Towards Pixel-Level OOD Detection for Semantic Segmentation

by

Matthew Angus

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

Waterloo, Ontario, Canada, 2019

Abstract

There exists wide research surrounding the detection of out of distribution sample for image classification. Safety critical applications, such as autonomous driving, would benefit from the ability to *localise* the unusual objects causing an image to be out of distribution. This thesis adapts state-of-the-art methods for detecting out of distribution images for image classification to the new task of detecting out of distribution pixels, which can localise the unusual objects. It further experimentally compares the adapted methods to a new dataset derived from existing semantic segmentation datasets, proposing a new metric for the task. The evaluation shows that the performance ranking of the compared methods successfully transfers to the new task.

Acknowledgements

I would like to thank my supervisor and colleagues for their helpful feedback and constructive criticism.

Dedication

To the friends and family that have supported me throughout my life.

Table of Contents

List of Tables ix

List of Figures x

Notation xiii

1 Introduction 1

2 Background 4

2.1 Convolutional Network Primitives 4

2.1.1 Convolution 4

2.1.2 Pooling 5

2.1.3 Batch Normalisation 5

2.1.4 Activation Functions 7

2.2 Image Processing 8

2.3 Out-of-Distribution Detection 10

2.3.1 Auto-Encoders 10

2.3.2 Generative Adversarial Networks 11

2.3.3 Pixel-Wise OOD 11

3 Experimental Setup **13**
3.1 Network Architecture 13
3.2 Adaptations for Semantic Segmentation 17
3.2.1 OOD Detection Methods 18
3.2.1.1 Dropout 19
3.2.1.2 Max Softmax 19
3.2.1.3 ODIN 20
3.2.1.4 Mahalanobis 21
3.2.1.5 Confidence Estimation 23
3.2.1.6 Entropy 25
3.2.1.7 Summary 26
3.2.2 Adapting Datasets 26
3.2.2.1 Training 27
3.2.2.2 Evaluation 27
3.2.2.3 Dataset Statistics 30
3.3 Performance Metrics 30
3.3.1 AUROC 31
3.3.2 AUPRC 32
3.3.3 FPRatTPR 33
3.3.4 DE 34
3.3.5 MaxIoU 34
3.4 Removing Boundary Artefacts 34
3.5 Experiments 36

4 Results and Discussion **38**
4.1 RQ1: Architecture Modifications 38
4.2 RQ2: Class Boundary Artefacts 39
4.3 RQ3: Method Performance 39

4.4 RQ3: Qualitative Discussion 40
4.5 RQ4: Runtime comparison 45
4.6 Threshold Selection 48

5 Conclusions and Future Work **50**
5.1 Limitations 50
5.2 Future Work 51

References **52**

List of Tables

3.1 Summary of key properties of the methods compared 26

4.1 mIoU values for modified PSPNet architectures showing efficacy for original segmentation task 38

4.2 Change in performance of each method after removing class boundary artefacts on the SUN dataset. All values are scaled $\times 10^3$. Bolded values indicate when the performance increases. The arrow beside each metric indicates if larger values are better ($\uparrow$) or smaller values are better ($\downarrow$). 39

4.3 Runtime comparison of all methods 48

List of Figures

1.1 Image from the LostAndFound dataset [38], where two unlikely objects (storage crates) are almost entirely incorrectly predicted to be road. The Max Softmax method – described in detail later – clearly highlights these crates as OOD. (best viewed in colour) 2

2.1 Example of a 3×3 convolution, sliding weights over the input feature map [16]. 5

2.2 3×3 max pooling example [16] 6

2.3 ReLU function 8

2.4 Example image from the SUN dataset [56], with labels for each task. . . . 9

3.1 The squence of Convolution, Batch Nomalisation, then ReLU is a ConvBlock. 14

3.2 Sequence of operations for a single ResNetBlock. The thick blocks represent a depth of 4 times that of the smaller blocks. The decrease in depth is a bottleneck in the ResNetBlock. The dimensions below the ConvBlocks indicate the internal convolution kernel shape. 14

3.3 Sequence of operations for a the ResNet feature extractor. The thick blocks represent a depth of 4 times that of the input. The dimensions below the convolution blocks indicate the kernel shape. The text below a ResNetBlock takes the from $n\,@\,f \times d$, where f is the height and width multiplier, d is the depth of the block, and n is the number of times that block configuration is repeated. Each decrease in height and width represents halving each. Here the first ConvBlock halves the height and width. 15

3.4 The PSPNet architecture with ResNet50 backbone. The number under each average pooling layers and the resize layers indicate the change in scale (height and width). The kernel size is shown below each ConvBlock. 16

3.5 Example of how c influences the prediction vector, giving “hints” by increasing the correct prediction when c is low. 23

3.6 Example image and ground truth from the Cityscapes [11] dataset. Best viewed in colour. 27

3.7 2D greyscale Perlin noise example. 29

3.8 Examples of colour noise images with different lattice scales. 29

3.9 Fraction of dataset for each class. 30

3.10 Examples of ROC curves for different qualities of classifiers, with corresponding area under the curve values. (best viewed in colour) 32

3.11 Examples of PR curves for different qualities of classifiers, with corresponding area under the curve values. (best viewed in colour) 33

3.12 Examples of IoU curves for different qualities of classifiers, with corresponding maximum values (best viewed in colour). 35

3.13 Stages of boundary artefact removal process from Max Softmax OOD prediction 36

4.1 Comparison of methods on different datasets. The arrow on the y-axis label indicates if a larger value is better (↑) or a smaller value is better (↓). Each group of bars are labelled by the dataset used for evaluation. Note the change in scale for the DE plot. 41

4.2 Reference class colours for ground truth and predictions. Best viewed in colour. 41

4.3 Examples take from the Cityscapes dataset (best viewed in colour). Since this is the training set there are no OOD pixels. See Figure 4.2 for Ground Truth and Class Prediction colour mappings. 42

4.4 Examples taken from the modified SUN datatset (best viewed in colour). These examples are well classified with respect to the performance metrics. The OOD pixels in the ground truth are labelled with bright yellow. 43

4.5 Examples taken from the modified SUN datatset (best viewed in colour). These examples are poorly classified with respect to the performance metrics. The OOD pixels in the ground truth are labelled with bright yellow. . 44

4.6 Examples taken from the modified LostAndFound dataset (rows 1 to 6), normal noise (row 7), uniform noise (row 8), and Perlin noise (row 9) – best viewed in colour. The OOD pixels in the ground truth are labelled with bright yellow. 46

4.7 Comparison of methods performance on the modified SUN dataset against runtime. The arrow on the y-axis label indicates if a larger value is better (↑) or a smaller value is better (↓). 47

4.8 Thresholded OOD predictions for ODIN, using MaxIoU and ROC to select the optimal threshold. Black pixels are ID predictions, Yellow pixels are OOD predictions. 49

Notation

The notation followed is that of Goodfellow *et al*. [18] with some minor modifications.

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector, ordered collection of scalars
$\boldsymbol{A}$	A matrix, ordered collection of vectors
$\mathsf{\mathbf{A}}$	A tensor, ordered collection of matrices
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}, \mathbb{N}$	The set of real and natural numbers
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, *i.e.* the elements of $\mathbb{A}$ that are not in $\mathbb{B}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathbf{\mathsf{A}}$
$\mathbf{\mathsf{A}}_{:,:,i}$	2-D slice of a 3-D tensor

Linear Algebra Operations

$\boldsymbol{A}^T$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^{-1}$	Inverse of $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathbf{\mathsf{X}}} y$	Tensor containing derivatives of y with respect to $\mathbf{\mathsf{X}}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. For brevity $f(\boldsymbol{x})$ is also used without $\boldsymbol{\theta}$
$\log(x)$	Natural logarithm of x
$\|\|\boldsymbol{x}\|\|_p$	l_p norm of $\boldsymbol{x}$
$\mathbf{1}_{\text{condition}}$	is the indicator function, *i.e.* 1 if the condition is true, 0 otherwise

The function notation is often abused to simplify equations. Functions with scalar arguments can be given vector, matrix, or tensor input. This implies that the function is applied element wise. For example, let $f : \mathbb{R} \mapsto \mathbb{R}$, then $\boldsymbol{b} = f(\boldsymbol{a})$ is shorthand for $b_i = f(a_i)$ for all valid i.

Datasets and Distributions

$\boldsymbol{x}^{(i)}$	The i-th example (input) from a dataset
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$

Chapter 1

Introduction

The surge in availability of data and programmable graphics processing units (GPUs) has contributed to a drastic increase in the popularity of machine learning (ML) over the last few years. It has spread into many new fields with a wide variety of applications. Some of these applications are safety critical, such as autonomous driving, where errors in the output of an ML algorithm could be fatal. This thesis will look at a way to improve the existing methods in order to make them more trustworthy.

The inherent risk when using machine learning methods in safety critical areas is that neural networks are not robust or fault tolerant. That is, neural networks typically output a confidence value that can't always be trusted. Neural networks are a series of non linear operations, therefore it is not trivial to analyse when an erroneous state has been reached.

Supervised learning approaches are trained using datasets that have inputs and target outputs. The training datasets are assumed to have some underlying, unknown or unknowable, data-generating distribution. Given an input that is not from this data generating distribution, the output should not be trusted. This type of input is termed "out of distribution" (OOD). One mode of action to make neural networks more robust to these OOD examples is to detect when such an example is encountered. This is the task of OOD detection.

A large body of research exists for detecting entire images as OOD for the task of image classification. Image-level OOD detection outputs a classification for the entire image. This coarse level of detection may be inadequate for many safety critical applications. Most of any image taken from an onboard camera will have the majority of its pixels as "in distribution" (ID), *i.e.* an image of a road scene with cars, people, and roadway, but an unusual object that was not part of the training set may cause only a small number of OOD pixels. Extending the framework to semantic segmentation networks, that is networks that perform pixel-level classification, will allow each pixel to have an "in" or "out of" distribution classification. Applied to autonomous driving, groups of pixels classified as OOD could be considered as unknown objects. Depending on the location of the unknown objects, a planner could then proceed with caution or hand over control to a safety driver. Another application is automatic tagging of images with OOD objects, which could then be sent for human labelling. Figure 1.1 shows a segmentation failure case where OOD detection is beneficial. In the centre image, the two crates are predicted as road. The right image of this figure shows the result of pixel-level OOD detection using one of the proposed methods, which clearly shows higher values for the unusual objects.

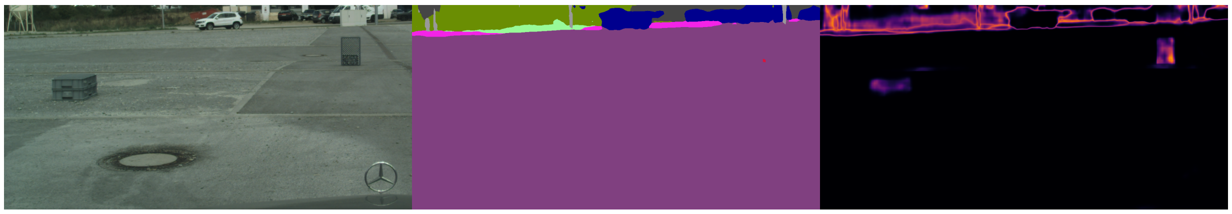


Figure 1.1: Image from the LostAndFound dataset [38], where two unlikely objects (storage crates) are almost entirely incorrectly predicted to be road. The Max Softmax method – described in detail later – clearly highlights these crates as OOD. (best viewed in colour)

This thesis adapts existing state-of-the-art image-level OOD detection and uncertainty estimation methods to the new task of pixel-level OOD classification and compares their performance on a new dataset designed for this task. In addition to adapting the methods, it addresses the question of whether the best-performing image-level methods maintain their performance when adapted to the new task. In order to answer this question, it also proposes pixel-level OOD detection performance metrics, drawing from both existing image-level OOD detection and semantic segmentation performance metrics. Furthermore, a new dataset for pixel-level OOD detection with test images that contains both pixels that are in distribution and pixels that are out of distribution. Somewhat surprisingly, the evaluation shows that the best performing pixel-level OOD detection methods were derived from image-level OOD detection methods that were not necessarily the best performing on the image-level OOD detection task.

In summary, the contributions of this paper are the following:

- adaptation of image-level OOD detection methods to pixel-level OOD detection and their evaluation;

- introduction of a pixel-level OOD detection evaluation dataset derived from existing segmentation datasets; and
- a new metric for pixel-level OOD detection, called MaxIoU.

This thesis is organised as follows: Chapter 2 covers the necessary background on convolutional neural networks and the image processing tasks of concern, concluding with the related and alternative OOD detection methods. Chapter 3 provides details on the experiments that are performed. It describes the neural network used, what changes are needed when moving from image-level OOD detection to pixel-level detection, the performance metrics used, a post-processing step introduced to remove artefacts from the resulting OOD predictions, and finally the research questions and the experiments behind them. Chapter 4 contains results and discussion. Chapter 5 presents the conclusions and future work.

Chapter 2

Background

This chapter covers the formulation of common image-processing neural-network operations as well as some common architectures for semantic segmentation networks. It concludes by discussing some related OOD detection methods. These concepts form the foundation for the topics covered by this thesis. Please refer to Appendix for details on the notation used.

2.1 Convolutional Network Primitives

Convolutional neural networks (CNNs) are very prevalent in computer vision, and drastically outperform other neural network architectures on image processing tasks. This is due to their inherent exploitation of spatial correlations in images and the benefits of weight sharing for training speedup. This section briefly covers some network primitives that are common to image processing, and are needed to understand this work.

2.1.1 Convolution

The convolution operator is a method of weight sharing. It reduces the number of weights required, compared to fully connected networks, while leveraging the spatial correlations in images. In general, the convolution operator is a method of composing two functions. In the case of neural networks these functions are discrete. Given an input feature map

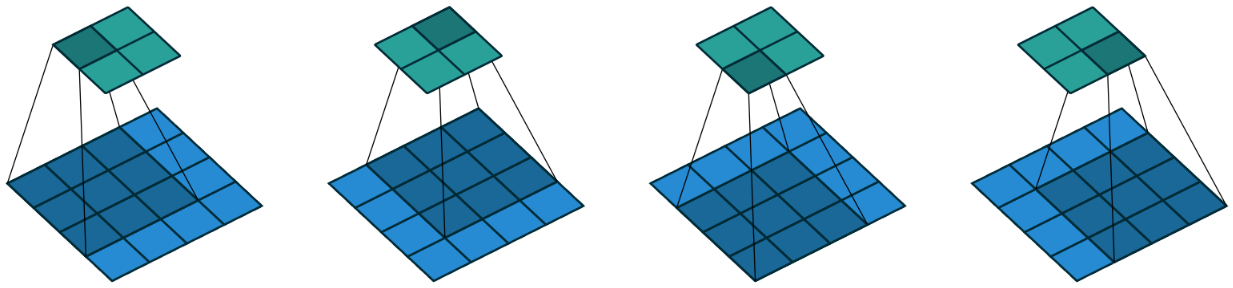


Figure 2.1: Example of a 3×3 convolution, sliding weights over the input feature map [16].

$\mathbf{X} \in \mathbb{R}^{h \times w \times d_1}$ and a weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n \times d_1 \times d_2}$, the discrete convolution is:

$$\begin{aligned} Y_{i,j,k} &= (\mathbf{W} * \mathbf{X})_{i,j,k} \\ &= \sum_{p=1}^{m} \sum_{q=1}^{n} W_{p,q,:,k} \cdot \mathbf{X}\left[i + \left(p - \left\lceil \frac{m}{2} \right\rceil\right), j + \left(q - \left\lceil \frac{n}{2} \right\rceil\right), :\right] \end{aligned} \tag{2.1}$$

where the range of i and j depend on whether padding with zeros is desired. When padding with zeros, $y \in \mathbb{R}^{h \times w \times d_2}$. Since indexing into the $\mathbf{X}$ tensor is non-trivial, $\mathbf{X}[\cdot]$ is used for better clarity. This operation is often viewed as sliding the weight matrix over an input feature map as shown in Figure 2.1.

There are several variants of the convolution operator, such as atrous (or dilated) [7], and transposed convolutions [58]. These variants have similar definitions, but with variations on the indexing used.

2.1.2 Pooling

Pooling is often used as a method for dimensionality reduction. Pooling summarises the activations of the input over some neighbourhood [18]. The summary function can vary, but it is usually the maximum value, or the mean of the neighbourhood. Figure 2.2 shows an example of pooling using the maximum summary function.

2.1.3 Batch Normalisation

Batch normalisation [23] is a method of reducing the covariate shift. The covariate shift was originally described as the change in distribution between the training dataset and

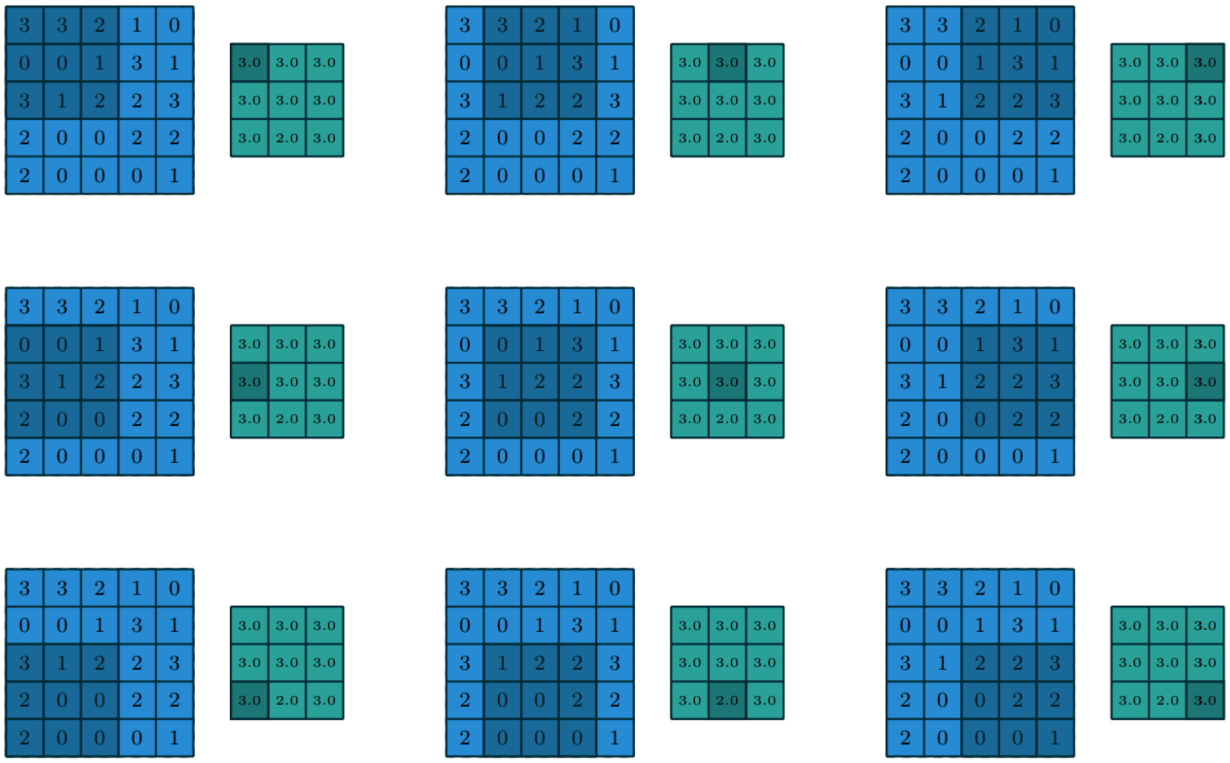


Figure 2.2: 3×3 max pooling example [16]

the evaluation dataset [47]. The same principle can be applied to the input of each layer. Let $f_l(\boldsymbol{x};\theta_l)$ be the l^{th} transformation function of the network, where $\boldsymbol{x}$ is the output of the previous layer and θ_l are the parameters to optimise over. Let $\boldsymbol{u} = f_l(\boldsymbol{x};\theta_l)$, and $\boldsymbol{v} = f_{l+1}(\boldsymbol{u};\theta_{l+1})$ for some layer l and input $\boldsymbol{x}$. During training, θ_l will be updated many times, which in turn will change the distribution of $\boldsymbol{u}$. This distribution change is the internal covariate shift problem [23], since θ_{l+1} will require updating to compensate for the shift.

Image Whitening could be performed for each layer over the whole dataset; however, it is inefficient as well as not differentiable everywhere [23]. Therefore, Ioffe and Szegedy [23] developed the batch based version which does not perform decorrelation. Batch normalisation is computed as:

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}^{(i)} \tag{2.2}$$

$$\boldsymbol{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^2 \tag{2.3}$$

$$\hat{\boldsymbol{x}}^{(i)} = \frac{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \tag{2.4}$$

$$\boldsymbol{y}^{(i)} = \gamma \cdot \hat{\boldsymbol{x}}^{(i)} + \beta \tag{2.5}$$

where $\boldsymbol{\mu}$ is the batch mean, $\boldsymbol{\sigma}^2$ is the batch variance, $\hat{\boldsymbol{x}}^{(i)}$ is the normalised input, ϵ is a small value added for numerical stability, γ and β are learned parameters, and $\boldsymbol{y}^{(i)}$ is the output. $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are tracked using a moving average during training to approximate the population mean and variance that is used during inference. Here $\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}$ are the inputs and outputs of the batch normalisation layer, with a batch with size m.

2.1.4 Activation Functions

Activation functions are non-linear functions applied to the output of a transformation layer (convolution, batch normalisation, *etc.*). Rectified Linear Unit (ReLU) [24] is a common activation function. It is a piecewise linear function, meaning it has the advantage of easier optimisation and better generalisation [18]. The function $\text{ReLU} : \mathbb{R} \to \mathbb{R}$ is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{2.6}$$

ReLU is applied element-wise to the input. Figure 2.3 shows a plot of the ReLU function.

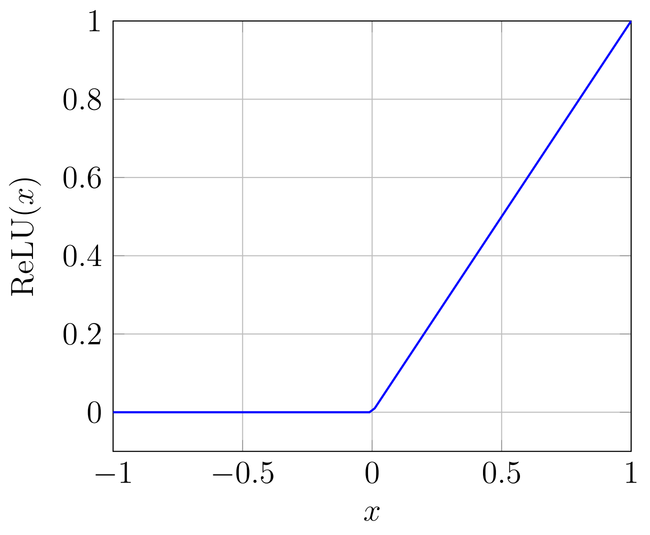


Figure 2.3: ReLU function

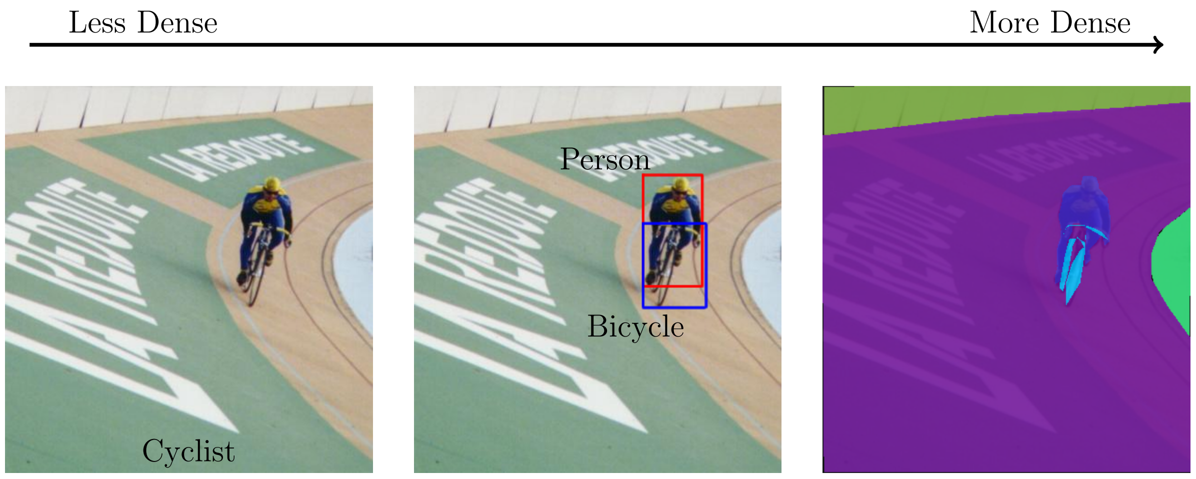
The next function is used to normalise the final outputs so that they are in the range $[0, 1]$, which represents the probability distribution of a random variable with n possible values [18]. This function is called the softmax function and is defined as softmax:$\mathbb{R}^n \mapsto \mathbb{R}^n$

$$\text{softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.7}$$

The softmax function, in general, is applied to the last layer of a neural network to produce class probabilities.

2.2 Image Processing

There are three main image classification tasks. Each of these tasks have a different level of localisation and density of output. They are: image-level classification, object detection, and semantic segmentation. Image-level classification does not have any localisation, and typically only has one main object in an image to classify. The goal of object detection is to locate foreground objects of interest within an image. This consists of determining a minimal bounding box that the whole object lies within and providing a classification of said object. This thesis is not concerned with regressing bounding boxes, therefore object detection is outside the scope of this work. Semantic segmentation has significantly more



(a) Classification (b) Object Detection (c) Semantic Segmentation

Figure 2.4: Example image from the SUN dataset [56], with labels for each task.

fine grained localisation, where the goal is to classify individual pixels, outlining the shape of all object classes. Segmentation sometimes has the added difficulty of classifying all pixels in an image, both foreground and background. Figure 2.4 shows an example of an image with each of these types of labels.

Many of the top performing segmentation architectures borrow their feature extractor from image classification literature [17]. A possible factor for using feature extractors from image classification is due to the lack of large scale datasets such as ImageNet [12] with over 14 million images, with labels for other tasks. The major innovations for segmentation are how these common feature extractors are augmented. The final dense layers are removed and replaced with some form of upscaling. Another common practice is to create some skip connections from shallow layers in the network to penultimate (or near penultimate). This allows the network to incorporate high-level features with low-level features into the final prediction.

Segnet [2] uses VGG16 [49] as an encoder, and mirrors the architecture with max pooling replaced with un-pooling (using max pooling indices) to increase resolution. DeepLabv3+ [8] uses the Xception [9] feature extractor with a shallow to deep skip connection, multi-scale atrous convolutions, and bilinear up-sampling. PSPNet [59] uses the ResNet [20] feature extractor with multi-scale average pooling. PSPNet is used in this work, and is described in much greater detail later.

2.3 Out-of-Distribution Detection

This section will give a brief overview of the categories of OOD detection, as well as how well-suited they are to localising the OOD source from within the input. Note that there are three related terms throughout the literature – anomaly detection, novelty detection and out-of-distribution detection. They are often used interchangeably, however, a distinction will be made here. Anomaly detection will refer to the task of detecting oddities within a sequence of data, and possibly localising those oddities. OOD detection will refer to the detection of samples that are oddities with respect to the data generating distribution.

OOD detectors have many applications. The common trend between these applications is the notion of rare events. The common positive classes are well-represented in data. However, there may only be a few labelled samples of the rare positive event. Some examples in the literature include: detecting brain tumours [39], detecting breast cancer [10], and detecting obstacles in fields for automated farming robots [41].

There are a number of techniques in the literature to detect OOD samples. Zhou *et al*. [60] use a threshold on the confidence output from a random forest. Ding *et al*. [15] estimate the level set function of the training data (density estimates) and threshold the density to achieve an ID/OOD decision boundary. Bodesheim *et al*. [5] train a kernel null space model with the k-nearest neighbours of the test image from the training images. These approaches are algorithmic or classical ML based methods. There are many methods that use support vector machines (SVMs) [45, 51, 25], and many that use kernel methods [34, 6, 42]. Given the success of deep learning in computer vision, the focus of this section will be on methods that have been applied to deep learning. There are a number of classes of neural networks that are used to detect OOD samples. The main types are classification, auto-encoders, and generative adversarial networks. The classification methods are used for adapting to pixel-level OOD detection, therefore they are discussed later.

2.3.1 Auto-Encoders

Auto-encoders (AEs) are a form of unsupervised learning that have two components – the encoder and the decoder. The encoder attempts to compress the input maintaining only the necessary information (called the latent features), then the decoder attempts to reconstruct the input from the compressed latent features. One method of detecting OOD samples is using the reconstruction error [40, 52]. The assumption is that the AE learns the manifold that the data lies on, and inputs that do not exist on that manifold will not be properly reconstructed. Denouden *et al*. [13] extend this by doing a weighted sum of the reconstruction error and the Mahalanobis distance of the encoded latent features from the mean of the training dataset. The latent features have also been used for the distance to the k^{th}-nearest neighbour to determine OOD samples [19].

2.3.2 Generative Adversarial Networks

Another class of network architectures used are generative adversarial networks (GANs). A GAN has two components, the generator denoted G and a discriminator denoted D. D is trained to distinguish between real images from a dataset and images generated by G. At the same time, G is given a random vector $\boldsymbol{z}$ sampled from some pre-determined distribution, and is trained to output images that are meant to "fool" D. G and D are trained via a minimax game (minimising w.r.t. G and maximising w.r.t. D). Schlegl *et al*. [44] use a GAN for OOD detection. First, through backpropagation, a vector $\boldsymbol{z}$ is iteratively updated such that $G(\boldsymbol{z})$ is close to the image in question $\boldsymbol{x}$, and $\boldsymbol{z}$ is close to the pre-determined distribution. $|\boldsymbol{x} - G(\boldsymbol{z})|$ is used to detect OOD pixels, and a convex combination of the G loss and D loss are used to determine if the sample $\boldsymbol{x}$ is OOD. Wang *et al*. [55] perform a very similar anaylsis on the MNIST dataset [29].

2.3.3 Pixel-Wise OOD

As of writing, there are very few works researching pixel-level OOD detection. Bevandic *et al*. [3] train a segmentation network with two datasets – one ID and one OOD dataset. The network learns to classify between the two on a per pixel-level, and then it is compared to the Max Softmax baseline. The major flaw with this approach is that the network learns its weights directly from OOD samples and is specific to an OOD dataset.

Pham *et al*. [37] create an open world dataset. Known object labels are drawn from the COCO dataset [33], and labels drawn from the NYU dataset [48] are relabelled as unknown if the class doesn't exist in COCO. Using a class specific object detector, a boundary detector and simulated annealing, a generic object instance-level segmentation algorithm is developed, which is evaluated on the new dataset. This approach splits the image into visually distinct connected regions, meaning that two portions of the same object could be detected as two objects. Thus, it is orthogonal to pixel-level OOD detection.

Sabokrou *et al*. [43] perform anomaly detection with localisation on surveillance videos. They use AlexNet [28] pre-trained on ImageNet [12] as a feature extractor. Only the output of the second or third layer is used, thus maintaining a smaller receptive field for easier localisation. The input to AlexNet is the past six images pixel-wise averaged in groups of two (*i.e.* three images). The mean and covariance are computed over all extracted features from the second AlexNet layers across the three inputs. An initial three class classification is performed using the Mahalanobis distance of each feature. Two thresholds are used, one to determine normal and one to determine abnormal features. Features between these two thresholds are considered suspicious. Suspicious features are fed to the third layer of AlexNet and then a second distance threshold classifier. The main assumption in this work is that a video sequence is from a stationary camera with most pixels being normal.

Concurrent work has been released since the development of this thesis. Blum *et al.* [4] compare different uncertainty estimation methods applied pixel-wise and evaluated on a new dataset. The dataset is created by extracting animals from the COCO dataset and placing them in Cityscapes images. The animals are considered OOD pixels, while the rest are ID. They also create a hand made dataset that is similar to the automatically generated one. The animal images are manually cut out of images from the internet, then placed on top of Cityscapes images.

Chapter 3

Experimental Setup

This chapter focuses on the details of the experiments, neural network, training, and post processing algorithms. More specifically, this chapter starts with the network architecture, and the OOD detection methods, followed by the datasets used for training and evaluation, then the performance metrics used, an algorithm for the removal of boundary artefacts, and finally the research questions and their accompanying experiments.

3.1 Network Architecture

The PSPNet [59] network architecture was chosen with the ResNet50 feature extractor [20] for the experiments in this thesis. The two driving factors for this network are: near top performance on the Cityscapes benchmark [11] (top 16, with only 2.4 less mIoU than the top scoring algorithm) and the final operation before the softmax classification layer being a bi-linear upsample. The upsample is important for two reasons. The first is that there is a clear relationship between the softmax values and a pixel prediction. The second is that each spatial location in the penultimate layer has a fixed relationship to each output pixel. Both of these ensure a fair comparison between methods that use the softmax values directly, the Confidence method which has auxiliary pixel predictions, and the Mahalanobis method that uses the penultimate layer activations.

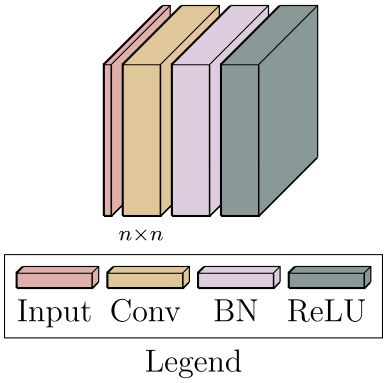


Figure 3.1: The squence of Convolution, Batch Nomalisation, then ReLU is a ConvBlock.

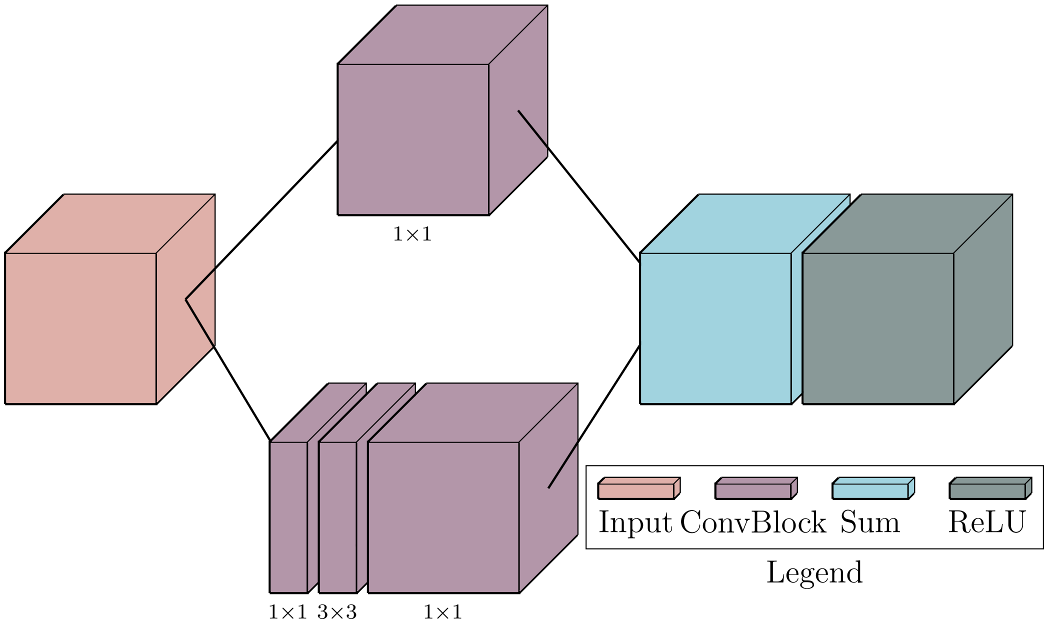
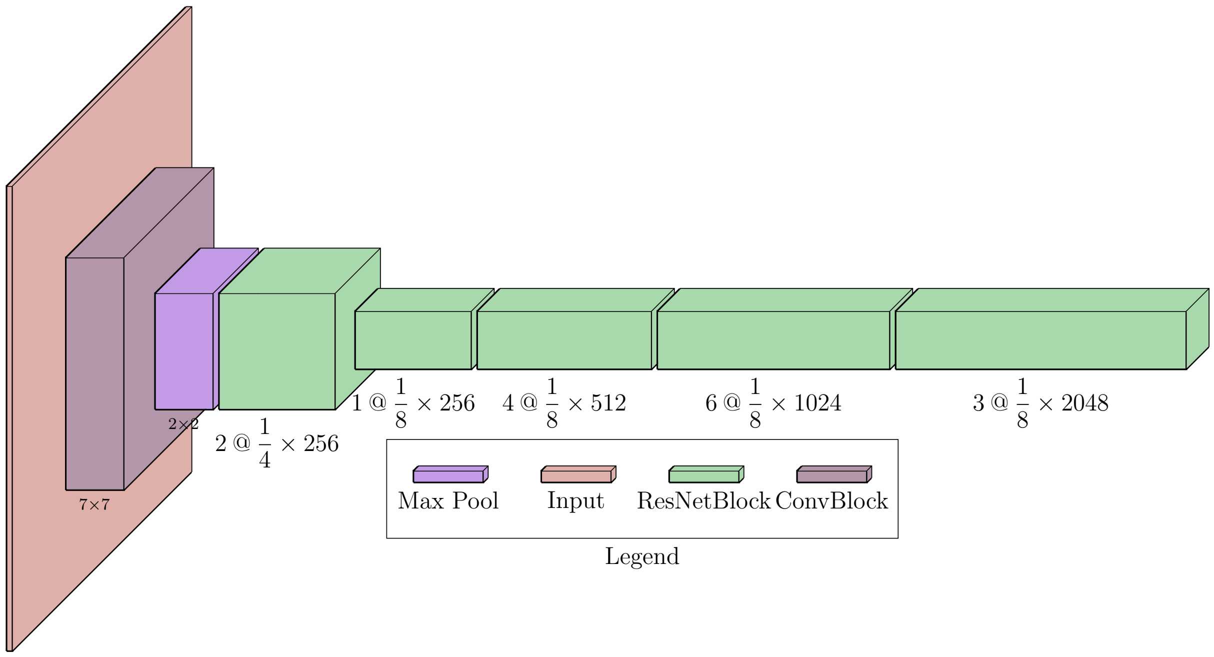


Figure 3.2: Sequence of operations for a single ResNetBlock. The thick blocks represent a depth of 4 times that of the smaller blocks. The decrease in depth is a bottleneck in the ResNetBlock. The dimensions below the ConvBlocks indicate the internal convolution kernel shape.



7×7
2×2
$2 @ \frac{1}{4} \times 256$
$1 @ \frac{1}{8} \times 256$
$4 @ \frac{1}{8} \times 512$
$6 @ \frac{1}{8} \times 1024$
$3 @ \frac{1}{8} \times 2048$
Max Pool
Input
ResNetBlock
ConvBlock
Legend

Figure 3.3: Sequence of operations for a the ResNet feature extractor. The thick blocks represent a depth of 4 times that of the input. The dimensions below the convolution blocks indicate the kernel shape. The text below a ResNetBlock takes the from $n @ f \times d$, where f is the height and width multiplier, d is the depth of the block, and n is the number of times that block configuration is repeated. Each decrease in height and width represents halving each. Here the first ConvBlock halves the height and width.

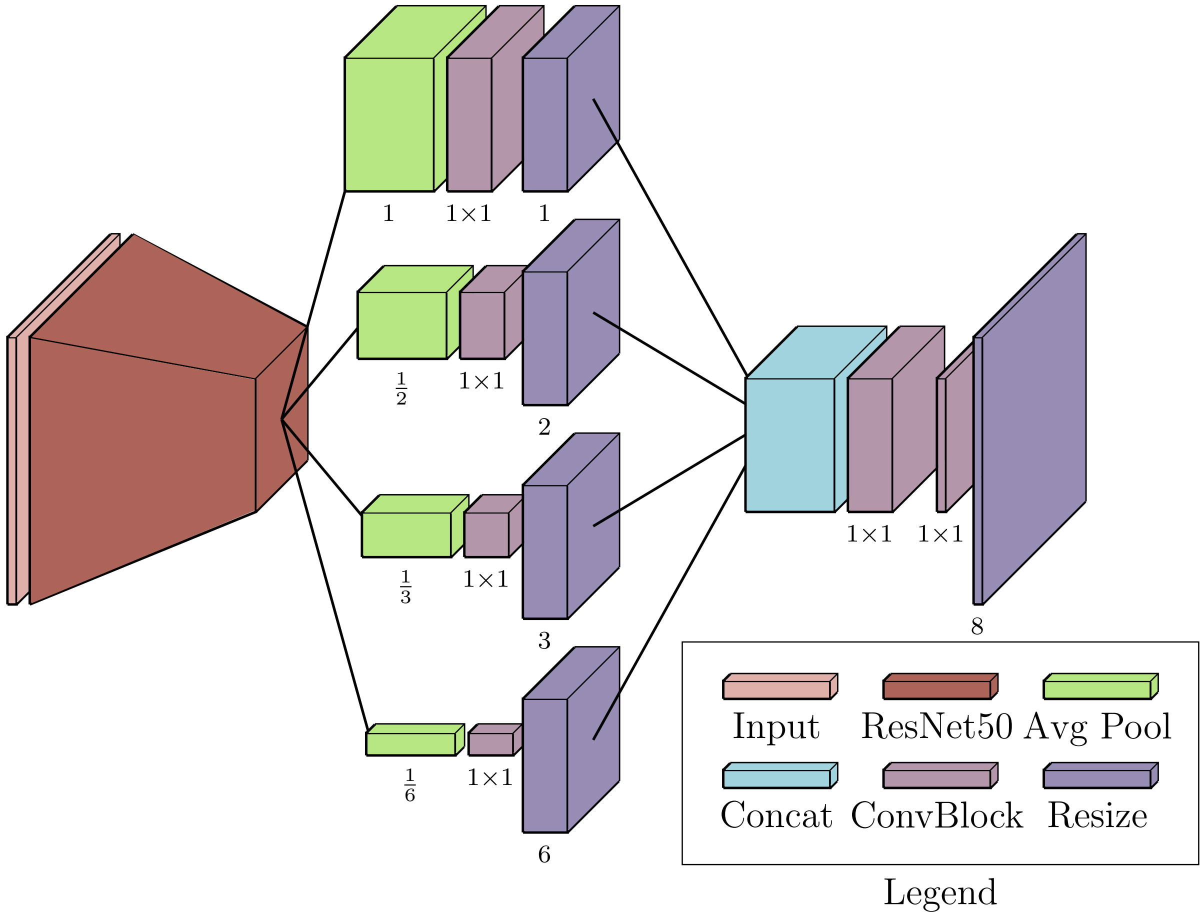


Figure 3.4: The PSPNet architecture with ResNet50 backbone. The number under each average pooling layers and the resize layers indicate the change in scale (height and width). The kernel size is shown below each ConvBlock.

The ResNet architecture has two composable parts. The first composable component will be called a "ConvBlock". This block is a sequence of a convolution, then batch normalisation (BN), followed by the ReLU function as shown in Figure 3.1. The second composable component is the "ResNetBlock". Figure 3.2 shows that each ResNetBlock has two paths. The top path is the shortcut or skip connection. This is a 1×1 ConvBlock to change the depth to match the output of the bottom path, if needed. The bottom path consists of two bottleneck ConvBlocks followed by a ConvBlock to expand the depth to the desired output. These two paths are summed element wise before applying the ReLU function. The skip connections have been shown to flatten out the loss landscape [31], as well as improve performance [20]. The full ResNet feature extractor is shown in Figure 3.3.

Figure 3.4 shows the full PSPNet architecture. Zhao *et al.* [59] state that the addition of the four different scaling branches allows the network to capture different scales of features allowing for better classification of both large and small objects.

The loss function used to train the network is the cross entropy loss [18] with l_2 norm weight regularisation. Let $\hat{\boldsymbol{Y}} \in \mathbb{R}^{n \times c}, \boldsymbol{Y} \in \mathbb{R}^{n \times c}, \boldsymbol{W}_l \in \mathbb{R}^{v_l}$ be the output of the network after softmax, the target one-hot vector, and the weights of layer l respectively. Where n is the number of pixels, c is the number of classes, and v_l is the dimension on the l^{th} layer. The total loss is therefore:

$$\mathcal{L}_c = -\frac{1}{n} \sum_i Y_{i,:} \cdot \log(\hat{Y}_{i,:}) \tag{3.1}$$

$$\mathcal{L}_w = \frac{1}{L} \sum_l ||\boldsymbol{W}_l||_2^2 \tag{3.2}$$

$$\mathcal{L} = \mathcal{L}_c + \lambda \cdot \mathcal{L}_w \tag{3.3}$$

where L is the number of layers, and λ is a hyperparameter ($\lambda = 0.0001$ in all experiments). The ADAM optimiser [27] is used with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

3.2 Adaptations for Semantic Segmentation

There are two main concerns when considering the differences between OOD detection for image classification and semantic segmentation. Both adapting OOD detection methods, and adapting datasets to asses performance have a number of challenges that need to be addressed in order to properly conduct experiments.

3.2.1 OOD Detection Methods

There are many OOD detection methods that are candidates for to pixel-level detection. To limit the scope of this work, three criteria are used to select existing image-level OOD detection methods to adapt to pixel-level OOD detection. First, the candidate methods are top performers on image classification datasets. Second, they must be computationally feasible for semantic segmentation. Third, they must apply to deep learning models. The Dropout method [26] and the Entropy method do not meet the first criterion, but they are included as an existing baseline for uncertainty estimation applied to pixel-level OOD classification. Max Softmax [21], ODIN [32], Mahalanobis [30] and Confidence [14] fit all the above criteria. One method that is excluded from the experiments because it does not meet the second criterion is an ensemble method by Vyas *et al.* [54], with an ensemble of K leave out classifiers. AEs and GANs are also excluded as the goal is to add to semantic segmentation networks. In general, images and architectures for semantic segmentation are larger than for image classification, and therefore an ensemble method is much less feasible for segmentation than classification due to GPU memory limitations.

The following are descriptions of the selected image-level OOD detection methods and any modifications that are necessary to adapt them to pixel-level.[1] Each original/adapted method produces a value that can be thresholded to predict whether an image/pixel is OOD. All metrics used to evaluate performance are threshold independent, therefore, no thresholds are discussed in this section.

Each subsection uses notation and variable names close to the original works, so that readers can refer to said works for more detail. Variable names should not be carried between sections unless explicitly stated. Assume the segmentation and image classification neural network are functions $f : \mathbb{R}^n \to \mathbb{R}^{n \times c}$ and $g : \mathbb{R}^n \to \mathbb{R}^c$ respectively, where n, c are the number of pixels and the number of classes respectively. Unless otherwise stated, f and g do not include softmax output. For brevity, images are represented by flattened vectors, and segmentation output is a matrix where each column is the predicted probability of a class. The set of pixels $\mathbb{P}$ will be used with typical subscripts $i, j \in \mathbb{P}$. For example $f(\boldsymbol{x})_i$ is the i^{th} prediction vector (*i.e.* the predicted distribution over the classes) of the result of $f(\boldsymbol{x})$.

[1]Code for each method discussed in this section is available at `https://github.com/mattangus/fast-semantic-segmentation`

3.2.1.1 Dropout

Background. Dropout [50] can be used as a regularisation technique in neural networks to limit the number of weights that can be used in a single forward pass. A multiplier of 0 or 1 is randomly chosen for each neuron at train time. Typically at test/evaluation time, all neurons are kept (all multipliers are set to 1). However, using dropout at test time can act as a surrogate for uncertainty. Kendall *et al.* [26] use the variation in predictions of a model that uses dropout at test time to compute an estimate of model uncertainty.

Repeatedly sampling new multipliers generates new outputs. The mean $\boldsymbol{\mu}$ and variance (not covariance) $\boldsymbol{S}$ can be computed over all of these outputs. The uncertainty score is then

$$\boldsymbol{v} = \frac{1}{c} \sum_{j=0}^{c} S_{:,j} \tag{3.4}$$

Note, this method is not originally intended for OOD detection. Visually, however, there seems to be a correlation between uncertainty and out of distribution. Since there are no published methods for pixel-level OOD detection, Dropout is included in this comparison.

Adaptation. The mean of variances $\boldsymbol{v}$ is used to predict if a pixel is OOD. However, in the experimental evaluation, each metric has many thresholds with a specific increment between them. For any two distinct $i, j \in P$ the absolute difference between v_i and v_j is smaller than this increment. Therefore $\boldsymbol{v}$ is scaled by a factor λ, which equals 400 in all experiments.

$$\boldsymbol{v}' = \lambda \cdot \boldsymbol{v} \tag{3.5}$$

3.2.1.2 Max Softmax

Background. Hendrycks and Gimpel [21] show that the max softmax value can be used to detect image-level OOD examples as a baseline for image classification. The max softmax value for a prediction $\boldsymbol{p} = g(\boldsymbol{x})$ is:

$$S_{\hat{y}}(\boldsymbol{p}) = \max_j \operatorname{softmax}(\boldsymbol{p})_j \tag{3.6}$$

The max softmax value $v = S_{\hat{y}}(\boldsymbol{p})$ is used to determine if an input sample is OOD.

Adaptation. Applying Max Softmax to segmentation is done per pixel. Let $\boldsymbol{P} = f(\boldsymbol{x})$, then

$$v_i = \max_j \operatorname{softmax}(P_i)_j \tag{3.7}$$

3.2.1.3 ODIN

Background. Liang *et al.* [32] create a similar method to the max softmax value, dubbed ODIN. This method adds temperature scaling and input preprocessing. The softmax function in Equation 3.6 is modified to include a temperature value t, given $\boldsymbol{p} = g(\boldsymbol{x})$:

$$S_{\hat{y}}(\boldsymbol{p}; t) = \max_j \operatorname{softmax}\left(\frac{\boldsymbol{p}}{t}\right)_j \tag{3.8}$$

Liang *et al.* found that perturbing the input in the direction of the gradient influences ID samples more than OOD samples, thus further separating ID and OOD examples. The input preprocessing step is:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \epsilon \cdot \operatorname{sign}(-\nabla_{\boldsymbol{x}} \log S_{\hat{y}}(\boldsymbol{p}; t)) \tag{3.9}$$

where ϵ is a hyperparameter chosen from a set of 21 evenly spaced values starting at 0 and ending at 0.004. The best temperature value is chosen from a predefined set of temperatures $\{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$. These hyperparameters are selected by performing a grid search while testing on a small subset of the OOD dataset.

The final value $\boldsymbol{v}$ is used to determine if a sample is OOD.

$$\boldsymbol{v} = S_{\hat{y}}(\tilde{\boldsymbol{x}}; t) \tag{3.10}$$

Adaptation. Similar to the adaptation of Max Softmax, Equation 3.8 is modified to be applied per pixel. Given a prediction map $\boldsymbol{P} = f(\boldsymbol{x})$:

$$S_{\hat{y}}(\boldsymbol{P}; t)_i = \max_j \operatorname{softmax}\left(\frac{P_{i,:}}{t}\right)_j \tag{3.11}$$

The preprocessing step is then:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \epsilon \cdot \operatorname{sign}\left(-\sum_i \nabla_x \log S_{\hat{y}}(\boldsymbol{P}; t)_i\right) \tag{3.12}$$

where ϵ is a hyperparameter chosen from a set of 21 evenly spaced values starting at 0 and ending at 0.004. The best temperature value is chosen from a predefined set of temperatures $\{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$.

The temperature-scaled and preprocessed max softmax score $v_i = S_{\hat{y}}(\tilde{\boldsymbol{x}}; t)_i$ is used to predict if pixel i is OOD.

3.2.1.4 Mahalanobis

To simplify indexing, this section will use $\boldsymbol{a}^{(c)}$ to denote class specific quantities.

Background. Lee *et al*. [30] use the Mahalanobis distance for detecting OOD samples. The Mahalanobis distance is the number of standard deviations a vector is away from the mean, generalised to many dimensions. The class sample mean $\boldsymbol{\mu}^{(c)}$ and sample covariance $\boldsymbol{\Sigma}$ are computed by Equation 3.13 and 3.14 respectively. These values are computed for the penultimate layer g_l of the network g.

$$\boldsymbol{\mu}^{(c)} = \frac{1}{k^{(c)}} \sum_{i:y^{(i)}=c} g_l(\boldsymbol{x}^{(i)}) \tag{3.13}$$

$$\boldsymbol{\Sigma} = \frac{1}{k} \sum_{c} \sum_{i:y^{(i)}=c} (g_l(\boldsymbol{x}^{(i)}) - \boldsymbol{\mu}^{(c)})(g_l(\boldsymbol{x}^{(i)}) - \boldsymbol{\mu}^{(c)})^T \tag{3.14}$$

where $k^{(c)}$ is the number of examples of class c, and $k = \sum_c k^{(c)}$. Using these quantities the Mahalanobis distance $M^{(c)}(\boldsymbol{x})$ for a sample $\boldsymbol{x}$ and class c is:

$$M(\boldsymbol{x}; c) = \sqrt{(g_l(x) - \boldsymbol{\mu}^{(c)})^T \boldsymbol{\Sigma}^{-1} (g_l(x) - \boldsymbol{\mu}^{(c)})} \tag{3.15}$$

The minimum distance is found over each class:

$$M(\boldsymbol{x}) = \min_c M(\boldsymbol{x}; c) \tag{3.16}$$

An input preprocessing step, computing $\tilde{\boldsymbol{x}}$, is performed using the gradients of the minimum distance. According to Lee *et al*. [30], this step separates the in and out of distribution examples to a greater extent. $\tilde{\boldsymbol{x}}$ is computed by:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \epsilon \cdot \text{sign}\left(-\nabla_{\boldsymbol{x}} M(\boldsymbol{x})\right) \tag{3.17}$$

where ϵ is a hyperparameter chosen from a set of 21 evenly spaced values starting at 0 and ending at 0.004. The best ϵ is found by evaluating on a held out test set. Lee *et al*. feed $M(\tilde{\boldsymbol{x}})$ to a logistic regression model to predict if the input is OOD.

Adaptation. Each feature vector at every spatial location in the penultimate layer is assumed to be normally distributed. Each distribution is parameterized by the pixel class mean $\boldsymbol{M}^{(c)}$ and global class covariance $\boldsymbol{\Sigma}^{(c)}$ for a class c. The global class covariance is computed for all pixels of a given class, independent of their location. Initial tests showed that using per-pixel class means and a global class covariance has better performance than global or per-pixel class mean and per-pixel class covariance, therefore they are used throughout. Note that the labels are resized to match the height and width of the penultimate layer using nearest neighbour interpolation. Assume that the penultimate layer is computed by $f_l : \mathbb{R}^n \to \mathbb{R}^{p \times d}$, where p is the number of spatial locations, and d is the depth of the feature map. The two quantities $\boldsymbol{M}^{(c)}$ and $\boldsymbol{\Sigma}^{(c)}$ are computed as follows:

$$\boldsymbol{M}^{(c)} = \frac{1}{\boldsymbol{k}^{(c)}} \odot \sum_i \mathbf{1}_{\boldsymbol{y}^{(i)}==c} \odot f_l(\boldsymbol{x}^{(i)}) \tag{3.18}$$

$$\boldsymbol{Z}^{(i)(c)} = \mathbf{1}_{\boldsymbol{y}^{(i)}==c} \odot \left(f_l(\boldsymbol{x}^{(i)}) - \boldsymbol{M}^{(c)}\right) \tag{3.19}$$

$$\boldsymbol{\Sigma}^{(c)} = \frac{1}{||\boldsymbol{k}^{(c)}||_1} \sum_i \sum_{j=1}^{p} \left(Z_{j,:}^{(i)(c)}\right) \left(Z_{j,:}^{(i)(c)}\right)^T \tag{3.20}$$

where $\mathbf{1}_{\boldsymbol{y}^{(i)}==c}$ is the indicator function for the label vector $\boldsymbol{y}^{(i)}$, $\boldsymbol{k}^{(c)} = \sum_i \mathbf{1}_{\boldsymbol{y}^{(i)}==c}$ is a vector of the number of instances of a class for each spatial location, and $\frac{1}{\boldsymbol{k}^{(c)}}$ is the element wise reciprocal. Note that $\boldsymbol{M}^{(c)} \in \mathbb{R}^{p \times d}$, $\boldsymbol{Z}^{(i)(c)} \in \mathbb{R}^{p \times d}$, and $\boldsymbol{\Sigma}^{(c)} \in \mathbb{R}^{d \times d}$. Here the Hadamard product $\odot$ follows broadcasting semantics, meaning that if $\boldsymbol{A} \in \mathbb{R}^p$ and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, then $\boldsymbol{A} \odot \boldsymbol{B} \in \mathbb{R}^{p \times q}$ since $\boldsymbol{A}$ is repeated q times to have the same shape as $\boldsymbol{B}$.

Each spatial location has a class distance and minimum distance computed by:

$$M(x;c)_i = \sqrt{\left(f_l(\boldsymbol{x})_{i,:} - \boldsymbol{M}_{i,:}^{(c)}\right)^T (\boldsymbol{\Sigma}^{(c)})^{-1} \left(f_l(\boldsymbol{x})_{i,:} - \boldsymbol{M}_{i,:}^{(c)}\right)} \tag{3.21}$$

$$M(x)_i = \min_c M(x;c)_i \tag{3.22}$$

This increases the number of matrix multiplications required to compute each pixel distance. Due to hardware memory limitations, a dimensionality reduction layer is needed after the penultimate layer, reducing the depth from 512 to 32 via a 1×1 convolution. An input preprocessing step is also performed. The new input $\tilde{x}$ is computed by:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \epsilon \cdot \operatorname{sign}\left(-\sum_i \nabla_x M(x)_i\right) \tag{3.23}$$

Instead of a logistic regression layer, the minimum distance is normalised to have zero mean and unit variance. The sigmoid function σ is applied to clip to the interval $[0, 1]$:

$$\boldsymbol{v}_i = \sigma\left(\frac{M(x)_i - \mu}{s}\right) \tag{3.24}$$

where μ and s are the mean and standard deviation of all $M(x)$ computed over the whole training dataset. v_i is used to determine if pixel i is OOD. The prediction values are resized, with bi-linear interpolation, to the original input size.

3.2.1.5 Confidence Estimation

Background. DeVries and Taylor [14] train a secondary branch of an image classification network to output a confidence value c. For a prediction vector and confidence value $\boldsymbol{p}, c = g(\boldsymbol{x})$ a new prediction $\boldsymbol{p}'$ is computed by:

$$b \sim B(0.5) \tag{3.25}$$
$$c' = c \cdot b + (1 - b) \tag{3.26}$$
$$\boldsymbol{p}' = c' \cdot \boldsymbol{p} + (1 - c') \cdot \boldsymbol{y} \tag{3.27}$$

Where B is a Bernoulli distribution and $\boldsymbol{y}$ is the one-hot ground truth vector. c gives "hints" to the network when there is a low confidence, and b is used to limit those hints. Figure 3.5 shows an example of how c changes the updated prediction p'.

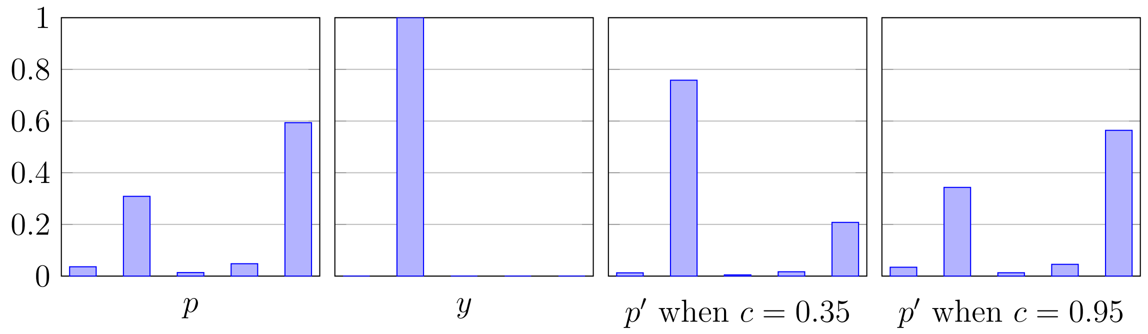


Figure 3.5: Example of how c influences the prediction vector, giving "hints" by increasing the correct prediction when c is low.

The negative log likelihood loss is then applied to the new p':

$$\mathcal{L}_t = -\log(\boldsymbol{p}') \cdot \boldsymbol{y} \tag{3.28}$$

This loss is lower when c is lower, therefore a regularisation term is added to force c to 1 (*i.e.* high confidence):

$$\mathcal{L}_c = -\log(c) \tag{3.29}$$

The total loss is then:

$$\mathcal{L} = \mathcal{L}_t + \lambda \mathcal{L}_c \tag{3.30}$$

where λ is a hyperparameter. See [14] to understand the trade off between $\mathcal{L}_t$ and $\mathcal{L}_c$.

Next, a preprocessing step is applied. $\tilde{\boldsymbol{x}}$ is computed using the gradients w.r.t. $\mathcal{L}_c$, which makes the predicted confidence higher. DeVries and Taylor state that this preprocessing step further separates the in and out of distribution examples.

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \epsilon \cdot \operatorname{sign}\left(\nabla_{\boldsymbol{x}} \mathcal{L}_c\right) \tag{3.31}$$

Let $\tilde{\boldsymbol{p}}, \tilde{c} = f(\tilde{\boldsymbol{x}})$, then $\tilde{c}$ is used to determine if an image is OOD.

Adaptation. Similar to the image classification counterpart, a secondary branch is added to the network f. Therefore $\boldsymbol{c}, \boldsymbol{P} = f(\boldsymbol{x})$, where $\boldsymbol{c}$ is the confidence vector and $\boldsymbol{P}$ is the prediction matrix. The new branch is trained by creating a new prediction $\boldsymbol{P}'$ as:

$$\boldsymbol{b} \sim B(0.5) \tag{3.32}$$

$$\boldsymbol{c}' = \boldsymbol{c} \odot \boldsymbol{b} + (1 - \boldsymbol{b}) \tag{3.33}$$

$$\boldsymbol{P}' = \boldsymbol{c}' \odot \boldsymbol{P} + (1 - \boldsymbol{c}') \odot \boldsymbol{Y} \tag{3.34}$$

where B is a Bernoulli distribution (sampled i.i.d. for each element in $\boldsymbol{b}$) and $\boldsymbol{Y}$ is the one-hot ground truth vector associated with each spatial location of the input. Again, $\odot$ follows broadcasting semantics. The mean negative log likelihood is then applied to the new $\boldsymbol{P}'$:

$$\mathcal{L}_t = -\frac{1}{n} \sum_i \log(P'_{i,:}) \cdot Y_{i,:} \tag{3.35}$$

The same regularisation term is added, and averaged over all spatial locations to force each c_i to 1 (*i.e.* high confidence):

$$\mathcal{L}_c = -\frac{1}{n} \sum_i \log(c_i) \tag{3.36}$$

$$\mathcal{L} = \mathcal{L}_t + \lambda \mathcal{L}_c \tag{3.37}$$

where $\mathcal{L}$ is the total loss that is used to train the network, and λ is a hyperparameter. λ is 0.5 in all experiments.

Each pixel has a preprocessing step is applied to at test time, and is computed using the gradients of the $\mathcal{L}_c$ loss. Note, that the adapted $\mathcal{L}_c$ sums over all the pixel confidence predictions c_i, therefore the gradient implicitly sums over output pixels as well.

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \epsilon \cdot \operatorname{sign}\left(\nabla_{\boldsymbol{x}} \mathcal{L}_c\right) \tag{3.38}$$

Let $\tilde{\boldsymbol{P}}, \tilde{\boldsymbol{c}} = f(\tilde{\boldsymbol{x}})$, then $\tilde{\boldsymbol{c}}_i$ is used to determine if pixel i is OOD.

3.2.1.6 Entropy

Background. Shannon entropy [46] is an information theoretic concept, that is used to determine how much information a source contains. The entropy equation is $H : \mathbb{R}^n \rightarrow \mathbb{R}$:

$$H(\boldsymbol{x}) = -\sum_i x_i \cdot \log x_i \tag{3.39}$$

Since H was developed for probabilities, $\boldsymbol{x}$ must behave like a probability distribution meaning:

$$\sum_i x_i = 1 \tag{3.40}$$

$$\forall i, x_i >= 0 \tag{3.41}$$

Hendrycks *et al.* [22] train an image-level classifier with a third outlier dataset that is disjoint from both the training and OOD dataset. An auxiliary loss is added minimising the entropy of predictions of outlier images. The network learns to predict uniform values for all classes. The max softmax value is then used at test time to determine if a sample is OOD.

Adaptation. The entropy function is applied to the softmax output, which satisfies the properties in Equations 3.40 and 3.41. Let $\boldsymbol{P} = f(\boldsymbol{x})$ then

$$\boldsymbol{S} = \operatorname{softmax}(\boldsymbol{P}) \tag{3.42}$$

$$\boldsymbol{v} = H(\boldsymbol{S}) \tag{3.43}$$

v_i is used to determine if pixel i is OOD.

3.2.1.7 Summary

Table 3.1 shows a summary of the key properties of the adapted pixel-level methods to be compared. “Preprocessing” indicates if the method includes input preprocessing based on the gradients of the output. “Modified Network” indicates if the method requires modification of the base network architecture. “Modified Method” indicates if the adapted method has significant changes from the corresponding method being adapted. “Classification” indicates if the method has an image-level OOD detection counterpart in the literature.

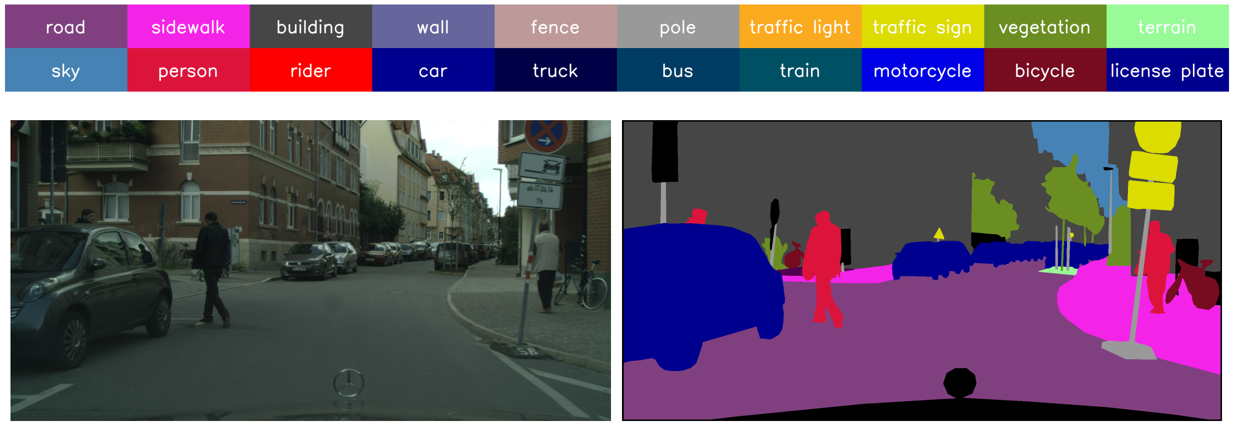
Max Softmax and ODIN do not require any modifications to the network as they both use the output logits of the network. The Dropout method requires adding dropout layers periodically. Mahalanobis has a larger GPU memory footprint, so a dimensonality reduction layer is required. This modification is the only one driven by hardware limitations; the other modifications are inherent to each method. The Confidence method requires that a new branch is added to predict the confidence value. The majority of methods also seem to benefit from preprocessing based on the gradients of the network. Although Dropout could be applied to image-level OOD detection, I am unaware of any prior work doing so.

Method	Preprocessing	Modified Network	Modified Method	Classification
Dropout	×	✓	×	×
Max Softmax	×	×	×	✓
ODIN	✓	×	×	✓
Mahalanobis	✓	✓	✓	✓
Confidence	✓	✓	×	✓
Entropy	×	×	×	×

Table 3.1: Summary of key properties of the methods compared

3.2.2 Adapting Datasets

Previous work on image-level OOD detection designates a dataset used for training as the ID dataset, and an OOD dataset for testing. This framework is not easily extended to semantic segmentation, as any two datasets may share features with the ID dataset. For example, people may exist in both datasets, but one contains indoor scenes and the other contains outdoor scenes. This section describes the datasets used in this work and any modifications required to convert the labels to be binary pixel-level classification, from the original segmentation labels.



(a) Input Image (b) Ground Truth

Figure 3.6: Example image and ground truth from the Cityscapes [11] dataset. Best viewed in colour.

3.2.2.1 Training

The motivation behind this work is the application of OOD detection for the purpose of increasing safety, specifically for autonomous driving. Therefore, the dataset used for training the weights of all networks is the unmodified Cityscapes train dataset [11]. This dataset contains road scenes labelled with 19 classes for training and benchmarking semantic segmentation for autonomous driving (excluding the license plate label, which is not evaluated). Figure 3.6 shows an example image from the Cityscapes train set as well as the list of the classes. Pre-trained weights from ImageNet [12] are used for initialising the ResNet backbone.

3.2.2.2 Evaluation

The diverse SUN dataset [56] is used as the main evaluation dataset. The main motivation for using the SUN dataset is that it has a large variety of scenes (*e.g.* street, forest, and conference room), as well as a large variety of labels (*e.g.* road, door, and vase). In total, there are 908 scene categories and 3819 label categories. Anonymous label submissions are ignored, as their validity is not confirmed.

The SUN dataset labels have to be modified before the dataset can be used for evaluating OOD detection[2]. The approach for modifying this dataset is similar to the open world

[2]Code for converting the SUN dataset is available at `https://github.com/mattangus/SUN-Scripts`

dataset created by Pham *et al*. [37]. Let $C = \{\text{person}, \text{car}, \text{ignore}, ...\}$ be the set of labels available in Cityscapes. Let $S = \{\text{person}, \text{roof}, \text{chair}, ...\}$ be the set of labels available in the SUN dataset. Ambiguous classes such as "wall" (can appear indoor and outdoor) or "path" (a sidewalk or a dirt path in the woods) are sorted into a third set $A \subset S$. The map $\mathcal{M} : S \to C \cup \{\text{OOD}\}$ is defined as:

$$\mathcal{M}(s) = \begin{cases} s & \text{if } s \in C \setminus A \\ \text{ignore} & \text{if } s \in A \\ \text{OOD} & \text{otherwise} \end{cases} \tag{3.44}$$

For every image in the SUN dataset, each pixel label l_i gets a new label $l'_i = \mathcal{M}(l_i)$. Pixels with the "ignore" class are given a weight of 0 during evaluation.

All the images in the SUN dataset have various dimensions. To prevent artefacts in the input after resizing, only images that have more than $640 \cdot 640 = 409,600$ pixels are used. All images are resized to the same size as Cityscapes (1024×2048).

The SUN dataset is split into a train set and an evaluation set (25%/75% split). It is stressed that the training set is only used to select the best hyperparameters of each method.

Following previous work on image-level OOD detection, two synthetic datasets are used as well. A third noise dataset is also used, which is not used in previous works. They are random uniform noise images, random normal noise images, and Perlin noise [36]. Each of these datasets have the entire image labelled as OOD. The LostAndFound dataset [38] is also used for qualitative analysis. The reason for using this dataset for qualitative analysis only is that the labels of this dataset are very sparse, with only the road class and new objects having labels.

One issue with the noise datasets used in previous works is that they are unstructured noise. The high frequency information in both uniform and normal noise is easily captured by small kernels, like those used by convolutional neural networks. To combat this, a Perlin noise [36] dataset is used as well. Perlin noise is a smooth noise generation technique. This is accomplished by randomly selecting gradients at lattice points on a grid. The number of lattice points determines the scale of the noise. Figure 3.7 shows an image generated from Perlin noise. The dataset is generated from three noise images for each red, green, and blue colour channels. Figure 3.8 shows two lattice scales of colour Perlin noise. The scale used for all experiments is 10.

All OOD train/evaluation datasets used are mixed with Cityscapes train/evaluation sets.

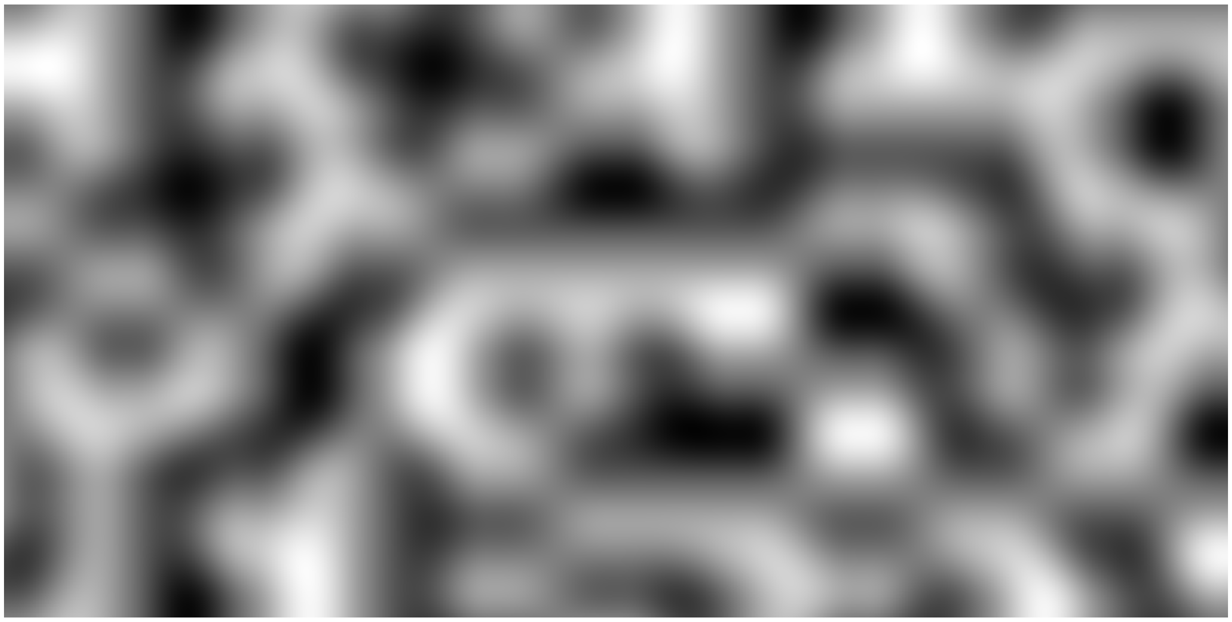
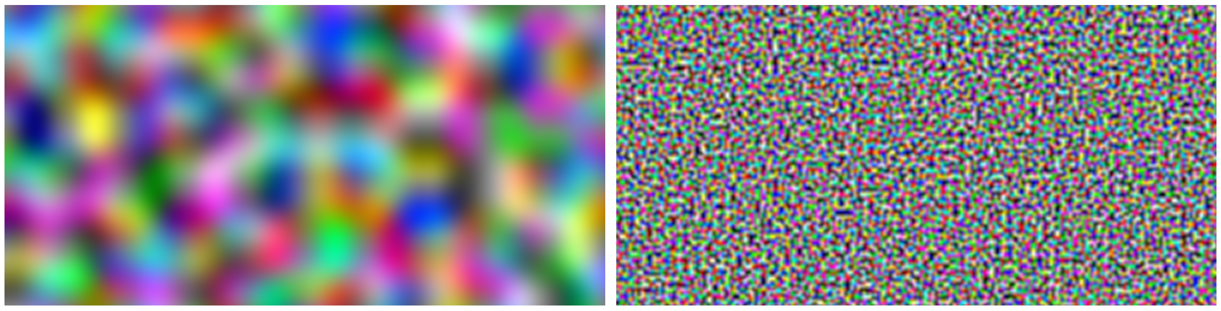


Figure 3.7: 2D greyscale Perlin noise example.


(a) Lattice scale 10

(b) Lattice scale 100

Figure 3.8: Examples of colour noise images with different lattice scales.

3.2.2.3 Dataset Statistics

The created dataset has different characteristics to Cityscapes, as shown by the frequency plot in Figure 3.9. The most important conclusion from the distribution of classes is that the modified SUN dataset has 40% OOD pixels. A prevalent issue in OOD detection is the very small ratio of OOD to ID samples. The number of ID pixels is similar to the number of OOD pixels, so the ratio in this dataset is not a barrier.

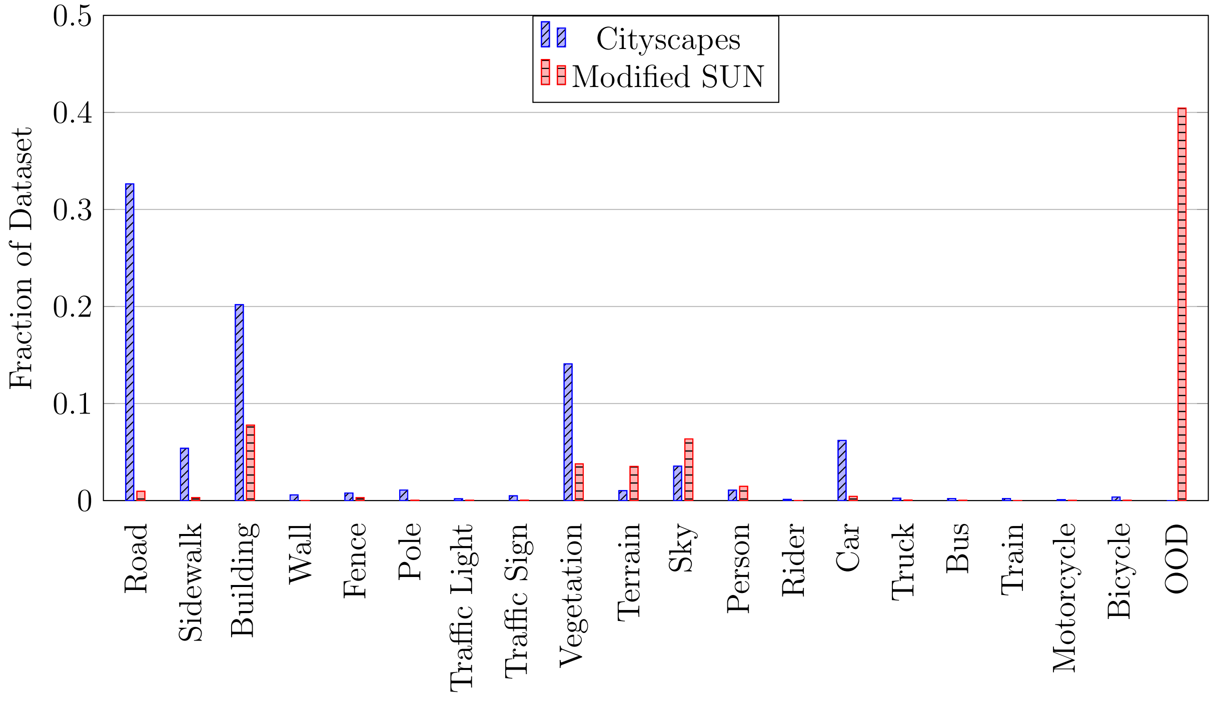


Figure 3.9: Fraction of dataset for each class.

3.3 Performance Metrics

There are five metrics used to evaluate the performance of each model. The first four listed below are the same metrics used by most previous works on OOD detection [32, 21, 30, 14]. Since the output of semantic segmentation is so much larger than image classification, and cannot be kept in memory, the below metrics must be approximated. This is done by using 400 linearly spaced thresholds between 0 and 1 and tracking all true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each threshold. For a given threshold in image-level OOD detection, each image has a prediction that contributes to or increments one of $\mathrm{TP}, \mathrm{TN}, \mathrm{FP}, \mathrm{FN}$. In pixel-level OOD detection, each pixel contributes to one of $\mathrm{TP}, \mathrm{TN}, \mathrm{FP}, \mathrm{FN}$ for a given threshold—accumulated across all images. In the following sections $\mathrm{TP}, \mathrm{TN}, \mathrm{FP}, \mathrm{FN}$ are used as functions with a threshold input mapping $[0, 1] \mapsto \mathbb{N}$. For example, given a threshold $t \in [0, 1]$ there are $\mathrm{TP}(t)$ true positives.

3.3.1 AUROC

The first metric is the area under the receiver operating characteristic (ROC) curve. The ROC curve is found by plotting the false positive rate ($\mathrm{FPR}(t)$) $\frac{\mathrm{FP}(t)}{\mathrm{FP}(t)+\mathrm{TN}(t)}$ against the true positive rate ($\mathrm{TPR}(t)$) $\frac{\mathrm{TP}(t)}{\mathrm{TP}(t)+\mathrm{FN}(t)}$. This is a parametric curve $\mathrm{ROC}(t) = (\mathrm{FPR}(t), \mathrm{TPR}(t))$, therefore the AUROC is:

$$\mathrm{AUROC} = \int_0^1 \mathrm{TPR}(t) \frac{d\mathrm{FPR}(t)}{dt} dt \tag{3.45}$$

This can be computed exactly for a given dataset by sorting the test examples by the OOD score assigned. Let s_i be the score of the i^{th} lowest scoring pixel, and let N be the total number of pixels. For brevity $s_{-1} = 0$.

$$\mathrm{AUROC} = \sum_{i=0}^{N} \mathrm{TPR}(s_i) \cdot (\mathrm{FPR}(s_i) - \mathrm{FPR}(s_{i-1})) \tag{3.46}$$

This version exactly computes the AUROC, since all possible thresholds are taken into account. Any finer grained thresholds would be redundant, *i.e.* $\mathrm{FPR}(s_i) - \mathrm{FPR}(s_i + \epsilon) = 0$ when $s_i + \epsilon < s_{i+1}$. This computation is tractable for image-level OOD detection, since each image only has one score. The number of scores for pixel-level OOD detection is the number of total number of pixels in the dataset, which makes the exact computation much less feasible. Let $\tau_i = \frac{i}{k}$ for $i \in \{0, 1, ..., k\}$, be a sequence of precomputed thresholds ($k = 400$ in all experiments). Again $\tau_{-1} = 0$. The approximation used is the composite trapazoid rule with variable subintervals:

$$\mathrm{AUROC} \approx \frac{1}{2} \sum_{i=0}^{k} \cdot (\mathrm{TPR}(\tau_i) + \mathrm{TPR}(\tau_{i-1})) \cdot (\mathrm{FPR}(\tau_i) - \mathrm{FPR}(\tau_{i-1})) \tag{3.47}$$

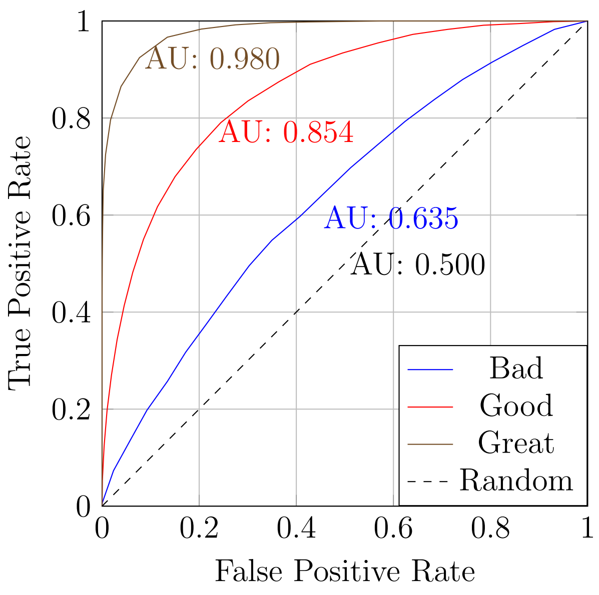


Figure 3.10: Examples of ROC curves for different qualities of classifiers, with corresponding area under the curve values. (best viewed in colour)

Figure 3.10 shows examples of ROC curves for varying qualities of classifiers, as well as a random classifier for a baseline.

The AUROC metric is very sensitive when there is a vastly disproportionate number of positives and negatives. However, as shown in Figure 3.9, there are many OOD pixels in the modified SUN dataset.

3.3.2 AUPRC

The next metric is the area under the precision recall (PR) curve. The PR curve is found by plotting the TPR(t) (or recall) the against precision (PRE(t)) $\frac{\mathrm{TP}(t)}{\mathrm{TP}(t)+\mathrm{FP}(t)}$. This is a parametric curve $\mathrm{PRC}(t) = (\mathrm{TPR}(t), \mathrm{PRE}(t))$, therefore the AUPRC is:

$$\mathrm{AUPRC} = \int_0^1 \mathrm{PRE}(t) \frac{d\mathrm{TPR}(t)}{dt} dt \tag{3.48}$$

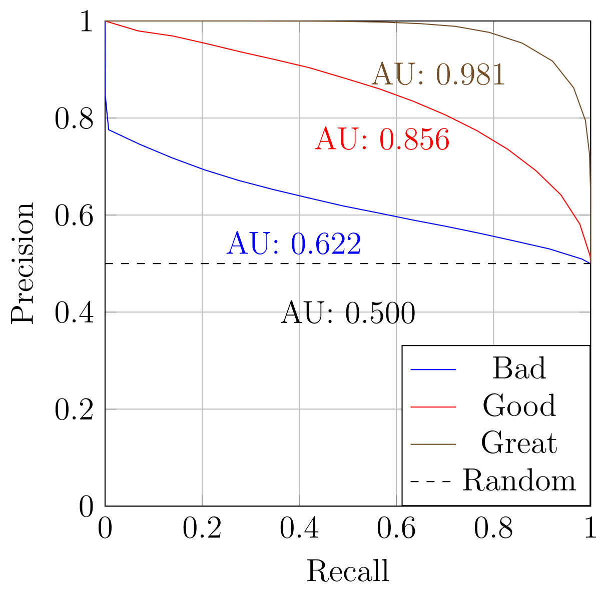


Figure 3.11: Examples of PR curves for different qualities of classifiers, with corresponding area under the curve values. (best viewed in colour)

The same analysis for approximating AUROC is applied to AUPRC, resulting in the foll-wing approximation:

$$\text{AUPRC} \approx \frac{1}{2}\sum_{i=0}^{k} \cdot (\text{PRE}(\tau_i) + \text{PRE}(\tau_{i-1})) \cdot (\text{TPR}(\tau_i) - \text{TPR}(\tau_{i-1})) \qquad (3.49)$$

Figure 3.11 shows the corresponding PRC curves for the same classifiers shown in Section 3.3.1.

3.3.3 FPRatTPR

The next metric is FPR at 95% TPR (FPRatTPR for short). It is extracted from the ROC curve, and can be interpreted as the probability that an OOD sample is classified as ID, when the TPR is set to be 95%. This is computed by finding a threshold t such that $TPR(t) = 0.95$, then simply computing $FPR(t)$. This metric is undefined for a perfect classifier (*i.e.* when AUROC $= 1$).

3.3.4 DE

Detection Error (DE) is the probability that a misdetection will occur with a 95% TPR, assuming that there are an equal number of OOD and ID samples. Computed as $0.5 \cdot (1 - \text{TPR(t)}) + 0.5 \cdot \text{FPR(t)}$, where t is a threshold such that $TPR(t) = 0.95$. Note that the minimum value for DE is not zero; it is 0.025. As with FPRatTPR, this metric is undefined for a perfect classifier. This metric is an affine transformation of the FPRatTPR. It is included as it is used throughout the literature.

3.3.5 MaxIoU

MaxIoU is introduced, inspired from the semantic segmentation community. Mean intersection over union (mIoU) is the canonical performance metric for semantic segmentation. mIoU for segmentation does not have a threshold value, as segmentation is normally a multi-class problem. In this case, the binary classification has a single value to threshold. The equation for MaxIoU is:

$$\text{MaxIoU} = \max_t \frac{\text{TP}(t)}{\text{FP}(t) + \text{FN}(t) + \text{TP}(t)} \tag{3.50}$$

Figure 3.12 shows the IoU curves for different classifiers. Notice that the peaks are not aligned.

Maximising the IoU quantity punishes false positives more than AUROC. The optimal threshold is generally greater, resulting in fewer positive predictions than for AUROC. To verify that the MaxIoU is complimentary to AUROC, the optimal thresholds selected by each metric were experimentally compared. The mean absolute difference between the threshold selected via Youden index [57] and that chosen via MaxIoU was found to be 0.039.

3.4 Removing Boundary Artefacts

One observation for all OOD predictions in Figure 4.3 is that predicted class boundaries are visually correlated with high prediction values. This is an issue because even ID pixels are predicted to be OOD when a class boundary is nearby. Classical computer vision techniques can be used to remove these artefacts from the prediction as a postprocessing step. The steps are as follows:

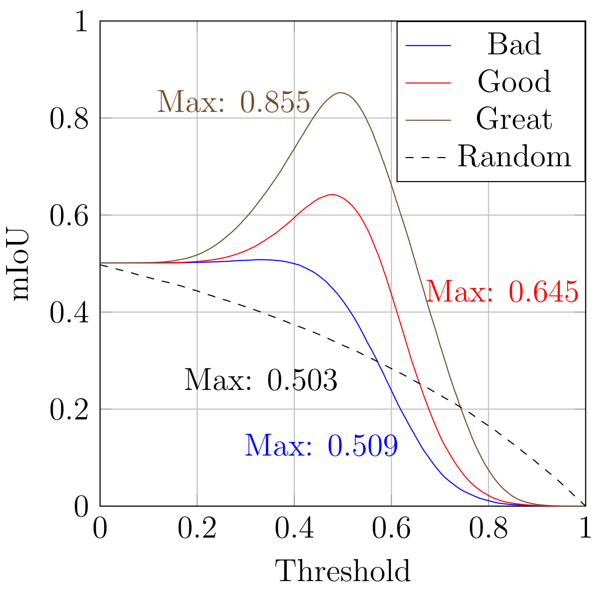
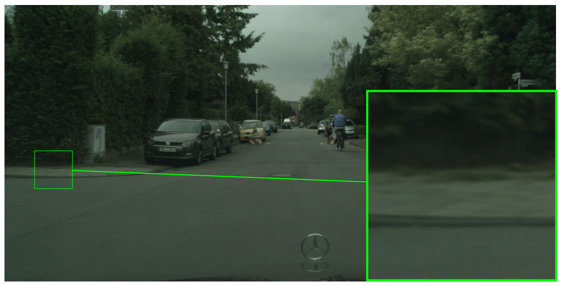
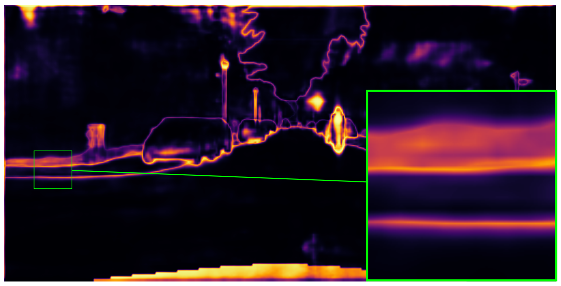


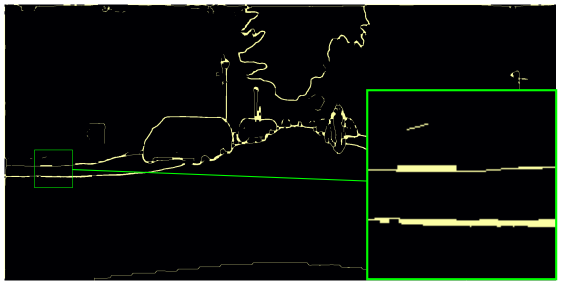
Figure 3.12: Examples of IoU curves for different qualities of classifiers, with corresponding maximum values (best viewed in colour).

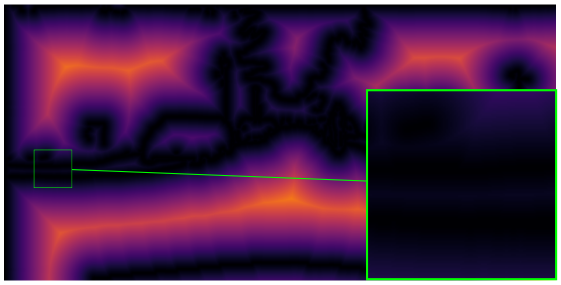
1. Apply Canny edge detector to OOD prediction output
2. Apply morphological close (dilate then erode)
3. Compute the distance transform of the remaining edges
4. Threshold distance to 10 pixels maximum
5. Normalise distance map from 0 to 1
6. Element wise multiply the result from previous step by the original output values

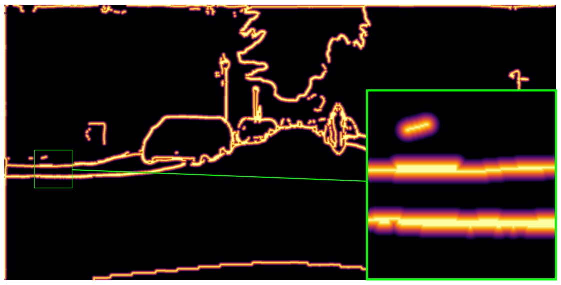
Figure 3.13 shows the stages of the postprocessing algorithm as described in the steps above. The zoomed-in portion shows a group of pixels in a shadow (that should be kept), above a line that is the class boundary between the sidewalk and road (that should be removed). The image after removing these artefacts is a better quality result, with fewer class boundary pixels highlighted.


(a) Input image


(b) OOD prediction probabilities


(c) Extracted Edges, from (b) (step 1 and 2)


(d) Distance Transform (step 3)


(e) Thresholded and inverted distance transform (step 4 and 5)

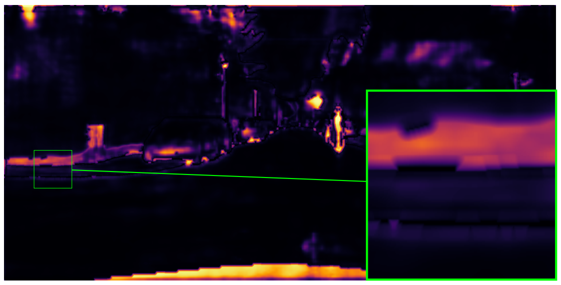

(f) $(1 - (e)) \times (b)$ (step 6)

Figure 3.13: Stages of boundary artefact removal process from Max Softmax OOD prediction

3.5 Experiments

There are four main research questions (RQ) that this thesis focuses on. Each question has an associated experiment.

- **RQ1:** *Do the required modifications to architecture and loss functions negatively affect the semantic segmentation performance of the network?*
- **RQ2:** *To what extent does removing class boundary artefacts affect OOD detection performance?*
- **RQ3:** *Which OOD detection method performs the best?*
- **RQ4:** *Which methods are the most computationally feasible?*

To answer **RQ1**, the semantic segmentation performance is evaluated on Cityscapes using the standard class mean intersection over union metric. As long as the performance drop of modified networks is not too large, the modifications will not interfere with the primary segmentation task. The evaluation is done on the Cityscapes evaluation dataset. **RQ2** is answered by removing class boundary artefacts and observing the effect on the OOD detection performance according to all metrics. **RQ3** is answered by evaluating each method on the datasets described in Section 3.2.2. Finally, **RQ4** is addressed by comparing the average runtime on specific hardware.

Chapter 4

Results and Discussion

The following sections describe the results of the four experiments, and addresses their research questions.

4.1 RQ1: Architecture Modifications

Table 4.1 shows the mIoU of PSPNet trained on Cityscapes and evaluated on Cityscapes. The modifications to the network only slightly degrade the performance. The maximum difference between the unmodified network and any modified version is 0.0161. This difference is acceptable since the focus of this paper is the OOD metrics.

Network	mIoU	Difference
PSPNet	**0.6721**	-
PSPNet + dim reduce	0.6701	0.0020
PSPNet + dropout	0.6593	0.0128
PSPNet + confidence	0.6560	**0.0161**

Table 4.1: mIoU values for modified PSPNet architectures showing efficacy for original segmentation task

4.2 RQ2: Class Boundary Artefacts

Visually, the effect appears to be good; however, the impact on the OOD detection performance is very small. Table 4.2 shows the change in performance after removing class boundary artefacts for all of the methods. The values displayed are the values after removal minus the value before. In the majority of cases the performance increases; however, the increase is very small. The reason for such a small increase is that there is a relatively small number of pixels that are corrected compared to the size of the image. It is important to note that an application that is concerned with identifying unknown objects may still benefit from this step.

Method	AUROC ↑	AUPRC ↑	MaxIoU ↑	FPRatTPR ↓	DE ↓
Dropout	-1.302	**1.405**	0.000	0.519	0.260
Max Softmax	**3.743**	**6.190**	**0.201**	**-0.459**	**-0.229**
ODIN	**1.829**	**2.709**	**0.751**	**-1.115**	**-0.558**
Mahalanobis	**2.108**	**2.487**	**1.291**	**-2.395**	**-1.197**
Confidence	**0.296**	**1.150**	**0.011**	1.064	0.532
Entropy	**3.193**	**2.261**	-0.004	5.032	10.063

Table 4.2: Change in performance of each method after removing class boundary artefacts on the SUN dataset. All values are scaled $\times 10^3$. Bolded values indicate when the performance increases. The arrow beside each metric indicates if larger values are better (↑) or smaller values are better (↓).

4.3 RQ3: Method Performance

Figure 4.1 shows the comparison of the performance of the different methods. Each graph shows a different metric (*c.f.* Section 3.3) for each dataset. Max Softmax, ODIN, and Mahalanobis follow the same trend as their image classification counterparts, increasing in performance in that order. For image-level OOD detection, the Confidence method outperforms the Max Softmax baseline. However, for pixel-level OOD detection it performs worse.

Across each metric, Dropout has the biggest performance increase from the modified SUN dataset to the random uniform and random normal datasets, moving from worst to near top performance. The Confidence method seems to mostly learn to highlight class boundaries, as that is where prediction errors are likely to occur. Therefore, the prediction loss $\mathcal{L}_t$ in Equation 3.35 forces lower confidence levels. This makes it less suitable for the random uniform and normal datasets, where the network predicts one single class for the majority of the input.

The metrics' values reported in the original works for the image-level OOD detection for all methods (except Dropout) are very close to 1 (~0.95 and above). The drop in performance for pixel-level OOD detection could be due to features that cause large disruptions at the pixel-level, but they would not affect an entire image, for example, shadows, occlusion, and far away objects. The actual reason should be investigated in future work.

4.4 RQ3: Qualitative Discussion

Figure 4.2 contains the Cityscapes class colours for reference. Figure 4.3 shows images and the OOD prediction values for the Cityscapes dataset. Note, that all of these pixels are considered ID. The purpose of this figure is to show normal execution of each method, however there are several interesting observations to be made. In all images and all methods cyclist and rider are often confused, which seems to lead to higher OOD prediction values. However, when the bicycle is without a rider, as shown in row 5, OOD values are lower. Rows 1, 3, and 8 have buses and trucks contained within them. These classes are underrepresented in the training data, so it is no surprise that the predicted OOD values are higher. Rows 7, 9, and 10 have strollers, suitcases, and a dog. These classes are unlabelled in Cityscapes, and they are highlighted by most methods.

Figure 4.4 shows some more successful classifications, and Figure 4.5 shows some failure cases that are both drawn from the modified SUN dataset. One common theme is that when road is predicted near the bottom portion of any image, the OOD prediction is much lower. There also seems to be some confusion between water and the road class, further exacerbating the problem. Row 7 is particularly interesting for two reasons. The first being that even the splashes from the truck are classified as road. The second reason is that row 7 is a typical road scene other than the flooding. One very likely explanation for why these cases fail is that all of the methods heavily rely on the correct classification of pixels.

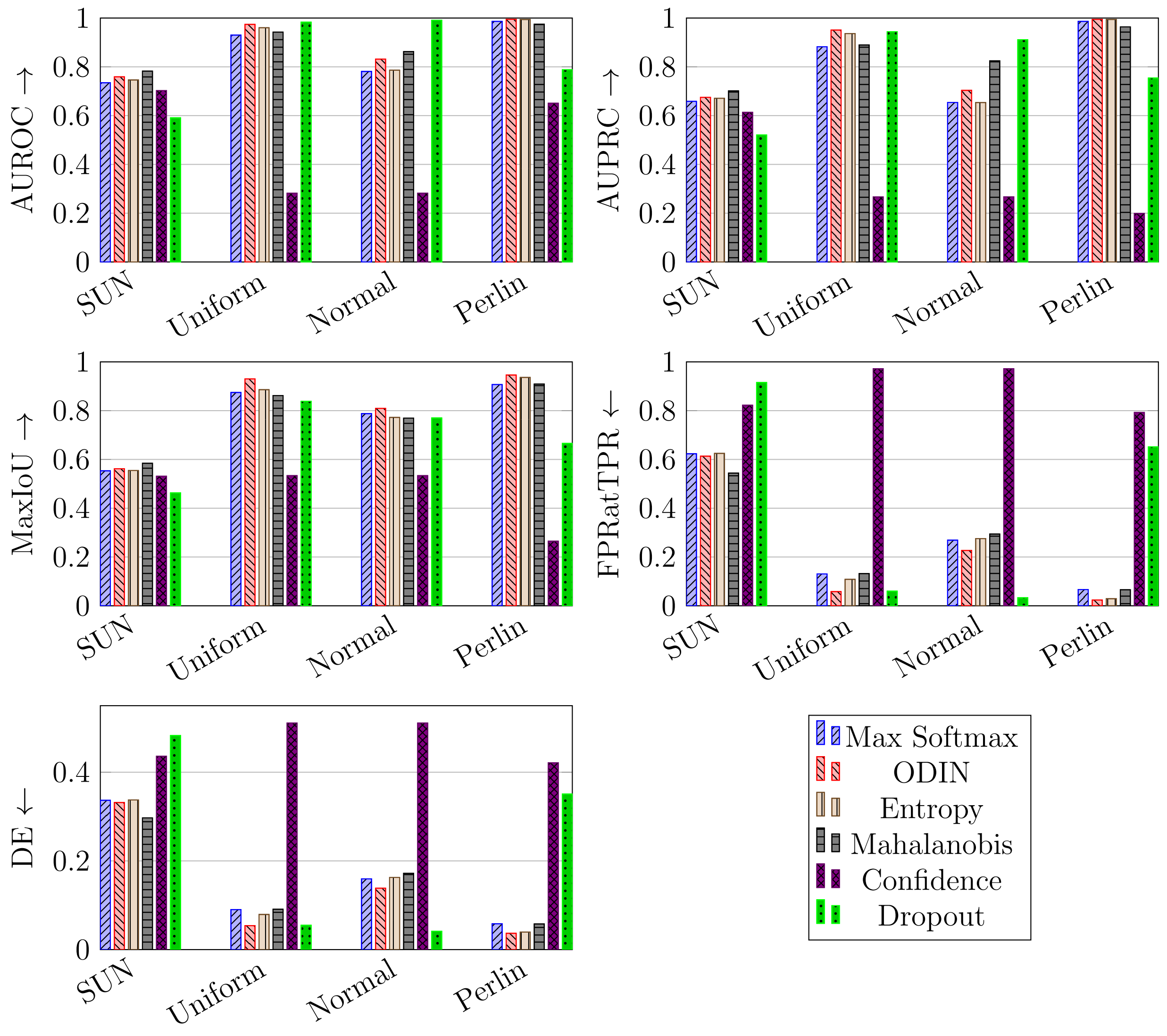


Figure 4.1: Comparison of methods on different datasets. The arrow on the y-axis label indicates if a larger value is better (↑) or a smaller value is better (↓). Each group of bars are labelled by the dataset used for evaluation. Note the change in scale for the DE plot.

road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain
sky	person	rider	car	truck	bus	train	motorcycle	bicycle	license plate

Figure 4.2: Reference class colours for ground truth and predictions. Best viewed in colour.

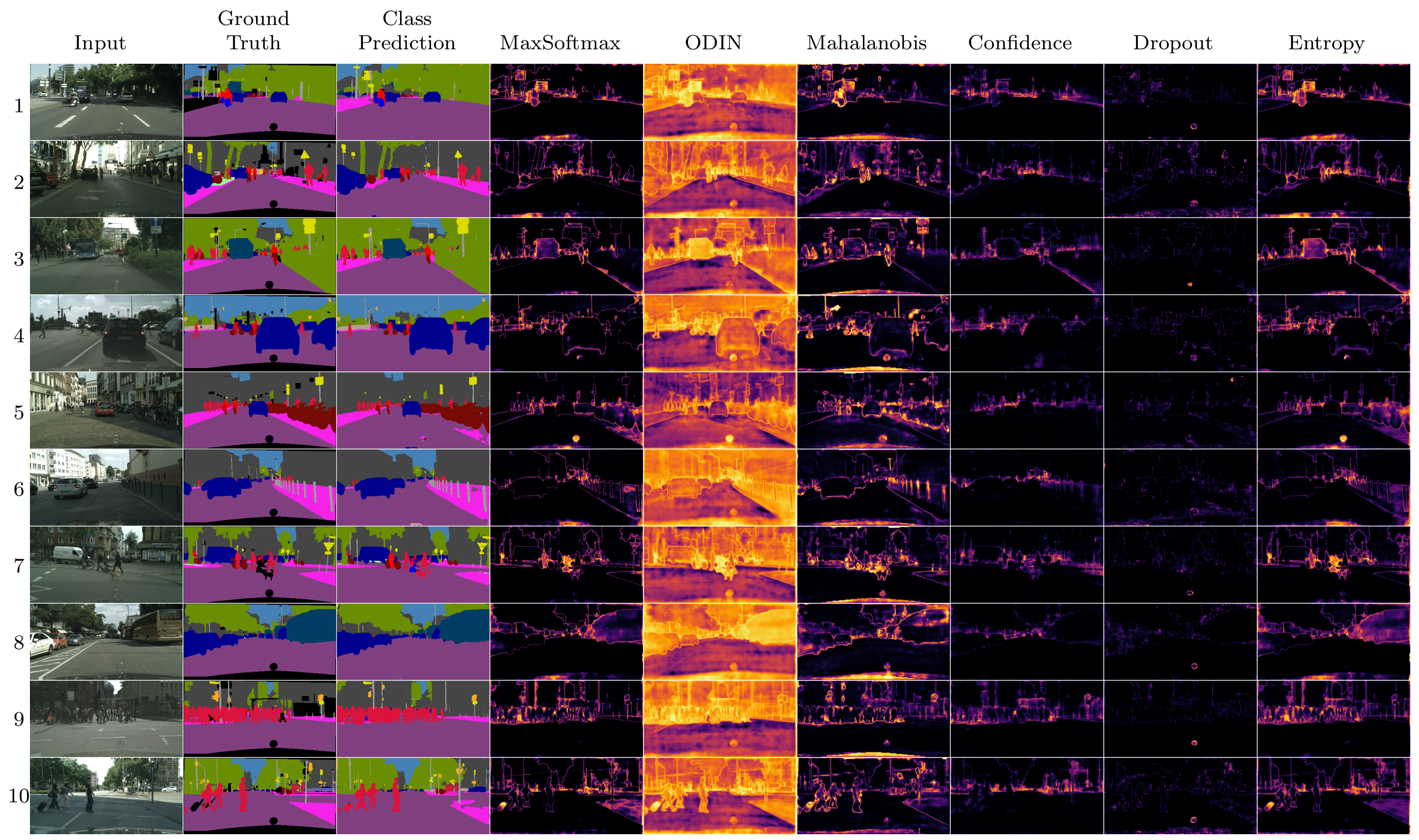


Figure 4.3: Examples take from the Cityscapes dataset (best viewed in colour). Since this is the training set there are no OOD pixels. See Figure 4.2 for Ground Truth and Class Prediction colour mappings.

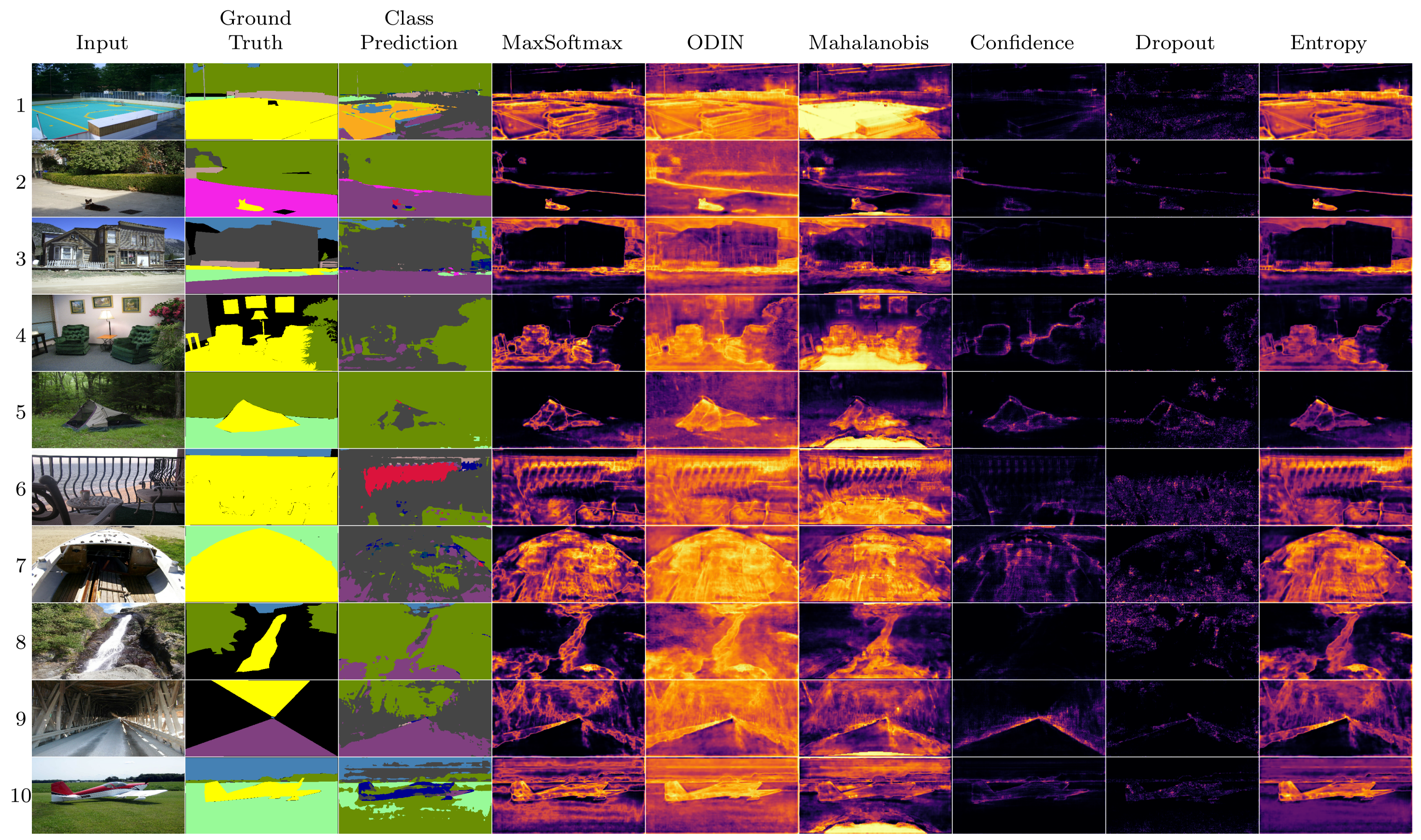


Figure 4.4: Examples taken from the modified SUN datatset (best viewed in colour). These examples are well classified with respect to the performance metrics. The OOD pixels in the ground truth are labelled with bright yellow.

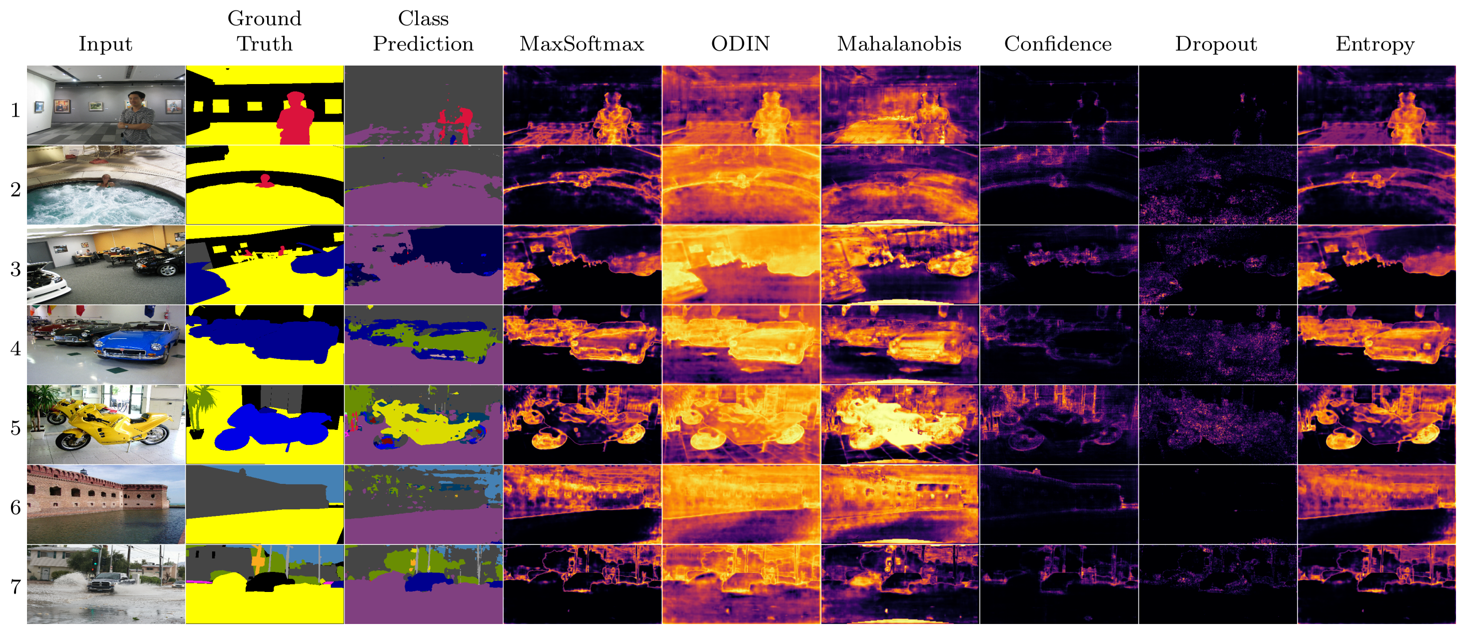


Figure 4.5: Examples taken from the modified SUN datatset (best viewed in colour). These examples are poorly classified with respect to the performance metrics. The OOD pixels in the ground truth are labelled with bright yellow.

Figure 4.6 shows examples from the LostAndFound dataset (rows 1 to 6) as well as from each random noise dataset (rows 7 to 9). The LostAndFound images have quite a diverse set of objects that are mostly picked up by all methods. Examining the Perlin noise class prediction helps to explain why there is a performance drop when compared to the other noise types. Interiors of most objects from Cityscapes are predicted to have a low OOD score. Since the prediction is not a single class for the entire image, the predicted OOD values are less uniform as well.

The image from the LostAndFound dataset in row 4 of Figure 4.6 is an example related to the motivation behind this paper, where the unknown object is localised. The removed car bumper lying in the road is highlighted by each method, even when the prediction has only small blobs of non-road that could be ignored as outliers in the primary task of semantic segmentation.

4.5 RQ4: Runtime comparison

The input is normally distributed images with each red, green, and blue pixel component sampled i.i.d. The runtime is averaged over at least 400 iterations, ignoring the first iteration as a warm up step. The programming language used is Python [53] with almost all operations are implemented using the TensorFlow [1] library, some (as few as possible) are implemented using NumPy [35]. The GPU used is an Nvidia Tesla V100.

Table 4.3 shows the runtime comparison of all methods. Unsurprisingly, Entropy and Softmax are fastest by a large margin, since there is only a few pixel-wise operations added at the end of the unmodified PSPNet. ODIN and Confidence have a large increase over Entropy and Softmax because backpropagation is added, thus effectively requiring two forward passes, and one backward pass of the network. Mahalanobis has backpropagation as well as many added matrix multiplications for the distance calculation, causing a larger increase. Dropout is very inefficient as the forward pass is computed many times (10 for this experiment). Dropout is implemented as a batch of 10 of the same image, thus the computation is not linearly scaling with number of iterations.

Figure 4.7 shows a comparison of performance against runtime. This comparison shows that there is a trade off between performance and runtime for the Entropy, ODIN, and Mahalnobis methods. Applications that are not as time sensitive would be best off with the Mahalanobis method, and time sensitive applications are best suited for Entropy or Softmax.

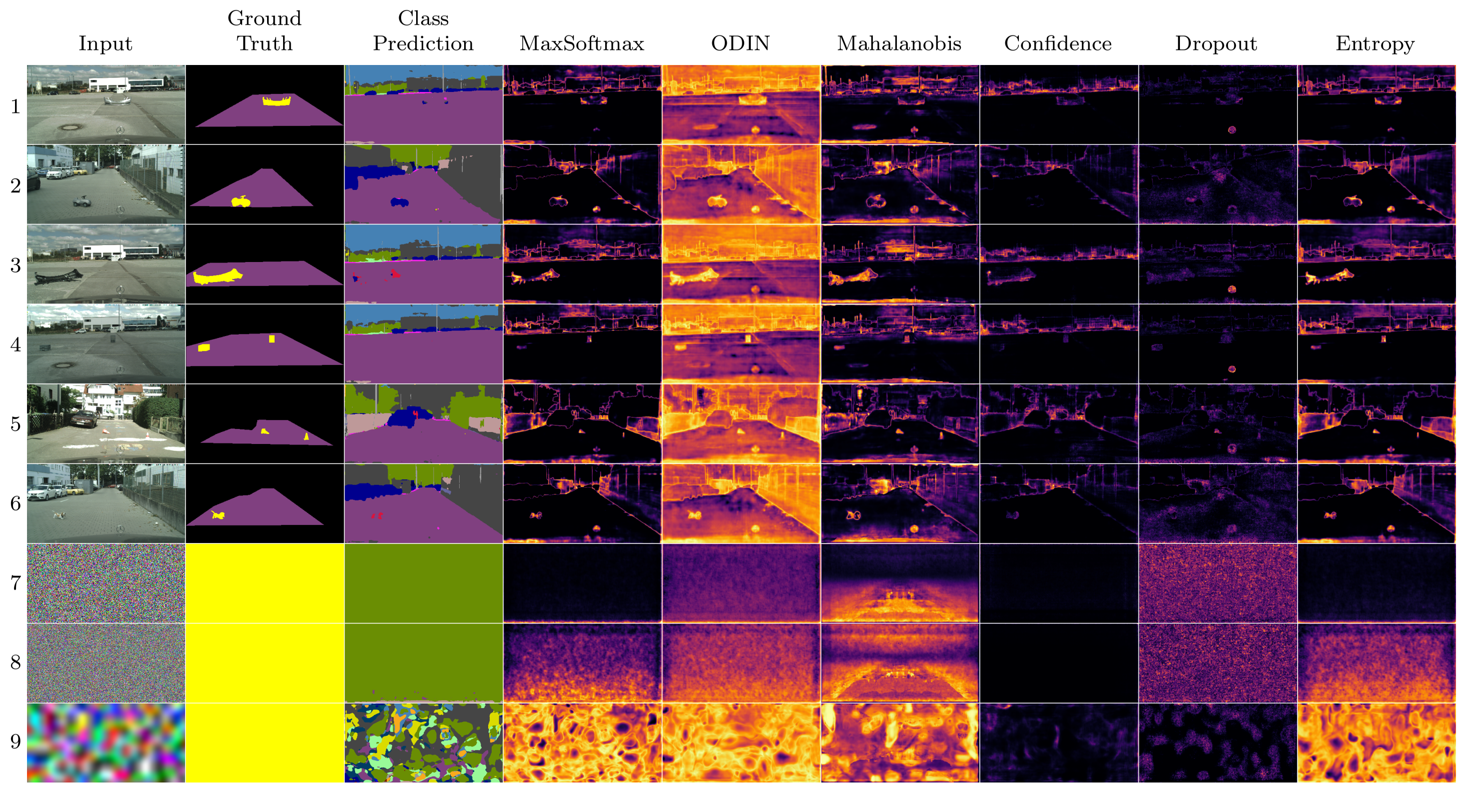


Figure 4.6: Examples taken from the modified LostAndFound dataset (rows 1 to 6), normal noise (row 7), uniform noise (row 8), and Perlin noise (row 9) – best viewed in colour. The OOD pixels in the ground truth are labelled with bright yellow.

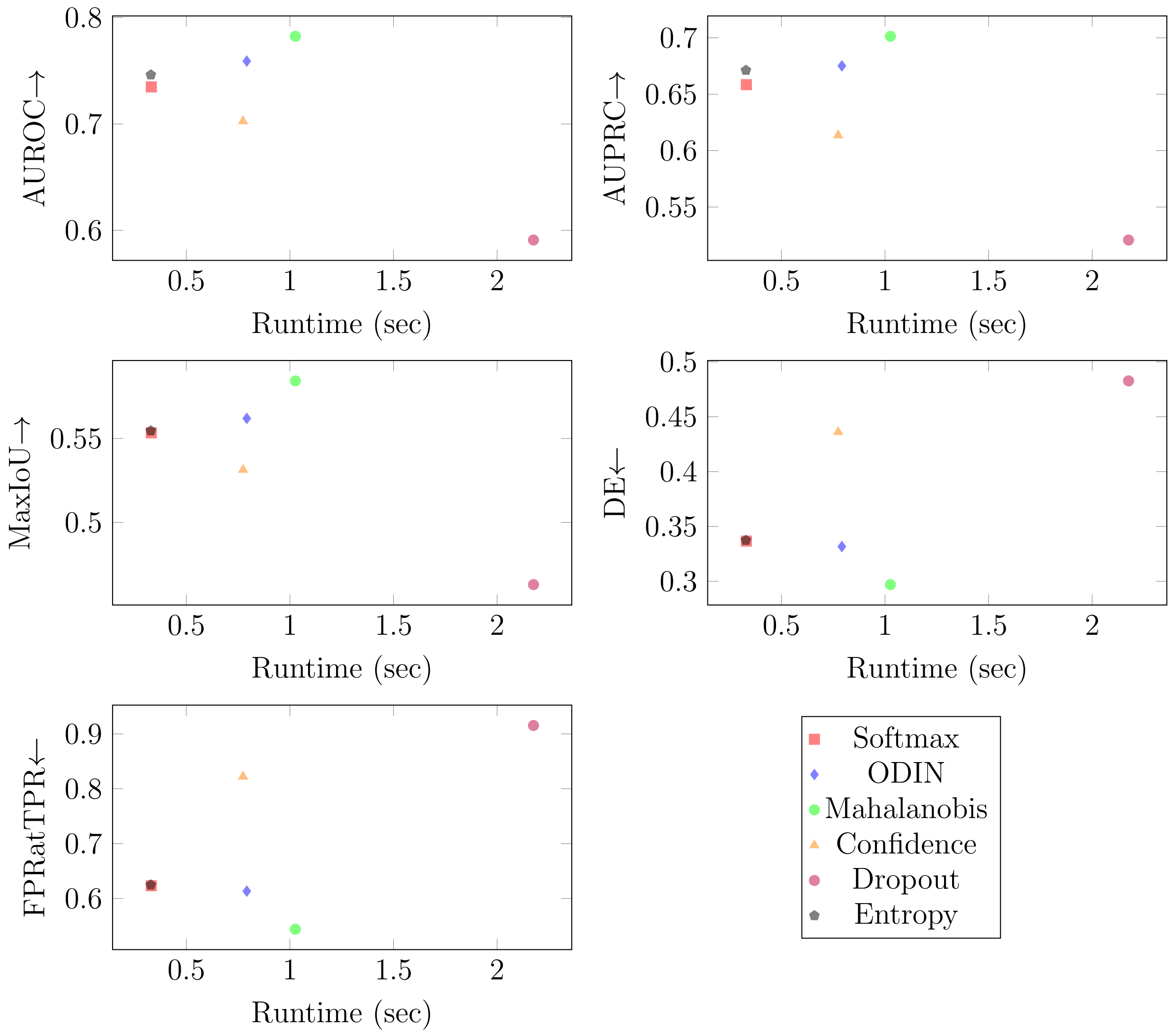


Figure 4.7: Comparison of methods performance on the modified SUN dataset against runtime. The arrow on the y-axis label indicates if a larger value is better (↑) or a smaller value is better (↓).

Method	Runtime (sec)
Softmax	**0.3300**
ODIN	0.7919
Entropy	**0.3288**
Mahalanobis	1.0262
Confidence	0.7739
Dropout	2.1756

Table 4.3: Runtime comparison of all methods

4.6 Threshold Selection

As discussed in Section 3.3.5, MaxIoU suggests a different threshold than ROC curve. Figure 4.8 explores this further, with a comparison of the OOD predictions from the ODIN method using MaxIoU and ROC. Rows 1 and 2 show images where the majority of pixels are ID, and rows 3 and 4 show images where a much larger number of pixels are OOD. The threshold selected by MaxIoU has strengths over ROC for tasks where the majority of pixels are ID. Rows 1 and 2 illustrate MaxIoU selecting a much tighter region around the OOD object. Rows 2 and 3 illustrate that the ROC threshold has tighter boundaries to the OOD region. Note, that the threshold is selected per image to illustrate the differences between the two metrics. In practice, the threshold would be selected based on the whole dataset.

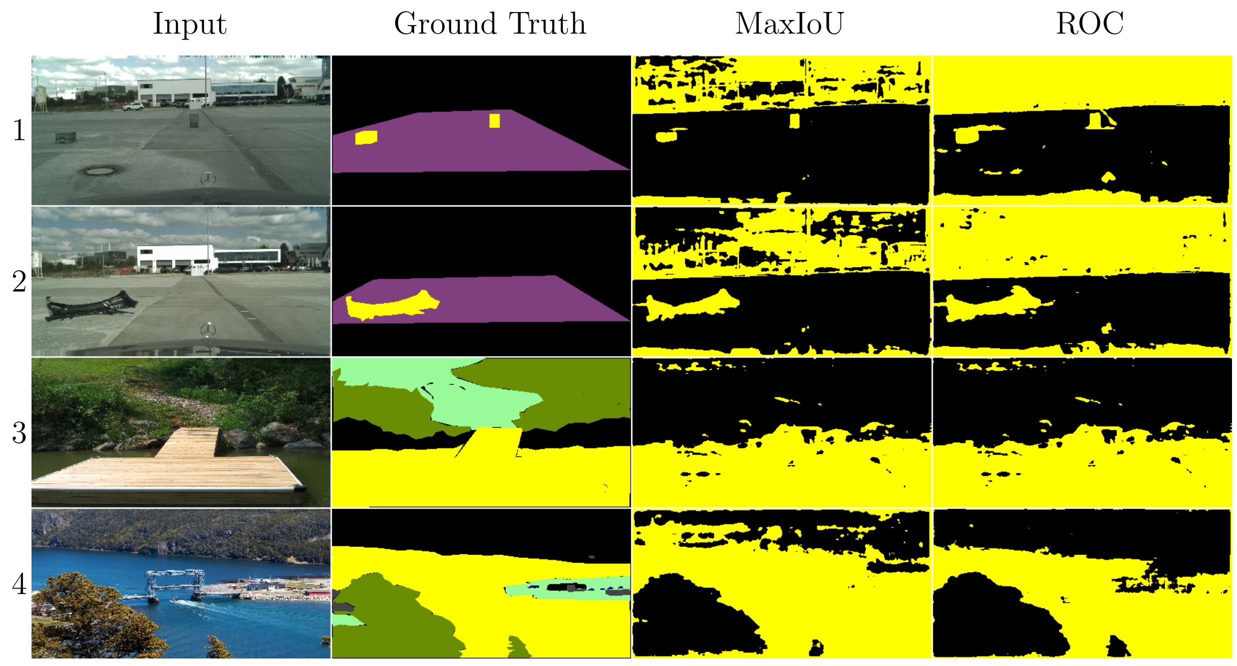


Figure 4.8: Thresholded OOD predictions for ODIN, using MaxIoU and ROC to select the optimal threshold. Black pixels are ID predictions, Yellow pixels are OOD predictions.

Chapter 5

Conclusions and Future Work

In this thesis, several methods for detecting OOD pixels were adapted from image-level OOD detection, as well as uncertainty estimation methods applied to pixel-level OOD detection. These methods were compared using metrics previously established by OOD detection works, as well as a new metric – MaxIoU – that has roots in the semantic segmentation task. This thesis also contributed a new dataset for pixel-level OOD classification which is derived from a semantic segmentation dataset that have common classes but also unique ones with respect to the ID Cityscapes dataset. The experiments performed are the first steps in OOD detection at a pixel-level.

5.1 Limitations

There is great room for improvement for pixel-level OOD detection. One shortcoming of all methods compared in this thesis is the ability to distinguish between class boundary pixels and OOD pixels. Classical computer vision techniques could be used to visually fix this problem, but the performance increase was negligible. The ODIN and Mahalanobis methods have the best all around performance, beating the Dropout and Confidence methods by a significant margin. Therefore, the ODIN and Mahalanobis methods should be considered the baseline for further research in pixel-level OOD detection.

The dataset proposed has a few limitations. The first being the applicability to autonomous driving. There are many road scenes in the SUN datatset, however, the majority are indoor. The networks could be leveraging context of the indoor scene as an indicator of OOD pixels.

One problem that plagues all works on OOD detection is that lack of formal definition of what OOD means. The result is that some research that seem very similar may actually be computing very different quantities. Once a precise definition is made, then better datasets targeting that definition can be made as well.

A limitation specific to semantic segmentation is the ignore class that contains many unknown or unlabelled pixels. Often these patches of pixels labelled ignore have a high OOD prediction value, meaning that OOD and ignored pixels may not be separable.

5.2 Future Work

The main limitation of this work is that it was an initial exploratory comparison. Much more should be done to understand why Mahalanobis and ODIN out perform the other methods. Understanding the faults of pixel-level OOD detectors is crucial for progress. A possible future study would include categorising the failure cases of a detector. For example, understanding why a flooded road is not highlighted, and what makes that different to shadows falsely being highlighted. Great improvements in deep learning applied to image processing were made when spatial correlations were taken advantage of (*i.e.* the convolution). All methods used in this thesis are applied per pixel, not using information of neighbouring pixels. Incorporating this information might have great performance boosts.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from https://www.tensorflow.org.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[3] Petra Bevandic, Ivan Kreso, Marin Orsic, and Sinisa Segvic. Discriminative out-of-distribution detection for semantic segmentation. *CoRR*, abs/1808.07703, 2018.

[4] Hermann Blum, Paul-Edouard Sarlin, Juan I. Nieto, Roland Siegwart, and Cesar Cadena. The fishyscapes benchmark: Measuring blind spots in semantic segmentation. *CoRR*, abs/1904.03215, 2019.

[5] Paul Bodesheim, Alexander Freytag, Erik Rodner, and Joachim Denzler. Local novelty detection in multi-class recognition problems. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 813–820. IEEE, 2015.

[6] Paul Bodesheim, Alexander Freytag, Erik Rodner, Michael Kemmler, and Joachim Denzler. Kernel null space methods for novelty detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3374–3381, 2013.

[7] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[8] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[10] Pawel Cichosz, Dariusz Jagodziński, Mateusz Matysiewicz, Łukasz Neumann, Robert M Nowak, Rafał Okuniewski, and Witold Oleszkiewicz. Novelty detection for breast cancer image classification. In *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016*, volume 10031, page 1003135. International Society for Optics and Photonics, 2016.

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[13] Taylor Denouden, Rick Salay, Krzysztof Czarnecki, Vahdat Abdelzad, Buu Phan, and Sachin Vernekar. Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance. *arXiv preprint arXiv:1812.02765*, 2018.

[14] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[15] Xuemei Ding, Yuhua Li, Ammar Belatreche, and Liam P Maguire. Novelty detection using level set methods. *IEEE Transactions on Neural Networks and Learning Systems*, 26(3):576–588, 2014.

[16] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[17] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] Jia Guo, Guannan Liu, Yuan Zuo, and Junjie Wu. An anomaly detection framework based on autoencoder and nearest neighbor. In *2018 15th International Conference on Service Systems and Service Management (ICSSSM)*, pages 1–6. IEEE, 2018.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[21] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[22] Dan Hendrycks, Mantas Mazeika, and Thomas G Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[24] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009.

[25] Vilen Jumutc and Johan AK Suykens. Multi-class supervised novelty detection. *IEEE transactions on pattern analysis and machine intelligence*, 36(12):2510–2523, 2014.

[26] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[30] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *arXiv preprint arXiv:1807.03888*, 2018.

[31] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6391–6401, 2018.

[32] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[34] Juncheng Liu, Zhouhui Lian, Yi Wang, and Jianguo Xiao. Incremental kernel null space discriminant analysis for novelty detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 792–800, 2017.

[35] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. [Online; accessed June 3, 2019] `http://www.numpy.org/`.

[36] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July 1985.

[37] Trung Pham, Vijay BG Kumar, Thanh-Toan Do, Gustavo Carneiro, and Ian Reid. Bayesian semantic instance segmentation in open set world. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018.

[38] Peter Pinggera, Sebastian Ramos, Stefan Gehrig, Uwe Franke, Carsten Rother, and Rudolf Mester. Lost and found: detecting small road hazards for self-driving vehicles. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1099–1106. IEEE, 2016.

[39] Marcel Prastawa, Elizabeth Bullitt, Sean Ho, and Guido Gerig. A brain tumor segmentation framework based on outlier detection. *Medical image analysis*, 8(3):275–283, 2004.

[40] Charles Richter and Nicholas Roy. Safe visual navigation via deep learning and novelty detection. *Robotics: Science and Systems XIII*, 2017.

[41] Patrick Ross, Andrew English, David Ball, Ben Upcroft, and Peter Corke. Online novelty-based visual obstacle detection for field robotics. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3935–3940. IEEE, 2015.

[42] Volker Roth. Kernel fisher discriminants for outlier detection. *Neural computation*, 18(4):942–960, 2006.

[43] Mohammad Sabokrou, Mohsen Fayyaz, Mahmood Fathy, Zahra Moayed, and Reinhard Klette. Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes. *Computer Vision and Image Understanding*, 172:88–97, 2018.

[44] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International Conference on Information Processing in Medical Imaging*, pages 146–157. Springer, 2017.

[45] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.

[46] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[47] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[48] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[51] Ingo Steinwart, Don Hush, and Clint Scovel. A classification framework for anomaly detection. *Journal of Machine Learning Research*, 6(Feb):211–232, 2005.

[52] Benjamin Berry Thompson, Robert J Marks, Jai J Choi, Mohamed A El-Sharkawi, Ming-Yuh Huang, and Carl Bunje. Implicit learning in autoencoder novelty assessment. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, volume 3, pages 2878–2883. IEEE, 2002.

[53] G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995. https://www.python.org/.

[54] Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L Willke. Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 550–564, 2018.

[55] Huan-gang Wang, Xin Li, and Tao Zhang. Generative adversarial network based novelty detection usingminimized reconstruction error. *Frontiers of Information Technology & Electronic Engineering*, 19(1):116–125, 2018.

[56] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

[57] William J Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950.

[58] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Robert Fergus. Deconvolutional networks. In *Cvpr*, volume 10, page 7, 2010.

[59] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.

[60] Qi-Feng Zhou, Hao Zhou, Yong-Peng Ning, Fan Yang, and Tao Li. Two approaches for novelty detection using random forest. *Expert Systems with Applications*, 42(10):4840–4850, 2015.