including this one.

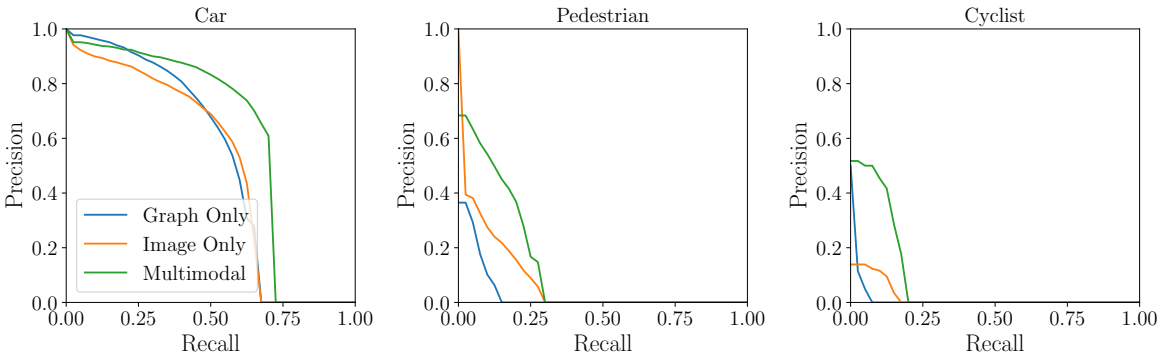

Figure 5.4: The precision-recall curves show that the multimodal approach improves detection performance for all classes. The effect is more pronounced for smaller objects.

5.2.3 Qualitative Results

Figure 5.2 shows typical performance of the object detection system. The annotations show the volume overlap for each predicted bounding box with the ground truth; highlighting that the principle issue with the system is in the fine-grained localisation component of detection. Whilst there are some egregious false positives, such as the potted plant, these are not the primary source of errors. Instead, the errors are dominated by limited volume overlap – note that many initially impressive looking predictions fail to meet the stringent 70% criteria.

Figure 5.5 shows a qualitative comparison between the predictions of the unimodal baselines and the multimodal system. Inspection of these results, along with other visualisations omitted for space, presents a dominant theme in the errors made by the unimodal baselines. The image-only model struggles due to the frequent occluded objects in the dataset – as a result, it frequently predicts "ghost" objects behind observed vehicles and pedestrians, as it cannot exploit 3D information to determine the precise 3D boundaries needed. The graph-only point cloud approach has a different problem – it strongly associates protrusions from the flat road surface with objects, and

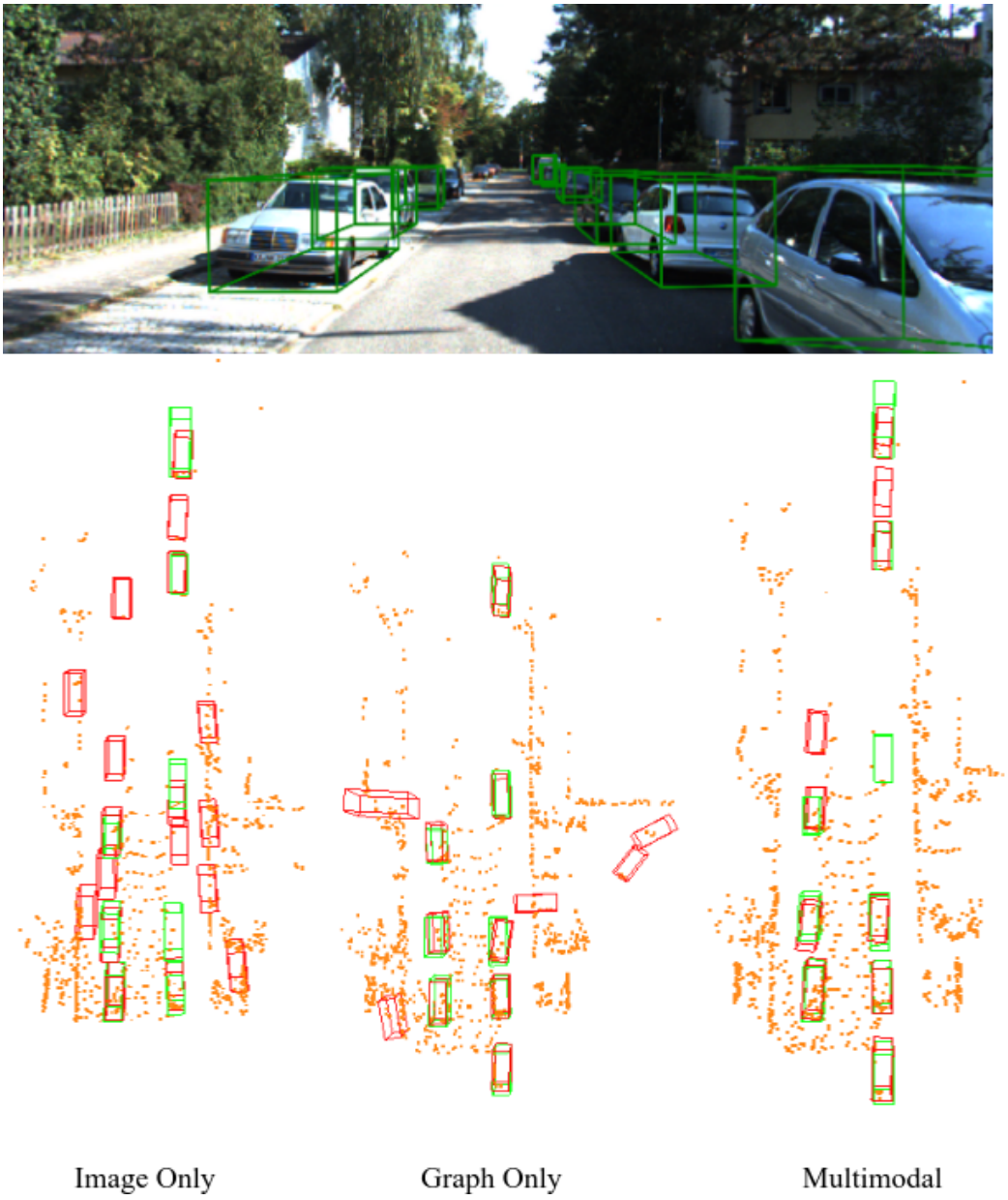

Figure 5.5: Qualatative comparison of the approaches. Red indicates prediction, green ground truth. The image-only approach detects ghosts behind visible objects. The graph-only approach is confused by protrusions from the ground that are not objects of interest. The multimodal approach helps to eliminate these two classes of error.

	Car	Pedestrian	Cyclist
Graph Only	68.9	5.74	4.55
Image Only	64.5	10.9	3.60
Multimodal	**75.9**	**17.9**	**15.3**

Table 5.3: The multimodal fusion scheme improves performance across all object detection classes by a large margin.

with thus sometimes mistake distant trees for pedestrians, bushes for cars and similar. The combined, multimodal approach allows for the elimination of these two classes of error.

5.2.4 Quantitative Comparison

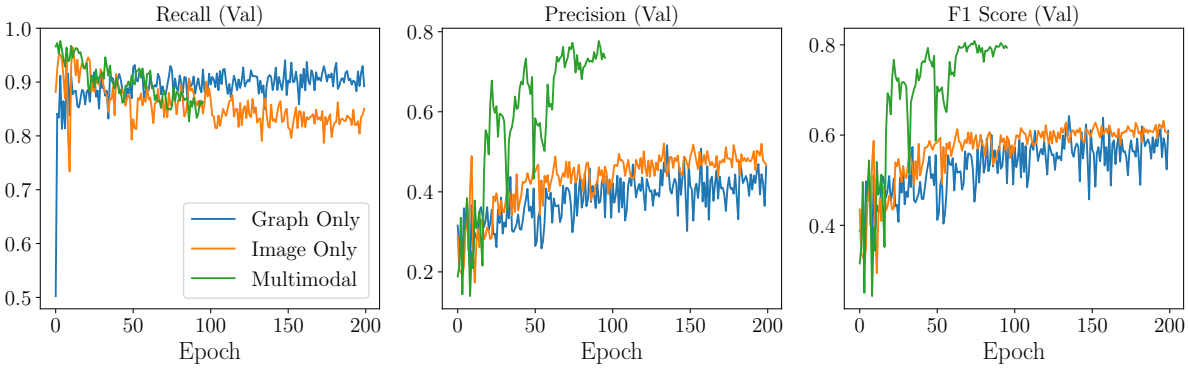

Figure 5.6: The recall, precision and F1 score may be computed for the binary "objectness" classification task throughout training. The multimodal model exhibits a signifigant improvment in recall that corresponds to our intuition that the modalities help correct each other.

Table 5.3 compares the MAP scores of the baselines and the multimodal method on the 3D object detection task. The multimodal approach dramatically improves performance for all classes, strongly confirming the central hypothesis of this thesis. Inspecting the model progress in Figure 5.6 indicates that this is because of a dramatic improvement in the objectness pre-

diction achieved by combining the two approaches. Note that the massive improvement in F1 is due to a reduction in false positives (e.g. an improvement in precision). This directly corresponds with our qualitative intuitions from inspecting model output, where we noted that the improvements appear to be due to each modality allowing for the correction of errors from the other.

5.2.5 Standardised KITTI Comparison

The results thus far have shown compelling evidence that multimodal fusion greatly improves the performance of the single shot 3D object detection pipeline. Comparisons to highly tuned object detectors reflect less well, however, on the single shot approach. The principal advantage, speed, is indeed carried over from the 2D case. On a modern GPU, this pipeline runs at 25+ frames per second, or 35ms per frame, with no optimisation. However, the model struggles to capture the fine-grained pose information needed to achieve high scores at the 70% threshold. It is typical of other authors who focus on speed to present data at the 50% level for the validation set. Comparing to work that presents this metric, we see that this single shot approach is competitive, but not yet optimal, as shown in Table 5.4. Table 5.5 presents comparisons to state of the art approaches at the 70% threshold. These results make it clear that further work could be invested in fine-tuning the hyperparameters of this design to improve the overall performance, but this does not undermine the central result – that multimodal fusion dramatically improves the performance of the proposed architecture over unimodal baselines with the same structure.

	Easy	Moderate	Hard	Time (ms)
MV(BV+FV) [51]	**95.74**	**88.57**	**88.13**	240
F-PC CNN (MS) [74]	87.16	87.38	79.40	500
BirdNet [75]	88.92	67.56	68.59	110
This Work	75.86	63.20	62.23	**33**

Table 5.4: This table shows the MAP for Car detections on KITTI for works reporting the 50% threshold. The single shot fusion approach presented here is less accurate, but much faster.

	Easy	Moderate	Hard	Time (ms)
Frustum PointNet [61]	70.39	81.20	62.19	170
AVOD-FPN [50]	**87.16**	**87.38**	**79.40**	500
This Work	27.60	23.73	21.28	**33**

Table 5.5: This table shows the MAP for Car detections on KITTI for works reporting the 70% threshold. As in the 50% case, the single shot fusion approach presented here is less accurate, but much faster.

Chapter 6

Summary and Conclusions

This project makes several contributions, including

- A unifying description of several recent approaches to deep graph processing in terms of learned set functions for neighbourhood aggregation.
- A small library, DeepGraph, that implements an API based on this description efficiently and in a way that is compatible with modern, high-level ML frameworks.
- A small library, MultiModal2D3D, that allows 2D and 3D data to be combined through differentiable projection and back-projection.
- Empirical results that show these techniques may be combined to achieve very fast single-shot 3D bounding box prediction that gives compelling results on a challenging dataset.
- Ablations that show the multi-modal nature of the scheme is critical to model performance; and that neither 2D or 3D data alone is sufficient to support this approach.

However, there remains a great deal of work to be done. There is a clear gap between the system proposed here and the most accurate state of the art. There are many approaches that could be taken to closing it, not least of which would be devoting more time to hyperparameter optimisation. The

flexibility of projection in the network could also be used to introduce more cross connections between the two modalities.

Recent work on automatically determining where to put these connections could potentially be applied [59]. The graph network itself could also be further developed – in particular, the graph is currently held constant throughout training. Learning to automatically adapt it to reflect the structure of the scene would allow for dynamic adaptation of the size of each node's receptive field. This might make it easier to distinguish objects.

Another exciting research direction would be to incorporate multiple cameras into the system. The point cloud could operate as a shared communication channel between views that only partially overlap.

The world around us is three dimensional. Powerful technology, such as high definition cameras and LIDAR, allow us to capture rich information about our environment in useful but varied ways. The fusion techniques explored here provide a new way to interpret this information faster than ever before, and I hope that the software tools developed will be used by future researchers to further expand our 2D view of this 3D world.

Bibliography

[1] C. Cangea, P. Veličković, and P. Liò, "Xflow: 1d-2d cross-modal deep neural networks for audiovisual classification," *arXiv preprint arXiv:1709.00572*, 2017.

[2] A. Ephrat, I. Mosseri, O. Lang, T. Dekel, K. Wilson, A. Hassidim, W. T. Freeman, and M. Rubinstein, "Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation," *arXiv preprint arXiv:1804.03619*, 2018.

[3] T. Gandhi and M. M. Trivedi, "Pedestrian collision avoidance systems: A survey of computer vision based recent studies," in *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*, pp. 976–981, IEEE, 2006.

[4] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

[5] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3354–3361, IEEE, 2012.

[6] J. G. Daugman, "Lecture notes in computer vision," 2016.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, IEEE, 2009.

[8] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[9] "Perspective projection - tex - latex stack exchange." https://tex.stackexchange.com/questions/96074/more-elegant-way-to-achieve-this-same-camera-perspective-projection-mode Modified with permission under the CC BY-SA 3.0 [**?**] (Accessed on 05/21/2018).

[10] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[13] 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*, pp. 91–99, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[15] 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.

[16] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, *et al.*, "Going deeper with convolutions," CVPR, 2015.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, vol. 1, no. 2, p. 4, 2017.

[20] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2018.

[21] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," in *Advances in neural information processing systems*, pp. 4974–4983, 2017.

[22] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *CoRR*, vol. abs/1801.07829, 2018.

[23] P. Veličković, D. Wang, N. D. Lane, and P. Liò, "X-cnn: Cross-modal convolutional neural networks for sparse datasets," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pp. 1–8, IEEE, 2016.

[24] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[25] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[26] M. Aubry, U. Schlickewei, and D. Cremers, "The wave kernel signature: A quantum mechanical approach to shape analysis," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pp. 1626–1633, IEEE, 2011.

[27] J. Sun, M. Ovsjanikov, and L. Guibas, "A concise and provably informative multi-scale signature based on heat diffusion," in *Computer graphics forum*, vol. 28, pp. 1383–1392, Wiley Online Library, 2009.

[28] 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, pp. 1150–1157, Ieee, 1999.

[29] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 3384–3391, IEEE, 2008.

[30] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 3212–3217, IEEE, 2009.

[31] H. Ling and D. W. Jacobs, "Shape classification using the inner-distance," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 2, pp. 286–299, 2007.

[32] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung, "On visual similarity based 3d model retrieval," in *Computer graphics forum*, vol. 22, pp. 223–232, Wiley Online Library, 2003.

[33] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3d point clouds with strongly varying density.," *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 3, no. 3, 2016.

[34] J. A. Montoya-Zegarra, J. D. Wegner, L. Ladickỳ, and K. Schindler, "Mind the gap: modeling local and global context in (road) networks," in *German Conference on Pattern Recognition*, pp. 212–223, Springer, 2014.

[35] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, "On the segmentation of 3d lidar point clouds," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2798–2805, IEEE, 2011.

[36] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, "Semantic3d. net: A new large-scale point cloud classification benchmark," *arXiv preprint arXiv:1704.03847*, 2017.

[37] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *European conference on computer vision*, pp. 44–57, Springer, 2008.

[38] C. Cadena and J. Košecká, "Semantic segmentation with heterogeneous sensor coverages," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2639–2645, IEEE, 2014.

[39] D. Munoz, J. A. Bagnell, and M. Hebert, "Co-inference for multi-modal scene analysis," in *European Conference on Computer Vision*, pp. 668–681, Springer, 2012.

[40] C. Zhang, L. Wang, and R. Yang, "Semantic segmentation of urban scenes using dense depth maps," in *European Conference on Computer Vision*, pp. 708–721, Springer, 2010.

[41] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 689–696, 2011.

[42] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," in *Advances in neural information processing systems*, pp. 2222–2230, 2012.

[43] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[44] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *arXiv preprint arXiv:1708.02002*, 2017.

[45] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," *arXiv preprint arXiv:1711.06396*, 2017.

[46] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 1355–1361, IEEE, 2017.

[47] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas, "Fpnn: Field probing neural networks for 3d data," in *Advances in Neural Information Processing Systems*, pp. 307–315, 2016.

[48] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, pp. 945–953, 2015.

[49] A. Kanezaki, "Rotationnet: Learning object classification using unsupervised viewpoint estimation.," *CoRR*, 2016.

[50] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander, "Joint 3d proposal generation and object detection from view aggregation," *arXiv preprint arXiv:1712.02294*, 2017.

[51] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *IEEE CVPR*, vol. 1, p. 3, 2017.

[52] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron,

et al., "Theano: Deep learning on gpus with python," in *NIPS 2011, BigLearning Workshop, Granada, Spain*, vol. 3, Citeseer, 2011.

[53] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[54] F. Chollet *et al.*, "Keras." `https://github.com/keras-team/keras`, 2015.

[55] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, pp. 1025–1035, 2017.

[56] H. Wildenauer and A. Hanbury, "Robust camera self-calibration from monocular images of manhattan worlds," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 2831–2838, IEEE, 2012.

[57] J. Deutscher, M. Isard, and J. MacCormick, "Automatic camera calibration from a single manhattan image," in *European Conference on Computer Vision*, pp. 175–188, Springer, 2002.

[58] X. Yu, N. Jiang, L.-F. Cheong, H. W. Leong, and X. Yan, "Automatic camera calibration of broadcast tennis video with applications to 3d virtual content insertion and ball detection and tracking," *Computer Vision and Image Understanding*, vol. 113, no. 5, pp. 643–652, 2009.

[59] L. Karazija, "Automatic inference of cross-modal connection topologies for X-CNNs," Master's thesis, University of Cambridge, 2017.

[60] A. Mousavian, D. Anguelov, J. Flynn, and J. Košecká, "3d bounding box estimation using deep learning and geometry," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 5632–5640, IEEE, 2017.

[61] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," *arXiv preprint arXiv:1711.08488*, 2017.

[62] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," *arXiv preprint arXiv:1705.07115*, 2017.

[63] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," in *CVPR*, 2018.

[64] P. Veličković, "Tikz/convolutional cross-connection at master petarv-/tikz." `https://github.com/PetarV-/TikZ/tree/master/Convolutional%20cross-connection`.

[65] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *arXiv:1802.02611*, 2018.

[66] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv preprint arXiv:1606.00915*, 2016.

[67] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 447–456, 2015.

[68] R. L. Cook, "Stochastic sampling in computer graphics," *ACM Transactions on Graphics (TOG)*, vol. 5, no. 1, pp. 51–72, 1986.

[69] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals for accurate object class detection," in *Advances in Neural Information Processing Systems*, pp. 424–432, 2015.

[70] 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*, pp. 2147–2156, 2016.

[71] C. Vogel, S. Roth, and K. Schindler, "An evaluation of data costs for optical flow," in *German Conference on Pattern Recognition*, pp. 343–353, Springer, 2013.

[72] K. Yamaguchi, D. McAllester, and R. Urtasun, "Efficient joint segmentation, occlusion labeling, stereo and flow estimation," in *European Conference on Computer Vision*, pp. 756–771, Springer, 2014.

[73] E. Ohn-Bar and M. M. Trivedi, "Fast and robust object detection using visual subcategories," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pp. 179–184, IEEE, 2014.

[74] X. Du, M. H. Ang Jr, S. Karaman, and D. Rus, "A general pipeline for 3d detection of vehicles," *arXiv preprint arXiv:1803.00387*, 2018.

[75] J. Beltran, C. Guindel, F. M. Moreno, D. Cruzado, F. Garcia, and A. de la Escalera, "Birdnet: a 3d object detection framework from lidar information," *arXiv preprint arXiv:1805.01195*, 2018.

[76] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

Appendices

Appendix A

ModelNet40 Classifier Implementation

The following is executable, not pseudocode. Imports and data feeding have been stripped.

```
graph_input = Input(shape=[N_PTS, K], dtype='int32')
pts_input = Input(shape=[N_PTS, D], dtype='float32')

# Initial feature extraction
# If you want to include point features, handle it here.
paired = PairingLayer(K)([pts_input, graph_input])
x = paired
x = MLP([32, 64, 128])(x)
x = MaxAggLayer(norm=True)(x)

for _ in range(3):
  prev_x = x
  x = PairingLayer(K)([x, graph_input])
  x = MLP([64, 64, 128+64])(x)
  x = MaxAggLayer(norm=False)(x)
  x = AsymAdd()([prev_x, x])
  x = LayerNorm()(x)

x = Dense(1024)(x)
x = MaxAggLayer(norm=True)(x)
x = Dropout(0.5)(LeakyReLU(0.125)(Dense(256)(x)))
```

```
x = Dropout(0.5)(LeakyReLU(0.125)(Dense(128)(x)))
x = Dense(N_CLASSES, activation='softmax')(x)

model = Model([graph_input, pts_input], x)
```