An Application of Out-of-Distribution Detection for Two-Stage Object Detection Networks

by

Taylor Denouden

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Abstract

Recently, much research has been published for detecting when a classification neural network is presented with data that does not fit into one of the class labels the network learned at train time. These so-called out-of-distribution (OOD) detection techniques hold promise for improving safety in systems where unusual or novel inputs may results in errors that endanger human lives. Autonomous vehicles could specifically benefit from the use of these techniques if they could be adapted to detect and localize unusual objects in a driving environment, allowing for such objects to be treated with a high degree of caution.

This thesis explores the modification of a selection of existing OOD detection methods from the image classification literature for use in a two-stage object detection network. It is found that the task of detecting objects as being OOD is difficult to define for object detection networks that include a high-variance background class label, but that these methods can instead be adapted for detecting when background regions are incorrectly classified as foreground and when foreground objects of interest are incorrectly classified as background in the final layers of the network. It is found that some methods provide a slight improvement over the baseline method that uses softmax confidence scores for detecting these kinds of errors.

Acknowledgements

I would like to thank Krzysztof, Rick, Vahdat, Buu, Sachin, Matt and all my other colleagues for the useful insights, suggestions, and encouragement they offered to me throughout this program.

Dedication

This is dedicated to my family and friends. Thanks for all the supportive words, occasional distractions, and spaghetti.

Table of Contents

List of Tables

List of Figures

Abbreviations

AP	average precision , 22
AUPR	area under the precision-recall curve ix,
AUROC	area under the receiver-operating characteristic curve , 23
AV	autonomous vehicle , 2
BG	background , 2
CIFAR	Canadian Institute for Advanced Research
CNN	convolutional neural network , 14
Conv	convolutional
FC	fully connected
FG	foreground xi,
FN	false negative ix,
FP	false positive ix,
FPR	false positive rate , 23
GPS	Global Positioning System
GPU	graphics processing unit , 1
ID	in-distribution ix,

IDD India Driving Dataset ix,

IoU intersection over union , 21

KITTI Karlsruhe Institute of Technology and Toyota Technological Institute

LiDAR light distance and ranging

LSUN Large-Scale Scene Understanding

mAP mean average precision , 23

MC Monte Carlo , 38

ML machine learning , 1

MNIST Modified National Institute of Standards and Technology

MSE mean squared error , 11

NMS non-maximum suppression , 29

NN neural network , 1

OBD On-board Diagnostics , 27

ODIN out-of-distribution detector for neural networks

OOD out-of-distribution iii,

OODL optimal out-of-distribution discernment layer x,

OSVM one-class support vector machine , 36

R-CNN region-based convolutional neural network

ReLU rectified linear unit , 8

ResNet Residual Network

RGB red-green-blue

ROI region of interest , 30

RPN	region proposal network xii,
SVM	support vector machine , 19
TN	true negative ix,
TP	true positive ix,
TPR	true positive rate , 23
VGG	Visual Geometry Group
VOC	visual object classes
YOLO	You-Only-Look-Once

Chapter 1

Introduction

Neural networks (NNs) have been successfully applied to increasingly difficult tasks over the past decade. This is largely due to the rising power and availability of graphics processing units (GPUs) and the increased availability of datasets necessary to implement these models. The impressive ability of some NNs to perform tasks with superhuman accuracy and speed has also led to their adoption and use in a few safety-critical environments. In these environments, the poor performance of the model could cause serious damage, human injury, or death. Unfortunately, the complex nature of these models makes it difficult or impossible to verify with traditional methods that they will perform the correct action given some input.

During development, NNs models are typically tested and assured as being reasonably accurate on a set of input samples which are analogous to the data used to train it. There is, however, serious risk when the model encounters data that is different from the limited domain of expected inputs: a type of data often referred to as being OOD. Research has shown that when machine learning (ML) models like NNs encounter OOD samples as input, they often make an output prediction with high-confidence, despite being poorly equipped to handle such data [16, 30]. This is problematic because it may cause a complex system that relies on these predictions to perform an inappropriate and dangerous action.

An abundance of recent ML literature aims to rectify this problem by detecting when model input data falls far outside of the data-generating distribution of previously encountered samples where performance has been empirically measured as being good. This body of literature is collectively known as OOD detection. The main idea behind these methods is that when an OOD input is given to the ML model, the OOD detection system can detect it as being a sample in which the model has a high risk of making a poor prediction.

It would then, for example, be possible to let a human take over from an automated system that is insufficiently trained to handle that input.

One safety-critical application where ML systems are employed is in autonomous vehicles (AVs). This thesis focuses on one AV subsystem in particular: an environmental perception system called object detection. Object detection systems are responsible for taking input sensor data feeds, such as camera images, and identifying the locations and labels of objects of interest in that scene. The objects of interest are defined during model training, but typical examples in AVs are cars, pedestrians, cyclists, and other road users. By identifying these objects, other AV subsystems can use the information to inform driving behaviour and actions.

In their usual implementation, object detection systems sometimes incorrectly identify the presence of objects of interest when none are there. These FP indications of objects can be unsafe if they cause the vehicle systems to take evasive actions to avoid non-existent objects; such behaviour would be surprising and confusing for other road users near the AV. Examples of these kinds of errors are shown in Figure 1.1. Alternatively, the object detection network may fail to detect an object entirely. These FN errors are likely even more severe than FP errors since these objects would be ignored entirely by the system. Examples of FN errors from an AV object detector are shown in Figure 1.2.

This thesis explores the extent to which FP and FN errors can be reduced or eliminated using OOD detection methods. In image classification, OOD detection methods are typically tested using samples from a class the model has not learned. Because object detectors learn a background (BG) class, it is not straightforward to introduce new classes in this way for testing in this way. However, it is presumed that in the space of a data generating distribution, FP and FN objects should lie far from the set of embedded TP and TN samples such that OOD detection methods can be used to detect them. Various methods are tested to validate this hypothesis and to establish a baseline experimental procedure for comparing OOD detection algorithms for FP and FN detection in the 2D object detection domain.

Experiments are performed separately on two AV object detection datasets: KITTI [11] and the IDD [41]. One dataset at a time, the predictions from a two-stage object detector are obtained and labelled as being TP, FP, TN, or FN cases. The performance of each OOD detection method for the task of FP and FN detection are first reported for the KITTI dataset. KITTI was used for determining hyperparameters for the various OOD detection methods, and so these results are expected to have higher performance than would each method in a deployed driving environment. To address this issue, the FP and FN detection experiments are repeated on the IDD dataset. The IDD was not used

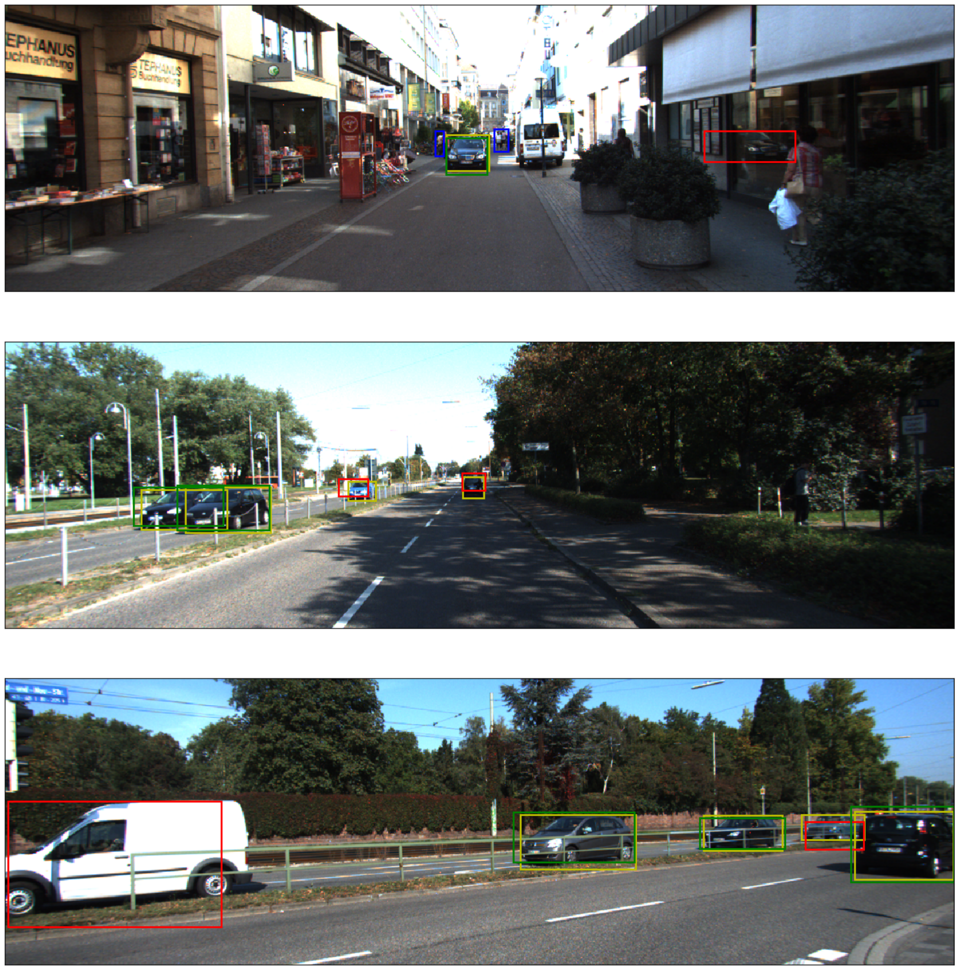

Figure 1.1: Example of FP car detection. Red boxes are FP car detections, yellow boxes are the ground truth annotations, green boxes are TP car detections, and blue boxes are unlabelled image regions of the “Don’t Care” class in KITTI. In these samples, the object detection network has suggested that a car is present in an area where one does not exist. Don’t Care regions are shown as they occasionally contain Cars and other FG objects that are unlabelled.

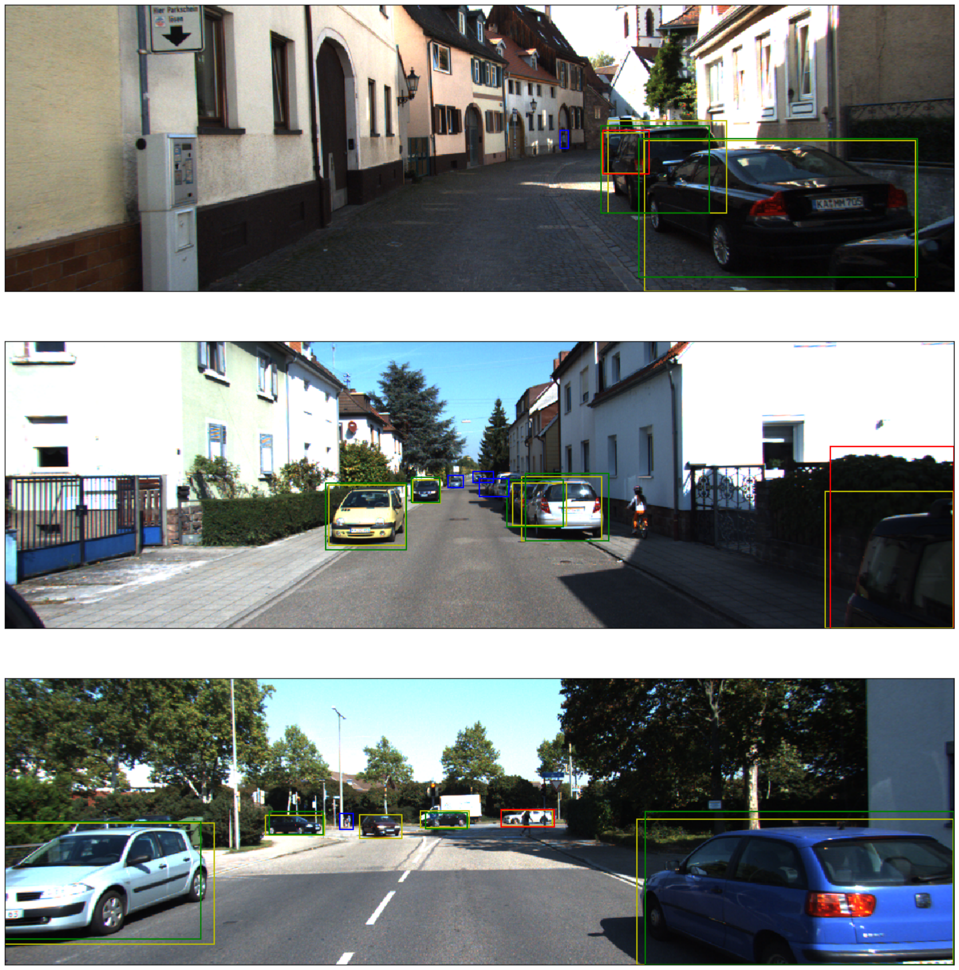

Figure 1.2: Example of FN car detections. Red boxes are FN car detections, yellow boxes are the ground truth annotations, green boxes are TP car detections, and blue boxes are unlabelled image regions of the "Don't Care" class in KITTI. In these samples, the object detection network initially proposed that the red boxes were objects, but discarded them in the later layers by classifying them as background. Any cars that were detected by the model in the Don't Care regions are not shown.

for training or hyperparameter optimization and provides unbiased performance results for each OOD detection method.

The contributions of this thesis are to:

- Define what it means to be OOD and formulate the tasks the OOD detection methods are useful for in the object detection application space.
- Show how OOD detection methods from the image classification literature can be modified to operate within a two-stage object detection network to detect when FP and FN errors have occurred.
- Establish a framework for generating datasets that can be used to compare the efficacy of different OOD detection methods for FP and FN detection[1].

Chapter 2 begins with a review of neural network and machine learning basics important for understanding the OOD detection methods used, as well as a high-level review of object detection and OOD detection. Chapter 3 formulates the problem this work solves, describes evaluation metrics used, details specific OOD detection techniques from the image classification literature as well as the modifications required to use them in the object detection network, and details the production of datasets for the OOD detection experiments. The results and discussion of experimental results are presented in Chapter 4. Finally, Chapter 5 concludes the findings of this work and discusses possible improvements to the methods used and research directions for further exploration.

[1]Implementation code is available at https://github.com/tayden/simple-faster-rcnn-pytorch

Chapter 2

Background

This chapter summarizes the concepts required for understanding the experimental setup and results that follow. It begins with an overview of basic neural network concepts, followed by an overview of methods that are more specific to image processing using neural networks. Following this, object detection networks are summarized in brief, the out-of-distribution detection task is described, and finally, metrics used for both object detection and OOD detection are summarized.

2.1 Machine Learning

Machine learning is a subset of artificial intelligence that includes mathematical techniques for estimating functions that will map some input data to a desired output. These mathematical techniques are typically used when the function $f(x')$ is difficult to explicitly specify. They can be classified into two broad categories, based on the kind of data used to estimate the function parameters. *Unsupervised* methods are only provided with a dataset of possible inputs x' and are often used to find structure or patterns that are intrinsic to the dataset [29]. In contrast, in *supervised* models, parameters are estimated using a dataset with examples of both the inputs x' and corresponding desired outputs y' of f [29]. Supervised learning, which is the focus of this thesis, can be broken down by task into two subtypes: classification or regression.

2.1.1 Classification

Classification is the task of mapping inputs x' to outputs $y' \in \{1, ..., C\}$ where C is the number of categorical classes. Classification tasks require that y is a categorical or nominal value in a finite set of possible classes [29]. If $C = 2$, the task is called a binary classification problem, while for any $C > 2$, the task is called multi-class classification. If it is the case that labels y' are not mutually exclusive and the model should potentially output more than one category for each input, the task is called a multi-label classification problem.

It is most common for a classification model to provide a single discrete categorical output for each input and it should be assumed this is the case unless otherwise specified [29]. Examples of classification problems include predicting the name of an object most prominently featured in an image or predicting if the content of an email is spam or a fishing attempt.

2.1.2 Regression

Regression is the task of predicting a function that can map an input x' to a continuous output $y' \in \mathbb{R}$ [29]. Examples of regression tasks include predicting the selling price of a home given details about its size, etc., and predicting the location of a robot in a room given the information from various on-board sensors.

2.2 Feedforward Neural Networks

Feedforward neural networks are a kind of machine learning model that approximate a function f through a series of intermediate computational layers. These layers are parameterized by values Θ such that desired outputs y' are approximated by the function $\hat{y} = f(x'; \Theta)$. Feedforward models are some of the simplest architectures of deep neural networks in that data x' passes through one or more intermediate layers of the model to the output layer with no cyclical connections between the output and any of the intermediate layers [13].

These kinds of models are referred to as networks because of how they are composed of a series of relatively simple functions. The network f is typically composed of simpler layer functions f^1, f^2, and f^3 as follows: $f(x') = f^3(f^2(f^1(x')))$. The depth of the network refers to how many of these layer functions are composed in this way. The outermost layer (f^3 in the example) is referred to as the output layer while functions f^1 and f^2 are referred

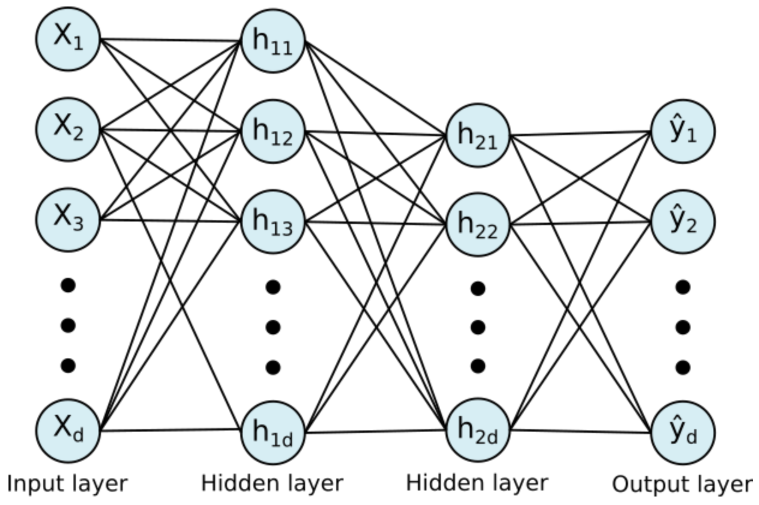

Figure 2.1: Example of a feedforward neural network.

to as hidden layers [13]. The desired output of the hidden layers is not explicitly specified by the training dataset. The structure of a simple feedforward neural network can be seen in Figure 2.1

The following sections describe in more detail some aspects of feedforward neural networks, how they are trained (i.e. the parameters Θ are optimized), and how generalization capability of f can be encouraged such that the model remains performant when presented with previously unseen, yet analogous, input data.

2.2.1 Activation Functions

Activation functions refer to a set of functions used for computing the hidden layer output values [13]. These functions take a linear combination of the input data via layer weights W, add a bias term b, and then output values of a non-linear transformation of that computation. That is, a typical activation neuron has the form $h = f_{act}(W^T x' + b)$. There are several commonly used activation functions f_{act} with different properties based on the range of outputs, the magnitude of gradients at different points, and other characteristics.

It is possible to use linear activation functions, however, these only allow function approximation by simple linear regression. Non-linear activation functions allow for learning more complex functions and allow for neural networks to behave as universal function approximators [13]. Some commonly used activation functions are the rectified linear unit (ReLU) and Softmax functions.

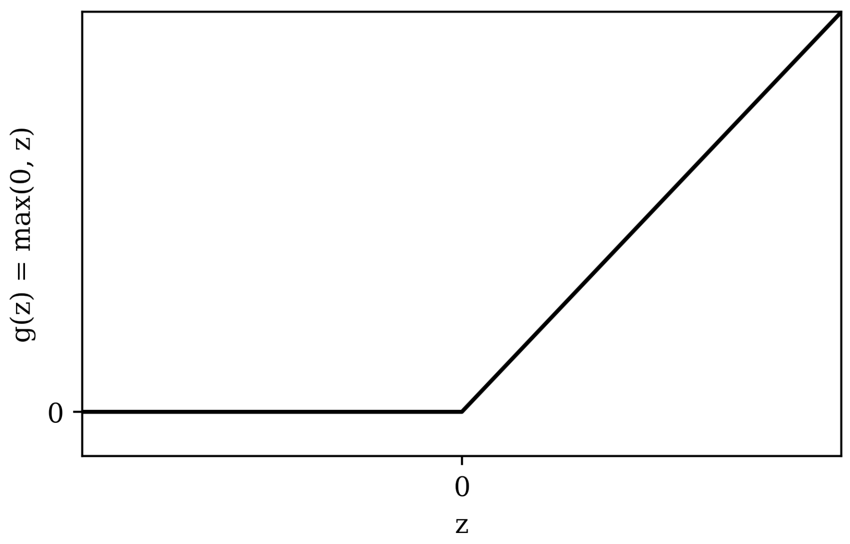

Figure 2.2: Output of the ReLU activation function.

ReLU

The ReLU function is a piece-wise linear activation function, which retains many of the properties of linear functions that make them easy to optimize with gradient-based methods [13]. ReLU is defined by Equation 2.1 and a visualization of the output is shown in Figure 2.2. ReLU activation functions are the default recommended non-linear activation function for use within neural networks, although others also exist [13]. While the gradient of the ReLU function is undefined at 0, software implementations typically define it heuristically as either the left or right derivative which allows the function to work well in practice [29].

$$\text{ReLU}(z) = max(0, z) \tag{2.1}$$

Softmax

The softmax activation function is often used as the final activation function in multi-class classification neural networks. It is used to normalize the outputs of a neural network to the interval $(0, 1)$ and change them such that the sum of all outputs is equal to 1. This is useful in practice when it is desirable to have outputs that can be interpreted as probability scores for each of the possible output classes. The formula for the softmax function is given by

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{2.2}$$

2.2.2 Loss Functions

The loss function of a neural network is a function that indicates how well the model is performing at the task it is supposed to accomplish. In supervised learning problems, this is often some kind of distance measure between the training data labels y' and the value predicted by the model $\hat{y} = f(x')$. These loss functions, which are interchangeably called objective functions, are meant to be used such that the value they output can be maximized or minimized using gradient-based optimization methods (covered in §2.2.3) [13]. The choice of loss function depends on the objective of the model. The loss functions used within this thesis are covered in more detail below.

Cross-Entropy Loss

Cross-entropy loss is used to measure the performance of a model with categorical output probabilities in the range $(0, 1)$ [9]. In the context of classification neural networks, the cross-entropy loss is applied to the output of a final softmax activation layer $\hat{y}$ and a one-hot representation of the ground truth label y'. That is, a one-hot vector of class label y' will be a vector of zeros of size C, the number of classes, with a value of 1 at the index corresponding to the ground-truth label of data x'. Given these representations, cross-entropy is then defined as

$$\mathbb{H}(\hat{y}, y') = -\sum_{k}^{C} y'_k \log \hat{y}_k \tag{2.3}$$

Cross-entropy loss is minimized when the predicted distribution $\hat{y}$ matches the distribution of the ground truth, and increases as the predicted class diverges from the true label [9]. When there are only two possible ground truth classes, it is more common to use the simpler binary cross-entropy formula, which encodes the label y' as a single 0 or 1, rather than a one-hot vector

$$\mathbb{H}_{binary}(\hat{y}, y') = -y' \log(\hat{y}) + (1 - y') \log(1 - \hat{y}) \tag{2.4}$$

L2 Loss

L2 or mean squared error (MSE) loss is used to minimize the difference between the ground truth and model-predicted value. The L2 loss function is defined by Equation 2.5. L2 loss is used in regression models and increases with the square of the distance between predicted output ground truth value. As a result L2 loss is sensitive to outliers in the data that may skew the model due to their effect on drastically increasing the loss value [36].

$$L2(\hat{y}, y') = \sum_{i=0}^{n} (y'_i - \hat{y}_i)^2 \tag{2.5}$$

L1 Loss

L1 loss is used in similar situations as mean squared error loss. L1 loss has an advantage over L2 in that the effect of outliers is reduced, which may result in a better model when there is noise present in the data [36]. It is defined as

$$L1(\hat{y}, y') = \sum_{i=0}^{n} |y'_i - \hat{y}_i| \tag{2.6}$$

Smooth L1 Loss

Smooth L1 loss is a combination of L1 and L2 loss that reduces the effect of outliers but also reduces the magnitude of the loss when the difference between the true and predicted labels is small (see Equation 2.7) [12]. This prevents the predicted output of the model oscillating around a local minima because of the magnitude of the gradients being too large. A plot showing the differences in the outputs of smooth L1, L1, and L2 losses are shown in Figure 2.3.

$$L1_{smooth}(\hat{y}_i, y'_i) = \begin{cases} \frac{1}{2}(y'_i - \hat{y}_i)^2, & \text{if } |y'_i - \hat{y}_i| \leq 1 \\ |y'_i - \hat{y}_i| - 0.5, & \text{otherwise} \end{cases} \tag{2.7}$$

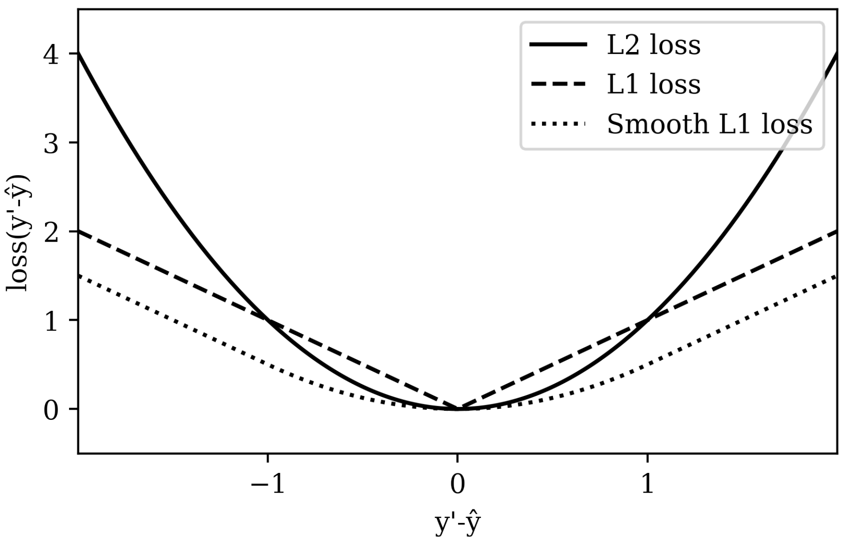

Figure 2.3: Some regression loss functions.

2.2.3 Gradient Descent

Gradient descent is a method for optimizing the parameters Θ of model $f(x')$. It specifically refers to the problem of minimizing the output of a loss function $L(y', f(x'))$. That is, an optimized model should perform well at the task of predicting y' given $\hat{y} = f(x')$. To do this optimization, the loss is calculated for the current parameters (*i.e*, weights and biases) of the model then the derivative $f'(x') w.r.t x'$ is calculated. The derivative of the model describes how a small change in the model parameters will improve the predicted output of the model, $\hat{y}$. Because we are aiming to minimize the loss function, we update the model parameters with $f(x') = f(x') - \epsilon f'(x')$, where ϵ is a hyper-parameter called the learning rate. The learning rate is a typically small fraction that optimizes the model in smaller steps such that loss minima are not over-shot by the update step. This prevents the learning algorithm from oscillating around a local minimum rather than decreasing the error more steadily [13].

Should $f'(x')$ equal zero, the model is said to have converged. This point on the manifold is not guaranteed to be the global minimum point for the model but is more often a local minimum or saddle point in the manifold. The performance of the model on a separate test dataset using this set of parameters is typically evaluated and reported.

2.2.4 Regularization

Regularization is a set of methods for improving the generalization capabilities of a machine learning model without drastically increasing the training error [13]. Without regularization, gradient descent optimizations will usually perform very well for the training data but poorly on any new data that was unseen during training, despite containing the same classes of image content, text, etc. When this undesirable scenario occurs, the model is said to have over-fit the training data. Two kinds of regularization used within this thesis are briefly described below: weight decay and dropout.

Weight decay

Weight decay is the penalization of large model weights as a means to prevent over-fitting the model to the training dataset. L2 regularization is one common method for achieving this and does so by adding an error term to the loss function. The modified optimization objective for a model with L2 regularization is defined

$$J(\boldsymbol{w}) = \mathrm{Loss}(\boldsymbol{w}) + \boldsymbol{w}^T \boldsymbol{w} \tag{2.8}$$

Alternatives to this kind of weight regularization also exist, such as L1 regularization, which instead penalizes using the absolute magnitude of the weight parameters:

$$J(\boldsymbol{w}) = \mathrm{Loss}(\boldsymbol{w}) + |\boldsymbol{w}| \tag{2.9}$$

Dropout

Dropout is a method that effectively allows training an ensemble of models simultaneously without the expensive computation or memory overhead of doing so the traditional way (*i.e.* by optimizing multiple models initialized with different random parameters) [13]. Dropout does this by randomly turning off some fraction of specified model layer outputs for a batch of data used in a single training pass of the model. The computation of the loss function and the gradient descent update operation are performed using the network outputs that were computed using the subset of model parameters. This random nullification of layer neurons (*i.e.* a single weighted parameter value in a layer) encourages the model to learn an internal representation with a level of redundancy. This is useful for learning more robust features from the dataset and often improves the generalization capability of the model.

Dropout is most often implemented as a layer in a feedforward neural network that sets some fraction of the output of the previous layer to zero before continuing the forward pass of the network. During evaluation time, these layers are no longer needed and no sampling of the input is performed. Instead, the output of the dropout layer is normalized using the same fraction used to randomly sample neurons at train time.

2.3 Convolutional Neural networks

Convolutional neural networks (CNNs) are a specialized kind of model for processing data with grid-like spatial correlations between the input features [13]. An example of data with this property is 2-dimensional image data. CNNs form the basis for many state-of-art image classification neural networks such as DenseNet [18] and Wide Residual Networks [46]. The difference between a simple feedforward neural network like that shown in Figure 2.1 and a CNN is that a CNN makes use of at least one convolutional layer. These layers are a modification of the more simple linear vector multiplication layers (called fully-connected or dense layers) and account for relationships between adjacent data features.

2.3.1 Convolution operation

The convolution operation is most easily understood as a mathematical operation between some ordered input data matrix and a kernel matrix which slides over the input. The kernel matrix is used to compute a matrix dot product with the corresponding input data covered by the kernel function. The output of this operation is a single value in a convolved matrix output, commonly called the feature map. This definition is visualized and clarified by Figure 2.4.

More formally, for multi-dimensional arrays of parameters as in machine learning, convolution is defined

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.10}$$

Where S is the output convolved matrix, i and j are indices in the matrix, I is an input multidimensional array (such as an image), and K is the kernel matrix [13]. Here, K is assumed to be zero everywhere except for a finite set of points for which we store the values. In the context of machine learning, these values are shared weight parameters learned via gradient descent.

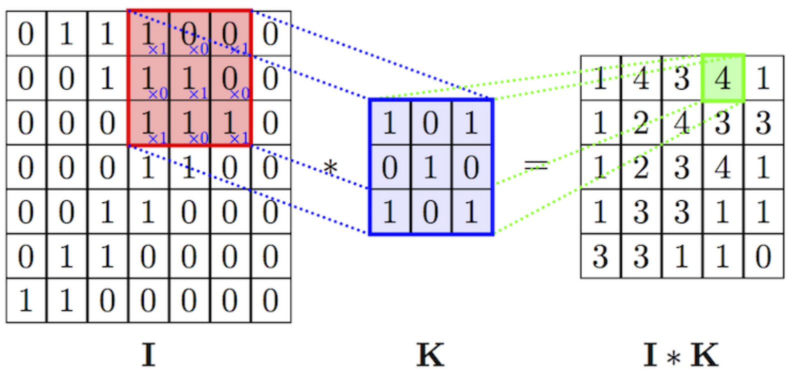

Figure 2.4: The convolution operation. The dot product of kernel K is computed with image region I to produce a single value in the feature map output [42].

The shape of the non-zero elements of K is described as the size of the kernel, with square 3x3 or 5x5 matrices being common in machine learning literature. The step size the kernel makes as it iterates over the input data is referred to as the stride. Finally, the input matrix I is sometimes modified before the convolution by padding the perimeter with zeros or some other value such that the output of the convolution operation is a matrix with some desired shape. Manipulating the kernel size, stride, and padding all affect the shape of the output feature map and also define how the spatially correlated input features are mapped to the hidden representation.

Finally, the convolution operation is usually defined as a layer in a neural network. It is common to have multiple convolutional kernels of the same size and stride but with different randomly initialized parameters in a single convolutional layer of a NN. An individual kernel in one of these layers is called a filter. Each of the filters or kernels is convolved with the input data to produce a set of feature map outputs.

2.3.2 Pooling

Pooling is an operation used in CNNs to encourage the model to be invariant to small translations in the input data. To do this, a filter similar to the convolutional operation is used to summarize the values of a layer in the spatial neighbourhood. An example of a pooling layer includes max pooling. Max pooling slides a window over some input data and outputs the maximum value in that kernel neighbourhood as an output feature. Similar to convolutional layers, pooling layers also have a defined kernel size and stride. Average pooling is another common kind of pooling layer, with the output being computed as the statistical mean of the values in the kernel neighbourhood.

2.3.3 CNN as a Feature Extractor

CNNs are commonly used as feature extractors to construct a representation of the input image data for use in a different neural network. These feature extractors typically consist of the early convolutional layers of a pre-trained CNN classification network. The feature extractor network thus outputs the hidden feature representation of an input image after multiple convolutions and pooling layers.

The advantage of using these feature extractors is that one can select a pre-trained model with high performance that may be difficult to obtain in an end-to-end training fashion. The features output by these models typically represents the structural composition of image data in terms of simpler functional components, textures, and colours.

2.4 Object Detection

Object detection systems are an example of one complex image processing neural network which uses all of the previously defined concepts in this chapter. Object detection is the task of detecting, labelling, and regressing the location of an undefined number of objects in an image. In 2D object detection, the objects of interest (or FG classes) that the network is supposed to detect are defined at training time by providing examples of image scenes (and/or other sensor data) that contain those objects along with correct label information. The label data consist of bounding boxes that give the spatial location of each object in the image and a categorical identity describing what it is, e.g. a car, horse, or bicycle. There are other kinds of object detection networks that use 3D data rather than 2D, with the difference being that some form of depth information is provided as an input to the network. This thesis, however, focuses on 2D object detection. There are two overarching architectures for 2D object detection models, two-stage and single-stage models.

2.4.1 Two-stage object detectors

Two-stage object detectors are neural networks that, given some input image, first propose bounding boxes that correspond to a positive detection for an "object" and then refines these bounding box predictions to more tightly bound each object as well as classifies what is contained in that box [12, 34]. These two stages of the network are referred to as the RPN and the classification and bounding box regression stage. Faster R-CNN, a foundational two-stage approach is covered in detail in Section 3.2.1.

Two-stage object detectors are often the most accurate of the two architecture types in object detection challenges [11, 7, 31]. The primary drawback of these methods is the relatively slow inference time they have compared to single-stage object detectors.

2.4.2 Single-stage object detectors

The second kind of 2D object detection architecture is the single-stage detector. Single-stage detectors address the issue of long inference time required by many two-stage object detectors. They do this by proposing bounding box locations and classifying the contained object simultaneously [32, 26]. There are various approaches to accomplishing this [33, 26, 25]. One example of a single-stage detector, the YOLO object detection network, places a grid over the input image and predicts classification scores for any object contained in that box as well as a set of regressed bounding boxes that correspond to that contained object [32]. The most likely class and bounding box for objects are output by the network after filtering overlapping boxes and discarding those predicted as containing only background imagery.

2.5 Out-of-distribution Detection

OOD detection is the task of identifying when a data sample is different in some way from the data that were available during model training [16, 17, 24, 3]. The data that are available at train time is assumed to be representative of the in-distribution (ID) class; these samples are thought of as being "normal" in the colloquial sense of the term. Data that are OOD are then considered to be samples that are dissimilar in some way from the ID samples, or "abnormal". The degree to which an OOD sample is dissimilar from the ID set varies, but OOD samples are typically defined as being those that are so dissimilar from the ID data that the classification model does not have an appropriate label for them or as being so unusual that we would expect the model to make incorrect predictions for them [30].

As a concrete example, a model that is trained to label pictures of pet cats and dogs should not be expected to label pictures of horses or elephants correctly. A well-performing

OOD classification model would continue to label cat and dog images correctly and label these other kinds of animals as "OOD". Unfortunately, the concept of OOD is somewhat vague in the literature and it is not clear how a cat and dog classifier should be expected to label a picture of a cougar or lynx. It is possible to justify labelling these animals as either "cat" or as "OOD". Which of these two labels is correct is subjective and can only be defined based on the application context of the model.

Special techniques are needed to detect OOD instances because models are unable to effectively incorporate an OOD class label and samples into the standard model training procedure. This is because it is difficult or impossible to collect a meaningful sample of OOD inputs, as they theoretically occupy the entire space of possible inputs outside of the ID data. It is an intractable problem to produce a meaningful sample that is representative of this entire space.

Numerous methods have been proposed to avoid explicitly modelling the OOD class. The majority of these methods assume that the ID dataset has an unknown and unknowable underlying data generating distribution from which the ID samples are drawn. OOD detection methods model this ID data-generating distribution in some way and compare new samples to it. When a sample is determined to be unlikely as being generated by the ID data-generating distribution, it is labelled OOD. In this way, we can think of the task of OOD detection as a one-class classification problem where the space of ID samples is modelled by using the training set. Samples that poorly fit the model of inlier data are then classified as OOD. How we model the data generating distribution and do this one-class bounding on the ID set can be broadly divided into five categories [30]:

1. Probabilistic approaches: Methods where a probability density function is fitted to the ID data. Samples lying in the low probability regions are labelled OOD.

2. Distance-based approaches: The distance between a sample and the ID points is calculated. If the sample is not within a cluster of data or is significantly far from a known ID sample, it is labelled OOD.

3. Reconstruction-oriented approaches: A model is created to efficiently compress ID samples to a subspace and then decompress them with minimal loss of information. Under the assumption that OOD samples cannot be effectively reconstructed using this same model, the reconstruction loss from these models can be used to detect OOD samples.

4. Domain-based approaches: A boundary is fitted around the ID data in the original or a transformed space and data outside that boundary is labelled OOD. The boundary

may rely on only a few of the ID data points. An example is a one-class support vector machine (SVM) model.

5. Information-theoretic techniques: The information content of the dataset is determined using information-theoretic techniques. When a new sample is presented, the entropy of the ID dataset plus the new sample is computed. OOD samples are assumed to significantly increase the entropy of the dataset whereas a new ID sample would not.

Each of these methods can be designed to produce a single score to a given sample, called the "novelty score". Novelty scores may be a distance metric, probability value, or other descriptive statistics depending on the OOD detection method used. However, they always describe the similarity a new sample has with previously seen ID data. By setting a threshold on the novelty scores the OOD detector behaves as a binary classifier. That is, we can let p_{train} be the distribution of the training data and $p_{train}(x)$ be the likelihood that sample x is drawn from p_{train}. We estimate novelty score $\hat{p}_{train}(x) \propto p_{train}(x)$ such that $\hat{p}_{train}(x) < \delta$ denotes that x is OOD with δ being a tuneable novelty threshold.

Figure 2.5 visualizes how novelty score increases as samples become increasingly dissimilar from the training dataset. For a model that detects cars, we would expect cars from the training dataset to have the lowest novelty scores. Cars in the test dataset, which are likely to be similar to the training dataset but haven't been seen by the model yet, are expected to have comparatively higher novelty scores. Cars that are from a different dataset, perhaps being a make and model not in the training or test data, would have higher novelty scores than the test data. Very strange looking cars, like those that might appear as a parade float, would have a higher novelty score still and, finally, the highest novelty scores would be assigned to images that contain objects that are not cars at all. Adjusting the threshold of novelty in the OOD detector changes the threshold of abnormality at which a sample is classified as ID or OOD.

Identifying OOD samples is useful for many applications. Some examples include: medical applications such as those that employ probability-based approaches to detect cases of emphysema in chest x-rays [2]; banks and auditing firms extensively use OOD detection methods to determine whether or not credit card transactions or applications are illegitimate [38]; and finally, engineers have used OOD detection to determine if offshore drilling platforms are damaged [39] and to check for abnormal sensor measurements in civil water distribution networks [28].

Finally, there are other related terms in the literature for the task of OOD detection. Namely, OOD detection is often interchangeably called "novelty", "anomaly", and "outlier"

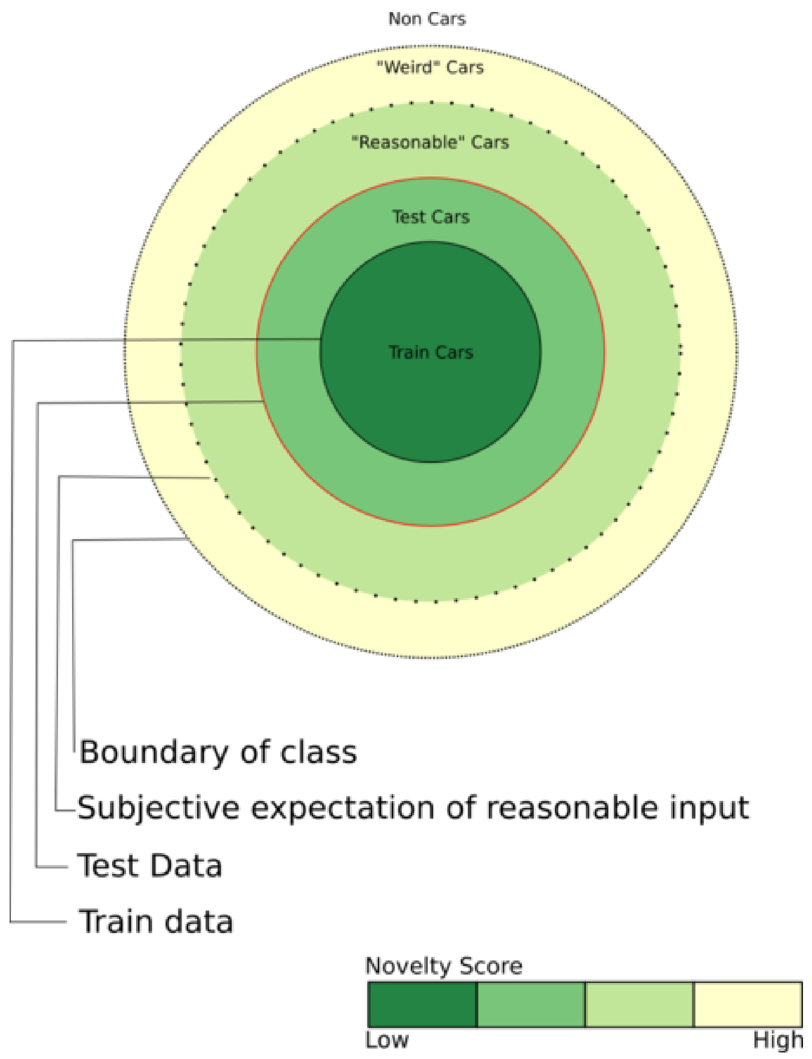

Figure 2.5: Different expected novelty scores relative to a set of ID car images. Data in the train set are usually best modelled by the OOD detector and therefore are expected to have the lowest novelty score. A good OOD detector should show ID test data and anything that reasonably matches the ID data to also have low novelty scores. Thresholding the novelty score determines how different a car is required to be before being classified as OOD.

detection. There is no universally agreed-upon definition for each of these tasks and the use of each term is largely related to the domain in which each technique is applied [30]. Novelty detection is sometimes distinguished as the task of detecting when something new and different from the existing data appears, while anomaly and outlier detection focus on the detection of abnormal or low-probability events. It is not particularly useful to distinguish these terms for this work, so all methods are collectively referred to as OOD detection.

2.6 Performance Metrics

This section details several performance metrics used in the OOD detection and Object Detection literature.

2.6.1 Intersection over union

Intersection over union (IoU), in the context of object detection, is a metric that describes how well a predicted bounding box represents the ground-truth bounding box. The name of this statistic is self-descriptive and it is computed as the area of set intersection between the predicted and ground-truth rectangles divided by the area formed by the set union of these two rectangles. An IoU value of 1 would indicate that the two bounding boxes are identical, while predicted bounding boxes that are larger than, smaller than, or non-overlapping with the ground truth would have a value less than 1.

2.6.2 Precision and Recall

Precision and recall are measures that describe the performance of a classification model. Each metric uses the TP, FP, and FN rates. The true/false designation of each of these values indicates whether or not a model accurately predicted the label or condition of a given sample. The positive/negative designation indicates the binary ground-truth label or condition for that sample [29]. Given these values, precision and recall are defined by Equations 2.11 and 2.12, respectively

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.11}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.12}$$

2.6.3 Average precision

Average precision (AP) is a summary of the precision/recall curve for the ranked output of an object detection network. The recall is the proportion of all positive samples ranked above a given rank, while precision is the proportion of samples above the rank which are from the positive class [8]. A curve is generated from this series of precision and recall scores calculated for each threshold that bisects the data on the confidence scores. AP summarizes the precision values for 11 different evenly-spaced recall values ([0, 0.1, 0.2, ..., 1] using Equation 2.13 [8]. The precision for each recall level r is interpolated by taking the maximum precision measured for the method where the corresponding recall is greater than or equal to r (Equation 2.14) [8].

$$\text{AP} = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_{\text{interp}}(r) \tag{2.13}$$

$$p_{\text{interp}}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}) \tag{2.14}$$

For the sake of clarification, in object detection precision is the measure of TPs above a ranked confidence threshold, and recall is the fraction of TPs above the ranked confidence threshold to the total number of ground truth objects for that class in the dataset. A prediction is considered to be correct if it has an IoU with a ground truth object bounding box label above a predefined threshold (e.g. 0.5) and additionally has labelled the predicted object correctly. Having high AP for one of the FG object classes means that the majority or all of the TP predictions have a higher confidence score than FP detections and that these cases would be separable using some threshold on the confidence score. The incorporation of recall in the calculation means that high AP implies, for a particular class, most or all of the ground truth objects were detected by the model. AP tells us how well the model is doing at detecting all objects and how easily we can filter out extraneous predictions by setting a threshold on the confidence score.

2.6.4 Mean average precision

Mean average precision (mAP) is a metric used to evaluate the performance of object detection networks with more than one FG class. It is simply the arithmetic mean of the AP scores calculated for each of the FG classes.

2.6.5 AUROC

The area under the receiver-operating characteristic curve (AUROC) metric is a threshold independent performance measure for evaluating detection networks that output some real-valued confidence or distance measure. AUROC is essentially a statistic that describes how well the positive and negative classes can be separated by the output confidence measure values without requiring that a threshold be determined beforehand. AUROC is calculated by sorting the output predictions by the confidence score and calculating the true positive rate (TPR) and false positive rate (FPR) for each possible threshold on that sorted data. The TPR is plotted against the FPR and the area under the curve that is formed by interpolation is taken to be the AUROC [16]. With an equal number of positive and negative samples, the baseline AUROC of a random classifier will be 0.50, while a classifier that always gives positive samples a higher confidence value than negative samples will have an AUROC value of 1.0.

It should be noted that the AUROC assumes that the count of positive and negative samples in the dataset are equal and that it may be a misleading statistic when this is not the case. In these experiments, this requirement that the number of positive and negative samples are equal does not hold. The performance of each method in terms of AUROC is given in these experiments to remain consistent with the reported statistics in other OOD detection literature.

2.6.6 AUPR

The AUPR is the second most common threshold independent performance measure for evaluation detection networks with positive and negative samples. The AUPR, like AUROC, also sidesteps the issue of having to determine an appropriate threshold for a classifier but is more appropriate for datasets with skewed numbers of positive and negative samples. The metric is calculated similarly as AUROC and AP, in that the data is sorted by the output confidence score and the precision and recall at each possible threshold are calculated using Equations 2.11 and 2.12. These values are plotted and AUPR is calculated as the area under this curve [16].

In OOD detection, AUPR is reported as two separate metrics, AUPR (in) and AUPR (out). The difference between these metrics is which of the classes is considered the positive class in the precision and recall calculations. For AUPR (in) the positive class is the ID samples. For AUPR (out) the OOD samples are considered positive.

2.6.7 FPR @ 95% TPR

The FPR @95% TPR metric describes the FPR on the AUROC curve when the TPR is at least 95% [24]. This metric describes the probability that the model will misclassify an OOD sample as ID when the ID samples are correctly classified at least 95% of the time.

2.6.8 Detection error

The detection error describes the probability of a misclassification on either an OOD or ID sample when ID samples are classified correctly at least 95% of the time. The detection error is calculated as [24]:

$$P_e = \frac{n_+}{n}(1 - \text{TPR}) + \frac{n_-}{n}\text{FPR} \tag{2.15}$$

where n_+, n_- are the number of positive and negative samples, respectively and n is the total number of samples. Here, negative samples are considered to be the OOD class.

2.6.9 Risk-Coverage plots

For the sake of safety, it can be better to abstain from classifying a data sample when it has been determined that any classified output is likely to be incorrect. The separated and unclassified samples can then instead be labelled by a human or treated with a high degree of caution to improve the safety of the system. However, abstaining from classification by the model has an associated cost and should be avoided if there is little or no safety benefit from doing so. In a system where it is possible to abstain from classifying samples, a threshold for doing so must be determined that takes into consideration the cost of abstaining vs. the improvement in system safety.

One way of measuring the safety benefit of abstaining from classification is by looking at the classification error of the remaining dataset the model classifies. Previous work proposes plotting this classification error (risk) over the fraction of the remaining data (coverage)

[43]. These plots take the form of a curve where points on the curve are determined by calculating the risk and coverage of the dataset over all possible novelty thresholds δ. If the model labels all samples where the novelty score of the sample is less than δ, then the risk is the fraction of incorrectly model labelled samples over the total number of model labelled samples. Coverage is the fraction of samples less than δ over the total number of samples in the dataset. Examples of Risk-Coverage plots are shown in Section 4 (Figures 4.6, 4.8, 4.2, 4.4).

Chapter 3

Experimental Setup

This chapter details the datasets used in this thesis, object detector model architecture and training, and OOD detection algorithms modified from the image classification literature. The experiments done using these components are also described. This includes the formulation of the problem that OOD detection methods can address in object detection networks; namely, the tasks of false positive and false negative error detection.

3.1 Datasets

Two different autonomous vehicle driving datasets were used in this work: the KITTI Vision Benchmark Suite [11] and the IDD [41]. KITTI data was used to train the object detection model and to fit OOD detection models as required. The KITTI validation set was used to determine the OOD detection method hyper-parameters. The IDD was used as a test set for OOD detection evaluation. KITTI and the IDD are described in more detail in the next sections.

3.1.1 KITTI Vision Benchmark Suite

The KITTI dataset has sensor measurements and human-generated labels for various AV related tasks. These measurements were captured by a sensor-equipped vehicle driving around the urban and rural areas of Karlsruhe, Germany. KITTI was designed with a public leaderboard for researchers to submit model predictions for a test set of samples. It ranks these submissions by scoring them against a withheld set of class and bounding box annotations. There are many AV related tasks researchers can compete in including optical flow, visual odometry, object detection, and object tracking. Sensor data available for download includes high-resolution colour and grayscale video camera stills, Velodyne LiDAR captures, and GPS localization data [11].

This work used the training set and labels provided by KITTI for 2D object detection. Single-camera RGB images provided by KITTI were used as model input. The label information provided by KITTI contained coordinates for the bounding boxes of each object in the image and the class of each object. Labelled object classes provided by KITTI are: “Car”, “Van”, “Truck”, “Pedestrian”, “Person_sitting”, “Cyclist”, “Tram”, “Misc”, and “Don’t Care”. The “Don’t Care” objects denote regions of the image that are unlabelled. The intention is that these regions are not used when training the model and that evaluation ignores predictions made in these areas of the image [11].

KITTI does not provide a set of validation data, so one was created by splitting the training set of size 7481 approximately in half. This split was done the same as in [5], such that the training and validation images did not come from the same video sequence. In total, this results in a set of 3260 training images and 3424 validation images.

3.1.2 India Driving dataset

The IDD is similar to the KITTI [11] dataset, but it was collected around Hyderabad, Bangalore, India [41]. IDD is provided with similar sensor data and labels as KITTI, but in a much more challenging environment with less structured use of the road and a larger diversity in the types of objects which appear.

Similar to KITTI, IDD also has public benchmarks for the task of semantic segmentation and instance segmentation. The task of instance segmentation requires 2D object detection labels, thus IDD provides object class and localization labels. There are more types of labelled objects in IDD than in KITTI; labels for the following objects are provided: bicycle, bus, traffic sign, train, motorcycle, car, traffic light, person, vehicle fallback, truck, autorickshaw, animal, caravan, rider, trailer. There are no unlabelled regions in the images [41].

Sensor data is provided in the form of stereo images from a front-facing camera, GPS points, and LiDAR and On-board Diagnostics (OBD) data. As with KITTI, only the single-camera front-facing image data is used in the experiments of this thesis.

Training, validation, and test data splits of size 31 569, 10 225, and 4 794 are provided in the IDD. Only the validation set was used for the experiments in this thesis to evaluate

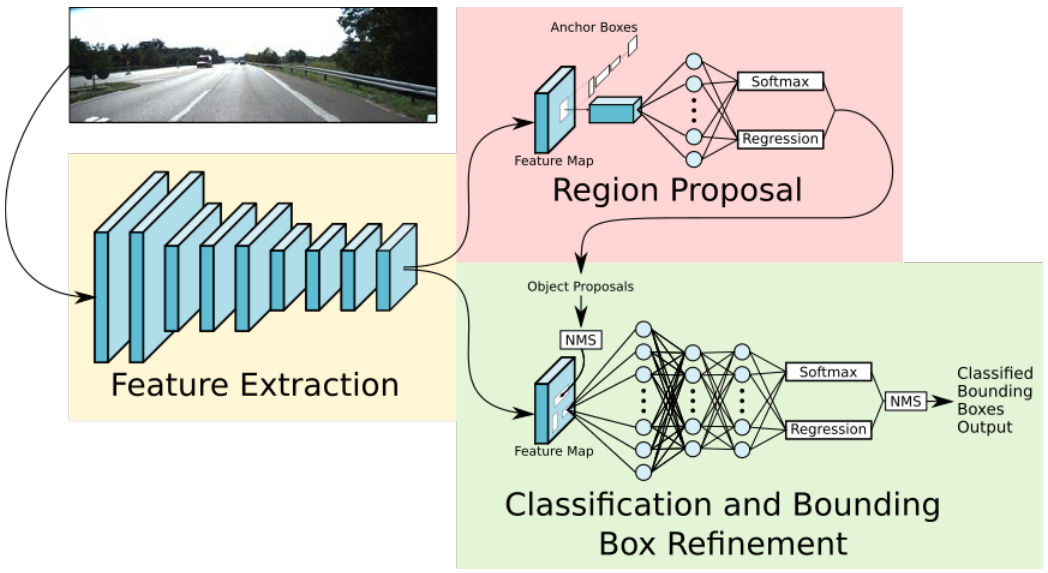

Figure 3.1: The Faster R-CNN architecture. A feature map extracted via a series of convolutional layers is shared between the region proposal network and the final classification and bounding box regression layers.

the proposed OOD detection methods. The validation split was used instead of the test split because label information is not provided for the test split.

3.2 Model

3.2.1 Faster R-CNN

Faster R-CNN is a foundational and popular two-stage object detection model. It leverages an end-to-end convolutional neural network to extract features from the input image and then shares this feature representation with a region proposal subnetwork and classification and regression subnetwork [34]. The feature extraction layers are typically taken from a pre-trained classification network. Popular sources for this feature extractor are the VGG16 [37] and Resnet-50 [14] networks, which have been trained for classification on the ImageNet dataset [35]. This feature sharing architecture can be seen in Figure 3.1.

The RPN consumes the feature representation of the image and a set of predefined sliding window boxes called the anchor boxes. The anchor boxes are the set of boxes that

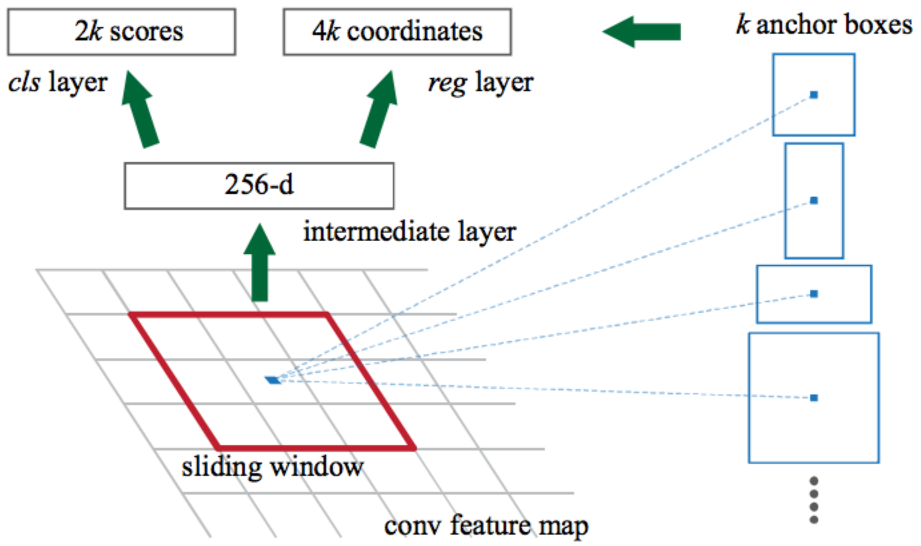

Figure 3.2: The RPN network in Faster R-CNN. The k anchor boxes slide over the feature map and output an objectness score (*cls*) and refined bounding box (*reg*) [34].

the network initially uses to slide at some stride length over the image and classify if, at that location, the image contains an object of interest. These anchor boxes are defined at multiple scales and aspect ratios to give a set of k initial guesses at the size and shape of an object. The output of the RPN is an objectness score and a regressed bounding box that slightly improves the shape of the anchor box to more accurately bound the object it potentially contains [34]. These anchor boxes are regressed and classified for all k boxes over all locations the boxes stride over throughout the image. The architecture of the RPN network is shown in Figure 3.2.

A fixed number of the RPN output boxes are passed to the successive layers of the neural network. The number of these output "region proposals" is typically fixed as being the t proposals with the highest confidence of containing a FG object (often called the "objectness" score). A filter called non-maximum suppression (NMS) next eliminates boxes that have too low of an objectness score using a fixed threshold, and also eliminates boxes that have high overlap with other predicted object boxes. In the case that multiple boxes have high overlap, the box with the highest objectness score is used while the others are removed before the next stage [34]. The amount the boxes overlap is determined using the IoU score, detailed in section 2.6.1. This IoU threshold at which two boxes are determined to overlap can be tuned to relax or restrict the rate at which overlapping boxes are eliminated.

Following NMS, the remaining object detection boxes are reshaped to the same size via a method referred to as region of interest (ROI) pooling. This is done so that these features can be fed to the next fully-connected layers, which perform final object classification and regression. The final layers of the network are typically composed of multiple fully-connected layers and often mirror the final layers from the original classification network which the feature extraction network was extracted from. The final classification layer uses the softmax function to output a normalized probability distribution of size $C + 1$, where C is the number of FG classes. One extra class is added for the classification of BG. A classification of BG means that the proposed bounding box contains no object of interest and should be ignored or eliminated. Because of the fixed number of proposals output by the RPN, it is common to eliminate many BG proposals in the final stages of the network. Another NMS function may be used at this stage to further eliminate overlapping boxes at this stage, often with a more restrictive IoU threshold.

Faster R-CNN is trained in an end-to-end fashion, meaning all the component sub-networks are trained together simultaneously. The final classification layer, final regression layer, and RPN network all have individual loss functions which are summed together and used to update the model parameters via back-propagation. The classification outputs use cross-entropy loss, and regression layer outputs use smooth L1 loss [34]. The RPN is also trained using a technique called hard-negative mining. That is, the RPN is provided with samples of anchors that contain labelled objects and non-objects, such that it learns FG and BG classes using explicit examples of each. Hard-negative mining uses BG data that is misclassified with high confidence to have the model learn to distinguish between samples it is particularly confused about. More details about RPN training are available in the original Faster R-CNN paper [34].

3.2.2 Training configuration

A Faster R-CNN network with a VGG16 [37] feature extraction backbone was trained using the KITTI train set. To quickly obtain an object detection model with suitably high performance, only a single class, cars, was used during training. That is, the model is designed to detect only cars in the image scenes.

To train the RPN, a maximum of 256 object targets were created in a 1:1 split of positive and negative targets for each image at train time. The positive object targets were created by taking the anchor boxes with an IoU greater than 0.7 with the ground truth bounding boxes. An equal number of anchors corresponding to BG areas were also sampled on an image-by-image basis. Areas that the model predicts as BG with low confidence were favoured here to force the network to learn this class based on difficult samples. Care was also taken not to sample areas that overlapped the KITTI "dontcare" areas that could contain unlabelled FG objects. The IoU threshold to be considered a background sample was required to be less than or equal to 0.3 with any of the ground truth FG object bounding boxes. If anchor boxes had an IoU with a FG bounding box between 0.3 and 0.7, they were ignored and not used as training examples for the RPN.

To train the final layers of the network, a maximum of 128 targets were generated in a 1:3 split of positive FG to negative BG samples per image. The precise number of each was determined by the available number of positive samples. The ground truth targets were generated such that bounding boxes were considered correct if the IoU with a ground truth bounding box was above 0.5 and considered negative if it was less than 0.5. This IoU threshold is higher than at the RPN stage to encourage better bounding boxes being output by the model. In the RPN, The required IoU is also lower to accommodate the lower IoU values that occur as a result of the initial anchor box proposals being less refined than the final bounding box outputs. As with the RPN training, areas that overlapped with "dontcare" regions were not used as targets for training the final layers of the network. Two drop-out layers were also used in the final fully-connected layers of the network with a drop rate of 50%.

NMS was used to eliminate overlapping object bounding boxes that had an IoU with one another greater than or equal to 0.7 after the region proposal network. In cases where boxes did overlap with IoU greater than 0.7, the box with the highest objectness score was kept while the others were discarded. A second NMS layer was used after the final classification and regression layers of the network with an IoU threshold of 0.3. Using two NMS layers reduced the number of object proposals before the final classification layers to improve the inference time of the network and subsequently eliminated most of the multi-detection per single object errors in the final stages of the network.

The model was trained for 14 epochs, with a batch size of 1. The number of epochs was chosen based on the number which was required to achieve good performance training on the Pascal VOC object detection dataset [8, 6]. A batch size of 1 was chosen to simplify implementation by using existing Faster R-CNN code from [6]. The initial learning rate of the model was set to 0.001 and was reduced to 0.0001 at the end of the ninth training epoch.

The performance of the network was measured in terms of AP, a standard object detection evaluation metric described in detail in Section 2.6.3. For the KITTI validation split, the AP was measured to be 0.71 for the car class when the IoU threshold to be considered correct was set to 0.5. At an IoU threshold of 0.7, AP for car detection was 0.50. While this model performance is not competitive with state-of-the-art object detection, this thesis focuses on the task of OOD detection, and the obtained performance was deemed satisfactory for this purpose.

Appendix A shows the NN layer architecture of the PyTorch Faster R-CNN model that was used for the experiments in this thesis in more detail.

3.3 Out-of-distribution detection

This section details the implementation of various OOD detection methods on the Faster R-CNN object detection network. All the methods used were adapted from the domain of image classification, and any modifications that were necessary to enable them to work in the two-stage object detection architecture are described.

3.3.1 Max softmax

Previous work in image classification established a set of metrics as a baseline for comparing different OOD detection methods in terms of ability to separate ID from OOD samples using some measure of novelty [16]. To establish this baseline, the authors used the maximum output confidence score from the output of a neural network softmax layer as their novelty metric. The authors argue that while OOD samples have been anecdotally noted as assigning high confidence outputs to OOD samples, these scores are still often lower than on ID samples. As a result, the maximum value output by the softmax activation function can be used as a novelty score for doing OOD detection. Equation 3.1 describes how the max softmax score s is obtained for input sample $\mathbf{x}^*$.

$$s(\mathbf{x}^*) = \max_c P(y_c|\mathbf{x}^*; \mathcal{D}) \tag{3.1}$$

Here, y_c is the class label c and $P(y_c|\mathbf{x}^*; \mathcal{D})$ is estimated using the softmax output of a NN trained with dataset $\mathcal{D}$.

This thesis adopts this same metric as a baseline measure for detecting FP objects output by the object detection network. This corresponds to the same setup as an unmodified object detection network and serves as the baseline detection score to beat.

3.3.2 ODIN

An improved version of the Max Softmax based OOD detection approach, called ODIN improves performance over Max Softmax by scaling the logit layers before the softmax output by a constant value called temperature, as well as perturbs the image input by a small amount [24]. Empirically, these modifications are shown to improve the separability of ID and OOD samples. The authors perform experiments on CIFAR-10 and CIFAR-100 as two sets of ID images. OOD images are provided to the classification networks from other RGB image datasets such as TinyImageNet [44], LSUN [45], as well as Uniform and Gaussian noise datasets. The specifics of temperature scaling and input perturbation are described in more detail in the next sections.

Temperature Scaling

The temperature scaling in ODIN works as follows: assuming a neural network $\boldsymbol{f} = (f_1, ..., f_N)$ trained to classify N classes, a label $\hat{y}(\boldsymbol{x}) = \text{argmax}_i S_i(\boldsymbol{x}; T)$ is computed as the temperature scaled softmax output of logit values f_i using Equation 3.2 [24].

$$S_i(\boldsymbol{x}; T) = \frac{\exp(f_i(\boldsymbol{x}/T)}{\sum_{j=1}^{N} \exp(f_j(\boldsymbol{x}/T)} \tag{3.2}$$

The value for the temperature T is a hyper-parameter which the authors in the original paper optimize using a fraction of the test images [24].

Input perturbation

In ODIN, the input data is perturbed given Equation 3.3:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \epsilon \ \text{sign}(-\nabla_{\boldsymbol{x}} \text{log} S_{\hat{y}}(\boldsymbol{x}; T)) \tag{3.3}$$

where ϵ is the perturbation magnitude and $S_{\hat{y}}(\boldsymbol{x}; T)$ is the maximum temperature-scaled softmax output value given by the network for image $\boldsymbol{x}$ [24]. Two passes of the network are required to do input perturbation. The first pass is used to calculate a gradient of the cross-entropy loss with respect to the input sample, which is done using the back-propagation algorithm. The input is then modified by subtracting perturbation magnitude ϵ times the sign of the input gradient. The perturbation magnitude hyperparameter ϵ is optimized using a fraction of the test images [24].

Modifications

Hyperparameter optimization

While optimizing the temperature and input perturbations using a holdout subset of the test images was done by [24] in the original ODIN paper, this is inappropriate for the task of OOD detection. The reason that this is inappropriate is that the types of OOD images the network is supposed to handle cannot be known before test time in the typical OOD detection setup. Instead, this work uses a dataset that is different from the test set to optimize hyperparameters, as done by [1]. The task of detecting false positive objects in AV driving scenes allows for the straightforward production of a validation set for optimizing hyperparameters. False positives from the KITTI validation data can be used to optimize hyperparameters T and ϵ for good FP detection performance on that data, before finally evaluating performance on the IDD test set. A similar method is used for FN detection, with hyperparameter optimizations being done to maximize FN detection performance.

Temperature scaling and input perturbation

Because input perturbation modifies the image input to the object detection network, there is no guarantee that the RPN will propose regions that contain objects the same way as when the input is not perturbed. Perturbations often reduce reduces the confidence of object proposals significantly; the result being that most or all proposal boxes then get eliminated by NMS.

To rectify this, the object proposals and object scores from the RPN are retained from the first pass through the network. The perturbed image is used to produce a new feature map representation of the input to use with these original object proposals in the later layers of the network. This produces the desired updates to the posterior class scores as in the ODIN paper.

The most effective amount of input perturbation was determined by trying 21 evenly spaced values between 0 and 0.004 on the validation set and selecting the best performing value for evaluating the test set. This optimization follows the same method used by [24].

Unfortunately, temperature scaling is only useful for classifiers with three or more output classes. Scaling logit layers for a binary classifier does not affect the output. As such, temperature scaling is not used in the experiments in this thesis, and the default softmax function is used instead.

3.3.3 Mahalanobis Distance

Improving further on the results obtained using ODIN, a technique was proposed for effective OOD detection using the features from the penultimate layer of a dense neural network [23]. Using the training images as input, the authors calculate class-wise mean vectors of these features (Equation 3.4 and a single tied covariance matrix over all classes (Equation 3.5). The mean vectors and covariance matrix are then used to calculate the Mahalanobis distance that a given test sample is to each of the known classes, with the minimum distance to a class mean being used as the novelty score (Equation 3.6).

It is argued that this technique avoids the issue of using label over-fitted softmax outputs [23], which occurs in methods that use the softmax output as a novelty score [16, 24]. Furthermore, the authors highlight that using the Mahalanobis distance has a statistically meaningful interpretation of being the log of the probability density that a sample belongs to a particular class, under the assumption that the penultimate layer features of each class have a Gaussian distribution. The method achieved state-of-the-art OOD detection performance for classification. However obtaining those results requires additionally perturbing the input data as done by ODIN, as well as ensembling multiple Mahalanobis distance functions operating on different feature layers of the network [23]. Unfortunately, the weights for the ensemble approach were inappropriately determined using a held-out set of the OOD test data.

$$\hat{\mu_c} = \frac{1}{N_c} \sum_{i:y_c=c} f(x_i) \tag{3.4}$$

$$\hat{\Sigma} = \frac{1}{N} \sum_{c} \sum_{i:y_c=c} (f(x_i) - \hat{\mu}_c)(f(x_i) - \hat{\mu}_c)^T \tag{3.5}$$

$$M(x) = \max_{c} -(f(x) - \hat{\mu}_c)^T \hat{\Sigma}^{-1} (f(x) - \hat{\mu}_c) \tag{3.6}$$

Modifications

Two experiments were done using this method: one with and one without input perturbation. For each experiment, penultimate features were collected from the KITTI train set where the objects were classified correctly and had a high IoU (>0.7) score with a ground truth bounding box. These penultimate features (i.e. the dense layer prior to the logit layer of the network) were of size 4096 and were used to calculate the class-wise mean vectors (Equation 3.4) and a tied covariance matrix (Equation 3.5) required for Equation 3.6.

For the input perturbed Mahalanobis distance experiment, the same values were tested as in ODIN, and the hyperparameters that gave the best performance on the KITTI validation data were used [24].

The ensembling approach as done in [23] was not attempted in this work due to a fairly limited number of layers corresponding to individual objects in the final layers of the Faster R-CNN model. Should it be attempted at a later time, weights for each layer could be determined in a more experimentally sound way using the KITTI validation dataset, as was done for the ODIN experiments.

3.3.4 SVM and early-layer features

While it was found that using the penultimate layer from a trained classification network was useful for OOD detection [23], this idea has been expanded upon further. By searching for an OODL within the set of all hidden layers in a classification network, it is possible to achieve even more performant OOD detection for image classification [1]. For a given training set of images, the authors extract features from each of the hidden layers of the network. For convolutional layers, the values in each filter were averaged to reduce the size of the output. For each set of features from the k model layers, a one-class support vector machine (OSVM) [40] model is trained. For the k OSVM models, a validation set of OOD images was used to determine which performed best, thus selecting the OODL. The OSVM parameter nu is also optimized this same way over possible values 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, and 0.3. Finally, the radial basis function kernel is used in the OSVM models as it was found to perform best. [1].

Once an optimal OSVM model is found, a new sample can be assigned a novelty score by determining the distance at which the sample lies from the separating hyperplane of the model. OOD samples should theoretically fall on one side of the separating hyperplane, while ID samples lie on the other.

While it is a common mistake in OOD detection literature to use held-out test data to tune hyperparameters, it is possible to side-step this issue for image classification by using an image set that is different from the test and train datasets to select the OODL and to tune other OSVM hyperparameters [1]. From the validation set, the OODL is selected as the layer which gives the lowest OOD sample detection error on the validation set [1]. It was found that the OODL performs well for grayscale and RGB image classification datasets [1]. Furthermore, the authors improve their results by incorporating the input perturbation technique initially proposed for ODIN [24].

Modifications

Extracting meaningful early layer features on an object-by-object basis in an object detection model is a non-trivial task. In the original proposal of this method, the early hidden layer feature activations in the network directly correspond to the output classification, but in object detection, these features correspond to the entire image scene and not just a single object of interest. Therefore, only a subset of the total extracted features should be used to determine the novelty-score of a particular object.

Two techniques were used to overcome this issue. The first solution was to use features only from later layers of the network that corresponded to individual objects. These layers could directly be used to train OSVM models for OOD detection as done in [1]. The second method used was to obtain early layer features by re-extracting features with cropped sections of the input image scene. The image is cropped using each predicted bounding box to obtain a set of early layer features that correspond to that object. The same feature extractor used in the object detection model was used for this purpose. While these image crops lose some of the scene contexts that may have resulted in their original classification as objects, the features extracted do contain all of the information related to the proposed object itself.

Because the cropped images used to extract new sets of early layer features are not consistent in object detection, it is necessary to resize them to provide a set of fixed dimension samples to train the OSVM model with. To do this resizing, the feature map obtained after passing image crops through the feature extractor are averaged across the width and height axes. This produces a vector with a size equal to the number of convolutional filters in the OOD discernment layer being tested.

Three experiments were performed using SVM models. SVM(conv) is trained with a vector of 512 average pooled convolutional filters from the RPN network which correspond to individual objects. In Appendix A, these correspond to the layer ‘rpn[conv1]‘. In the second experiment, SVM(fc) uses a fully connected layer in the head of the Faster R-CNN network of size 4096. This layer corresponds to layer ‘head[classifier][0]‘ in Appendix A. Finally, SVM(early layer) uses features extracted from cropped image sections for each object using the feature extraction network. The OODL used for this experiment was determined as detailed above.

3.3.5 Predicted Entropy and Mutual Information

Other previous work has introduced methods for estimating the uncertainty in model predictions in a principled way, such that sources of uncertainty could be separated into contributing factors [27]. These factors were, namely, model uncertainty and data uncertainty. The predicted entropy estimates the total uncertainty in a model while the mutual information estimates the model uncertainty. Model uncertainty can be thought of as uncertainty due to an insufficient quantity of data and is also called epistemic uncertainty. The other component of the total uncertainty, data or aleatoric uncertainty, is caused by noise inherent to the dataset population and can not be resolved with more data.

To calculate these two measures of uncertainty, it is necessary to have a set of predictions from an ensemble of models. These can be produced by training multiple models with different weight initialization or, more efficiently, using the Monte Carlo (MC) Dropout sampling technique [10]. Creating an ensemble of NN models is computationally expensive, so it was proposed in [10] to instead sample outputs of a NN that was trained using dropout layers. To do this, dropout layers of a NN are left activate during evaluation and the predicted output class distribution is sampled K times. Because of the dropout layers, each iteration makes use of different active model neurons. This allows for obtaining multiple posterior distributions without expensive model ensembling.

The predicted entropy of an MC dropout sampled distribution of labels is defined as [27]:

$$\mathcal{H}[P(y|\mathbf{x}^*;\mathcal{D})] = -\sum_{i=1}^{C} P(\omega_k|\mathbf{x}^*;\mathcal{D}) \; \ln P(\omega_k|\mathbf{x}^*;\mathcal{D}) \tag{3.7}$$

where $P(\omega_k|\mathbf{x}^*;\mathcal{D})$ is the statistical mean of label w_c over K MC samples:

$$P(\omega_k|\mathbf{x}^*;\mathcal{D}) = \frac{1}{K}\sum_{c}^{K} P(\omega_c|\mathbf{x}^*;\mathcal{D}) \tag{3.8}$$

The mutual information is calculated as the total entropy (Equation 3.7) minus the expected data uncertainty:

$$\mathcal{I}[y,\theta|\mathbf{x}^*,\mathcal{D}] = \mathcal{H}[\mathbb{E}_{p(\theta|\mathcal{D})} \; P(y|\mathbf{x}^*;\theta)] - \mathbb{E}_{p(\theta|\mathcal{D})} \; \mathcal{H}[P(y|\mathbf{x}^*;\theta)] \tag{3.9}$$

for varying model weights θ obtained by MC dropout. For calculating $\mathbb{E}_{p(\theta|\mathcal{D})}$, each set of active model weights is considered to be equiprobable.

Modifications

Because the object detection model used in this thesis incorporated two dropout layers in the final layers, no modifications were required to implement the entropy and mutual information OOD detection metrics. For the experiments, $K = 10$ MC dropout samples were calculated for each sample $\mathbf{x}^*$ in the test and validation datasets.

3.4 Experiments

In the OOD detection literature from the image classification domain, a model is typically trained to predict the correct class of images from a predefined inlier dataset. Commonly, datasets such as MNIST [22], CIFAR-10, and CIFAR-100 [20] are used and are then considered to be representative of the ID set of images the network should perform well on [16, 24, 23]. These papers then introduce a new set of images containing completely different classes of images such as Omniglot [21], ImageNet [35], or random Gaussian or uniformly distributed noise to represent images that are OOD [24, 23]. The setup for these experiments is that a well-performing OOD detection method should be able to, on average, assign a higher novelty score to images from the set of OOD images than the ID images.

The OOD detection task cannot be approached the same way for object detection networks. Object detection networks employ a BG class as a classification outcome to eliminate object proposals that are predicted as being none of the FG classes. The result of this is that any kind of object or image crop that does not contain an accurately bounded FG class could be correctly classified as BG. It is, therefore, not immediately clear how to introduce objects to scenes that should are neither FG or BG (i.e. $\neg$ FG) and be labelled as a new class, OOD.

One naive solution to this problem is to introduce image scenes to the object detection network that contain objects known not to appear in the training set. As an example, the class of object “autorickshaw” appears commonly in the IDD and not in the KITTI dataset. Given an object detection network designed to detect cars as FG however, KITTI does contain other vehicles like vans, motorcycles, and bicycles that should be labelled BG. That is, the network is supposed to detect at least some other vehicles as BG. It becomes difficult to say whether or not an autorickshaw is OOD and falls outside of the space of non-car vehicles that the network should have learned as being BG.

To further illustrate this reasoning, consider the case where buildings with different styles of architecture relative to the KITTI dataset appear in the IDD. Such buildings are different than those contained in the inlier dataset, yet it is probably not useful to detect them as OOD objects from a safety perspective. However, it is difficult to say that these new buildings are less novel relative to KITTI than a new kind of vehicle like the autorickshaw. There is a scale of novelty that new objects appearing in a dataset have relative to the ID data and it is likely to be subjective, difficult, or impossible for a human to assign fair ground-truth OOD object labels to the appropriate parts of an image from a new dataset like the IDD. Such labelling efforts would require a fair evaluation of relative levels of novelty for all parts of the images in the test dataset, which is unlikely to be feasible.

A resolution to this problem is to avoid the intractable task of labelling a set of OOD test samples in a way that subjectively sorts normal-looking and unusual-looking samples. Instead, it is possible to create a set of ID data points from the set of objects the model correctly labels and create a set of OOD predictions from the model predictions that are incorrect. TPs (i.e. correctly labelled FG instances) act as the ID dataset, while FPs (i.e. boxes predicted as a FG class that are in fact BG) act as the OOD dataset. It stands to reason that FPs should be expected to measure higher in terms of novelty score than TP objects since the OOD detection method learns a representation of the FG class. A well-performing OOD detector could distinguish FPs from TPs using novelty scores.

Similarly, another task can be formulated for false negative detection. BG areas of the image that the network correctly predicts as being background (i.e. TNs) can be used as the ID dataset, while areas that the network predicts as BG that are in fact FG (i.e. FNs) can be used as the OOD dataset. An OOD detector can be formulated such that it learns a representation of the BG class. TN instances would be assigned a low novelty scores while FNs are assigned high novelty scores by an ideal OOD detector. The OOD detector would attempt to catch cases where FG objects are incorrectly discarded by the network due to the network erroneously labelling them as BG.

This thesis tests this theory by measuring the ability of various OOD detection techniques to identify BG incorrectly labelled as FG (i.e. false-positives) and in a separate experiment to identify when FG objects are incorrectly discarded as BG (i.e. false-negatives). These two experiments are done separately to allow for the case that it might be best to use different algorithms to detect FPs and FNs. The nine techniques summarized in section 3.3 for OOD detection were tested for this task. That is: Max Softmax [16], ODIN [24], Mahalanobis distance with and without input perturbation [23], SVM on hidden model features (late layer fully connected features (SVM (fc)), late layer convolutional features ((SVM (conv)), and the OODL layer from the feature extraction network (SVM (early layers))) [1], predicted entropy [27], and mutual information [27].

The performance of each of these techniques is evaluated in terms of the previously outlined OOD detection metrics: AUROC, FPR @95% TPR, detection error, AUPR(in), and AUPR(out). Furthermore, risk-coverage plots show the ability of each of these methods to reduce misclassification risk by deferring classification to a human or other low-risk system.

Novelty metrics from the FP detection experiments can also be adapted to score object prediction confidence used for calculating AP performance of the object detector. By taking the negative of the novelty score from each FP detection method, a measure of confidence that a prediction is a TP is obtained. Therefore, a novelty detection metric that does FP detection better than the baseline Max Softmax approach should show an improvement in AP. AP results are given for the object detector when using these alternative confidence scores.

3.4.1 FP and FN Datasets

FP detection

For the experiment of FP detection, the KITTI test split was used to generate the OOD detection model, the KITTI validation split was used to optimize hyperparameters as necessary, and the IDD validation split was used as the test set (hereafter referred to as "IDD test set"). For each dataset, outputs predicted by the object detector were labelled as being a TP, FP, TN, or FN.

The TP and FP labels are determined for each predicted FG object by comparing the coordinates of the predicted bounding box to the ground truth annotations. Predicted boxes with a label car were considered to be a TP if it there was an IoU with a car labelled ground-truth box that is greater than or equal to 0.7. Conversely, predicted boxed are labelled as FPs if the IoU was less than 0.7. Table 3.1 summarizes the number of TP and FP found in each of the datasets.

For methods requiring a set of TP samples to build the OOD detection model (such as Mahalanobis distance and SVM based methods), the TP objects predicted on the KITTI train split were used. These models never see FP objects before validation or test time. For methods that required optimizing hyperparameters via a grid search, a random selection of 10% of the KITTI validation split was used for efficiency. The IDD test dataset is not used for training or hyperparameter optimization.

	Kitti (train)	Kitti (val)	IDD (val)
FP count	3695	5409	4766
TP count	9420	8015	4112
TP+FP count	13115	13424	8878
FP/(TP+FP)	28.17%	40.29%	53.68%

Table 3.1: Summary of the number of TPs, FPs in each of the datasets used for FP detection with OOD detection methods.

FN detection

For the FN detection experiments, an identical process was followed with minor exceptions. Firstly, objects determined to be BG in the final layers of the network are retained and returned as output for given image scenes. Proposed BG regions that do not have an IoU greater than 0.7 with a FG object are considered to be TNs. If a region classified as BG has an IoU greater than or equal to 0.7 with a FG object, it means this proposal region should have been classified as FG but was incorrectly discarded. These are the FN boxes.

Because of the large class imbalance of TNs to FNs, the number of TNs was sub-sampled randomly before fitting each OOD detection method. For each dataset used, only a randomly selected 5% of the TNs were retained as output on an image-by-image basis. This sub-sampling changes the prior distribution of FNs and TN, but since all OOD detection model that required determining parameters were one-class classification models, this change in prior is deemed acceptable. A summary of the FN detection dataset before sub-sampling is given in Table 3.2, and after sub-sampling in Table 3.3.

	Kitti (train)	Kitti (val)	IDD (val)
FN count	294	95	150
TN count	292257	297263	278273
TN+FN count	292551	297358	278423
FN/(TN+FN)	0.100%	0.032%	0.054%

Table 3.2: Summary of the number of TNs, FNs in each of the datasets used for FN detection with OOD detection methods before sub-sampling the number of true negatives.

Again similar to the FP detection experiments, TNs from the KITTI train split were used to fit OOD detection models where required, the KITTI validation split was used to

	Kitti (train)	Kitti (val)	IDD (val)
FN count	294	95	150
TN count	16445	16498	20140
TN+FN count	16739	16593	20290
FN/(TN+FN)	1.756%	0.573%	0.739%

Table 3.3: Summary of the number of TNs, FNs in each of the datasets used for FN detection with OOD detection methods after sub-sampling 5% of true negatives on an image-by-image basis.

find well-performing hyperparameters for each method. The IDD test data was again used as a test set.

Chapter 4

Results and Discussion

4.1 False Positive Detection

It was found that OOD detection methods were only slightly effective for identifying cases where BG image scenery was incorrectly identified as being the car FG class. Across the methods, there is no consensus over the different performance metrics in terms of which OOD detection method performed best for this task. The ODIN and Predicted Entropy approaches generally performed well on the KITTI validation set over all metrics except AUPR(out) (Table 4.1) and were able to beat the baseline Max Softmax approach by a small magnitude. On the IDD test data, the best performing methods for FP detection were Max Softmax and Predicted Entropy (Table 4.2). Performance on these methods should be considered the baseline for FP object detection in future work related to this task.

Overall, none of the OOD detection method performed better than the Max Softmax approach by a significant margin. However, marginal improvement over Max Softmax by some of these techniques provides some promise that better results are obtainable still. One possible explanation for the comparable results across methods is that many of the FP detections from the Faster R-CNN network appear analogous to true car objects and, as a result also appear similar to the OOD detection methods. Figure 1.1 shows some of these cases, where objects with the ground-truth labels of "van" are detected as cars, as well as a case where the reflection of a car in a building window is incorrectly detected as a car. Furthermore, cases where the predicted bounding box had an IoU less than 0.7 with the ground truth bounding box would be considered FPs in these experiments, despite that, in some cases, these predictions may have been just a slight deviation from the ground truth bounding box. Relaxing the IoU threshold used to label predictions as TPs or FPs may show a larger improvement in some methods over the baseline approach since FPs would appear less similar to the TP data.

Input perturbation was judged to be of dubious effect for the Mahalanobis distance based OOD method. Using perturbation, there was an improvement in performance across metrics for the KITTI data, but not for the IDD data. Surprisingly, ODIN also did not improve the OOD detection performance over the Max Softmax approach as is often the case in the classification literature. It is possible that temperature scaling is very important for ODIN to perform well and that a future experiments may show an improvement with this method if temperature scaling can be effectively implemented. The SVM-based OOD detection methods did not perform well overall. A possible explanation for this is that, in the original paper, images where typically small and contained similarly scaled objects. The difference in scales may result in more varied feature map activation across the SVM train samples which negatively effects the performance of the SVM models. More investigation into the failure of the SVM methods to perform well is required.

Method	AUROC↑	FPR@95TPR↓	D.Err.↓	AUPR in↑	AUPR out↑
Max Softmax	0.8628	49.67%	31.45%	0.8982	**0.8072**
ODIN	0.8621	**49.40%**	**31.27%**	0.8980	0.8039
Mahalanobis	0.7444	82.31%	50.86%	0.7831	0.6849
Mahalanobis (w pert.)	0.7445	82.25%	50.82%	0.7828	0.6853
SVM (fc)	0.5557	93.63%	57.55%	0.6364	0.4488
SVM (conv)	0.6497	87.24%	53.78%	0.7184	0.5268
SVM (early layer)	0.6824	78.25%	48.71%	0.7651	0.5498
Predicted Entropy	**0.8631**	49.52%	31.40%	**0.8984**	0.8071
Mutual Information	0.8380	54.95%	34.60%	0.8798	0.7643

Table 4.1: KITTI validation split FP detection performance for various OOD detection methods. Best results are bolded and arrows indicate if a higher (↑) or lower (↓) score is better. For AUPR, “in” and “out” refer to whether or not the ID class or OOD class is considered to be the positive class, respectively.

Method	AUROC↑	FPR@95TPR↓	D.Err.↓	AUPR in↑	AUPR out↑
Max Softmax	0.8274	**55.20%**	**28.25%**	0.8199	**0.8312**
ODIN	0.8263	55.67%	28.56%	0.8198	0.8278
Mahalanobis	0.7759	69.67%	34.94%	0.7489	0.7701
Mahalanobis (w pert.)	0.7748	69.63%	34.91%	0.7486	0.7667
SVM (fc)	0.5992	90.18%	44.45%	0.5515	0.6090
SVM (conv)	0.6553	89.32%	44.05%	0.5921	0.6549
SVM (early layer)	0.4751	96.18%	46.26%	0.4411	0.5287
Predicted Entropy	**0.8281**	55.23%	28.26%	**0.8208**	0.8309
Mutual Information	0.8115	60.07%	30.50%	0.7990	0.8083

Table 4.2: IDD test split FP detection performance for various OOD detection methods. Best results are bolded and arrows indicate if a higher (↑) or lower (↓) score is better. For AUPR, "in" and "out" refer to whether or not the ID class or OOD class is considered to be the positive class, respectively.

4.1.1 Average Precision

The AP scores are shown after being computed using each OOD FP detection method to estimate prediction confidence of detected objects in the KITTI validation dataset (Table 4.3 and the IDD test dataset (Table 4.4). For both of the datasets, the Predicted Entropy method outperformed the baseline Max Softmax value used by default in Faster R-CNN. These results align with the AUPR(in) metrics in Tables 4.1 and 4.2. This is expected since AP and AUPR(in) compute and summarize the TP and FP rates of the object detector in similar fashions.

	IoU Threshhold	
Method	0.5	0.7
Max Softmax	0.7053	0.5009
ODIN	0.7049	0.5000
Mahalanobis	0.6883	0.4385
Mahalanobis (w pert.)	0.6894	0.4363
SVM (fc)	0.6350	0.3588
SVM (conv)	0.6688	0.4025
SVM (early layer)	0.6588	0.4272
Predicted Entropy	**0.7058**	**0.5011**
Mutual Information	0.7009	0.4917

Table 4.3: KITTI validation data Average Precision scores after using each FP OOD detection method to estimate the prediction confidence for each object.

	IoU Threshhold	
Method	0.5	0.7
Max Softmax	0.2238	0.1370
ODIN	0.2236	0.1369
Mahalanobis	0.2207	0.1259
Mahalanobis (w pert.)	0.2213	0.1255
SVM (fc)	0.1900	0.0928
SVM (conv)	0.2039	0.1010
SVM (early layer)	0.1688	0.0783
Predicted Entropy	**0.2241**	**0.1371**
Mutual Information	0.2224	0.1340

Table 4.4: IDD test data Average Precision scores after using each FP OOD detection method to estimate the prediction confidence for each object.

4.1.2 Method Hyperparameters

A summary of the hyperparameters used for each OOD detection method is given in Table 4.5. The minimum amount of perturbation tested provided the best performance for Mahalanobis (w. pert), suggesting that either lower amounts should be tested or that perturbation only degraded performance for this method. The OODL layer refers to the index of the feature extraction layer of the PyTorch VGG16 implementation, given in Appendix A.

Method	Epsilon	M.C. Samples	Kernel	Nu	OODL
Max Softmax	-	-	-	-	-
ODIN	0.0026	-	-	-	-
Mahalanobis	-	-	-	-	-
Mahalanobis (w pert.)	0.0002	-	-	-	-
SVM (fc)	-	-	rbf	0.001	-
SVM (conv)	-	-	rbf	0.001	-
SVM (early layer)	-	-	rbf	0.01	23
Predicted Entropy	-	10	-	-	-
Mutual Information	-	10	-	-	-

Table 4.5: Hyperparameters for each OOD detection method for the task of FP detection. The OODL refers to which layer in the VGG16 feature extraction network provided the best performance. The value of OODL corresponds to the PyTorch layer index of the implemented feature extractor that separates activation functions into discrete network layers. Epsilon is the perturbation magnitude hyperparameter.

4.1.3 FP Detection Risk-Coverage Plots

The Risk-Coverage plots for FP detection on both the KITTI and IDD datasets show risk increasing as dataset coverage increases, as expected. The best-performing methods in terms of risk-coverage trade-off were Predicted Entropy, Mahalanobis, Max Softmax, and ODIN, which performed similarly. For the KITTI validation dataset, Figure 4.1 shows the risk coverage for each method plotted individually and Figure 4.2 shows those same curves on a single plot for comparison purposes. Similarly, Figures 4.3 and 4.4 show individually plotted and compared risk-coverage curves for the IDD test data.

For both the KITTI and IDD datasets, no FP detection method was able to significantly outperform the baseline Max Softmax method for reducing misclassification risk. The plots for the KITTI validation data shows that the baseline misclassification risk for the model on that data is 40.29%. Similarly, the plot for the IDD test data shows a baseline misclassification risk of 53.68%. For both datasets, significantly lowering risk by means of redirecting a portion of the classification to a safer, yet more costly, handler involves drastically reducing coverage. This is likely to be infeasible for an practical AV system since it is desirable for the low-latency object detection network to perform the large majority of region classification tasks. As a result the OOD detection methods are deemed to be not-useful for improving safety by means of abstaining from classification as outlined in [43].

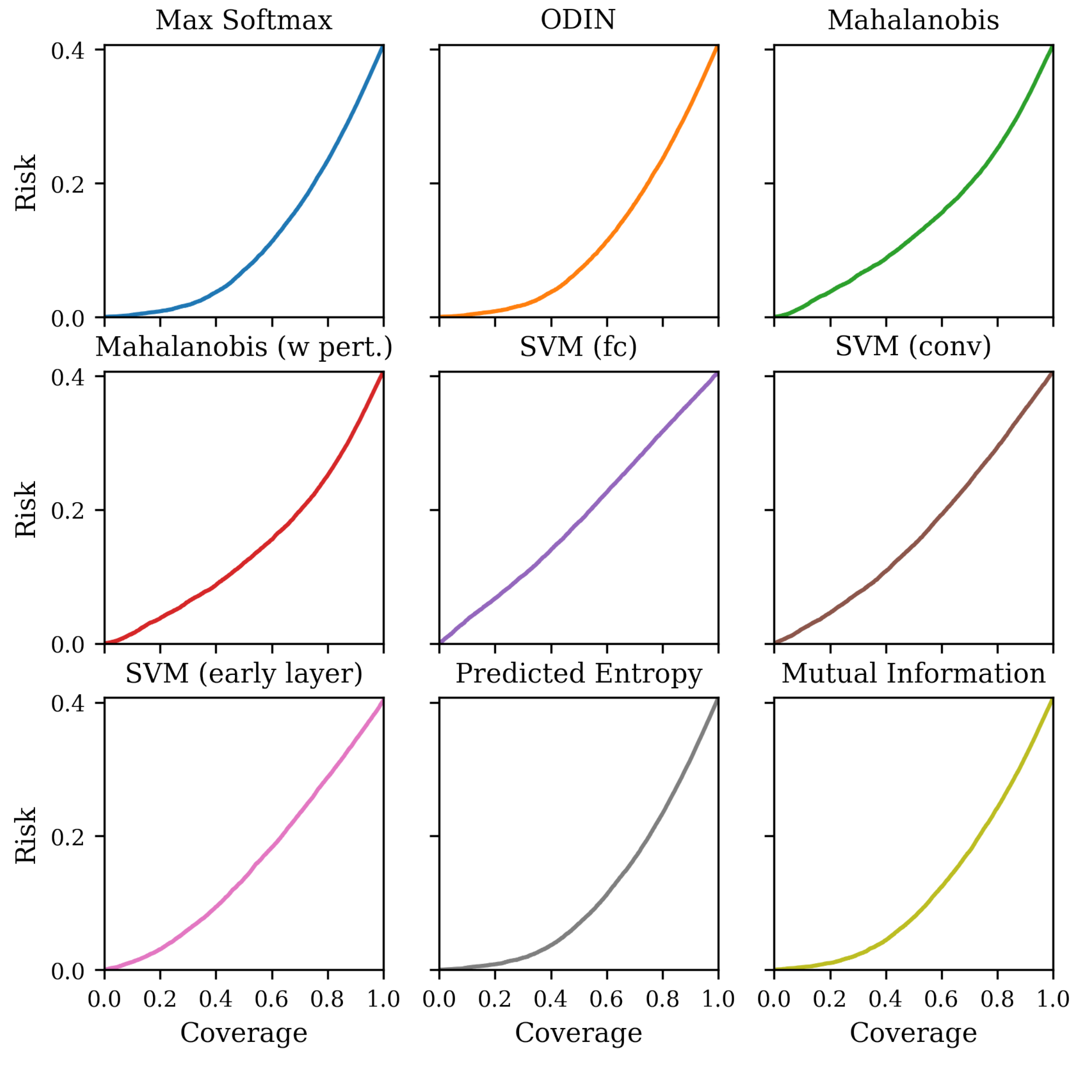

Figure 4.1: Risk-Coverage curves for FP detection on the KITTI validation set for various OOD detection methods.

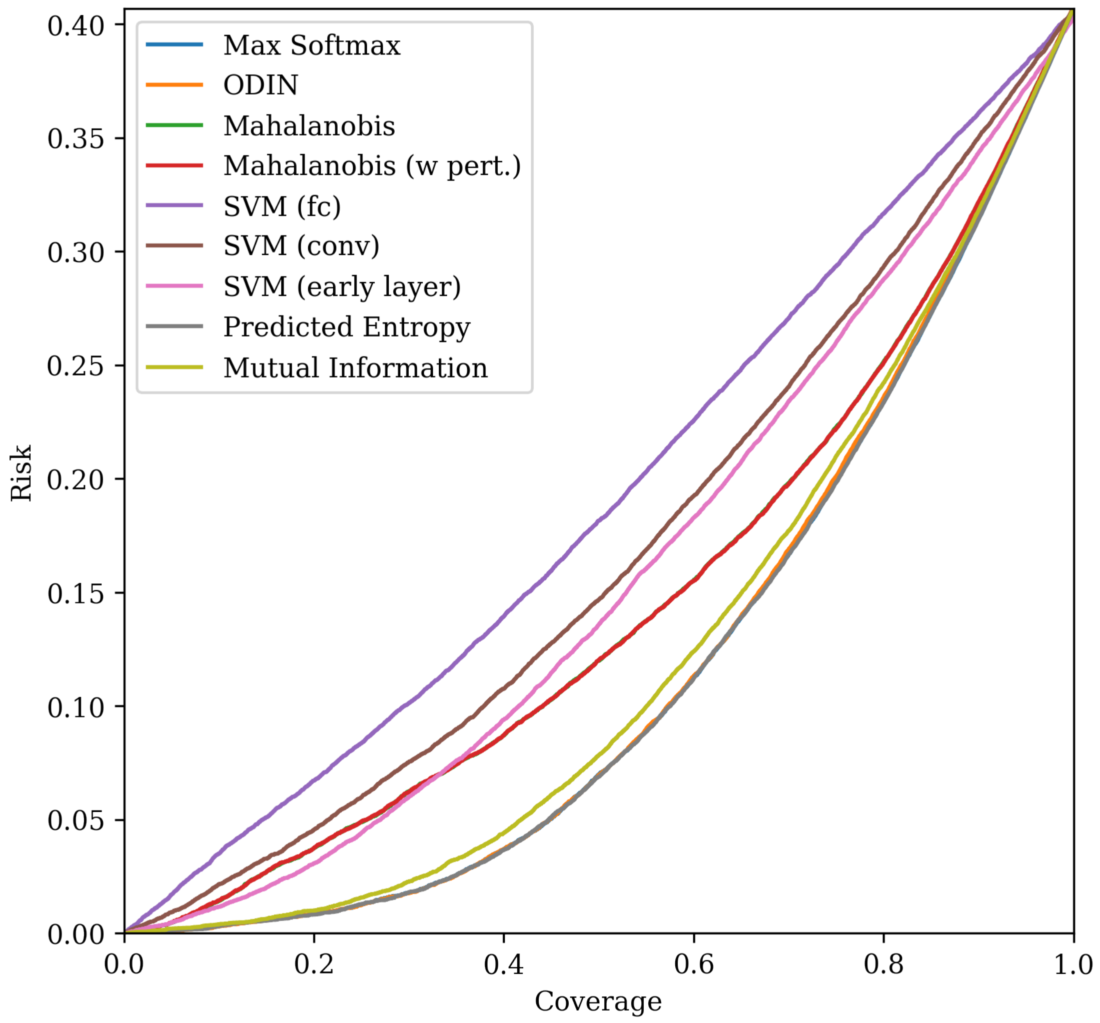

Figure 4.2: Comparison of risk-Coverage curves for FP detection on the KITTI validation set for various OOD detection methods.

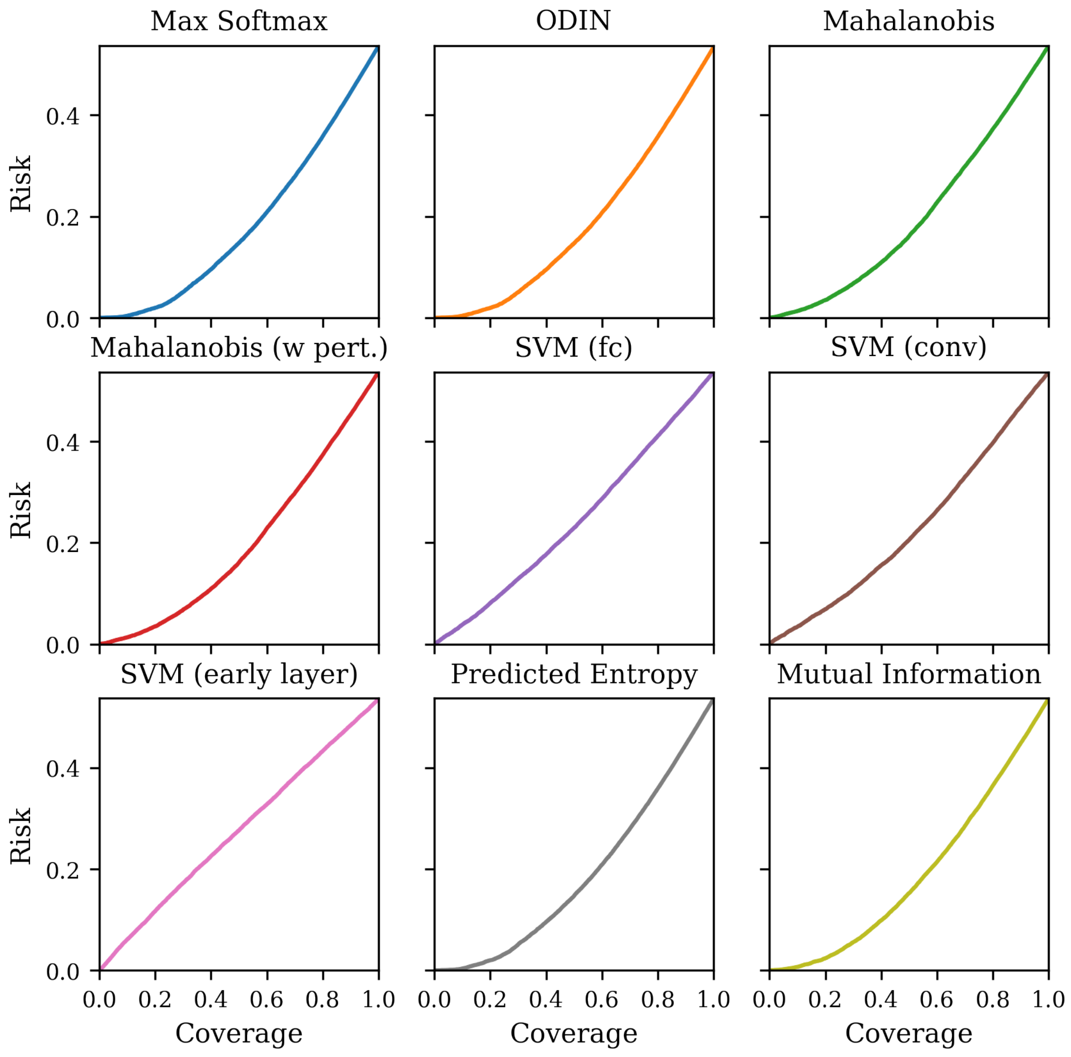

Figure 4.3: Comparison of risk-Coverage curves for FP detection on the IDD test set for various OOD detection methods.

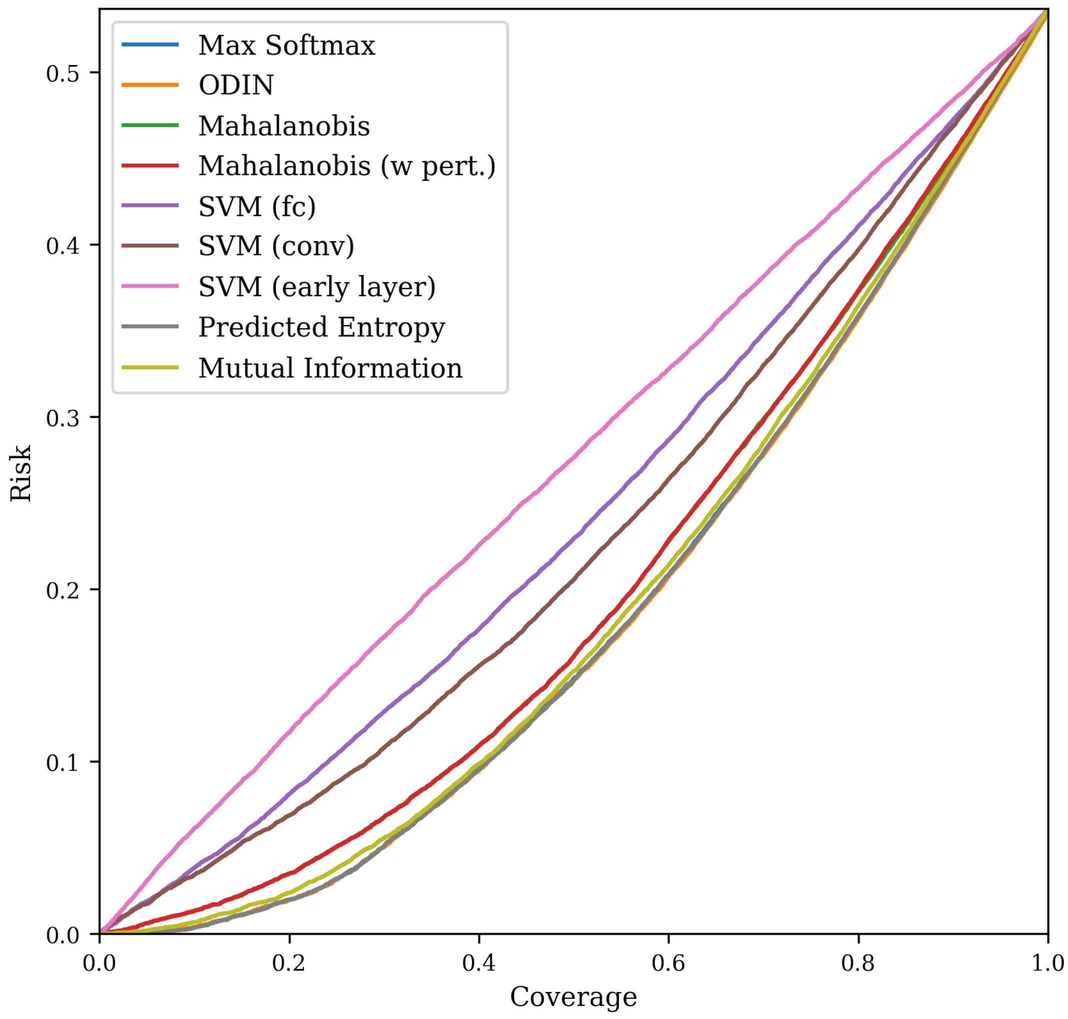

Figure 4.4: Comparison of risk-Coverage curves for FP detection on the IDD test set for various OOD detection methods.

4.2 False Negative Detection

It was found that OOD detection methods were useful for identifying cases where FG objects were incorrectly identified as being BG in the late layers of the object detection network. There is again, however, no consensus over the different performance metrics in terms of which OOD detection method performed best. For the KITTI validation data, The Mahalanobis OOD detection method outperformed other methods except in terms of AUPR(out), where Max Softmax performed best (Table 4.6). For the IDD test data, the information theoretical OOD detection methods, Predicted Entropy and Mutual Information, performed best over overall (Table 4.7.

For the FN detection experiments, the ratio of FNs to TNs was highly imbalanced. As a result, the metrics which weigh the performance of inlier and outlier classification equally are not particularly informative. As a results, AUROC, FPR @95% TPR, and Detection Error are judged to be poor measures of how well the rare instances of FN samples are detected. The most important metrics for these experiments was AUPR(out), which gives an indication of the precision and recall that OOD detection methods are able to provide for FN samples. In terms of AUPR(out) the best performing method for the KITTI validation data was Max Softmax, while for the IDD test data was Mutual Information.

The difference in performance for different OOD detection approaches across the KITTI and IDD datasets may be explained by the domain shift which occurs relative to the KITTI training data. Because the KITTI validation dataset is visually more similar to the KITTI training data, it is unsurprising that the method used to classify FG and BG objects at train time (i.e. Max Softmax) performed best. It does not seem unusual that Max Softmax would have reduced performance for OOD detection on the more unusual IDD dataset samples.

Finally, input perturbation was again of dubious benefit over the equivalent unperturbed OOD detection methods. ODIN did not clearly improve results over Max Softmax and the Mahalanobis method often worked better without perturbation of the input data. This suggests that the modified technique in which the entire image scene was perturbed based on the predicted object labels and bounding boxes needs further refinement. Its possible that for effective perturbation, the input image is required to be perturbed differently for each of the predicted objects, rather than they way it is done currently by backpropagating the summed loss over all objects to alter the entire input image scene. Object-by-object perturbation is however likely to be prohibitively computationally expensive.

Method	AUROC↑	FPR@95TPR↓	D.Err.↓	AUPR in↑	AUPR out↑
Max Softmax	0.8317	71.40%	72.59%	0.9984	**0.0895**
ODIN	0.8458	65.09%	64.73%	0.9988	0.0699
Mahalanobis	**0.9119**	**24.51%**	**24.39%**	**0.9994**	0.0426
Mahalanobis (w pert.)	0.9087	28.42%	28.28%	0.9994	0.0421
SVM (fc)	0.2215	99.93%	99.39%	0.9825	0.0034
SVM (conv)	0.3425	97.66%	97.13%	0.9915	0.0041
SVM (early layer)	0.4168	85.86%	85.40%	0.9944	0.0044
Predicted Entropy	0.8482	64.21%	63.86%	0.9988	0.0737
Mutual Information	0.8464	59.07%	58.75%	0.9988	0.0763

Table 4.6: KITTI validation split FN detection performance for various OOD detection methods. Best results are bolded and arrows indicate if a higher (↑) or lower (↓) score is better. For AUPR, “in” and “out” refer to whether or not the ID class or OOD class is considered to be the positive class, respectively.

Method	AUROC↑	FPR@95TPR↓	D.Err.↓	AUPR in↑	AUPR out↑
Max Softmax	0.9584	17.45%	17.32%	0.9995	0.2232
ODIN	0.9750	15.64%	15.56%	0.9998	0.3686
Mahalanobis	0.9730	14.00%	13.93%	0.9998	0.2800
Mahalanobis (w pert.)	0.9721	12.63%	12.57%	0.9998	0.3278
SVM (fc)	0.3955	99.46%	98.76%	0.9865	0.0088
SVM (conv)	0.3165	99.25%	98.55%	0.9855	0.0116
SVM (early layer)	0.6885	91.35%	90.71%	0.9957	0.0222
Predicted Entropy	**0.9766**	14.23%	14.15%	**0.9998**	0.3597
Mutual Information	0.9730	**12.61%**	**12.55%**	0.9998	**0.3717**

Table 4.7: IDD test split FN detection performance for various OOD detection methods. Best results are bolded and arrows indicate if a higher (↑) or lower (↓) score is better. For AUPR, “in” and “out” refer to whether or not the ID class or OOD class is considered to be the positive class, respectively.

4.2.1 Method Hyper-parameters

A summary of the hyperparameters used for each OOD detection method is given in Table 4.8. The minimum amount of perturbation tested provided the best performance for both

ODIN and Mahalanobis (w pert.), suggesting that either lower amounts should be tested or that OOD detection generally performs better without perturbation in these experiments. The OODL layer refers to the index of the feature extraction layer of the PyTorch VGG16 implementation, given in Appendix A.

Method	Epsilon	M.C. Samples	Kernel	Nu	OODL
Max Softmax	-	-	-	-	-
ODIN	0.0002	-	-	-	-
Mahalanobis	-	-	-	-	-
Mahalanobis (w pert.)	0.0002	-	-	-	-
SVM (fc)	-	-	rbf	0.001	-
SVM (conv)	-	-	rbf	0.001	-
SVM (early layer)	-	-	rbf	0.01	23
Predicted Entropy	-	10	-	-	-
Mutual Information	-	10	-	-	-

Table 4.8: Hyperparameters for each OOD detection method for the task of FN detection. The OODL refers to which layer in the VGG16 feature extraction network provided the best performance. The value of OODL corresponds to the PyTorch layer index of the implemented feature extractor that separates activation functions into discrete network layers.

4.2.2 FN Detection Risk-Coverage Plots

The Risk-Coverage plots for FN detection on both the KITTI and IDD datasets show much more varied performance across different OOD detection methods than in the FP detection experiments. The best performing methods were both of the Mahalanobis distance-based OOD detection techniques. Figure 4.5 shows the risk coverage curves on the KITTI validation set individually while Figure 4.6 compares these same curves using common axes. The same figures for the IDD test data are shown in Figures 4.7 and 4.8.

For the KITTI validation set, the Mahalanobis distance OOD detection method could be reduced from the baseline FN error rate (0.573%) to 0 by selecting a novelty threshold that provided dataset coverage of 60%. For the IDD test set, the non-SVM based OOD detection methods could reduce the baseline FNR or Risk from 0.739% to almost 0 and maintain dataset coverage around 80%. If it is desired to maintain dataset coverage over 90%, the Predicted Entropy, Mutual Information, and Softmax approaches give the best reduction in misclassification risk.

The reduction of risk to zero in these plots at relatively higher levels of dataset coverage show that the OOD detection methods are able to assign a high novelty scores to FN objects effectively for a number of methods. As a result, having a safe fallback system manually classify samples detected as being FNs could prevent the AV from mistakenly discarding FG instance and improve the overall safety of the system.

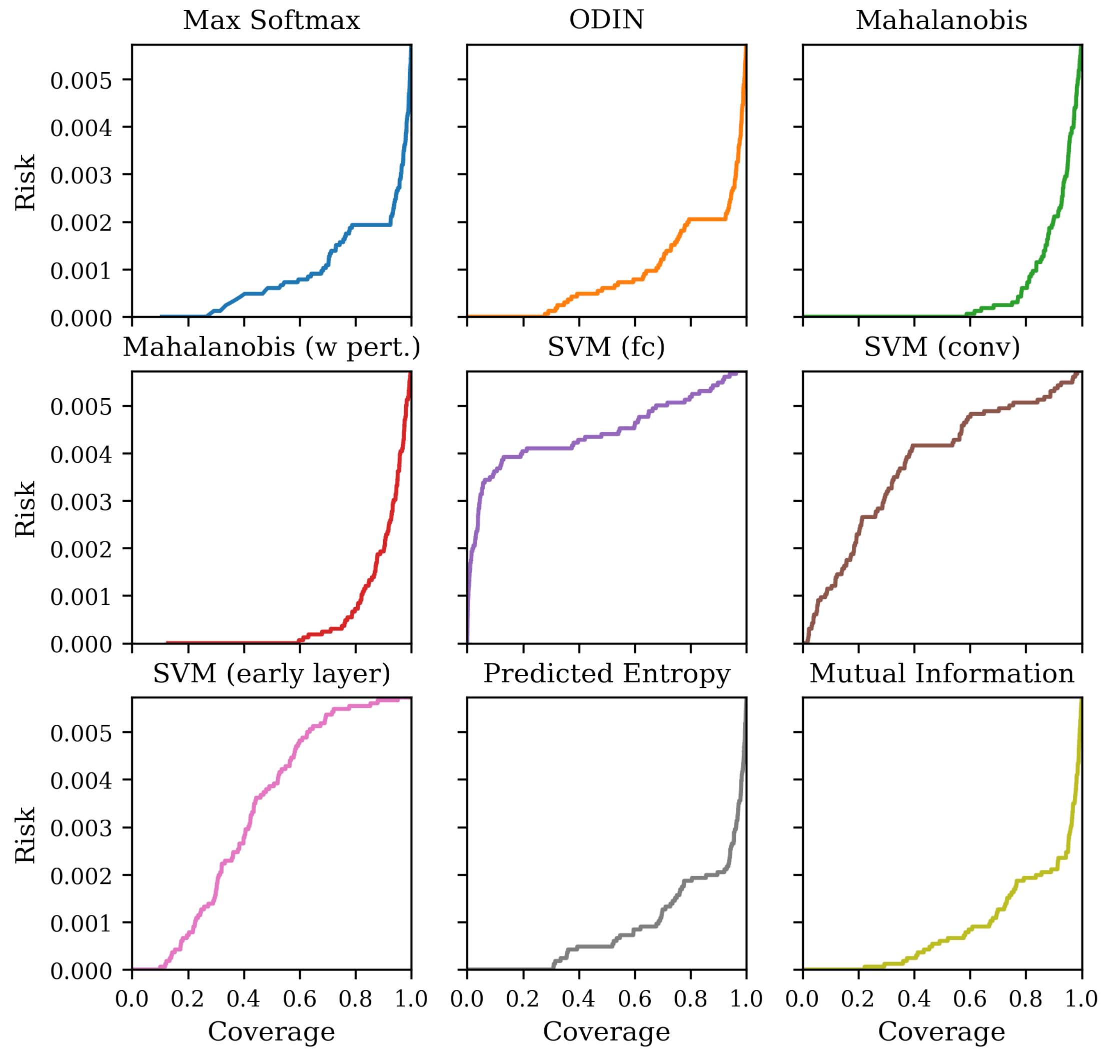

Figure 4.5: Risk-Coverage curves for FP detection on the IDD test set for various OOD detection methods.

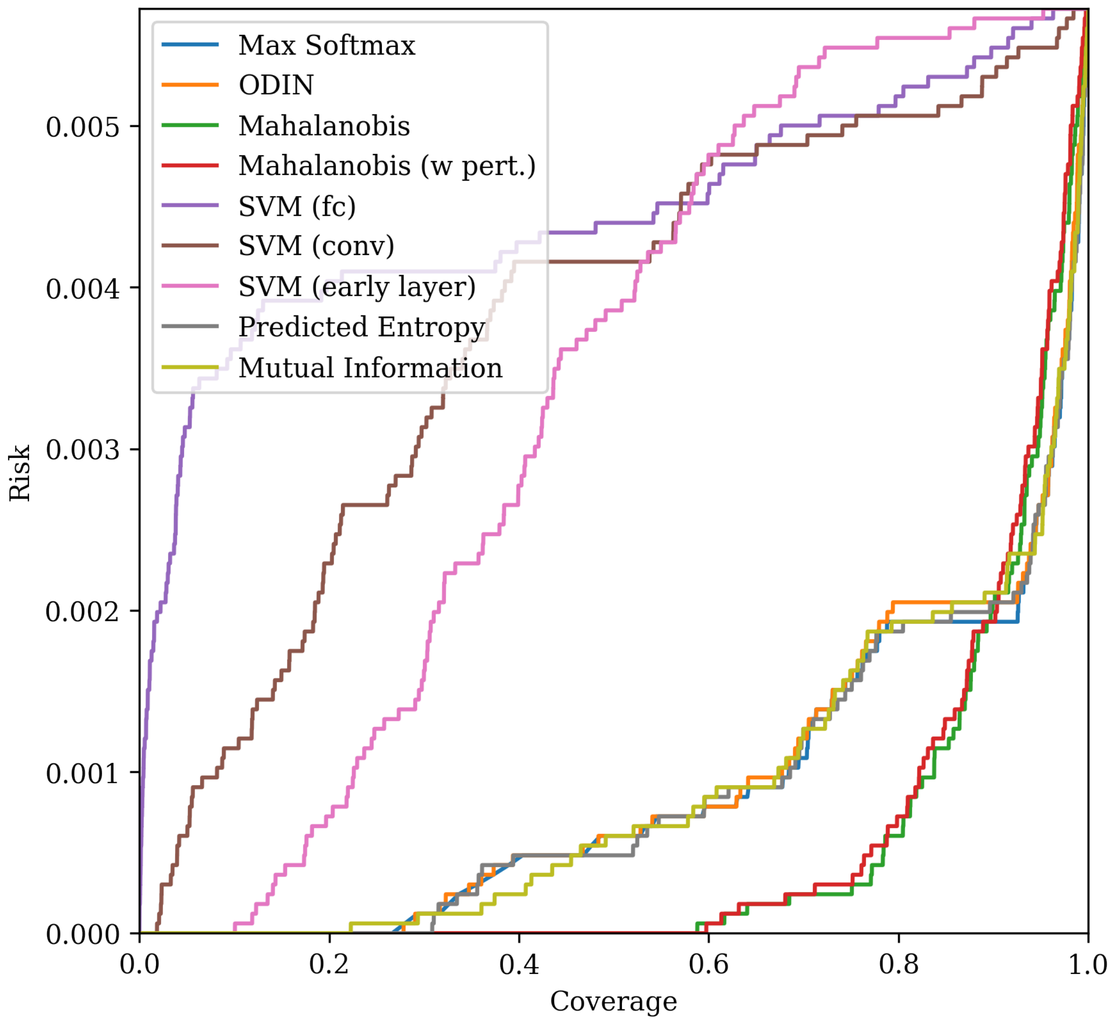

Figure 4.6: Comparison of risk-Coverage curves for FP detection on the IDD test set for various OOD detection methods.

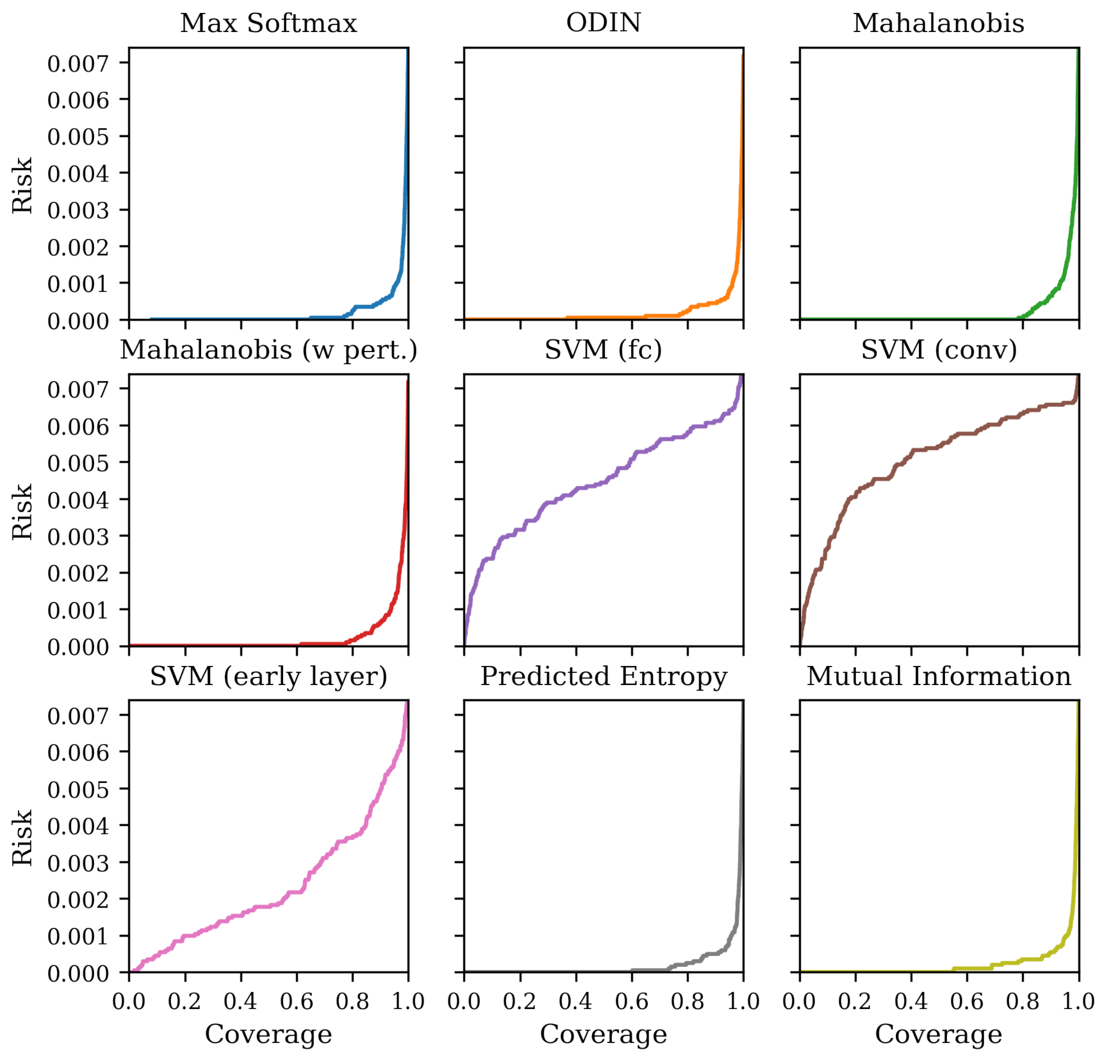

Figure 4.7: Risk-Coverage curves for FN detection on the IDD test set for various OOD detection methods.

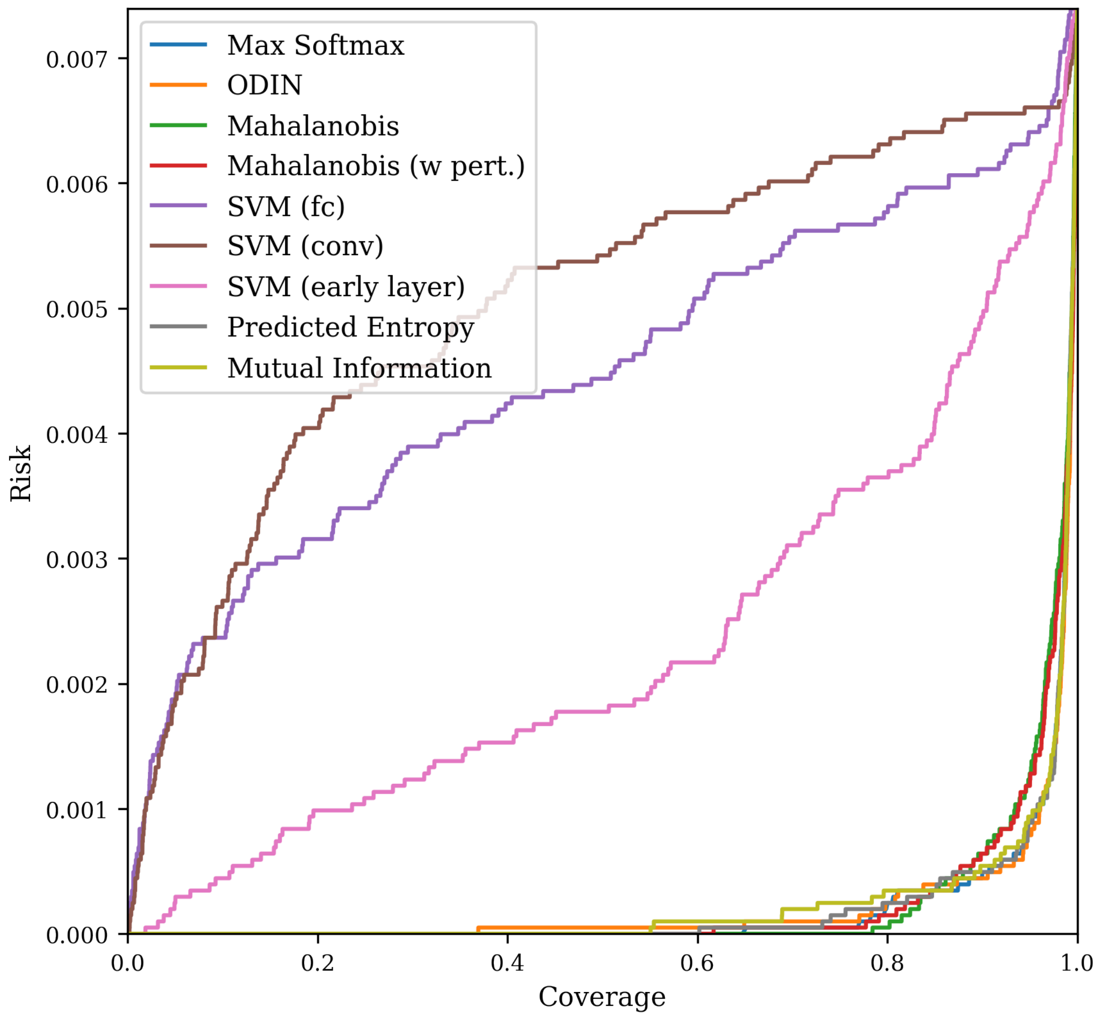

Figure 4.8: Comparison of risk-Coverage curves for FN detection on the IDD test set for various OOD detection methods.

4.3 Runtime comparison

Table 4.9 summarizes the runtime required for each method to output a novelty score for all objects in an image. Runtime was determined by averaging the runtime to obtain novelty scores for predicted FG objects (cars) over 10 iterations on the same image.

Determining a novelty score for each object using the SVM (early layer) method was by far the slowest technique since each proposal box was used to crop the original image and re-run that smaller image through the feature extraction network. A method for determining early layer features on an object-by-object basis from image-level feature map would significantly improve the runtime of this method and may improve results by incorporating object context within the features used by the SVM. It is currently not clear how to accomplish this optimization however.

The methods which only required one forward pass through the network, Max Softmax, Mahalanobis, SVM (fc), and SVM (conv) were the most efficient, while methods requiring multiple passes through the network like ODIN and Mahalanobis (w. pert) required more time. Similarly, the Predicted Entropy and Mutual Information Methods required more time to do Monte Carlo sampling via multiple passes through the final dropout layers of the network.

Method	Runtime (sec)
Max Softmax	0.126
ODIN	0.342
Mahalanobis	0.136
Mahalanobis (w pert.)	0.328
SVM (fc)	0.128
SVM (conv)	0.127
SVM (early layer)	12.3
Predicted Entropy	0.356
Mutual Information	0.357

Table 4.9: Inference time for each of the OOD methods. Times are averaged over 10 iterations on inference for the same input image scene.

Chapter 5

Conclusion and Future Work

This thesis established a framework for employing methods from the OOD detection literature for use within a two-stage object detection network. This included formulating a sensible application of these methods in the presence of a BG class. OOD methods were found to be applicable and sometimes useful for the tasks of false positive and false negative object detection in the predicted outputs of a Faster R-CNN object detector. A framework was described for generating samples of correctly and incorrectly labelled FG objects and BG image areas for testing FP and FN object detection methods. Multiple OOD detection methods from the image classification literature were modified to allow estimating a score of object novelty for objects predicted by the network.

Overall, the performance of OOD detection methods was found to only slightly improve performance over the baseline Max Softmax method. The use of the Predicted Entropy method, however, did provide a measurable boost to the AP of the model for detecting FG car objects. For FN detection, Predicted Entropy and Mutual Information were shown to be effective for detecting the small numbers of FN examples that were incorrectly eliminated in the final layers of the network.

Finally, the use of risk-coverage plots allowed for visualizing the trade-offs for having a network abstain from labelling samples with a high level of novelty relative to the training dataset. For FN samples, the risk of misclassification was shown to be reducible at the cost of using some slower, but risk-mitigating action to classify samples that appeared more unusual to the network.

5.1 Limitations and Future Work

This work explored a way that OOD detection methods could be applied in two-stage object detection networks, but there are some limitation to the approach taken. One major limitation was that the region proposal network of Faster R-CNN is not guaranteed to propose all regions containing foreground objects. As a result, OOD detection methods operating on features downstream from the RPNs have no opportunity to detect FNs that were not initially proposed by the RPNs. The FN detection experiments in this thesis were largely adapted from analogous FP detection methods, which allowed the use of the same OOD detection techniques for both tasks. A robust false negative detection system should operate instead or additionally at the earlier RPNs stage of the network to possibly allow catching all FN cases in an image. Such a system would be required to score all anchor box locations in an image for novelty though, which may be computationally expensive for some methods. Future work should explore implementing efficient FN detection methods that operate at this earlier RPNs stage. Furthermore, initial findings in the risk-coverage analysis show that it may be possible to bolster safety and reduce FN misclassification errors in an implemented driving system; it would be interesting to see if this pattern holds for a FN detector operating at the RPNs stage of a network.

This work also attempted to detect FP and FN errors on an object detection network designed to detect a single foreground object, cars. This was a limitation in that application of the temperature scaling method from the OOD detection literature was rendered useless by only having a single foreground class and the background class as classification outputs. Scaling logit layers before the softmax function of a binary classifier results in the same posterior as using unscaled logits, resulting in no effect for the improved distinction between ID and OOD samples as suggest by [24]. Furthermore, having a two-stage object detector for a single class provides little advantage over a simpler network that would only include the feature extraction and region proposal stages of Faster R-CNN. However, since one objective of this work was to detail how to implement these methods on a popular two-stage object detection architecture, the decision was made to retain the original architecture of the network. A single FG class was used in this work to simplified obtaining an object detection network with good performance on the KITTI dataset. The experiments in this thesis should ideally be repeated using a well-trained network designed to detect multiple FG object classes to allow the use of the temperature scaling method.

Another limitation of the methods employed was that the cases of FPs could have been divided into more specific categories on which to test the OOD detection methods. For example, it is likely to be easier to detect FP object proposals that do not overlap at all with a ground truth FG instance than it would be to detect a near-miss FP (i.e. a detection where the IoU with ground truth falls just below the required threshold). Another FP category could be created for detections occurring on analogous objects (e.g. detecting a van as a car), and detections of reflections or pictures of FG objects as the real thing (e.g. detecting a billboard that shows a car as a car). These different categories of FP objects present varying levels of danger to a safety-critical system and it would be useful to compare models based on what kind of mistakes they can catch. As a result, future work should consider these cases individually.

This thesis also strives to define the problem of OOD detection in object detection networks and discussed if the concept of OOD was sensible or useful when a catch-all BG class is one of the possible output class labels. While it was decided that declaring an object being OOD given this setup to be difficult or intractable, it may be possible to define the problem differently in a general object detection system that does not use a BG class label. A system that defines a class hierarchy of objects with broad class labels like “vehicle” as well as more narrow labels like “car”, “truck”, “van” could allow for a meaningful concept of OOD for previously unseen types of vehicles like autorickshaws. These new vehicles could be classified under the superclass of “vehicles”, but of an unknown type. Such a hierarchy would allow classifying at what semantic level the object becomes OOD, which could also be useful for determining what kind of risk-mitigating action should be taken as a result.

Panoptic segmentation networks [19] may also provide a potential solution to the ill-defined nature of the OOD object detection problem. [19] essentially performs semantic segmentation on an image scene with the addition of distinguishing between instances of the same class kind of object. These networks require all pixels in an image scene to have a class label and there is thus no BG label. Using OOD detection methods for segmentation, such as those proposed by [4] and [15], with a panoptic segmentation network, would likely allow for detecting anomalous object instances by clustering these areas with high pixel-level uncertainty. This is a promising direction for future work.

Finally, this work does not attempt to inform on how to implement OOD detection methods for use on more modern single-stage object detection networks. It should be possible to adapt the methods and techniques used to modern single-stage object detectors like SSD [26] and YOLO [33] and some investigation into doing so may be an interesting research topic. Because there are fewer stages at which BG objects are eliminated in single-stage object detection networks, it can be safely assumed that FN detection techniques would evaluate all possible objects, unlike in this work. However, these models present new challenges primarily in the number of proposal boxes that need to be evaluated, which may be computationally expensive for some OOD detection methods.

References

[1] Vahdat Abdelzad, Krzysztof Czarnecki, Rick Salay, Taylor Denounden, Sachin Vernekar, and Buu Phan. Detecting Out-of-Distribution Inputs in Deep Neural Networks Using an Early-Layer Output, 2019.

[2] Erdi C ¸ Allı, Keelin Murphy, Ecem Sogancioglu, and Bram Van Ginneken. FRODO: Free rejection of out-of-distribution samples: application to chest x-ray analysis. In *Medical Imaging with Deep Learning*, 2019.

[3] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete Problems in AI Safety. *CoRR*, abs/1606.0, 2016.

[4] Matthew Angus. *Towards Pixel-Level OOD Detection for Semantic Segmentation*. PhD thesis, University of Waterloo, 2019.

[5] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3D Object Proposals Using Stereo Imagery for Accurate Object Class Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1259–1272, 2018.

[6] Yun Chen. chenyuntc/simple-faster-rcnn-pytorch, 2019.

[7] Xinxin Du, H. Ang Marcelo Jr., Sertac Karaman, and Daniela Rus. A general pipeline for 3d detection of vehicles. *CoRR*, abs/1803.0, 2018.

[8] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[9] Brendan Fortuner. Loss Functions — ML Glossary documentation, 2017.

[10] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *33rd International Conference on Machine Learning, ICML 2016*, volume 3, pages 1651–1660, 6 2016.

[11] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[12] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2015 Inter, pages 1440–1448, 2015.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, Cambridge, Massachusetts, 1 edition, 2016.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, pages 770–778. IEEE Computer Society, 12 2016.

[15] Dan Hendrycks, Steven Basart, Mantas Mazeika, Mohammadreza Mostajabi, Jacob Steinhardt, and Dawn Song. A Benchmark for Anomaly Segmentation, 2019.

[16] Dan Hendrycks and Kevin Gimpel. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *Proceedings of International Conference on Learning Representations*, 10 2017.

[17] Dan Hendrycks, Mantas Mazeika, and Thomas G. Dietterich. Deep Anomaly Detection with Outlier Exposure. In *International Conference on Learning Representations*, 12 2019.

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 2261–2269, 2017.

[19] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Doll\'ar. Panoptic Segmentation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9396–9405, 2019.

[20] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *. . . Science Department, University of Toronto, Tech. . . .*, pages 1–60, 2009.

[21] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 12 2015.

[22] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. *http://yann.lecun.com/exdb/mnist/*, 2010.

[23] Kibok Lee, Kimin Lee, Honglak Lee, and Jinwoo Shin. A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks, 7 2018.

[24] Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks. In *Proceedings of International Conference on Learning Representations*, 6 2017.

[25] Tsung Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal Loss for Dense Object Detection. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 2999–3007, 8 2017.

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9905 LNCS, pages 21–37, 12 2016.

[27] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, volume 2018-Decem, pages 7047–7058, 2018.

[28] Stephen R. Mounce, Richard B Mounce, and Joby B Boxall. Novelty detection for time series data analysis in water distribution systems using support vector machines. *Journal of Hydroinformatics*, 13(4):672–686, 2011.

[29] Kevin Murphy. *Machine Learning: A probabilistic perspective.* MIT Press, Cambridge, Massachusetts, 1 edition, 2012.

[30] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[31] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 779–788, 6 2016.

[33] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *CoRR*, abs/1804.0, 4 2018.

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 6 2017.

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 12 2015.

[36] Rishabh Shukla. L1 vs. L2 Loss function, 7 2015.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.

[38] Karanjit Singh and Shuchita Upadhyaya. Outlier Detection: Applications And Techniques. *International Journal of Computer Science Issues*, 9, 2012.

[39] C. Surace and K. Worden. Novelty detection method to diagnose damage in structures: an application to an offshore platform. In *Proceedings of the International Offshore and Polar Engineering Conference*, volume 4, pages 64–70. International Society of Offshore and Polar Engineers, 1 1998.

[40] David M.J. Tax and Robert P.W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20(11-13):1191–1199, 1999.

[41] Girish Varma, Anbumani Subramanian, Anoop Namboodiri, Manmohan Chandraker, and C V Jawahar. IDD: A dataset for exploring problems of autonomous navigation in unconstrained environments. In *Proceedings - 2019 IEEE Winter Conference on Applications of Computer Vision, WACV 2019*, pages 1743–1751, 2019.

[42] Anh Vo. Deep Learning – Computer Vision and Convolutional Neural Networks – Anh Vo.

[43] William Wang, Angelina Wang, Aviv Tamar, Xi Chen, and Pieter Abbeel. Safer Classification by Synthesis. *arXiv e-prints*, 11 2017.

[44] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny ImageNet Visual Recognition Challenge, 2017.

[45] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop. *arXiv preprint arXiv:1506.03365*, 2015.

[46] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. In *British Machine Vision Conference 2016, BMVC 2016*, volume 2016-Septe, pages 1–87, 2016.

APPENDICES

Appendix A

PyTorch Faster R-CNN Architecture

A.1 Model Layers

The following is a printout of the layers in the Faster R-CNN architecture used for this thesis.

```
FasterRCNNVGG16(
  (extractor): Sequential(
    (0): Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=3, stride=1, padding=1)
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
    (5): Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=3, stride=1, padding=1)
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
    (10): Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
    (17): Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
    (24): Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
    (29): ReLU(inplace=True)
  )
  (rpn): RegionProposalNetwork(
    (conv1): Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
    (score): Conv2d(512, 18, kernel_size=1, stride=1)
    (loc): Conv2d(512, 36, kernel_size=1, stride=1)
  )
  (head): VGG16RoIHead(
    (classifier): Sequential(
      (0): Linear(in_features=25088, out_features=4096, bias=True)
      (1): ReLU(inplace=True)
      (2): Dropout(p=0.5, inplace=False)
      (3): Linear(in_features=4096, out_features=4096, bias=True)
      (4): ReLU(inplace=True)
      (5): Dropout(p=0.5, inplace=False)
    )
    (cls_loc): Linear(in_features=4096, out_features=8, bias=True)
    (score): Linear(in_features=4096, out_features=2, bias=True)
    (roi): RoIPooling2D()
  )
)
```