Out-of-Distribution Detection with Deep Hybrid Models

A thesis

submitted in fulfilment of the requirements for the degree

of

Master of Science (Research) in Artificial Intelligence

at

The University of Waikato

by

Paul-Ruben Schlumbom

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2024

Abstract

Deep learning systems suffer from "silent failures" where they make highly confident, but incorrect, predictions for input instances well outside of their training data. This motivates the development of out-of-distribution (OOD) detection for such systems: the ability to recognise when an input deviates significantly from the training data. The "deep hybrid model" (DHM) for image classification presented by Cao and Z. Zhang (2022a) uses a normalising flow to perform density estimation for OOD detection and addresses shortcomings of approaches that model pixel-space densities: it performs density estimation using classifier features. Remarkably, Cao and Z. Zhang (2022a) claim 100% detection accuracy on a number of common benchmarks but do not make their code available. As we find the principles behind the DHM interesting and sound, we reimplement it to either confirm its capabilities or understand why it falls short. We perform an extensive search over possible model configurations to maximise performance and provide a detailed record for best practice.

Although unable to achieve 100% detection accuracy in our experiments, the DHM delivers competitive performance with careful fine-tuning, while exhibiting great sensitivity to hyperparameter settings. We argue that this is predominantly due to an adversarial relationship between the classifier and the normalising flow that can result in the collapse of the feature space. We verify this by means of several synthetic datasets and show that one of the assumptions underlying the DHM architecture, that the feature extractor can be regularised to preserve input-space densities in feature space, is not satisfied, thereby providing an understanding of where the DHM falls short and informing the development of future OOD detectors based on modelling feature space densities. We also evaluate the DHM on a real-world dataset of endemic and invasive stink bugs in New Zealand that poses a fine-grained OOD problem due to the high visual similarity between the bug species. Low DHM performance, compared to OOD benchmarks, reveals the benefit of testing OOD systems in real-world settings.

Acknowledgements

First and foremost, I'd like to thank my supervisor, Professor Eibe Frank, for lending me his insightful guidance and advice, for his friendly encouragement, and for always having his door open to discuss any problems I encountered. The discussions we had were incredibly fruitful, and quite educative for me. I'm also very grateful for his ceaseless patience in reviewing the countless revisions I sent his way. The thesis wouldn't be half as refined as it is now without him.

My thanks also go to Professor Albert Bifet, who along with Professor Eibe Frank and Professor Bernhard Pfahringer originally brought me into the Te Ipu o te Mahara Artificial Intelligence Institute as a research programmer and encouraged me to pursue my interest in machine learning to the next stage.

On that note I'd also like to thank the Te Ipu o te Mahara Artificial Intelligence Institute at the University of Waikato for funding my degree, as well as providing me access to our plucky A100. My thanks also go to the TAIAO community, which at our various workshops and conferences has been a great source of inspiration with the various exciting machine learning projects everyone is working on.

My thanks must of course go the machine learning lab posse as well; Dr. Anany Dwivedi, Dr. Nick Lim, Dr. Guilherme Cassales, (soon to be) Dr. Hongyu Wang, Simone Mungari, Justin Liu, Yibin Sun, Nilesh Verma, Filippo Leveni, Ayman Chaouki, and Syuhaida Safian, who also kept me on track on the administrative side of things. I hope I haven't forgotten anyone. The advice was helpful, and the tea-break discussions were fun and most assuredly productive, if not always by an easily quantifiable measure.

Contents

Chapter 1

Introduction

Over recent years advances in deep learning have significantly enhanced machine learning capabilities, promising a tremendous range of new possibilities for autonomous systems. However, deep learning systems come with their own set of issues, and have proven to be particularly vulnerable to overconfidence, especially when faced with situations that fall outside of their training domain. This motivates the development of systems that are more sensitive to samples that fall outside of the distribution of the training data; this characteristic is called the out-of-distribution (OOD) detection capability of a system.

1.1 Motivation

Out-of-distribution detection is seen as a crucial component of reliable AI systems in safety-critical domains, such as autonomous vehicles or medical diagnosis systems (Amodei et al., 2016; Yang et al., 2021). Deep learning systems in particular are known to exhibit "silent failures", characterised by highly overconfident yet incorrect predictions (Guo et al., 2017). This behaviour in particular severely limits the deployment of deep learning based systems, despite their on average high performance on a number of tasks.

Imperfect systems can in principle still be a great productivity boost if a human operator can compensate for their shortcomings. For example, an autonomous bus that is *mostly* successful in navigating a city, while seemingly

failing at being a fully autonomous driving system, could still be useful if it were capable of handing remote control to a human overseer when it encounters a challenging scenario. Such an operator could be responsible for multiple vehicles at once, representing an overall boost in productivity.

As another example, one could imagine a medical system designed to diagnose and treat common ailments that works extremely well for most patients, but then confidently fails to recognise when a patient presents the symptoms of something rare but more serious, or even novel. A system capable of reliably identifying these out-of-distribution situations could divert such a patient to a skilled professional, enabling a much more affordable and scalable healthcare system. Out-of-distribution detection systems could potentially unlock deep learning applications for a much wider, more risk-averse set of domains.

Generally speaking, even a system that makes correct decisions 99% of the time will still be unusable in many domains if it is incapable of recognising when it encounters the 1% of cases it cannot handle reliably. While high in-distribution performance promises to considerably reduce the workload on human operators, the inability to recognize unusual situations (and therefore hand the problem to a human) could result in catastrophic costs that outweigh the potential benefits gained from increased automation.

There exists therefore a strong motivation to develop deep learning approaches that are far more sensitive to the distinctions between in-distribution and out-of-distribution inputs. In this thesis we investigate a proposed architecture, the deep hybrid model, for which remarkable performance has been reported.

1.2 The Deep Hybrid Model

Recently, Cao and Z. Zhang (2022a) proposed a deep learning architecture that they called a deep hybrid model, or DHM. This architecture augments a classification model with a highly flexible density estimator, a normalising flow,

to jointly model the distribution of the features alongside the in-distribution class likelihoods.

Normalising flows have commonly been used to operate in pixel space, particularly for image generation tasks (see for example Dinh, Sohl-Dickstein, and S. Bengio (2016) and Durk P Kingma and Dhariwal (2018)), where they prove to be quite effective at modelling high-dimensional data. As normalising flows compute an exact density estimate for their inputs, they also seem a natural fit for performing out-of-distribution detection tasks. However, it was found that normalising flows are highly ineffective for OOD detection, often assigning higher densities to out-of-distribution samples than to in-distribution samples (Nalisnick et al., 2018). This was found to be because normalising flows prioritise modelling low-level features in images and so struggle to recognise higher-level conceptual drift that is usually indicative of OOD instances.

The DHM therefore sought to resolve this shortcoming by having the normalising flow model the classifier features instead, which must necessarily capture high-level features for classification to be done effectively. The principle seems sound; however, Cao and Z. Zhang (2022a) report astonishing results of 100% in a number of OOD detection benchmarks, a claim that warrants closer scrutiny. At the same time, Cao and Z. Zhang (2022a) do not make their model publicly available.

This thesis investigates the DHM architecture: it attempts to replicate it and its results, and, finding it to fall short of reported performance, investigates why this should be. In so doing we explore extensively the requirements for effective out-of-distribution detection systems.

1.3 Overview

This thesis begins with a description of the relevant theory and technical knowledge required to understand the DHM and interpret its results, and then describes the experiments that were carried out, before discussing their results.

Chapter 2 begins with a description of the relevant theoretical background, briefly describing deep neural networks and going into more detail on the variants of convolutional neural networks, as these are very relevant to image processing tasks. The out-of-distribution task is described, and narrowed down in particular to density-based approaches to out-of-distribution detection. Finally, normalising flows are described in detail; a number of different approaches to implementing normalising flows are described, particularly residual normalising flows.

Chapter 3 goes into technical detail of how the various components of the DHM are implemented, as well as how the DHM itself is assembled. We also describe the datasets used in this thesis, and discuss the challenges they pose in an OOD detection framework.

Chapter 4 focuses on experimental results. Normalising flows are briefly demonstrated on synthetic datasets to explore their behaviour. We describe an extensive search through the possible configurations of the described DHM model in an attempt to replicate its performance. While we report high performance, we are unable to attain 100% OOD detection rates, and we attribute shortcomings in performance partially to collapsing features. We make the code for our DHM implementation publicly available[1]. We then describe the creating of a new synthetic dataset to investigate this behaviour.

Chapter 5 frames the issue as fundamentally revolving around the properties of the feature space, and discusses approaches to feature space regularisation. We then demonstrate that the DHM is vulnerable to a form of feature collapse that is a direct consequence of its architecture, and propose a loss penalty term to help address it. Finally, we apply the DHM on a real world problem of fine-grained OOD detection, with the aim of distinguishing invasive bugs from closely related endemic species.

Chapter 6 summarises the thesis and provides concluding remarks.

Finally, Chapter A contains the appendix.

[1] https://github.com/P-Schlumbom/deep-hybrid-models

Chapter 2

Background

2.1 Convolutional Neural Networks

2.1.1 Overview

Convolutional Neural Networks (CNNs) are a variant of neural networks that are specialised for image processing. They achieved success in the field early on, with one of the first practical applications being a handwritten digit classifier demonstrated by LeCun et al., 1995 in 1998. Later on, in 2012, Alex Krizhevsky et al. were able to demonstrate state of the art performance with a classifier trained on the ImageNet dataset (Krizhevsky, Sutskever, and G. E. Hinton, 2017); they also showed that large, deep CNNs could be trained efficiently on GPUs, making CNNs not only effective but also widely accessible and easily trainable. They remain a popular technique for complex computer vision tasks, and are widely applied.

The advantage of CNNs comes from the fact that they are specialised for image processing by exploiting certain properties that images tend to exhibit. Where basic Fully Connected (FC) neural networks treat each element of an input tensor as a unique feature, CNNs introduce the inductive bias of assuming spatial correlation. In other words, features that are close together in the input tensor are assumed to be correlated with one another, whereas features that are far apart are assumed to be uncorrelated. This allows CNNs to ex-

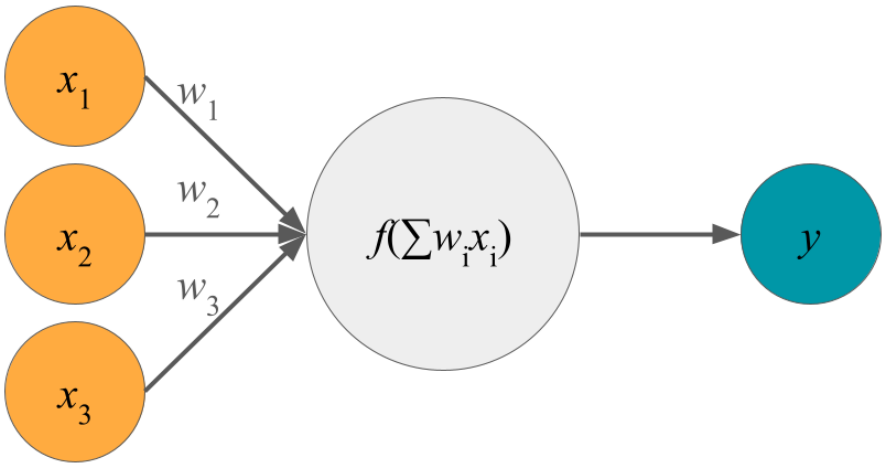


Figure 2.1: A single fully connected neural network layer.

tract as much or more information from images than FC networks, while using fewer parameters and therefore being more robust to overfitting (as there is less capacity for memorisation). In practice, this means that CNNs can achieve greater performance than FC networks in image processing tasks, potentially even for a lower computational cost, making them the vastly preferred option for computer vision applications. In the following section we first revisit the key concepts of fully connected networks before discussing convolutional networks in greater detail.

2.1.2 Fully Connected Neural Networks

2.1.2.1 Basic Principles

A fully connected neural network is organised into layers of "neurons" or units, where each neuron receives as input the weighted sum of the outputs of all neurons in the previous layer; hence the term, fully connected. The output of a neuron is then calculated by passing the weighted sum into a non-linear function. In more formal mathematical terms, this operation can be seen as a matrix operation. Given two layers l_1 and l_2 with m and n neurons, respectively, and a size-m vector of activations from the first layer of neurons

called $a_1 \in \mathbb{R}^{1\times m}$, we can calculate the pre-activation for each neuron in l_2 by multiplying a_1 with a weight matrix $W_1 \in \mathbb{R}^{m\times n}$ and adding the bias term $b_1 \in \mathbb{R}^{1\times n}$. The pre-activation is then passed into the non-linear function f to get the activations of layer l_2, $a_2 \in \mathbb{R}^{1\times n}$:

$$a_2 = f(a_1 W_1 + b_1)$$

A fully connected neural network can then simply be defined as a series of such matrix multiplications and activations. So, for example, a very simple 3-layer network with layers l_1, l_2, and l_3 (each with corresponding parameters W_i, b_i) can be defined as the operation:

$$output = f(f(f(input \times W_1 + b_1) \times W_2 + b_2) \times W_3 + b_3)$$

Practical implementations will also include a number of other components to improve performance, but this represents the core principle. The non-linear function f is crucial for achieving the complex representational capacity of deep neural networks, since without it the network would simply be a sequence of matrix multiplications, which can always be simplified into a single matrix multiplication. That is, without the nonlinearity at each layer, a neural network would never be anything more than a simple linear transformation, no matter the depth. The activation function must therefore be a function which is non-linear (i.e. not the identity operator) and, for the benefit of gradient descent, continuous. In practice, functions such as the sigmoid or ReLU activation are preferred.

The sigmoid function,

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

was originally quite popular and maps input values into a 0 to 1 range. However it has some issues, the most notable being the vanishing gradients problem: for input values of large magnitude, the sigmoid function has gradients that approach 0, which can cause problems for gradient descent operations. Additionally, the function is also relatively expensive to compute, since it relies

on both exponentiation and multiplication/division operators. Nowadays one of the most commonly preferred alternatives is the ReLU activation, which simply returns the identity for all $x > 0$ and 0 otherwise:

$$ReLU(x) = \max(0, x)$$

This is far more efficient to compute, reduces (but does not eliminate) the vanishing gradients effect, and still adheres to the non-linearity condition.

Any layer in a neural network that is neither the input nor the output layer is called a *hidden* layer. Any neural network with at least one hidden layer is technically considered a *deep neural network*. It is quite common, for historical reasons, to refer to a deep fully connected neural network as a *multi-layer perceptron*, abbreviated to *MLP*.

When it comes to image processing, MLPs suffer from some shortcomings due to both the inherently high dimensionality of even small images and the particular properties of image data. Dimensionality is the first problem: even for small $32 \times 32 \times 3$ RGB images, one is already dealing with 3072 dimensions, and in practical applications one would typically expect to deal with images on the order of hundreds of pixels to a side. In a fully connected network the number of parameters between any two layers l_i and l_j with m and n units respectively is $mn + n$; thus the number of parameters will scale at least linearly with the size of the input, or quadratically with the width of the input image. Moreover in practice, the number of units of the hidden layers will be increased as well to accommodate the increased complexity of the data. This yields a massive explosion in the number of parameters required to handle even relatively small images, and also quickly introduces issues of overfitting.

The second issue is that the inductive bias of MLPs ignores some important properties of image data: namely, the spacial correlation of pixels and, by extension, the spacial invariance of features. The inductive bias of a machine learning algorithm refers to the underlying set of assumptions it makes about the structure of the patterns to be learnt. MLPs implicitly treat each image as a single global pattern to be matched. This both makes them more liable to

simply memorise training images - leading to poor testing performance - and ignores the fact that relevant features in an image tend not to be determined by the value of any specific pixel, but rather to patterns of values across multiple neighbouring pixels. Plainly, if there are two images of a football, but one contains the football in the upper left corner while the other has it in the lower right corner, these are both still images of a football, despite the large difference in values of specific pixels.

2.1.3 Convolutional Layers and the CNN

Convolutional neural networks are designed to address these shortcomings. Convolutional neural networks take advantage of spacial correlation properties of images to process visual features using only a fraction of the parameters.

2.1.3.1 The Convolutional Layer

In the context of deep learning, the convolution operation involves taking the weighted sum of a set of neighbouring inputs and passing the result through a non-linear activation function. This is very similar to what happens in the fully connected network, with the crucial difference that this is only done to a local neighbourhood of inputs. The same weights are then reused to compute the output of another neighbourhood, and another, until the whole input has been processed. In this way a relatively small number of parameters can be used to process large inputs; and this approach necessarily captures the relationship between neighbouring inputs rather than matching global patterns. This can be easily demonstrated with a 1D example: consider a sequence of values $x_0, x_1, ..., x_N$. We then apply a convolutional kernel of size 3 with weights w_1, w_2, w_3, bias b, and activation function f. We can compute the first activation, a_0, with the operation:

$$a_0 = f(x_0 w_0 + x_1 w_1 + x_2 w_2 + b)$$

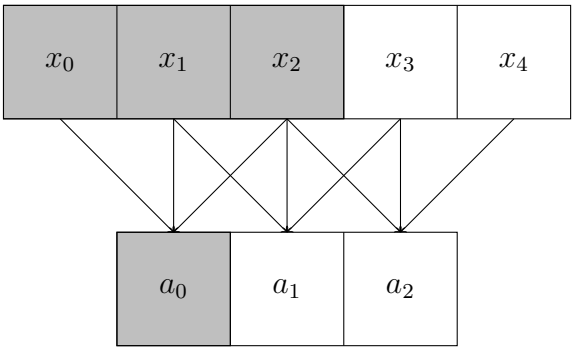


Figure 2.2: Example of a filter of size $k = 3$ convolved over a 1D input array of size 5, resulting in an output array of size 3.

(Alternatively we might pad the sequence to the left of x_0 with zeroes and begin the convolution on the input $0, x_0, x_1$). Then we can shift the kernel one step to the right and compute:

$$a_1 = f(x_1 w_0 + x_2 w_1 + x_3 w_2 + b)$$

And so on, until we get the sequence of activations $a_0, a_1, ..., a_{N-2}$ in this particular case. Note of course that if we vary the size of the kernel, the step, or pad the beginning and end of the sequence, we will vary the length of the resulting activation sequence; see Figure 2.2, which provides an example with a 1D array of size 5.

The kernel size k determines the size of the input field. Naturally, a larger k will result in a greater area of the input grid being captured with each step, with the tradeoff being that a larger number of parameters is required. In typical image processing tasks, where the input is a 2D grid, the convolutional kernels will likewise be a 2D grid (see Figure 2.3), and so the number of parameters rises quadratically in the size of the kernel. In the extreme case where k is the same as the width of the image, we are back to having a fully connected neural network, with the associated problems mentioned earlier. When CNNs were becoming popular, it was common to vary the size of k significantly, searching for a balance between the presumed increased capacity of larger kernels to capture complex patterns against the increased parameter cost. However, in contemporary architectures $k = 1$, 3, or 5 are the most com-

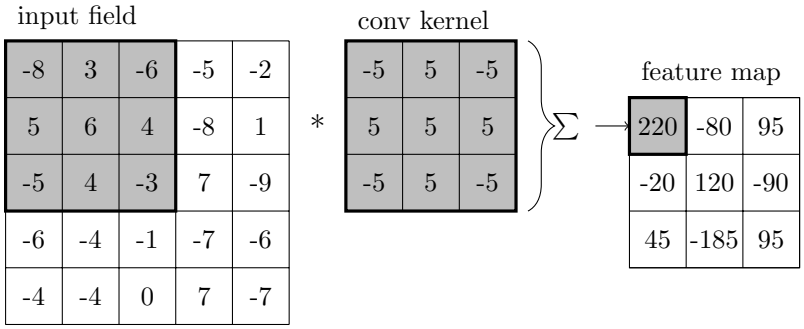


Figure 2.3: In a 2D convolution, a filter of learned weights (the convolutional kernel) is convolved over the input image, resulting in the feature map. The activation function is then applied to this feature map.

monly used values; this is far more efficient parameter wise and the stacking of multiple convolutional layers means small kernels can still capture large areas of the input in aggregate.

Naturally, a larger kernel size will also reduce the size of the output. This can be avoided by padding the ends of the input by $\lfloor k/2 \rfloor$, usually with zeroes. One can also vary the step size (or stride) s of the convolutional operation. So far the examples have involved shifting the kernel by one step for each operation ($s = 1$) but this can be increased. A greater stride means faster processing of the input and a smaller output size - which will also speed up the processing of subsequent layers - but it may also lose some information from the input due to sparser sampling. It is common to use larger strides in earlier layers to quickly reduce the size of the input to be processed.

Generally speaking, the size of a convolutional layer's output will be

$$w' = \lfloor \frac{w - k + 2p}{s} \rfloor + 1, \tag{2.1}$$

where w (i.e. 'width') is the size of the input, k is the kernel size, p is the number of padded values added to the ends of the sequence, and s is the stride.

2.1.3.2 Channels

Colour images are typically encoded as three stacked matrices corresponding to red, green, and blue intensities - i.e., three colour channels. The convolutional

kernel will take as input all three channels at the same time as a 3D tensor - a stack of three 2D tensors of the same size, one for each channel. Convolutional layers will also typically have multiple kernels, such that each can specialise in capturing a different input feature. For each patch from the input, these kernels will be applied in parallel, and each will produce its own output. The output produced by a particular kernel in a convolutional layer is often called a feature map, or filter map. The feature maps produced by a convolutional layer will then be stacked together much as the RGB layers are stacked together in a colour image, and so each feature map is also called a channel. This stack of feature maps is often called a convolutional image, and it can be processed in exactly the same way as the original RGB image was; however, the filters in the next convolutional layer may require more dimensions - one for each channel.

As mentioned before, the operation of a single kernel is the same as that of a single neuron in a fully connected neural net, but it is applied repeatedly across the input (convolutional) image. Therefore the number of kernels (or filters) in a convolutional layer is the number of neurons in this layer, and so the number of kernels in a convolutional layer is also called the convolutional layer's 'width'.

2.1.3.3 Pooling Layers

Another important building block of the CNN is the pooling layer. The pooling layer is essentially a downsampling step: it calculates an aggregate value for the pixels in a particular input patch (on a per-channel basis), typically either the maximum or the average, and produces a smaller output convolutional image. This has the dual benefit of first, helping to make the new representation slightly invariant to translations of a few pixels (which in most cases should not matter) and second, reducing the input space for the next layer - thus reducing the computational cost and memory requirements, making the network more efficient. Naturally, there is the usual trade-off to be made in that if the input

space is reduced too aggressively, the CNN might lose useful information for making its predictions, and as one would expect there is a point where further increases in efficiency incur a loss in performance. Classifiers used throughout this work often use a maxpooling layer - which takes the maximum value in an input patch - with a pooling window size of 3, a step of 2, and a padding size of 1. This will reduce the width of an input by half.

2.1.3.4 The Convolutional Neural Network

The Convolutional Neural Network (CNN) is, at its most basic, simply a stack of convolutional layers, as alluded to before. In practice, other mechanisms, such as pooling layers, are added in as well. Finally, it is common to also stack one or several fully connected layers on to the end of a CNN. The original LeNet (LeCun et al., 1995) and AlexNet (Krizhevsky, Sutskever, and G. E. Hinton, 2017), for example, consisted purely of convolutional, maxpooling, and fully connected layers. However, since AlexNet's success in 2012 there has been an explosion of variants to the basic convolutional architecture, which further improved upon both the efficiency and performance of the basic CNN architecture.

Some notable contributions include the Network in Network (NiN) approach (Lin, Q. Chen, and Yan, 2013), which introduced the use of 1x1 convolutional kernels. These can essentially provide a non-linear transformation of the input feature maps while preserving their dimensions and information content, which was shown to provide some benefits. NiN also introduced the idea of replacing the final fully connected layers entirely with an average pooling operation over the final convolutional image, where this final image would have one channel for each class. This idea has been adopted in a number of other architectures. Around this time, VGGNet (Simonyan and Zisserman, 2014) demonstrated the value of substantially increasing the depth of CNNs, while also relying on smaller kernel sizes. GoogLeNet (Szegedy et al., 2015) experimented with combining multiple different kernel sizes in the same layer, to

capture multiple features of different scales at the same time; both of these architectures were able to achieve significant improvements over AlexNet. However, of particular interest with respect to this thesis is the ResNet architecture (He et al., 2016) introduced shortly afterwards.

2.1.4 Residual Networks

2.1.4.1 Introduction

Residual Networks, commonly referred to as ResNets, were initially introduced by He et al. (2016), and were intended to address a number of issues identified with basic CNNs, with the goal of creating much deeper networks. The approach achieved considerable success, with up to 152 layer models (8 times deeper than VGGNet) being demonstrated on ImageNet, and a 1000 layer model being demonstrated on the CIFAR-10 dataset. To understand both the challenges ResNets were intended to address, as well as the key components they are composed of, it is worth going over the concepts of vanishing and exploding gradients, and batch normalisation, first.

2.1.4.2 Vanishing and Exploding Gradients

As mentioned above, VGGNet showed early on that increasing the depth to CNNs also improved their performance - a result generally expected for deep neural networks. Since each layer essentially captures features made up of lower-level features captured by the previous layer, it follows that the deeper a deep neural net is, the greater the complexity of the features it can capture. However, early attempts at creating deeper DNN architectures quickly ran into the vanishing and exploding gradients issues. This is a problem where, without careful adjustments of the weights in each layer, the activations during the forward pass (and therefore the gradients during the backwards pass) will either grow or shrink at an exponential rate, depending on the distribution of the parameters in each layer.

To see why this would be, consider a single fully-connected layer with a ReLU

activation. Let its bias b be 0 for simplicity, and let its weight matrix w have mean $\mu_w = 0$ and some variance σ_w^2. We now pass in an input x with mean $\mu_x = 0$ and variance σ_x^2. By definition, $\sigma_x^2 = E[x^2] - E[x]^2$, so in this case we have:

$$\sigma_x^2 = E[x^2] \tag{2.2}$$

Similarly, we also have:

$$\sigma_w^2 = E[w^2] \tag{2.3}$$

Every unit in the output pre-activation is computed by the dot product between D-dimensional input x and the corresponding column in w; that is, for unit i of the pre-activation h, we have $h_i = \sum_j^D w_{ij}x_j$. Since both w and x had mean 0, we expect that $\mu_h = 0$ as well, and so again we have $\sigma_h^2 = E[h^2]$. It follows then that:

$$\begin{aligned}
\sigma_h^2 &= E[(\sum_j^D w_{ij}x_j)^2] \\
&= E[\sum_j^D w_{ij}x_j \sum_j^D w_{ij}x_j] \\
&= \sum_j^D E[w_{ij}x_j]E[w_{ij}x_j] \\
&= \sum_j^D E[w_{ij}]E[x_j]E[w_{ij}]E[x_j] \\
&= \sum_j^D E[w_{ij}^2]E[x_j^2] \\
&= DE[w^2]E[x^2]
\end{aligned} \tag{2.4}$$

By the identities established in equations 2.2 and 2.3, we finally get:

$$\sigma_h^2 = D\sigma_w^2\sigma_x^2 \tag{2.5}$$

When we now calculate the activations a using the ReLU function, half of

these values are of course set to 0 (assuming h is symmetric around 0), thus halving the variance:

$$\sigma_a^2 = \frac{1}{2} D \sigma_w^2 \sigma_x^2 \tag{2.6}$$

Bearing in mind that a will be used as the input for the next layer, we can generalise this statement to $\sigma_{a_{n+1}}^2 = \frac{1}{2} D \sigma_w^2 \sigma_{a_n}^2$. For a network with N layers (assuming for the sake of simplicity that all are of size D and initialised with $\mu = 0$ and variance σ_w^2), we can see that the variance of the final output a_N will be:

$$\sigma_{a_N}^2 = \frac{1}{2^N} D^N \sigma_w^{2n} \sigma_{a_o}^2 \tag{2.7}$$

In other words, looking at the term in Equation 2.6, whenever $\frac{1}{2} D \sigma_w^2 \neq 1$ we can expect either an exponential increase or an exponential decrease in the variance of the activations as the network becomes deeper.

Note that, when the gradients of the parameters with respect to some loss function are backpropagated through the network, the gradients are still multiplied by the (transposed) weight matrices of the network, and so their variances are still dependent on the size and variance of the weights in each layer. In other words, the gradients also risk exploding or vanishing for unsuitable initialisations of the layer weights.

The problem can be mitigated by careful initialisation of the weights, i.e. solving $\frac{1}{2} D \sigma_w^2 = 1$ for σ_w^2 for each layer, which yields:

$$\sigma_w^2 = \frac{2}{D} \tag{2.8}$$

This is known as the He initialisation (He et al., 2015), and other initialisation approaches exist as well. However, good initialisations can only mitigate the gradient problems, not outright eliminate them. The purpose of training is to update the weights w, and inevitably these will deviate from the ideal

distribution. Since the deviation is exponential in the depth of the network, we can see that the deeper the network, the more sensitive it must become to even slight deviations in the weight variances; in effect, deep neural networks will become more unstable the deeper they are.

2.1.4.3 Batch Normalisation

Another relevant technique that now sees widespread adoption in deep learning computer vision architectures is batch normalisation. Where previously we described He initialisation as a means of controlling the expected variance of layer activations - thus mitigating vanishing and exploding gradients - batch normalisation takes a more direct approach by normalising the outputs of a layer to a particular mean and standard deviation.

Specifically, given a batch of inputs X output by some previous layer, batch normalisation computes, for each channel i, the mean μ_i and standard deviation σ_i across that batch. Each channel c_i in each sample is then updated with:

$$c_i \leftarrow \frac{c_i - \mu_i}{\sigma_i + \epsilon} \tag{2.9}$$

where ϵ is some small non-zero number to prevent a division by zero error. Each channel is then rescaled to mean δ and standard deviation γ:

$$c_i \leftarrow \gamma c_i + \delta \tag{2.10}$$

In this way the mean and variance of each output channel can now be explicitly set. A running average will also be kept for each μ_i and σ_i during training, so that they can be used for individual samples when the trained model is deployed.

2.1.4.4 Residual Networks

The ResNet architecture introduced the residual block, which reformulated the problem from simply transforming an input (i.e. $y = f(x)$) to one of estimating the residual of the transformation ($y = x + f(x)$). Typically, each residual

block consists of a batch normalisation operation, followed by a ReLU activation and a convolutional layer. Ordinarily, the activation function is applied after the linear transformation; however in this case, this would result in $f(x)$ being always positive, and so a ResNet with ReLU activations at the end of every residual block would in effect only be able add to the input, restricting its representational capability. The convolutional layer in the residual block is typically a 3×3 convolutional layer with padding and stride adjusted appropriately to produce an output of the same width and height as the input. This is typically achieved by setting stride and padding to 1; a quick substitution into Equation 2.1 verifies that these parameters (i.e. $k = 3$, $p = 1$, $s = 1$) will produce an output of the same size. The number of output channels might be the same or different, but since the strength of CNNs comes partly from reducing the width and increasing the depth of the input (i.e., transforming the input from simple, spatially variant features to complex, spatially invariant features), at least some of the residual layers must increase the depth of the input. In order for the $x + f(x)$ operation to still be valid, in this case x will be passed through a single 1×1 convolutional layer with a number of output channels that matches that of $f(x)$.

One of the original intentions of ResNets was to enable the construction of deeper CNNs, and to yield the improved representational capacity that is expected with deeper networks. There are a number of reasons why deeper neural networks struggle to learn good solutions: firstly, the gradients of deeper networks tend to be more unstable, as even slight changes in the weights of earlier layers can result in considerable changes in activations to build up by later layers, as demonstrated in Section 2.1.4.2. It has also been observed that for deeper networks, the correlation between gradients for slight changes in the input x quickly drops off as the number of layers increases, a phenomenon Balduzzi et al. (2017) describe as "shattered gradients". This has led to the hypothesis that deeper neural networks struggle to produce a smooth loss function with respect to the weights, thus degrading the ability of gradient descent

based methods in optimising the parameters. As will be shown below, ResNets essentially keep the outputs from earlier layers, allowing the loss function to directly influence the updates of earlier layers.

To see how the residual architecture might aid in achieving greater practical depths, we can look at an example ResNet with 3 residual blocks.

Consider the first residual block, $f_1(x)$. The output of this block is naturally:

$$x' = x + f_1(x) \tag{2.11}$$

This output is then passed on to the second residual block, f_2:

$$\begin{aligned} x'' &= x' + f_2(x') \\ &= x + f_1(x) + f_2(x + f_1(x)) \end{aligned} \tag{2.12}$$

Finally we compute y by passing x'' into the third residual block f_3:

$$\begin{aligned} y &= x'' + f_3(x'') \\ &= x + f_1(x) + f_2(x + f_1(x)) + f_3(x + f_1(x) + f_2(x + f_1(x))) \end{aligned} \tag{2.13}$$

Thus the final output is essentially the combined output of 3 different networks (or 4, if one thinks of x as a network that computes the identity), each with a different depth. This ensures that each layer can contribute directly to the output, and gradients do not have to be propagated through multiple subsequent layers to compute updates for any particular layer. This is believed to be the reason why ResNet-based architectures tend to have smoother loss functions with respect to their gradients (Balduzzi et al., 2017; H. Li et al., 2018).

ResNets also eliminate the possibility of vanishing gradients, since the activations of earlier layers cannot be lost through the sequential multiplications of a standard neural network. Exploding gradients remain a problem: each residual block adds the variance of the residual function output to the variance of the original input, effectively doubling it even when the residual function is

well initialised, resulting in an exponential growth in variance of activations and, by extension, gradients.

For example, consider a residual block $x' = x + f_1(x)$, and suppose for simplicity x has a variance of 1, and f_1 is well initialised and preserves x's variance of 1. When both are added together again, x' is expected to have a variance of 2. Assuming the next residual block, $x'' = x' + f_2(x)$, has f_2 just as well initialised, x'' is therefore expected to have a variance of 4. Thus the variance doubles with each residual block. We can generalise this observation to say that a residual network where each residual function scales the variance of its input by c and the input x has a variance of σ_x, the variance is expected to grow after n residual blocks to $(c\sigma_x)^n$.

However, the inclusion of batch normalisation in each residual block eliminates this possibility, by re-scaling the outputs of each residual block to a target mean and variance.

Additionally, it is suspected that ResNets essentially act as ensembles of shallow neural networks, as later layers do not seem to contribute as much to a ResNet's performance as expected. For example, Veit, Wilber, and Belongie (2016) showed that removing individual residual blocks at test time had only minor impacts on performance, whereas this will cause catastrophic failures in sequential neural networks. Shuffling the residual blocks in a ResNet likewise has relatively minor impacts on error. They also found that earlier layers accounted for a disproportionately large amount of the total gradient updates during training, suggesting again that they are more relevant to the network's final performance. This indicated that the improved performance of ResNets could not be entirely attributed to their increased depths as initially expected These observations motivated the exploration of shallower, wider ResNets. Zagoruyko and Komodakis (2016) introduced Wide ResNets (WRNs) which traded depth for wider convolutional layers in the residual blocks (i.e. more channels per layer). They were able to show that a 16 layer WRN could match the performance of a 1000 layer ResNet of roughly equivalent parameter count,

while being significantly faster to compute due to the increased parallelism enabled by wider layers. It is these WRNs that were used in the Deep Hybrid Model architecture proposed by Cao and Z. Zhang (2022a), described in more detail in Section 3.2.

2.1.5 Sensitivity

A property of particular interest to us in this thesis is the *sensitivity* of a network; that is, how much its output changes in response to slight changes in the input. Spectral normalisation is a key component in influencing the sensitivity of a residual network, and is also one of the key building blocks of the deep hybrid model architecture.

2.1.5.1 Spectral Normalisation

One of the key building blocks of the Deep Hybrid Model architecture is spectral normalisation. The spectral norm of a matrix W, $\sigma(W)$, is the largest singular value of that matrix, in other words the square root of the maximum eigenvalue of the conjugate transpose of a matrix multiplied by itself (Miyato et al., 2018):

$$\sigma(W) = ||W||_2 = (\max_{\lambda}(W^H W))^{1/2} = \max_{||x||_2 \neq 0} \frac{||Wx||_2}{||x||_2},$$

where λ is the eigenvalue of $W^H W$. Note that in the case of a real-valued matrix, the conjugate transpose is simply the transpose of the matrix. In intuitive terms, the spectral norm of a matrix indicates the maximum factor by which a vector's magnitude will be scaled when multiplied with the matrix. In the context of deep learning this has obvious implications for stability: a spectral norm close to 1 on the weights of a neural network's layer would safeguard against vanishing or exploding gradients.

In the context of deep learning, spectral normalisation was first utilised by Yoshida and Miyato (2017) with the intention of improving regularisation. They argued that a crucial indicator of the generalisability of a model was

its sensitivity - that is, how much the loss function output can change in response to slight perturbations of the input. Given some neural network f, they show that $||f(x+\epsilon) - f(x)||_2$ (where ϵ is some small non-zero offset) can be minimized by keeping the spectral norms of the weight matrices low, and they add a regularization term to the loss function to penalise large spectral norms.

This idea was extended to spectral normalisation by Miyato et al. (2018), where the weight matrices are directly normalised to satisfy the condition $\sigma(W) = 1$:

$$\hat{W} = \frac{W}{\sigma(W)} \tag{2.14}$$

However, their implementation of spectral normalisation for convolutional layers reshapes the convolutional weight matrices to shape $c_{in} \times (c_{out} * k_h * k_w)$ and applies spectral normalisation to the resulting matrices. This is technically not a correct implementation, and Gouk et al. (2021) address this issue by deriving the true 2D weight matrix corresponding to the convolutional operation, for which the spectral norms are then computed. However, it is interesting to note that Miyato et al.'s approach still works quite well empirically. It should also be noted that Gouk et al. (2021) further adjust the normalisation by keeping the unnormalised weights when the spectral norm is below some coefficient c, i.e., W is set with:

$$\hat{W} = \frac{W}{\max(1, \frac{\sigma(W)}{c})} \tag{2.15}$$

Another important feature of spectral normalisation to note is its behaviour across multiple consecutive layers, as typically encountered in deep learning architectures.

Estimating the spectral norm Gouk et al. (2021) describe estimating the spectral norm via the power iteration method, which is a numerical approach for estimating the largest eigenvalue, and this is the method typically used now. Given the square matrix W, we make an initial guess x_0 for the eigenvector

and then compute

$$x_i := W^T W x_{i-1} \tag{2.16}$$

for n iterations. The estimate for the spectral norm is then calculated with:

$$\sigma(W) = \frac{||Wx_n||_2}{||x_n||_2} \tag{2.17}$$

Strictly speaking, the accuracy of the estimated spectral norm depends on the ratio between the largest and second largest eigenvalue; a small difference will lead to a low rate of convergence and will therefore require a greater number of iterations to produce an accurate estimate. However, it is common to use only a single iteration (i.e., $n = 1$) to estimate the spectral norm during training, as this has been found to be effective in practice (J. Liu et al., 2020).

2.1.5.2 Lipschitz Continuity

Spectral normalisation is particularly interesting in regards to ensuring Lipschitz continuity. A function f is Lipschitz continuous if there exists a Lipschitz constant L such that for every pair of points x and y in the domain of f, the following condition holds:

$$D_Y(f(x) - f(y)) \leq L \cdot D_X(x - y) \tag{2.18}$$

Where D_X and D_Y are distance metrics in the input and output spaces, respectively. In this work they are Euclidean distances unless mentioned otherwise. In other words, the Lipschitz constant for a function is the upper bound of the factor by which the distance between x and y can be changed when both are passed to f. It follows that constraining the spectral norm of a neural network's layer also sets the upper bound for its Lipschitz constant.

A useful feature of the Lipschitz constant is that the Lipschitz constant of a deep neural network as a whole is dependent on the Lipschitz constants of the individual components of that deep neural network. Tsuzuku, Sato, and Sugiyama (2018) show how the Lipschitz constants L_f and L_g for two functions f and g, respectively, will be bounded under the operations of composition:

$$f \circ g : L_f L_g \tag{2.19}$$

addition:

$$f + g : L_f + L_g \tag{2.20}$$

and concatenation:

$$(f, g) : \sqrt{L_1^2 + L_2^2} \tag{2.21}$$

From these rules we can derive the Lipschitz constants of more complex components in a deep neural network.

Identity function: For the identity function the Lipschitz constant is exactly 1.

ReLU activation: The ReLU activation does not change the input except by setting it to 0 for $x < 0$, so from 2.18 we have $L_{\text{ReLU}} \leq 1$.

Feedforward layer: Taking a single layer with a ReLU activation and weight matrix W, $f(x) = \text{ReLU}(W(x)+b)$: let the Lipschitz constant of $W(x)$ be L_W. The constant bias b has no effect on the gradient of the function and therefore has no effect on L_W, so $L_{W(x)+b} = L_W$. Since $L_{\text{ReLU}} \leq 1$, $L_f \leq 1 * L_W = L_W$.

Residual block: It then follows that a simple MLP with layers $l_1, ..., l_n$ and Lipschitz constants $L_1, ..., L_n$ will have its whole Lipschitz constant bounded by $\prod_{i=1}^{n} L_i$, i.e.

$$L_{\text{MLP}} \leq \prod_{i=1}^{n} L_i \tag{2.22}$$

For a residual block, we have $g(x) = x + f(x)$. Let the Lipschitz constant of f - usually an MLP - be L_f. Since the Lipschitz constant for the identity function is simply 1, it follows that:

$$L_{\text{resblock}} \leq 1 + L_f \tag{2.23}$$

Note that constraining the Lipschitz constant of the residual connection, L_f, also constrains the lower bound of the entire residual block Lipschitz constant. This is known as bi-Lipschitz continuity; see Appendix A.1 for the proof. This reaffirms the observation made in Section 2.1.4.4 about the resistance

Operation	**Lipschitz Constant Upper Bound**
identity	1
ReLU	1
max pooling	1
$f(x) + g(x)$	$L_f + L_g$
$f(g(x))$	$L_f L_g$
$x + f(x)$	$1 + L_f$

Table 2.1: Lipschitz constant upper bounds for various operations.

of residual networks to vanishing gradients; due to the lower bound on the residual block's Lipschitz constant, the rate at which activations can shrink is constrained to this lower limit.

Max pooling: Finally, max pooling layers are also 1-Lipschitz continuous at worst, since the maxpooling operation simply ignores a large proportion of the input elements. The Euclidean distance between two inputs after maxpooling can therefore be at most the same as the distance between the original inputs, in the specific case where all the elements ignored by the maxpooling operation happen to be zero in the inputs. See table 2.1 for a summary of the relevant Lipschitz constant upper bounds.

The neural network architectures discussed here rely on feedforward layers, residual blocks, and pooling layers; therefore the above rules can be used to calculate the upper bound of a Lipschitz constant for a DNN consisting of these components. Crucially, this indicates that controlling the Lipschitz bounds of individual network components is sufficient for controlling the Lipschitz bound for the overall network.

2.2 Out of Distribution Detection

Out of distribution detection (OOD) concerns the identification of inputs that do not come from the training data distribution. While intuitively seemingly straightforward, defining that distribution exactly is considerably less so. Initial discussions around OOD detection in the context of deep learning simply treated all non-training datasets as OOD (Amodei et al., 2016; Hendrycks and Gimpel, 2016); however, Ahmed and Courville (2020) brought up the distinction between covariate distributions and semantic distributions.

2.2.1 Covariate and Semantic Distribution Shifts

Covariate distribution shifts (alternatively referred to as low-level sensory anomalies (Ruff et al., 2021), low-dimensional semantics (Ahmed and Courville, 2020), population-level background statistics (Ren et al., 2019), or spurious OOD (Ming, Yin, and Y. Li, 2022)) refer to covariate shift in the data such that the class categories remain the same, while the distributions of features in the data may deviate from those of the training data. Meanwhile, as Ahmed and Courville (2020) pointed out, and as is now generally accepted in the OOD field (Yang et al., 2021), the focus of OOD detection should be on *semantic* shifts in the data labels. Indeed, Ahmed et al. argue that reliable OOD detection systems should ideally be robust against covariate shifts while simultaneously being sensitive to semantic shifts, this being more in line with the general goal in machine learning of creating models that generalise well. More formally, where an input has some (unknown) label $\hat{y}$ that is not in the training label set Y, it should be identified as OOD. Generally speaking, we would like to identify any shift in the label distribution from that of the training data, i.e., any distribution $p'(Y)$ such that $p(Y) \neq p'(Y)$ (where p and p' are distributions over the labels Y) (Yang et al., 2021).

This distinction is crucial, as early work was quick to point out that machine learning models will tend to model low-level feature distributions over

higher-level semantic ones, often resulting in OOD datasets being identified as in-distribution (ID) (Ming, Yin, and Y. Li, 2022; Kirichenko, Izmailov, and Wilson, 2020). This is perhaps to be expected, as low-level features reflect direct patterns in the input features, whereas semantic labels are assigned by humans a posteriori and may therefore reflect any arbitrary combination of low-level features. This would suggest that a supervised setting may be beneficial, if not necessary, for an effective OOD detector.

2.2.2 Taxonomy of OOD Approaches

The field of OOD detection is closely intertwined with multiple similar, yet technically distinct, areas of interest in machine learning revolving around identifying unusual samples, such as anomaly detection. Yang et al. (2021) propose a conceptual framework they call *Generalized OOD Detection*, that they separate into five separate sub-fields:

- **Anomaly Detection (AD)**: aims to identify samples that deviate from some defined normality. Can cover both covariate and semantic distribution shifts; in-distribution (ID) classification is not a concern and the ID samples are treated as belonging to a single "normal" class. AD is also often treated as a generalisation of novelty detection and outlier detection.

- **Novelty Detection (ND)**: primarily focused on semantic shift, novelty detection aims to identify when a sample comes from a new class not in the ID class(es). ID classification accuracy is not a priority. In contrast to AD, which permits the use of labelled anomalous samples during training, ND is unsupervised. ND assumes that the training data represents normality.

- **Open Set Recognition (OSR)**: aims to identify semantic shift in test samples like ND, but additionally aims to achieve high performance on

the ID classes. Typically treats the open set as an additional "unknown" class alongside the known ID classes; see for example Fang et al. (2021).

- **Out of Distribution detection (OOD)**: also aims to identify semantic shift in test samples, while maintaining high performance on ID data. High performance on ID data is relevant, as the goal is to build a usable system that is simultaneously capable of recognising when an input is OOD. There is a subtle difference to OSR in that where OSR is concerned with identifying "unknown classes" when they appear, OOD is concerned with recognising when an input does not belong to the ID classes. As a result, OSR does not permit the processing of e.g. adversarial examples. OOD approaches only need to reject anything that does not belong to the ID set, regardless of methodology.

- **Outlier Detection (OD)**: typically transductive rather than inductive, unlike the other methods; considers a whole dataset at once and identifies the samples that are too different from the rest according to some distribution measure; that is, outliers are assumed to be present in the dataset. May focus on either covariate or semantic distributions, and ID classification performance is not considered.

These definitions are by no means definitive, as some terms are used interchangeably in the literature or interpreted under a different framework. For example, Ruff et al. (2021) treat novelty and outlier detection as variants of anomaly detection. However, this framework is helpful for separating OOD from other forms of anomaly detection: crucially, we consider OOD detection to refer to models that can identify OOD inputs *in addition to* performing their primary task, ideally without sacrificing performance.

OOD could arguably be seen as a more generalised version of OSR, in the sense that OSR is specifically focused on the task of classifying an OOD sample as belonging to the "open set" class, using only knowledge gained from the ID classes. OOD encompasses the broader set of methods that allow

the recognition of OOD samples as semantically different from the ID data, including OSR, although the distinction in the literature is not necessarily clean-cut. However, in this work we focus on OOD detection rather than OSR. Although the method explored in this thesis focuses on identifying OOD samples using only knowledge gained from the ID training data, our goal is to identify deviations from the ID classes rather than building a model that robustly captures the open set. This makes a subtle but important difference for the direction of the research.

In the context of deep learning, OOD detection was first introduced by Hendrycks and Gimpel (2016), who use the maximum softmax probability (MSP) in the final output layer to distinguish between ID and OOD samples. As noted before, Hendrycks and Gimpel (2016) did not formally make the distinction between covariate and semantic shifts in the data that subsequent approaches delved into. However, their approach is simple to implement and is a popular baseline for comparing more sophisticated OOD approaches (see e.g. Ahmed and Courville (2020)). Yang et al. (2021) group the OOD detection methods developed since then into four main groups:

- **Classification:** classification based approaches focus on adjustments that ensure the classifier's softmax confidence scores more effectively correspond with the familiarity of the input data.

- **Distance:** distance based methods typically focus on using the distances between samples in feature space, with the idea that OOD samples ought to be further away from ID class clusters than ID samples.

- **Reconstruction:** reconstruction-based approaches assume that an encoder-decoder architecture will perform worse at reconstructing OOD samples than ID samples, allowing their identification.

- **Density:** density based approaches attempt to model the distribution of ID data directly, and then take samples that fall into low density regions to be OOD.

Additionally, Bitterwolf, Mueller, and Hein (2023) organise OOD approaches into logit-based and feature-based approaches, i.e. approaches that attempt to identify OOD samples either by looking at the final logit activations, or by looking at feature vectors in the final layers of a deep learning model, respectively. To this framework one could potentially also add a third category of data-based approaches, i.e. methods that attempt to identify OOD samples through direct modeling in the data space - this would include reconstruction based approaches and density-based approaches that attempt to model the training data directly. However, as mentioned above, modelling the data directly proves problematic for semantic OOD detection, as low-level features tend to dominate the learnt models. This thesis focuses on density-based approaches in the feature space, which will be described in more detail.

2.2.3 Evaluation

OOD performance also needs to be measured, and a set of commonly used measurements has developed in the literature to evaluate the performance of OOD methods. Following Hendrycks and Gimpel (2016), OOD detection metrics typically interpret the problem as a binary classification problem where an OOD detection method's output is classified as either ID or OOD. A continuous OOD score - whether the softmax activation value, a distance measure, or a probability density estimate - is provided for the ID testing set and an OOD testing set, which is then interpreted as either ID or OOD by means of a threshold.

Some methods occasionally report the f-score or FPR@TPR-x rate for a given threshold. In the case of the FPR@TPR-x measure, the threshold is selected such that the true positive rate (TPR) is x (typically 0.95), and the false positive rate (FPR) is measured here (see e.g. W. Liu et al. (2020)). However, such measures only capture a snapshot of ID-OOD separation at a particular threshold. Instead, Hendrycks and Gimpel (2016) argued for using threshold-agnostic measures, specifically Area Under the Receiver Operating

	Assigned N	**Assigned P**	
True P	FN	TP	recall = TPR = $\frac{\text{TP}}{\text{TP+FN}}$
True N	TN	FP	FPR = $\frac{\text{FP}}{\text{FP+TN}}$
	NPV = $\frac{\text{TN}}{\text{TN+FN}}$	precision = $\frac{\text{TP}}{\text{TP+FP}}$	

Table 2.2: Summary for how recall/TPR (True Positive Rate), False Positive Rate (FPR), precision, and Negative Predictive Value (NPV) are calculated from the False Negative (FN), True Positive (TP), True Negative (TN), and False Positive (FP) rates.

Curve (AUROC) and Area Under the Precision-Recall curve (AUPR) scores.

To calculate the AUROC score, we calculate the TPR and FPR (see Table 2.2) and then plot the TPR against the FPR for different choices of the threshold, from 0 - 1. The area under the resulting curve is then measured using the trapezoidal rule, giving the AUROC score. The AUROC score may be interpreted as the probability that the classifier will rank a randomly chosen true positive sample higher than a randomly chosen true negative sample (Fawcett, 2006).

However, AUROC is insensitive to large imbalance between the positive and negative classes. By comparing the TPR to the FPR, AUROC scores implicitly assume that positive and negative classes occur at roughly the same rate, since the TPR is only considering the positive class samples and the FPR is considering only the negative class samples. This leads to overly optimistic scores in the case where the positive class is much smaller than the negative class (Davis and Goadrich, 2006). For this reason AUPR is also often measured, which plots precision against recall (TPR). This captures how well the positive class is identified relative to the negative class. This also means that unlike AUROC, AUPR produces a different result depending on whether the ID class or the OOD class are treated as "positive", and papers often report both: AUPR-in for when ID samples are considered positive, and AUPR-out

for when OOD samples are considered positive (Hendrycks and Gimpel, 2016).

Finally, it is also worth noting that Humblot-Renaux, Escalera, and Moeslund (2023) raised some issues with AUROC and AUPR scoring for OOD methods, pointing out that these measures do not capture the amount of separation between ID and OOD data. Instead, they propose calculating the FPR and FNR (FNR = FN/(FN + TP)) rates for different thresholds to obtain AUFPR and AUFNR areas, which are then used to calculate what they call the Area Under the Threshold Curve (AUTC):

$$\text{AUTC} = \frac{\text{AUFPR} + \text{AUFNR}}{2}$$

Unlike AUROC and AUPR, smaller values are better for this metric; the smaller the score, the better is the separation between ID and OOD datasets.

2.2.4 Density Based Approaches

The focus in this work is on density-based approaches to OOD detection. These approaches involve modelling the ID data using some distributional model, and then performing OOD detection by rejecting samples as OOD that fall below some threshold density value. Basic approaches may involve simply fitting Gaussian mixture models (GMMs) to data (e.g., Lee et al. (2018) fits Gaussian distributions to the feature vectors of a trained classifier), but there is significant interest in more complex models, particularly normalising flows (NFs), that in principle ought to be able to capture more complex distributions.

Normalising flows (see Section 2.2.5) establish an invertible mapping from some input density (i.e., the unknown density governing the distribution of the data) to a well understood base density, such as a standard normal distribution. In this way, normalising flows promise to compute explicit densities for input data samples, in contrast to other generative machine learning approaches such as generative adversarial networks, which makes them promising candidates for identifying OOD samples. However, as mentioned in Section 2.2.1, Nalisnick et al. (2018) showed that normalising flows actually often assign higher densities

to OOD datasets than ID test sets, and Kirichenko, Izmailov, and Wilson (2020) argued this is because they model low-level feature distributions rather than high-level semantic distributions. Therefore, the challenge density-based approaches face is one of modelling the semantic distributions and ignoring irrelevant specifics of the covariate distributions.

There have been some approaches that attempt to address this directly; for example, Ren et al. (2019) looked at training a "background model" with data that is sufficiently augmented to destroy semantic information. The density estimate of this background model (in their case, they did not use a normalising flow) is then compared with the density estimate of the main model to try and cancel out the effect of covariate feature distributions on the final density estimate.

A different approach is to instead model the distributions of higher-level features in deep learning architectures, as these are assumed to more closely reflect the semantic distributions in the data. This started with Lee et al. (2018), although their method was adapted from Feinman et al. (2017), who applied a kernel density estimator on the feature layer with the goal of identifying adversarial samples rather than OOD samples. For OOD detection, Van Amersfoort et al. (2020) and J. Liu et al. (2020) achieved good results with their DUQ and SNGP methods, respectively. DUQ drops the final softmax layer of a deep learning model and trains a radial basis function model on the features; while strictly speaking not a density estimation method, their analysis of feature vector properties becomes relevant in later approaches to estimating densities in feature space. SNGP trains a Gaussian process on spectrally normalised features in order to estimate the likelihood of samples, and is commonly used as a benchmark due to its excellent single pass performance. While successful, van Amersfoort also identified a crucial problem with their approach, namely, the tendency for *feature collapse* to arise in the feature space without regularisation. This is discussed in more detail in Section 5.1; Amersfoort et al. (2021) were able to considerably reduce this effect by ensuring bi-Lipschitz continuity,

and Cao and Z. Zhang (2022a) adopted these concepts to deliver considerable improvements in performance.

2.2.5 Normalising Flows

The basic idea behind the normalising flow is to model a complex distribution (such as a set of images) by taking a simple, well understood distribution (such as a standard normal distribution) and finding an invertible (and therefore bijective) mapping from the simple base distribution to the more complex distribution underlying the data. The key idea is that because we can compute the density of a sample in the simple base distribution, we can use the invertible mapping to infer the density of the original data sample as well.

2.2.5.1 General Principles

Let us define $Z \in \mathbb{R}^D$ to be a D-dimensional random variable whose distribution is defined by the known and tractable probability density function p_Z - usually a standard normal density. This is our base distribution. Let us also define the data variable $X \in \mathbb{R}^D$ with the unknown data probability density function p_X (note that X has the same dimensionality as Z).

At its most basic, a normalising flow is a differentiable function $g : \mathbb{R}^D \to \mathbb{R}^D$ such that $g(Z) = X$, that has an invertible (and likewise differentiable) counterpart f such that $f(X) = f(g(Z)) = Z$. Being both invertible and differentiable in both directions, such a transformation is known as a diffeomorphism and ensures a bijective mapping between X and Z. The function g is often referred to as the *generator* function, as it generates samples of X from the base distribution. Meanwhile, $f = g^{-1}$ is commonly referred to as the *normalising* function, since it transforms a sample from the complex data distribution into a sample of the (typically standard normal) base distribution (see Figure 2.4). If we can find a way of constructing an arbitrarily complex g such that it adheres to the condition of invertibility, and is differentiable in both directions, then it should be possible to model any distribution p_X from

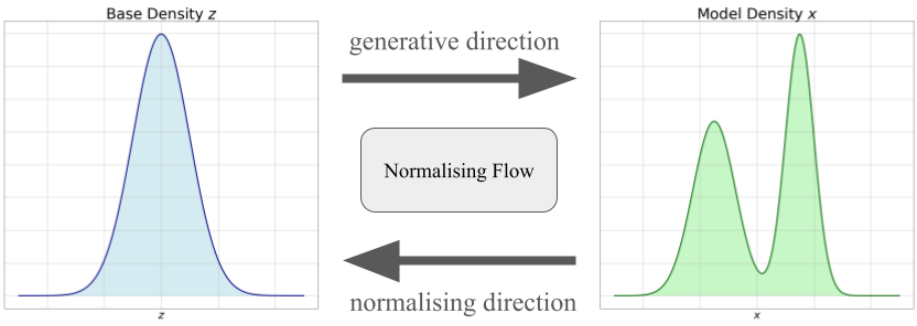


Figure 2.4: A normalising flow allows mapping both from the base density to the model density (the generative direction) and vice-versa (the normalising direction).

the base distribution p_Z.

We also note that bijectivity is preserved in composition; that is, if we have a set of bijective functions $g_1, g_2, ..., g_N$, then the function $g = g_1 \circ g_2 \circ ... \circ g_N$ is also bijective. By the chain rule, differentiability also holds in composition. This indicates that, similarly to how the relatively simple layers in a deep neural network can be stacked to create an arbitrarily complex function, a set of simpler but demonstrably bijective functions can also be stacked to create a much more complex bijective function. This is the principle followed by deep learning approaches to normalising flows.

In the context of deep learning, much initial interest in normalising flows actually focused on the generative direction g; popular examples of these include Nice (Dinh, Krueger, and Y. Bengio, 2014), RealNVP (Dinh, Sohl-Dickstein, and S. Bengio, 2016), and Glow (Durk P Kingma and Dhariwal, 2018). These demonstrated impressive performance in modelling high-dimensional image data, but it is worth noting that while the function learnt by a normalising flow must be invertible, the inverse may not necessarily be efficient to evalu-

ate. While much literature on normalising flows is primarily concerned about the generative direction g, our requirement to efficiently evaluate the density function for OOD means we are focused on the normalising direction f.

2.2.5.2 Technical Details

Transforming the distribution: To transform the base probability density p_Z into p_X, we must preserve its volume to obtain a valid probability density, i.e., for a non-linear transformation, any infinitesimally small δz must preserve its volume when transformed to the corresponding region δx (and vice-versa).

To understand how this can be done, we begin by considering a 1D case: given the region δz with known density $p_Z(z)$ along its range, it has an area - hence probability - of $p_Z(z)\delta z$. We also have a diffeomorphism f such that $f(x) = z$ (as we are interested in the normalising direction), and therefore have $f^{-1}(z) = x$. The region δz is mapped by f^{-1} to the region δx, with the unknown density $p_X(x)$ and area $p_X(x)\delta x$. Both areas must be the same to conserve probability:

$$p_X(x)\delta x = p_Z(z)\delta z, \tag{2.24}$$

therefore,

$$p_X(x) = p_Z(z)\left|\frac{\delta z}{\delta x}\right|. \tag{2.25}$$

Note that we take the absolute of $\frac{\delta z}{\delta x}$ since we are only interested in the rate of change, and not the direction of change. In our problem formulation we assume we only have x and f, so we modify the equation appropriately:

$$p_X(x) = p_Z(z)\left|\frac{\delta f(x)}{\delta x}\right|. \tag{2.26}$$

We now extend this reasoning to the D-dimensional case, where $x, z \in \mathbb{R}^D$. We begin by recognising that the absolute determinant of a D-dimensional matrix is the volume of the D-parallelotope formed by the rows of that matrix (Strang, 2009). Therefore the total probability of the region δz, which covers a D-dimensional hypercube, is now $p_Z(z)|\det(\delta z)|$, and likewise for δx the probability becomes $p_X(x)|\det(\delta x)|$. We can now modify Equation 2.26 to

get:

$$\begin{aligned} p_X(x) &= p_Z(z)\frac{|\det(\delta f(x))|}{|\det(\delta x)|} \\ &= p_Z(z)|\det(\delta f(x))|\left|\det(\frac{1}{\delta x})\right| \\ &= p_Z(z)\left|\det\frac{\delta f(x)}{\delta x}\right| \end{aligned} \tag{2.27}$$

Note that $\frac{\delta f(x)}{\delta x}$ is the Jacobian of f on x, J_f, where f is the normalising function. In other words, we get the probability density of a sample x by multiplying the density of the corresponding z by the rate at which z changes in its transformation to x.

As mentioned before, normalising flows will generally be defined as compositions of invertible functions, e.g., $g_1, g_2, ..., g_M$ with their invertible counterparts $f_M, f_{M-1}, ..., f_1$. We can modify Equation 2.27 to accommodate this chain of functions by taking the product of the Jacobians:

$$p_X(x) = p_Z(f(x)) \cdot \prod_{i=1}^{M}\left|\det\frac{\delta f_i(x_i)}{x_i}\right| \tag{2.28}$$

Here, x_i represents the intermediate value $f_i \circ f_{i-1} \circ ... \circ f_1 \circ x$, such that $x_M = z$. Note that, for simplicity of notation, we can re-write this as:

$$p_X(x) = p_Z(f(x)) \cdot |\det(J_f)| \tag{2.29}$$

The operation is commonly carried out in its log form, where it becomes:

$$\log(p_X(x)) = \log(p_Z(f(x))) + \log(|\det(J_f)|) \tag{2.30}$$

Excellent overviews of the principles and technical details underlying the implementation of normalising flows are provided by Papamakarios et al. (2021) and Simon J.D. Prince (2023).

Fitting the model: Typically, we fit the normalising flow model with the objective of maximising the likelihood of the model. That is, given p_X as the unknown true density of the data, and p_X^* as the estimated density modelled

by $f(X;\theta)^{-1}$, with trainable parameters θ (as we are interested in measuring the density, we implement f rather than g), the training objective we use is to maximise

$$\mathcal{L}(\theta) = \int p_X(x) \log p_X^*(x) dx. \tag{2.31}$$

Note that this is equivalent to the negative of the crossentropy between p_X and p_X^*, i.e., $\mathcal{L}(\theta) = -H(p_X, p_X^*)$. Recall that the crossentropy measures the difference between two distributions; when comparing some density q with a density p, crossentropy is defined as:

$$H(p, q) = -\int p(x) \log q(x) dx, \tag{2.32}$$

In other words, maximising the likelihood of the normalising flow model is equivalent to minimising the difference between the modelled distribution and the unknown true data distribution. This is also equivalent to minimising the Kullback-Leibler (KL) divergence between p_X and p_X^*, as we now show.

Since we have only a finite set of samples $\{x^i\}_{i=1}^N$, we cannot compute the integral in Equation 2.31 directly. Therefore, we approximate it with a Monte Carlo estimate:

$$\mathcal{L}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \log p_X^*(x_i), \tag{2.33}$$

where p_X^*, as defined in Equation 2.28, can be determined purely through the base distribution p_Z and the normalising direction function f. We can see that maximising this function is equivalent to minimising the KL divergence between p_X and p_X^* when considering the definition for the KL divergence:

$$KL(p||q) = -\int p(x) \log \frac{q(x)}{p(x)} dx, \tag{2.34}$$

and for the entropy of a distribution:

$$H(p) = -\int p(x) \log p(x) dx. \tag{2.35}$$

We can take Equation 2.31 to get

$$\begin{aligned}\mathcal{L}(\theta) &= \int p_X(x) \log p_X^*(x) dx \\ &= \int p_X(x) \log p_X^*(x) dx - \int p_X(x) \log p_X(x) dx + \int p_X(x) \log p_X(x) dx \\ &= \int p_X(x) \log \frac{p_X^*(x)}{p_X(x)} dx + \int p_X(x) \log p_X(x) dx \\ &= -KL(p_X, p_X^*) - H(p_X) \\ &= -KL(p_X, p_X^*) - const,\end{aligned} \tag{2.36}$$

where the entropy of the true distribution p_X is a constant and can be ignored. Therefore, maximising the probability of the samples is equivalent to minimising the KL divergence between the modelled distribution and the ground truth distribution. As a result, the maximum likelihood estimation objective is also known as the forward KL divergence.

2.2.5.3 Coupling Flows

So far, we have assumed the diffeomorphism f is a given; however, it still needs to be implemented. Since f should be able to represent arbitrarily complex transformations, it follows that we should use a deep learning architecture to represent it. However, ensuring the invertibility of a deep neural network is not straightforward, particularly when working with high-dimensional data such as images, and a number of different approaches have arisen over recent years to achieve this.

One of the earliest approaches were the coupling layers introduced by Dinh, Krueger, and Y. Bengio (2014) with their NICE architecture. Their approach was to take a D-dimensional input x and split it into two parts at dimension d, computing the two corresponding parts of the output y as follows:

$$\begin{aligned} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= \tau(x_{d+1:D}; \phi(x_{1:d})) \end{aligned} \tag{2.37}$$

Where τ is some invertible elementwise function parameterised by $x_{1:d}$, and ϕ is some arbitrary differentiable function. For example, Dinh, Krueger, and

Y. Bengio (2014) used $\tau = x_{d+1:D} + m(x_{1:d})$ where m is a multi layer perceptron, and Dinh, Sohl-Dickstein, and S. Bengio (2016) used $\tau = x_{d+1:D} \odot \exp(s(x_{1:d}) + t(x_{1:d}))$ where s and t are deep neural networks that compute scaling (i.e., multiplication) and translation (i.e., addition) parameters, and $\odot$ is the Hadamard (element-wise) product.

The benefit of this approach is that while τ must be invertible, ϕ does not have this requirement, since its inverse does not need to be computed to invert the coupling flow:

$$\begin{aligned} x_{1:d} &= y_{1:d} \\ x_{d+1:D} &= \tau^{-1}(y_{d+1:D}; \phi(x_{1:d})) \end{aligned} \tag{2.38}$$

This approach is easy to invert (in fact the inverse is just as fast to compute as the forward direction) and the determinant of the Jacobian is easy to compute: for the transformation $(x_{1:d}, x_{d+1:D}) \to (y_{1:d}, y_{d+1:D})$ we get the Jacobian (Papamakarios et al., 2021)

$$\begin{bmatrix} \mathbf{I} & 0 \\ \mathbf{A} & \mathbf{D} \end{bmatrix}, \tag{2.39}$$

where $\mathbf{D}$ is a $(D-d) \times (D-d)$ diagonal matrix representing the gradient of τ with respect to $x_{d+1:D}$. $\mathbf{I}$ is the $d \times d$ diagonal identity matrix (since $x_{1:d}$ is left unchanged) and $\mathbf{A}$ is the $(D-d) \times d$ full matrix corresponding to ϕ. If τ is an elementwise function, then $\mathbf{D}$ will be a diagonal matrix, and the determinant of the coupling block's Jacobian becomes the product of its diagonal elements, i.e., the non-zero elements of $\mathbf{D}$.

2.2.5.4 Autoregressive Flows

While coupling layers demonstrated some early success in generating high-dimensional data such as images, they have the limitation that a single coupling layer cannot represent an arbitrary transformation. As one component of the input is left unchanged and the other is transformed by a single function, not all dimensions can be transformed individually in a single coupling layer. This restricts the set of possible transformations, and therefore the universality, of

the transformation. In principle, a composition of D coupling layers could be a universal approximator provided the indices of the input vector are rotated by 1 after each flow (Papamakarios et al., 2021), but this may be a prohibitively deep model in practice.

This limitation led to a more generalised set of approaches known as *autoregressive flows*, first introduced by Germain et al. (2015), that take the basic principle behind the coupling flow to its logical extreme: instead of dividing the D-dimensional input into two components, we divide it into D components such that the dth element of y, y_d, is dependent upon all of $x_1, \ldots, x_d$, i.e.,

$$y_d = \tau(x_d; \phi(x_{1:d-1})) \tag{2.40}$$

for every d. As with the coupling flow, ϕ can be any arbitrary non-invertible function. Theoretically, a unique ϕ could be defined for every component, although that would be prohibitively expensive. Instead, Germain et al. (2015) used a single neural network where, given $x_{1:d-1}$, the weight matrix parameters corresponding to $x_{d:D}$ are masked out. Additionally, the output of the conditioner has to be defined for $d = 1$ where there are no $x_{i<d}$ to condition on; Germain et al. (2015) simply mask ϕ completely resulting in an output of 0.

The inverse, similarly to that of the coupling flow, can then be computed as:

$$x_d = \tau^{-1}(y_d; \phi(x_{1:d-1})) \tag{2.41}$$

So, for example, given an input x with three elements, the output y would be computed as

$$\begin{aligned} y_1 &= \tau(x_1; \phi) \\ y_2 &= \tau(x_2; \phi(x_1)) \\ y_3 &= \tau(x_3; \phi([x_1, x_2])) \end{aligned} \tag{2.42}$$

and the inverse would be computed as

$$\begin{aligned} x_1 &= \tau^{-1}(y_1, \phi) \\ x_2 &= \tau^{-1}(y_2, \phi(x_1)) \\ x_3 &= \tau^{-1}(y_3, \phi([x_1, x_2])) \end{aligned} \tag{2.43}$$

A single autoregressive flow, unlike coupling flows, is a universal approximator given sufficiently flexible choices for τ and ϕ (Kobyzev, Simon JD Prince, and Brubaker, 2020). The more flexible mechanism comes at a cost though: while the coupling layer is just as fast to compute in the forward and inverse directions, the autoregressive flow is asymmetric in computational efficiency. In one direction, any y_d can be computed in any order independently of the other $y_{e \neq d}$ (meaning $y_{1:D}$ can be computed in parallel), but in the inverse direction x_d depends on $x_{1:d-1}$, and must therefore be computed first. This means that one direction of the flow can be calculated quite efficiently, but the other direction is necessarily D times slower to compute. Therefore, the intended use (sample generation or density estimation) must be taken into account when designing an autoregressive flow.

Note that the determinant of the Jacobian of an autoregressive flow is efficient to compute: since each y_d depends only on $x_{1:d}$ (i.e., $\frac{\delta y_i}{\delta x_j} = 0$ for $i < j$), the Jacobian is a lower triangular matrix, and the determinant is simply the product of the diagonals (Durk P Kingma, Salimans, et al., 2016).

2.2.5.5 Residual Flows

Residual flows were initially introduced by Behrmann et al. (2019) and then refined by R. T. Chen et al. (2019). The basic principle is to take advantage of the residual block architecture, which, as described in Section 2.1.4.4, takes the form

$$x' = x + f(x) \tag{2.44}$$

The inverse of this operation is:

$$x = x' - f(x) \tag{2.45}$$

An analytical solution to this inverse does not exist, but it can be approximated if f is a contraction mapping; that is, if the Lipschitz constant L for f is strictly less than 1 (see Section 2.1.5.2), which can be enforced via spectral normalisation (Section 2.1.5.1), approximation is possible.

This is a consequence of the Banach fixed point theorem (Agarwal, Jleli, and Samet, 2018), which states that for a contractive mapping (i.e. a function f with $L < 1$) there is exactly one fixed point z such that $z = f(z)$; since the function is contractive, f will converge to z if it is repeatedly given its own output, after being given some arbitrary initial input z_0. That is, if $z_k = f(z_{k-1})$ for $k > 0$,

$$\lim_{k \to \infty} z_k = z \tag{2.46}$$

Therefore, when inverting the residual layer to find $x = x' - f(x)$, x can be found iteratively by making an initial guess x_0 and then repeatedly computing $x_k = x' - f(x_{k-1})$ for a number of iterations. From the Banach fixed point theorem, we can see that after k iterations,

$$||x - x_k||_2 \leq \frac{L^k}{1 - L} ||x_1 - x_0||_2 \tag{2.47}$$

This implies convergence is exponential in the number of iterations, meaning a relatively small number of iterations will yield a good approximation. Behrmann et al. (2019) also recommend using x' as the initial guess for x_0, since it was derived by adding to the original x and should therefore already be quite close.

This approach is less constrained than other approaches to invertibility, which through their structural constraints end up with diagonal or lower triangular Jacobians and as a result are less expressive. Residual flows have been found to be better at modelling distributions, often achieving similar or better results than coupling-based approaches such as Glow (Durk P Kingma and Dhariwal, 2018), while requiring fewer parameters and being less memory intensive to train (R. T. Chen et al., 2019).

This comes at the cost of the determinant being much more difficult to

compute. Behrmann et al. (2019) introduced a method of approximating the log of the determinant, which begins with the observation that the determinant of the Jacobian of the residual block $g(x) = x + f(x)$, $\det(J_g)$, must be positive given that $L_f < 1$, and therefore:

$$|\det(J_g)| = \det(J_g) \tag{2.48}$$

From this, we can use the identity from linear algebra that $\log(\det(A)) = \text{trace}(\log(A))$ and the observation that $g(x) = (\mathbf{I} + f)(x)$ to reformulate Equation 2.28 in terms of log operations:

$$\log(p_X(x)) = \log(p_Z(f(x))) + \sum_{i=1}^{M} \text{trace}(\log(\mathbf{I} + J_f^i)) \tag{2.49}$$

The trace of the matrix logarithm can be computed via a power series, which converges since $L_f < 1$ (Hall, 2013):

$$\begin{aligned} \text{trace}(\log(\mathbf{I} + J_f)) &= \text{trace}\left(\sum_{k=1}^{\infty} (-1)^{k+1} \frac{(J_f)^k}{k}\right) \\ &= \sum_{k=1}^{\infty} (-1)^{k+1} \frac{\text{trace}((J_f)^k)}{k} \end{aligned} \tag{2.50}$$

This leaves us with two problems: first, the trace of the Jacobian must still be calculated, which is inefficient to do exactly in high dimensional space ($\mathcal{O}(D^3)$ for a $D \times D$ matrix). Second, this is an infinite series, so it cannot be computed as is.

The first issue is resolved through the use of the Hutchinsons trace estimator (Hutchinson, 1989), which provides a stochastic estimate for the trace of a matrix:

$$\text{trace}(J) \approx \mathbb{E}_{p(v)}[v^\intercal J v] \tag{2.51}$$

Where $v \in \mathbb{R}^D$ is a vector whose elements are sampled independently from a distribution with $\mathbb{E}_v = 0$ and $\text{cov}(v) = 1$, i.e., typically $v \sim \mathcal{N}(0, 1)$.

The second issue, the infinite series, was initially addressed by Behrmann et al. (2019) by simply setting some truncation threshold n and computing only the first n iterations. However, R. T. Chen et al. (2019) showed that this introduced considerable bias: since the rate of convergence drops with

increasing D and increasing L_f, this method struggles to scale with higher dimensional data. Instead, they introduced an unbiased Russian roulette estimator, which after each iteration will compute the next iteration with some probability q (a hyperparameter) or stop. R. T. Chen et al. (2019) show that, if we call the kth term of the infinite series Δ_k, we can get an estimate for the sum of the series from (R. T. Chen et al., 2019):

$$\sum_{k=1}^{\infty} \Delta_k = \mathbb{E}_{n\sim p(N)} \left[\sum_{k=1}^{n} \frac{\Delta_k}{\mathbb{P}(N \geq k)} \right], \tag{2.52}$$

where the denominator $\mathbb{P}(N \geq k)$ indicates the probability of a randomly chosen stopping point N being greater than or equal to k, and is used to weigh the kth sample by its probability of being included in the series, thus compensating for the bias introduced by the random truncation of the series. R. T. Chen et al. (2019) used $p(N) = Geom(0.5)$ and found no issues with this choice, although in principle p could be any distribution under the condition that it provides support for all indices. This results in an unbiased stochastic estimator for the log determinant.

Combining this estimator with the Hutchinsons trace estimator, we can finally formulate the log density estimate for the whole residual block as follows:

$$\log p_X(x) = \log p_Z(f(x)) + \mathbb{E}_{n,v} \left[\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} \frac{v^\intercal [(J_f(x))^k] v}{\mathbb{P}(N \geq k)} \right] \tag{2.53}$$

Where $n \sim p(N)$ and $v \sim \mathcal{N}$. As the residual flow is composed of residual blocks, the log density estimate for the entire model with L residual blocks becomes, following Equation 2.28,

$$\log p_X(x) = \log p_Z(f(x)) + \sum_{l=1}^{L} \left\{ \mathbb{E}_{n,v} \left[\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} \frac{v^\intercal [(J_{fl}(x_{l-1}))^k] v}{\mathbb{P}(N \geq k)} \right] \right\} \tag{2.54}$$

where residual block l, with Jacobian J_{fl}, outputs the intermediate value x_l (i.e., $x = x_0$).

The end result is a flow that has no architectural constraints on its Jacobian and can model the relations between all variables in the input, producing a powerful model for learning complex distributions. Hence, this thesis focuses almost exclusively on the residual flow as the model of choice for normalising flows.

LipSwish Activation R. T. Chen et al. (2019) also introduced the LipSwish activation function, an evolution of the Swish activation function introduced by Ramachandran, Zoph, and Le (2017), who defined the Swish activation function as

$$\text{Swish}(x) = x \cdot \text{sigmoid}(\beta x), \tag{2.55}$$

where β can be either a constant or treated as a trainable parameter, and was intended to improve upon the standard ReLU function. LipSwish, in turn, was motivated by two requirements for the residual network: first, the activation function had to be contractive to preserve the contractive property of the overall network (i.e., it must have a Lipschitz constant ≤ 1). Second, it should have a second derivative that does not approach zero when the first derivative is close to one, as this can lead to vanishing gradient issues. Chen et al. observe that there generally tends to be a tradeoff between the Lipschitz constant of an activation function and non-vanishing gradients, making the enforcement of the Lipschitz condition potentially problematic.

However, they noted that the Swish function meets both of these conditions when it is scaled by $\frac{1}{1.1}$, creating the LipSwish activation function:

$$\text{LipSwish}(x) = \frac{\text{Swish}(x)}{1.1} = \frac{x \cdot \text{sigmoid}(\beta x)}{1.1}, \tag{2.56}$$

where the Lipschitz constant of this function $L_{\text{LipSwish}} < 1$ for all values of β. Chen et al. additionally enforce β to be strictly positive by passing it through the softplus function ($\text{softplus}(x) = \log(1 + \exp(x))$). The LipSwish activation function remains a standard activation function for residual flows.

Chapter 3

Methods

3.1 Overview

In this chapter we describe the tools used for the experiments in this thesis. In particular, we describe implementation details for the wide residual network architecture (Zagoruyko and Komodakis, 2016), the residual flow (R. T. Chen et al., 2019), and the deep hybrid model (Cao and Z. Zhang, 2022a). We make our implementation of the DHM publicly available[1]. We also describe training methods for each of these architectures. Additionally, we describe the datasets we use for our experiments, detailing their preparation and discussing their properties.

3.2 Deep Hybrid Models: a High-Level View

Deep hybrid models (DHMs) are a class of models for OOD classification problems that attempt to model both the data and the labels as a joint density; in other words, they combine discriminative and generative elements in a single end-to-end model. More formally, they are generally thought of as consisting of a model for $p(y|x)$ and one for $p(x)$, although these may share many if not most parameters. Put together, they model the joint probability $p(y, x) = p(y|x)p(x)$. While the concept of building hybrid models has been around for a

[1] https://github.com/P-Schlumbom/deep-hybrid-models

while (see, e.g., Raina et al. (2003)), originally with the goal of enabling semi-supervised learning, it was Kuleshov and Ermon (2017) who introduced "deep" hybrid models with an architecture that utilised a deep neural network for the classifier component and a variational autoencoder for generative component.

Nalisnick et al. (2019) first applied the DHM concept to OOD detection, with an architecture that trained a classifier on the latent variables of a normalising flow model. They noted that one of the benefits of their model was that the $p(y|x)$ and $p(x)$ components shared almost all of their parameters. H. Zhang et al. (2020) then built a DHM model that inverted this approach; an encoder (i.e., a deep neural network) $\phi : x \mapsto h$ was used to generate features h, which were then passed on to a classifier head $C : h \mapsto y$ for classification, and a flow-based model $F : h \mapsto z$ for density estimation. This model was built to perform open set recognition (OSR) rather than OOD detection, but their motivation was to address the same problem outlined in Section 2.2.1, namely, that density estimation methods (particularly flows) tend to be sensitive to low-level background statistics. Modelling the density of the features instead was meant to avoid this issue. They report superior OSR performance on natural images compared to several other state-of-the-art methods at the time.

Finally, Cao and Z. Zhang (2022a) adopted the architecture described by H. Zhang et al. (2020) for OOD detection, although they made a further adjustment to the ϕ encoder model: they ensure it is roughly volume-preserving via spectral normalisation, arguing that this forces the encoder to be roughly measure-preserving by preserving the volume of the transformation of x to h. This would ensure that the feature space maintains a representation that captures the densities in input space. This adopts the same concept as normalising flows use (see Section 2.2.5), where the change of variables formula can be used to keep track of the change in volume of a transformation via the determinant of its Jacobian.

However, as was covered in the normalising flows section, computing the

determinant is a considerable challenge if the target matrix is not in a triangular or diagonal form. Cao and Z. Zhang (2022a)'s proposed solution is to ensure the change in volume is roughly 1, in which case Equation 2.27 simplifies for the transformation from x to h to: $p(x) \approx p(h)$. Thus the densities of h should still correspond to those of the data, while being transformed into a set of features which still allow the classifier to discriminate the ID classes.

This can be achieved by controlling the upper bound of the Lipschitz constant of the encoder network, provided this network is a residual network. As we saw in Section 2.1.5.2, given a residual block $g(x) = x + f(x)$, controlling the upper bound of the Lipschitz constant L_g of the residual connection $f(x)$ also places a lower bound on the Lipschitz constant L_f of the entire residual block. Therefore, minimising the upper bound of L_f will constrain L_g to be close to 1, which in turn will ensure the measure-preserving property of the encoder architecture. As detailed in Section 2.1.5.1, this can be enforced by means of spectral normalisation. Cao and Z. Zhang (2022a) note however that, in practice, L_f should not be too tightly restricted lest the residual network learn a trivial mapping, such as for example $f(x) = 0$, which would turn every residual block into an identity operator. The spectral normalisation coefficient c (see Equation 2.15) should therefore be adjusted empirically for the best results.

Remarkably, Cao and Z. Zhang (2022a) report 100% performance on a number of metrics with their DHM approach. They train their model on both the CIFAR-10 and CIFAR-100 datasets, and in each case measure AUROC and AUPR scores for OOD detection against CIFAR-100/CIFAR-10 and SVHN. They claim a 100% score in every case. This seems especially astonishing considering that Fort, Ren, and Lakshminarayanan (2021) showed erroneous samples in the CIFAR-10 and CIFAR-100 test sets - through labelling error and semantic ambiguity - which calls into question whether a 100% AUROC score is possible at all for the CIFAR-10/CIFAR-100 OOD test. Unfortunately, Cao and Z. Zhang (2022a) have not published their code.

Group name	Parameters
bn1 + relu	-
conv1	$[3 \times 3, 16(2^{i-1})k]$
bn2 + relu	-
dropout	dropout rate = 0.3
conv2	$[3 \times 3, 16(2^{i-1})k]$

Table 3.1: Basic structure of the residual block in group i (starting at $i = 1$), which is repeated N times per group.

Nevertheless, the reasoning behind their DHM architecture seems sound, and it would be just as interesting to confirm that it works as it would be to understand why it does not. This thesis investigates the limits of this approach for out of distribution detection.

3.3 Wide Residual Network

3.3.1 Architecture Description

The classifier used by Cao and Z. Zhang (2022a) for their image processing DHM is the wide resnet (WRN) architecture introduced by Zagoruyko and Komodakis (2016). While the original resnet architecture was intended to enable deeper network structures, Zagoruyko and Komodakis (2016) showed that a shallower, wider resnet architecture (i.e., with more channels processed per layer) could match the performance of much deeper resnets with only a fraction of the parameters. It has since been a popular architecture for computer vision applications.

The WRN defines a specific format for the residual block (see Table 3.1), that consists of two 3×3 convolutional layers, each preceded by batch normalisation and a ReLU activation (Section 2.1.4.4 explains why the ReLU activation precedes the convolutional operation). Zagoruyko and Komodakis

Group Name	**Output Size**	**Parameters**
input	32×32	-
conv1	32×32	$[3 \times 3, 16]$
block1	32×32	$\begin{bmatrix} 3 \times 3, 16k \\ 3 \times 3, 16k \end{bmatrix} \times N$
block2	16×16	$\begin{bmatrix} 3 \times 3, 32k \\ 3 \times 3, 32k \end{bmatrix} \times N$
block3	8×8	$\begin{bmatrix} 3 \times 3, 64k \\ 3 \times 3, 64k \end{bmatrix} \times N$
bn1 + relu	8×8	-
avgpool1	1×1	-
fc1	C	$64k \times C$

Table 3.2: Wide residual network overall architecture. The residual block groups are defined separately, see Table 3.1. k is the number of channels in the residual blocks, where a basic residual network essentially has $k = 1$, N is the number of residual blocks per group, C is the number of classes.

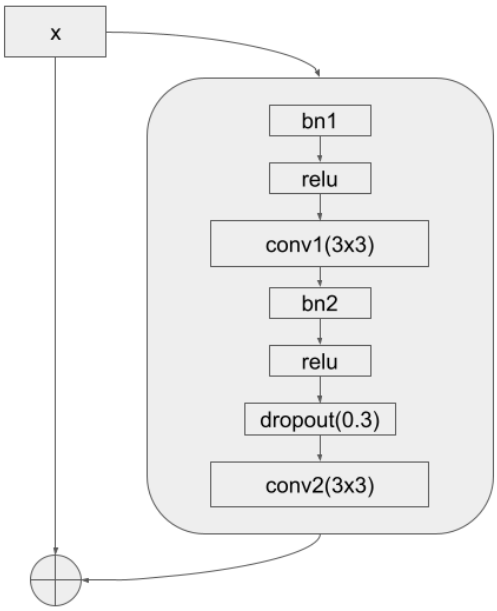


Figure 3.1: The basic WRN residual block architecture.

(2016) also introduce a dropout layer before the final convolutional operation. Dropout is a regularisation mechanism that zeroes out a random subset of activations during training, which is known to encourage deep learning architectures to learn general solutions that are more robust against overfitting (Srivastava et al., 2014). Zagoruyko and Komodakis (2016) empirically find a dropout rate of 0.3 for the CIFAR datasets, and this value is employed in this thesis by default.

The network as a whole has three "groups" of these residual blocks, where each group contains N residual blocks and the residual blocks for group i (where i starts counting at 1) have $16 \times 2^{i-1} \times k$ channels; i.e., the first group has $16k$ channels, the second group has $32k$ channels, the third $64k$ channels and so on (see Table 3.2). The parameter k sets the width of the WRN, where $k = 1$ defines a standard resnet architecture and any $k > 1$ is considered a WRN. Zagoruyko and Komodakis (2016) use a notation to describe a particular WRN architecture in terms of the number of convolutional layers in the network and k. As there are three groups, each with N residual blocks with 2 convolutional layers each, the three groups have $6N$ convolutional layers be-

tween them. Additionally, the first residual block of each group requires an extra 1×1 convolutional layer to allow for the expansion of channels by the group, producing an additional 3 convolutional layers. Finally, there is also the initial convolutional layer at the start of the network. In total, a WRN architecture will have $6N + 4$ convolutional layers and is denoted as WRN-$[6N + 4]$-k. For example, the standard WRN architecture used by Zagoruyko and Komodakis (2016), and commonly also used by other works (particularly e.g. Van Amersfoort et al. (2020) and Cao and Z. Zhang (2022a)), is WRN-28-10, set using $N = 4$ and $k = 10$. This architecture has 36.5 million trainable parameters and can achieve a 4% error rate on the CIFAR-10 test set (see Section 3.7.1). WRN-28-10 is used throughout this thesis as the classifier architecture by default unless specified otherwise.

3.3.2 Implementation and Training

Although the original WRN paper provides an open-source repository implementing the WRN architecture[2], they use a unique functional approach that is difficult to reconcile with traditional object-oriented approaches to defining deep learning models using pytorch. Therefore, the WRN was re-implemented in an object-oriented format to ease development, and it was confirmed to attain the same performance as the official implementation on the CIFAR-10 dataset (see Figure 3.2).

For CIFAR datasets, preprocessing consists of reflection padding the 32×32 images by 4 pixels on each side and taking random 32×32 crops; random horizontal flips are applied at a rate of 0.5. The images are also normalised on a per-channel basis with means of $\frac{125.3}{255}, \frac{123.0}{255}, \frac{113.9}{255}$ and standard deviations of $\frac{63.0}{255}, \frac{62.1}{255}, \frac{66.7}{255}$.

Training WRN-28-10 to attain the reported performance (i.e. 4% error on the CIFAR-10 test set) is done via stochastic gradient descent (SGD) with Nesterov momentum with a momentum of 0.9, and an initial learning rate of

[2]https://github.com/szagoruyko/wide-residual-networks/tree/master

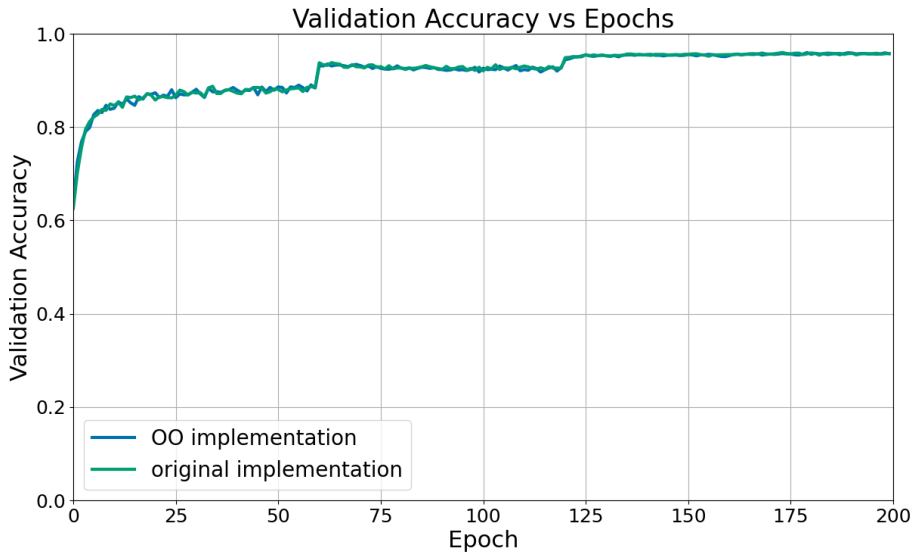


Figure 3.2: Validation accuracies of the original WRN implementation compared to the Object-Oriented (OO) re-implementation.

0.1. Weight decay of 0.0005 is used, and the learning rate is scaled by 0.2 after 60, 120, and 160 epochs. The model is trained for 200 epochs total using a batch size of 128, and using a crossentropy loss function. Variants of this approach to training yield similarly good results. Training with Adam optimisation (Diederik P Kingma and Ba, 2014), though a popular and effective optimisation algorithm, does not yield the same performance for this model.

3.4 Residual Flow

Cao and Z. Zhang (2022a) describe using a residual flow architecture used by H. Zhang et al. (2020), while adapting the official code for the residual flow (R. T. Chen et al., 2019))[3].

The general structure of the residual flow as implemented by Chen et al. consists of stacks of residual blocks, where in each residual block the residual function consists repeating pairs of a LipSwish activation (see Section 2.2.5.5) followed by a InducedNormLinear layer. The "InducedNormLinear" layer sim-

[3]https://github.com/rtqichen/residual-flows

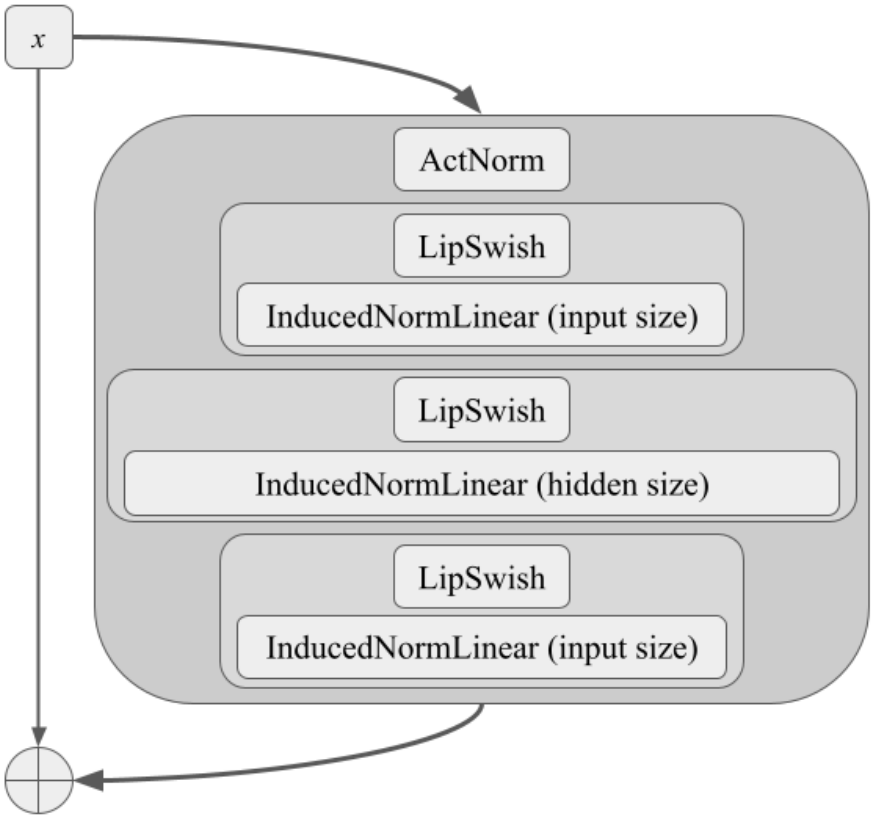


Figure 3.3: Example residual flow residual block architecture with three pairs of LipSwish/InducedNormLinear layers.

ply refers to a fully connected layer that has been modified to implement spectral normalisation. Where the first and final InducedNormLinear layers must necessarily have D neurons for a flow mapping D-dimensional data, the InducedNormLinear layers sandwiched between them may have some user defined dimension - often referred to as the "hidden dimension"(H. Zhang et al., 2020) or the intermediate dimension. See Figure 3.3 for an example residual connection with 3 such pairs.

Finally, each residual block is commonly also preceded and succeded by an *activation normalisation* or actnorm layer. Actnorm layers were introduced by Durk P Kingma and Dhariwal (2018) with their Glow architecture, and were intended to address the dependence of typical batch normalisation layers on the size of the batch. Since batch normalisation computes statistics over each minibatch during training, it necessarily loses precision for very small batches and fails completely for batch sizes of 1. However, for memory intensive tasks (such as generating high resolution images, as Glow was intended to do), this requirement for larger batch sizes can become practically prohibitive. Actnorm instead computes scaling and offset parameters - just as with batch normalisation - for an initial data batch to normalise it to zero mean and unit variance, but afterwards these scaling parameters are trained normally alongside the network's other parameters, with no further dependence on batch statistics.

Zhang et al. describe using an architecture with 10 residual blocks, where in each block the residual function consists of three LipSwish-InducedNormLinear pairs. In the first 6 blocks, a hidden dimension of 256 is used, and a hidden dimension of 128 is used in the final 4 blocks. Cao and Z. Zhang (2022a) describe, in their supplementary materials (Cao and Z. Zhang, 2022b), using the same architecture but using a hidden dimension of 640 instead of 256. It is unclear whether this substitution also applies to the final 4 layers.

The LipSwish activation layer parameter β is set by $\beta := \text{softplus}(a)$ (see Section 2.2.5.5), where a is a trainable parameter and initialised with $a := 0.5$.

Zhang et al. also describe passing input data through a logit transform before passing it on to the normalising flow. Logit transforms were introduced by Dinh, Sohl-Dickstein, and S. Bengio (2016) for their RealNVP architecture, where they use it to preprocess images. The logit transform is defined as:

$$y = \text{logit}(\alpha + (1 - 2\alpha)\frac{x}{255}), \tag{3.1}$$

bearing in mind that the logit function is defined as $\text{logit}(p) = \log(p) - \log(1 - p)$, and α is some small positive value that simply prevents the value $p = \alpha + (1 - 2\alpha)\frac{x}{255}$ from being exactly zero or exactly one, as either case would require computing $\log(0)$. The input variable x in this case is assumed to be an image with pixel values between 0 and 255, so here it is simply rescaled to a 0-1 range. The purpose of the logit transformation is to map values from a $(0, 1)$ range to a $(-\infty, \infty)$ range. In the case of modeling images, as was the aim with the first deep learning normalising flows, this is beneficial as it represents the bounded distribution of pixel values as an unbounded distribution, i.e., it prevents the normalising flow from predicting impossible pixel values in the generative direction.

However, in this case the normalising flow is receiving the (theoretically unbounded) features of an encoder model rather than the image itself, so the transformation is not appropriate. Nevertheless, Zhang et al. appear to describe using it when passing the features to the normalising flow. Cao and Z. Zhang (2022a) only describe using the same normalising flow architecture as Zhang et al., but do not explicitly mention whether or not they apply the logit transformation, thus making it unclear whether or not it was used. However, as is discussed later, this makes a crucial difference as a normalising flow with an initial logit transformation cannot handle the unbounded feature vectors without additional processing steps.

3.4.1 Normflows Implementation

Stimper et al. (2023) released a pytorch-based python library called "normflows" for implementing a large variety of normalising flows[4], including residual flows. Some experiments are carried out with "normflows" based residual flows (see Section 4.1.2), and variants of the DHM using a "normflows" based model were briefly explored. Residual flow networks are implemented following the structure above as closely as possible: 10 residual blocks with each residual connection consisting of three pairs of the "normflows" equivalent of the Lipschitz-constrained linear function and the LipSwish activation function. The "normflows" library also provides alternative options for the base distribution, such as a Gaussian mixture model, and it offers the option to train the parameters of the base distribution as well. In contrast, the standard residual flow structure predominantly used in this thesis uses a D-dimensional standard normal distribution as the base distribution, with no changes made to its parameters.

Since the idea behind the normalising flow is to transform some simple distribution to the target distribution, and since in theory *any* distribution should be valid as the base distribution, this functionality is mostly ignored.

3.5 Spectral Normalisation

Multiple implementations exist for spectral normalisation, although as an approach applied to deep learning relatively recently, there does not at the time of writing seem to be a standardised solution. Pytorch provides an implementation of spectral normalisation[5], but it appears to be based on the implementation by Miyato et al. (2018) that, as Gouk et al. (2021) noted, does not provide a strictly correct implementation for convolutional layers (see Section 2.1.5.1).

[4] https://github.com/VincentStimper/normalizing-flows

[5] https://pytorch.org/docs/stable/generated/torch.nn.utils.spectral_norm.html

Instead, throughout this work, the implementation developed by Amersfoort et al. (2021) for their DUE architecture[6] is used. Their implementation offers a clean, up to date implementation and is readily usable for fully connected, convolutional, and batch normalisation layers, making it perfectly suitable for the spectrally normalised WRN implementation used in this thesis.

The spectral normalisation components are parameterised by the number of power iterations (see Section 2.1.5.1), and the upper bound coefficient c (as described in Section 2.1.5.1, Equation 2.15). J. Liu et al. (2020) describe finding a single power iteration to be sufficient for estimating the spectral norm during training, and Cao and Z. Zhang (2022a) follow this as well (Cao and Z. Zhang, 2022a; Cao and Z. Zhang, 2022b). For c, Cao and Z. Zhang (2022a) report using an upper bound $c = 6$. They describe using a grid search to find this value, following J. Liu et al. (2020), who note that for convolutional layers the spectral norms will tend to be overestimated. A looser upper bound c tends to yield better results in practice, and so it is recommended to find the smallest value of c that does not impact performance.

3.6 Deep Hybrid Model

3.6.1 Structure

The WRN architecture and the residual flow architecture are assembled to recreate the DHM architecture as described by Cao and Z. Zhang (2022a). The WRN architecture is modified so that the final feature layer is also provided as an output, which is then directed to a residual flow architecture. The combined structure was implemented as a single pytorch class that allows both the WRN and the residual flow to be flexibly parameterised as needed. However, this also introduced another uncertainty, namely, the nature of the features to be passed on to the normalising flow. An inspection of the WRN architecture shows that there are several seemingly valid options for where the features

[6] https://github.com/y0ast/DUE/tree/main

should be extracted to be passed on to the normalising flow.

Recall that the WRN, after processing the input image with the three main residual groups, then passes the resulting convolutional image on to a batch normalisation layer, followed by a ReLU activation function, followed by an average pooling layer that effectively condenses the convolutional image into a 1×1 image with $64k$ channels. In the case of the WRN-28-10 architecture, the final convolutional image is an $8 \times 8 \times 640$ tensor, that is then downsampled to a $1 \times 1 \times 640$ tensor - which is simply treated as a 640-dimensional vector. One could reasonably argue that the immediate output of the residual blocks counts as the relevant features to model, since these are the direct outputs of the main feature-extracting body of the network, and since they still contain positional information about the features in the original image. Perhaps the batch normalisation and ReLU activation operations should also be included, since they are part of the feature processing pipeline - on the other hand, they might perhaps erase information that isn't relevant to the classification task, but might still aid in modelling the density. Alternatively, perhaps only the final output after average pooling should be used, since this after all represents the final feature that the classification is actually based on. It also holds the advantage of being of much lower dimensionality, considerably reducing the memory requirements of the DHM. Additionally, the fact that average pooling drops positional information about the features might be considered a boon, as we are interested in modelling the semantic (class-relevant) features rather than spurious background statistics.

The DHM paper does not specify where the features should be taken from, although an adjacent approach by Van Amersfoort et al. (2020) also describes utilising the features of a WRN-28-10 model, which in their case meant the flattened vector after average pooling. The flattened feature vector is seen as the preferred option in this thesis due to its computational efficiency and positional insensitivity, but the other approaches are explored as well.

3.6.2 Training

In their supplementary materials (Cao and Z. Zhang, 2022b), Cao and Z. Zhang (2022a) describe training the classifier portion of the DHM with a very similar method to that used for the WRN architecture: the parameters are trained with SGD, with Nesterov momentum with a momentum of 0.9, an initial learning rate of 0.05, and a weight decay rate of 0.0005. The learning rate is also scaled by 0.2 at 60, 120, and 160 epochs, and they train the model for 200 epochs with a batch size of 64. The same data augmentation as described for the WRN (see Section 3.3.2) is applied to the images.

Additionally, in this thesis, uniform noise is added to the training images, adopting a technique commonly used for training normalising flows (Uria, Murray, and Larochelle, 2013): since raw image data has a finite set of pixel values corresponding to the values 0-255, a density model could in principle assign arbitrarily high density to those exact values and zero to everything else. To resolve this issue, Uria, Murray, and Larochelle (2013) described adding values from a uniform distribution in the range [0, 1] to pixel values before normalisation during training, i.e., each D-dimensional sample $x \in \mathbb{R}^D$ is set to

$$x := x + u \tag{3.2}$$

where $u \sim U(0, 1)^D$. The technique is now regularly used (see, for example, Dinh, Krueger, and Y. Bengio (2014)). In this case the standard CIFAR-10/CIFAR-100 and SVHN datasets provided by pytorch are already rescaled to [0, 1], so instead we sample $u \sim U(0, \frac{1}{256})^D$. Although in the DHM architecture the normalising flow is training on feature vectors rather than the images themselves, the classifier component is nevertheless a deterministic transformation of the input - not to mention that in theory it should be modelling $p(x) \approx p(h)$ (see Section 3.2). We argue that it is therefore still reasonable to de-quantise the image data for the DHM.

For spectral normalisation, the upper bound $c = 6$ recommended by Cao and Z. Zhang (2022a) is used by default unless specified otherwise. As with

J. Liu et al. (2020), the number of power iterations used is set to 1.

The normalising flow portion possibly has a different training regime. Cao and Z. Zhang (2022a) describe using a separate weight decay rate of $16e-4$ for the normalising flow (Cao and Z. Zhang, 2022b). H. Zhang et al. (2020), who Cao and Z. Zhang (2022a) seem to derive their architecture from in many cases, describe using Adam optimisation with a learning rate of 0.0001 for the flow portion of the DHM, with the Adam optimiser updating the flow parameters in parallel to the SGD optimiser updating the classifier parameters during training. While Cao and Z. Zhang (2022a) do not explicitly mention using Adam optimisation, it seems to be a reasonable option, and practical experiments (see Sections 4.1.2) show that the residual flow responds well to Adam optimisation, while at the same time better performance is gained from WRN under SGD optimisation. Several different combinations of SGD and Adam optimisation for different parameter sets in the DHM are explored (see Section 4.2.1), but the default training regime uses SGD optimisation as described above for the WRN portion of the DHM, and Adam optimisation for the flow portion.

Different parts of the network can be trained with different optimisers simultaneously by specifying the optimisers for non-overlapping subsets of the DHM model, which the object-oriented design makes fairly straightforward to do: the class `DHM` is designed with three attributes, `dnn`, `flow`, and `fc`, that refer to the classifier network, the normalising flow network, and the final fully connected classifier layer, respectively. This setup allows one to specify which parameters each optimiser should be updating. During training, parameters can be updated as normal for a pytorch model, except this time one must remember to repeat the `optimiser.zero_grad()` and `optimiser.step()` operations for both optimisers (see Appendix A.2 for a basic demonstration).

3.6.2.1 Loss Function

Finally, given that the purpose of the DHM is to model the joint probability $p(x, y) = p(y|x)p(x)$, the loss function to be optimised combines loss for the discriminative and generative components in one function. The discriminative loss - i.e., the loss for the classifier component - is a standard categorical crossentropy loss for a K-class classification problem, and is derived from the Monte-Carlo approximation of Equation 2.32:

$$\mathcal{L}_{dnn}(x, y) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} p(y_k)\log p^*(f(x)_k), \tag{3.3}$$

where f is the full classifier component of the DHM (i.e., $C \circ \phi$), $f(x)_k$ is the logit value assigned by the classifier to class $k \in K$, and y is the ground truth label. The model's probability estimate for class $k \in K$, $p^*(f(x))_i$, is computed with a softmax function on the classifier's logit outputs:

$$p^*(f(x))_k = \frac{e^{f(x)_k}}{\sum_{k=i}^{K} e^{f(x)_i}}. \tag{3.4}$$

Therefore the full classifier loss function becomes:

$$\mathcal{L}_{dnn}(x, y) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} p(y_k)\log \frac{e^{f(x)_k}}{\sum_{k=i}^{K} e^{f(x)_i}}. \tag{3.5}$$

The normalising flow loss is computed as described in Section 2.2.5.2, Equation 2.33, with the aim of maximising the likelihood of the training samples:

$$\mathcal{L}_{nf}(\theta) - \frac{1}{N}\sum_{i=1}^{N}\log p_X^*(x_i), \tag{3.6}$$

Bearing in mind that the loss function should be minimised, so we take the negative of the likelihood.

The full loss $\mathcal{L}_{DHM}$ is then calculated from the sum of these two losses. Since $\mathcal{L}_{nf}$ can reach far greater magnitudes than $\mathcal{L}_{dnn}$, and therefore risks overshadowing it, it is scaled by the hyperparameter λ:

$$\mathcal{L}_{DHM} = \mathcal{L}_{dnn} + \lambda\mathcal{L}_{nf}. \tag{3.7}$$

The DHM proves to be highly sensitive to λ, which must be carefully set. We explore this in detail in Section 4.3.1.

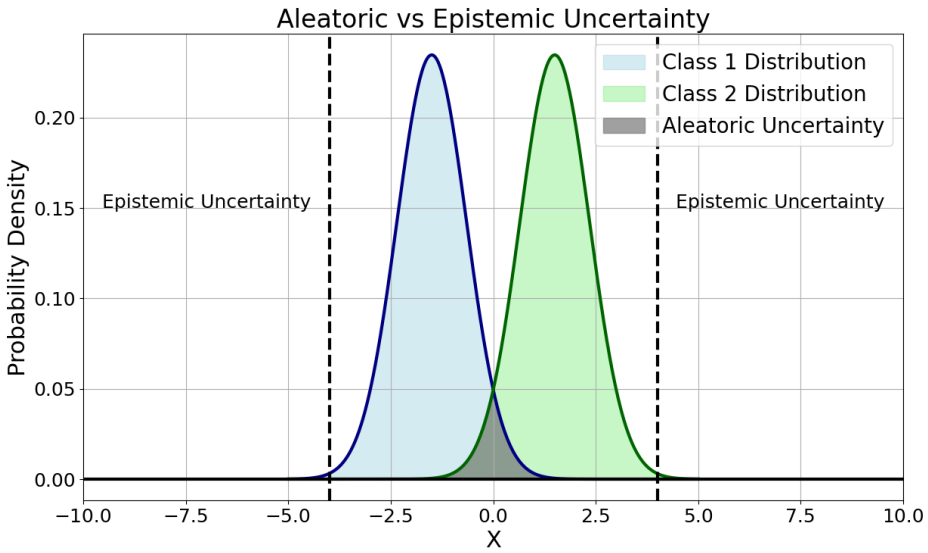


Figure 3.4: A 1D, 2 class example illustrating the difference between aleatoric and epistemic uncertainty.

3.6.2.2 Uncertainty Estimates

At inference, the joint probability calculated by the DHM is $p(x, y) = p(y|x)p(x)$, combining the likelihoods computed by the classifier and by the normalising flow. These two likelihoods can be interpreted as indications of the DHM's aleatoric and epistemic uncertainty, respectively; see Figure 3.4 for an illustration how we expect these uncertainties to manifest for a classifier.

Where $p(y|x)$ captures the model's confidence in which of the in-distribution classes an input best belongs to (and thus struggles with inputs that sit on the decision boundary in feature space), $p(x)$ captures the model's confidence that the input is likely at all given the training data. However, the aim with OOD detection is to recognise when a particular sample falls outside of the semantic distribution of the training data; that is, to capture epistemic uncertainty. In measuring the DHM's OOD detection performance, we therefore use only the $p(x)$ scores computed by the normalising flow component. Cao and Z. Zhang (2022a) use the same approach for measuring OOD performance.

3.6.3 Summary of Uncertainties

Clearly, several details regarding the exact implementation of the DHM are missing, complicating the re-implementation of the architecture. To summarise:

- **Residual flow structure:** The DHM paper seems to describe a homogeneous residual flow structure in terms of hidden dimension, while the paper it is based on describes a heterogeneous structure.
- **Logit transformation:** DHM appears to be based on an approach that describes using logit transformation on the features, despite being inappropriate for unbounded distributions.
- **Feature processing:** a number of valid options exist for processing the features before passing them on to the normalising flow.
- **Optimisation:** a number of valid configurations exist for how the different components of the DHM are optimised.

3.6.4 Evaluation

Given an input x, the normalising flow component of the DHM produces as output the vector z in the base distribution and the sum of the log determinants of the flow transformations. Since the base distribution p_Z is defined as the standard normal distribution, we can use Equation 2.54 to compute $\log p_X(x)$. We use this log probability to score the DHM's OOD detection capabilities, by comparing the log probabilities assigned to the in-distribution test set with the log probabilities assigned to some out-of-distribution test set.

As described in Section 2.2.3, we compute the AUROC, AUPR-in, and AUTC scores for each OOD dataset compared to the ID dataset, with a focus on the AUROC score in most experiments. For each OOD data set, the scores are calculated with respect to the ID test set.

3.7 Datasets

3.7.1 CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 datasets were introduced by Krizhevsky, G. Hinton, et al. (2009) and are popular, standardised image datasets for benchmarking visual machine learning tasks. Both are made readily available by the pytorch library.

The CIFAR-10 dataset consists of 60000 32×32 RGB (3 channel) images for 10 different classes, with 6000 images per class. It has a standard split into 50000 training images and 10000 test images, i.e., 5000 training images and 1000 test images per class. It was assembled from a much larger dataset of 80 million images, the "80 million tiny images" dataset (Torralba, Fergus, and Freeman, 2008), which itself is no longer available. The classes are mutually exclusive:

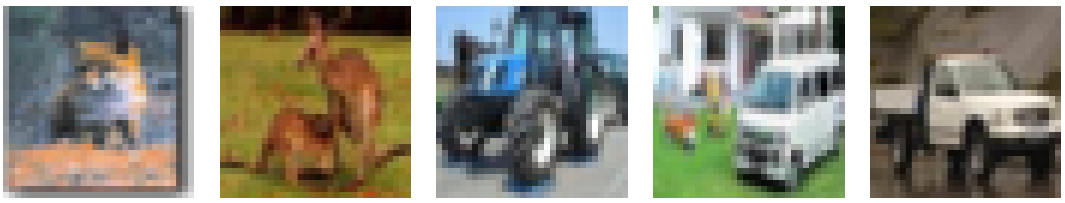
1. Airplane
2. Automobile
3. Bird
4. Cat
5. Deer
6. Dog
7. Frog
8. Horse
9. Ship
10. Truck

There is no overlap between "automobile" and "truck" (pickup trucks are not included), and the "automobile" class contains, according to the author, vehicles such as SUVs and sedans.

The CIFAR-100 dataset is sampled from the same dataset as CIFAR-10, except that it contains 100 classes instead of 10. It still consists of 60000 32×32 RGB images, with 50000 in the training split and 10000 in the testing split, meaning that there are 600 images per class with 500 for training and 100 for testing. While the classes are mutually exclusive, they have been chosen such that they can be organised into 20 "superclasses" of related concepts (see Table 3.3). Of crucial interest with respect to OOD detection is the fact

Superclasses	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Table 3.3: Superclasses and classes of the CIFAR-100 dataset



(a) “cat” (b) “deer” (c) “truck” (d) “car” (e) “truck”

Figure 3.5: Images in the CIFAR-10 dataset that do not match their class. 3.5a, 3.5b, and 3.5c depict the CIFAR-100 classes “fox”, “kangaroo”, and “tractor”. 3.5d depicts a van, which is ambiguous as it could arguably also be attributed to the “bus” category in CIFAR-100. 3.5e depicts a pickup-truck, a vehicle the creator of CIFAR-10 deliberately excluded to avoid confusion.

that both CIFAR-10 and CIFAR-100 are sampled from the same dataset, but contain mutually exclusive (though in some cases closely related) classes. This makes them an ideal test for OOD detection, as their covariate distributions can be expected to be extremely similar (i.e., the images in both datasets are generated by the same process), and only their classes - the semantic distributions - are different.

It should be noted though that the CIFAR-10 and CIFAR-100 classes are not necessary as exclusive as their class labels indicate; Fort, Ren, and Lakshminarayanan (2021) carried out a thorough investigation of these datasets and found multiple instances of mislabeled images, semantically ambiguous images, and classes that are simply difficult to distinguish visually at a 32×32 resolution. For example, in the CIFAR-10 test set they found images of a fox, a kangaroo, and a tractor (all CIFAR-100 categories) labeled as “cat”, “deer”, and “truck”, respectively (see Figures 3.5a, 3.5b, 3.5c). The CIFAR-10 categories “automobile” and “truck” also contain ambiguous samples that overlap with CIFAR-100 categories: for example, the “automobile” category and the CIFAR-100 “bus” category both contain images of vans (although they don’t seem to share identical images); see Figure 3.5d. Meanwhile, the CIFAR-10 category “truck” contains images of what are clearly pickup trucks

Figure 3.6: Examples of SVHN images, for digits 0 - 9.

(a CIFAR-100 category, see Figure 3.5e), despite Krizhevsky emphasising that pickup trucks had been deliberately left out of the CIFAR-10 dataset precisely to avoid such confusion. There are also a number of instances that, due to the 32×32 resolution of the images, simply don't seem possible to distinguish visually; examples include CIFAR-10 "truck" and CIFAR-100 "streetcar" samples, CIFAR-10 "deer" and CIFAR-100 "rabbit" samples, and CIFAR-10 "cat" against CIFAR-100 "otter" classes, among others. This is not intended as a criticism of the CIFAR-10 and CIFAR-100 datasets - after all, Fort, Ren, and Lakshminarayanan (2021) where only able to identify a handful of truly erroneously labelled samples - but to show that true 100% OOD detection performance should not be possible between these datasets. At the same time, the high degree of visual similarity between the classes makes the CIFAR-10/CIFAR-100 pair a particularly interesting and informative challenge for measuring OOD performance.

3.7.2 SVHN

The Street View House Numbers (SVHN) dataset was introduced by Netzer et al. (2011) and consists of 630420 32×32 RGB images of digits in house number plates, with 10 classes for digits 0 - 9 (see Figure 3.6). They are organised into three splits: a training set of 73257 images, a testing set of 26032 images, and an additional set of 531131 "easier" images that Netzer

et al. (2011) recommend using as additional training samples. As throughout this thesis we are mostly interested in using SVHN data as an OOD set, we mostly make use of the 26032 sample testing set. Similarly to the CIFAR datasets, SVHN is readily available via the pytorch library.

The SVHN dataset is of particular interest in the field of OOD detection, since Nalisnick et al. (2018) demonstrated that pure normalising flows trained on CIFAR-10 data will tend to assign higher likelihoods to SVHN samples, as discussed in Sections 2.2.1 and 2.2.4. Where the CIFAR-10/CIFAR-100 dataset pair is useful for measuring near-OOD performance (i.e. OOD detection with semantically similar classes), SVHN is an excellent option for identifying a focus on covariate rather than semantic distributions, as its semantic categories are far removed from the concepts covered by the CIFAR datasets. While achieving a high OOD detection score on SVHN with a CIFAR-10 trained model should be a seemingly easy task, actually getting a density based OOD method to consider SVHN samples as less likely than CIFAR-10 samples is also an important milestone to reach.

3.7.3 iNaturalist

We also work with a custom dataset based on the iNaturalist platform (iNaturalist.org, 2023). iNaturalist[7] is a citizen science platform that allows users to upload images and sound files of organisms they encounter; as such, it is an extremely large database that at the time of writing (February 2024) hosted 171 million observations covering 460437 species globally. This makes it a potentially very useful source for creating large visual datasets aimed at biological identification tasks. It should be noted that iNaturalist organises its records into "observations", where each observation is a package uploaded from a user and may contain multiple media articles - often, multiple images will be uploaded. Therefore there is a difference between the number of observations available of a species, and the number of unique images available of

[7]https://www.inaturalist.org

Figure 3.7: Example images from the stink bugs dataset. From left to right, the first four examples are of ID classes Dictyotus caenosus, Monteithiella humeralis, Cermatulus nasalis, and Nezara viridula. The final two are the OOD species Halyomorpha halys and Erthesina fullo. Images have been centre-cropped for display purposes.

that species. When building a database based on iNaturalist one must therefore consider the costs and benefits between including all images - which will mean a larger dataset, but potentially many groups of very similar images, risking overfitting - and only including one image per observation, which will yield a smaller dataset but with more diversity between samples.

We report results on a custom dataset aimed at identifying stink bugs not native to New Zealand. The New Zealand Ministry of Primary Industries (MPI) is concerned about the potential introduction of the Brown Marmorated Stink Bug (*Halyomorpha halys*)(Ministry for Primary Industries, 2023), which is an agricultural pest and has already spread to Europe and North America. At the same time, New Zealand has a number of native stink bug species that

are not a problem. This motivates an interesting use-case for OOD detection of training a classifier on endemic stink bugs and seeing how effectively the undesirable stink bugs can be identified.

In this problem, the endemic species that serve as the in-distribution (ID) samples are *Dictyotus caenosus* (brown shield bug), *Monteithiella humeralis* (Pittosporum shield bug), *Cermatulus nasalis* (brown soldier bug), and *Nezara viridula* (green vegetable bug), which the MPI notes for their visually similar appearance to the marmorated brown stink bug. As the OOD species, the MPI offers both *Halyomorpha halys* (brown marmorated stink bug, the target pest) and *Erthesina fullo* (yellow spotted stink bug) as examples of non-endemic species.

An image dataset of these species was constructed via automated requests to the iNaturalist API. While iNaturalist allows restricting queries to specific regions, there were very few images of the OOD species in New Zealand (6 for the brown marmorated stink bug, all from the same observation, and 7 for the yellow stink bug across 3 observations), while even the endemic species had limited samples; the most popular one, the green vegetable bug, only had 1482 observations and 1885 unique images. Instead, the global iNaturalist database was used to construct the stink bugs dataset, resulting in a dataset with a more usable number of samples per species (see Table 3.4).

To retrieve relevant images from the iNaturalist API, queries were submitted for research-grade (i.e., very reliable labels) observations with images for each taxon key. Since only 200 observations could be retrieved per query, these were iteratively retrieved. Each observation contained links to each image in the observation, so these were then used to directly retrieve the images.

As Table 3.4 shows, even when using the global database the number of samples is very unbalanced across the species, and fairly limited still for some species - particularly *Dictyotus caenosus*. There are a number of different techniques for handling unbalanced datasets - these are explored in more detail in Section 5.4 - but in terms of dataset preparation, we use two different

Group	Species	iNaturalist Taxon Key	Sample Count
ID	Dictyotus caenosus	388589	754
	Monteithiella humeralis	392922	1935
	Cermatulus nasalis	387368	2173
	Nezara viridula	141725	14550
OOD	Halyomorpha halys	81923	10000
	Erthesina fullo	359008	5856

Table 3.4: The stink bug dataset, with 4 ID species, 2 OOD species, the iNaturalist taxon key used to retrieve images for each species, and the number of images the dataset contains per species. The images were collected in December 2023.

approaches. The first approach is to simply load the entire dataset as is. The second approach involves capping each species at some maximum number of samples M when loading the data, which, depending on M, can balance out the class populations at the cost of the overall amount of training data. During loading, the dataset is split so that 80% of each species is assigned to the training set and the rest is assigned to the testing set. The images are preprocessed by resizing the smallest length to the target size (32 and 64 are tried), applying a random horizontal flip with probability 0.5, taking a random square crop using reflection padding of width 4, adding uniform noise, and finally normalising all channels to means 0.5, 0.5, 0.5, and standard deviations 0.25, 0.25, 0.25.

Chapter 4

Initial Experiments

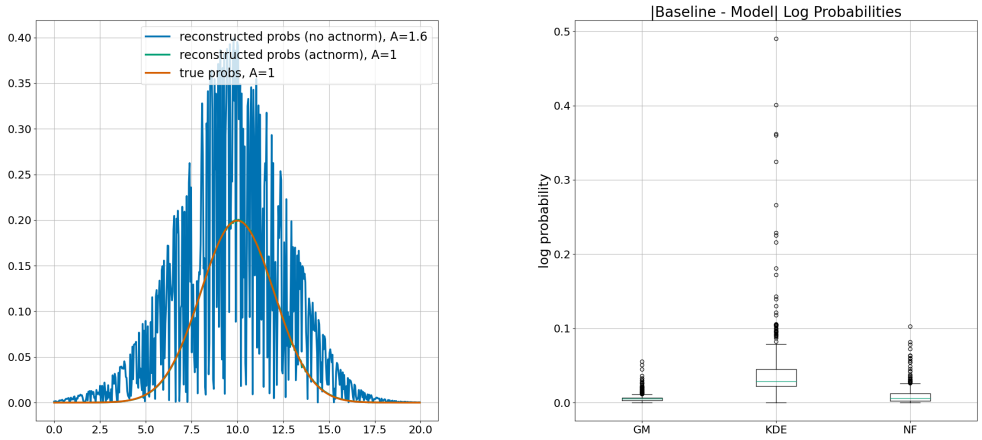
In this section, we explore the construction options for the DHM, which provides some insights into its operation. We begin with a brief overview of the behaviour of residual normalising flows on toy datasets.

4.1 Normalising Flows Comparisons

As a first step, we consider residual normalising flows on toy datasets to demonstrate their performance and investigate their learning behaviour. We also compare the residual flow implementation based on R. T. Chen et al. (2019)'s code and a residual flow implementation based on Stimper et al. (2023)'s normflows library.

4.1.1 1D Gaussian Dataset

First, we verify the normalising flow implementation on a simple task: fitting an off-centre 1D Gaussian distribution. This is the simplest task a normalising flow can be demonstrated on: it only has to learn the change in the Gaussian distribution's parameters. More specifically, we model the distribution $\mathcal{N}(10, 2)$. A key criterion for the validity of the implementation is the area under the estimated density function, which is expected to be approximately 1.



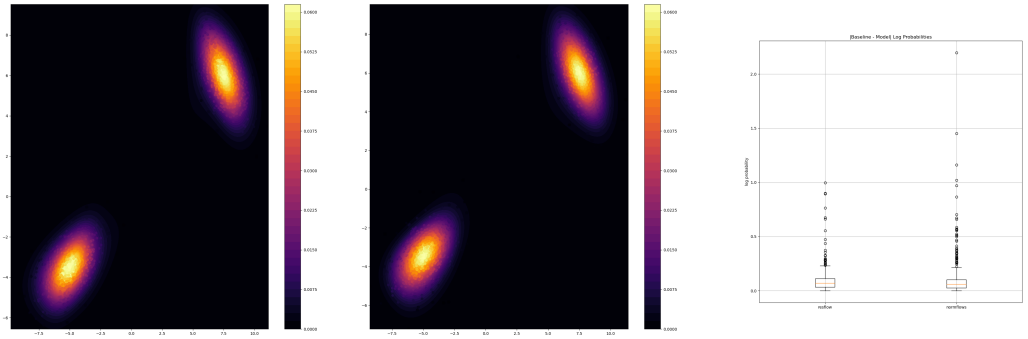
(a) (b)

Figure 4.1: a): normalising flow modelling a 1D Gaussian distribution, with and without activation normalisation. b): absolute log probability offsets between a fitted Gaussian model, a KDE, and the normalising flow, from the true distribution values.

A simple residual flow network with a single residual block and a single 16-dimensional hidden layer is sufficient for modelling this distribution effectively. This can be visualised by feeding the trained flow $x \sim \mathcal{N}(10, 2)$, calculating $p_X(x)$ according to Equation 2.54, and plotting the result over the ground-truth density (see Figure 4.1). This exercise also reveals the importance of activation normalisation (see Section 3.4), which made a large difference in the flow's ability to approximate the distribution even in this extremely simple example. Compared to simpler density estimation methods - fitting a Gaussian model to the data, or using a kernel density estimator (KDE) - the normalising flow performs roughly as well as the KDE and the GMM in terms of mean log density (see Table 4.1). An inspection of the absolute difference between the true distribution log density and the estimates of the models reveals that the normalising flow has a tighter spread of errors and a lower median log probability error than the KDE (see Figure 4.1b, where the median is included in the box plot as a green bar).

Model	Mean Log Probability	Mean Log Probability Error
GMM	-2.121	0.0053
KDE	-2.120	0.0368
NF	-2.121	0.0110

Table 4.1: 1D Gaussian modelling probabilities.



(a) (b) (c)

Figure 4.2: The original residual flow implementation (4.2a) and the normflows implementation (4.2b) trained on the same data for the same number of epochs. Errors for the normflows implementation are significantly higher (4.2c).

4.1.2 Mixture of Gaussians Dataset

Next, we compare the residual flow implementation based on the original code provided by R. T. Chen et al. (2019) with the normflows (Stimper et al., 2023) implementation on a 2D mixture of Gaussians dataset. We train both implementations on data sampled from a mixture of two 2D Gaussians, with the means $(7.5, 6.0)$ and $(-5.0, -3.5)$, and the covariance matrices $\left[\begin{smallmatrix} 1.25 & -0.75 \\ -0.75 & 1.80 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} 1.80 & -0.75 \\ -0.75 & 1.25 \end{smallmatrix}\right]$. We sample 50000 points as a training data set and another 1000 for the testing set.

When using the original residual flow implementation, a small architecture with 8 residual blocks and a single hidden layer of size 16 is sufficient for approximating the distribution to a high degree of accuracy after only 10 epochs of training (see Table 4.2). Even a model with a single residual block was ob-

Model	Log Probability	Mean Log Probability Error
Ground Truth	-3.818	0.0
Original Resflow	-3.836	0.087
Normflows Resflow	-3.837	0.085

Table 4.2: 2D Gaussian modelling probabilities.

Optimiser	Log Probability	Mean Log Probability Error
Adam	-3.836	0.087
SGD	-3.835	0.236

Table 4.3: Residual flow optimiser results.

served to achieve a decent performance. The same architecture implemented using the normflows library achieved similar mean log probabilities and mean absolute errors, but was found to have a greater spread in errors (see Figure 4.2c). Without providing any major advantage, it was decided not to use the normflows library further.

It was also found that using Adam optimisation was more effective for training the normalising flow than stochastic gradient descent (SGD). SGD results in less stable training: a learning rate that is too high results in not-a-number errors, but even when the learning rate is well adjusted to allow effective training, the resulting model still exhibits far greater log probability error than when trained with Adam optimisation (see Table 4.3). This observation motivates a preference for training normalising flows using Adam optimisation, since the exact training conditions for the normalising flow component of the DHM are not stated clearly by Cao and Z. Zhang (2022a), as discussed in Section 3.6.2.

4.1.3 Measuring the Volume Modelled by a 2D Normalising Flow

In the 2D case, the volume of the modelled distribution is expected to sum to 1. The volume can be approximated by creating a large grid, with a step size s, of samples that exhausts the area of significant support of the ground-truth density function, and measuring the density assigned by the normalising flow to each point. Let the density assigned to a point (x, y) be $p_{(x,y)}$. Then for a quartet of neighbouring points (x, y), $(x+s, y)$, $(x, y+s)$, and $(x+s, y+s)$, the volume of that grid square can be calculated by $s^2 \times (p_{(x,y)} + p_{(x+s,y)} + p_{(x,y+s)} + p_{(x+s,y+s)})/4$. The sum of all grid square volumes is then an approximation for the total volume modelled by the normalising flow. In the residual flow instance trained with Adam described above, we split the x and y between the minimum and maximum samples values along each respective axis into 400 steps, yielding a step size of 0.053 along the x-axis and 0.045 along the y-axis. We find the approximate volume to be 0.999, showing that the residual flow effectively preserves the volume of the base distribution, and that $p_X(x)$ can indeed be interpreted as a probability density.

4.2 Hyperparameter Exploration

As discussed in Section 3.6, there are a number of ambiguities in the exact implementation of the DHM architecture. We narrowed the uncertainties down to the configuration of the optimisation technique, the feature preprocessing stages, the use of activation normalisation, and the use of the logit transform. Additionally, we also consider the influence of the spectral normalisation coefficient of the feature encoder. The adjustment of the loss parameter λ, we leave for the next section, as investigating this parameter yields an important insight into the operation of the DHM. After completion of the hyperparameter exploration we select the best-performing combination as our interpretation of the DHM architecture, and achieve high performance, although we fail to

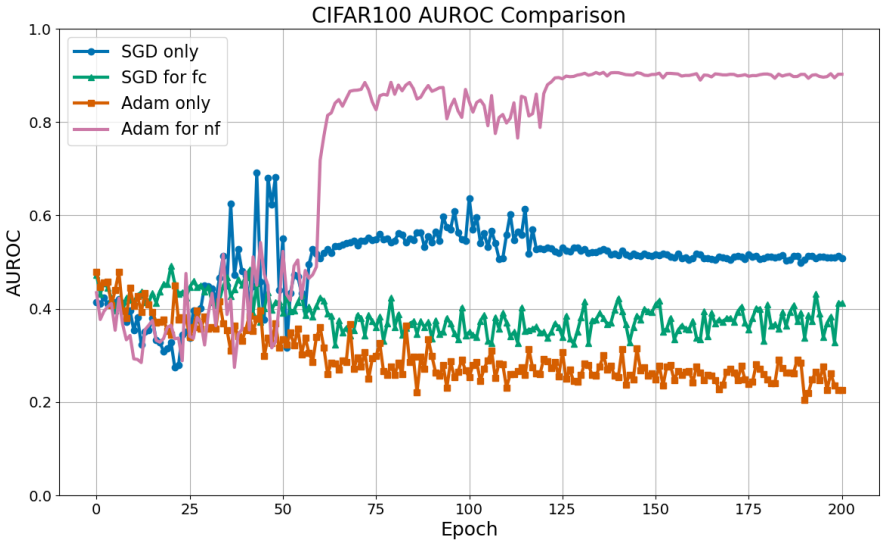


Figure 4.3: CIFAR-100 AUROC for different optimisation configurations.

achieve the 100% OOD detection rate claimed in the original paper.

4.2.1 Configuration of the Optimiser

As discussed in Section 3.6.2, there are multiple possibilities for configuring the optimisation of the DHM parameters. We consider the DHM to fundamentally consist of three components (the feature encoder, the normalising flow, and the final classifier layer), each of which could potentially be handled by a different optimiser. Empirically, the WRN architecture tends to learn better under SGD optimisation with scheduled learning rate drops, while normalising flows tend to learn more effectively under Adam optimisation. As described in Section 3.6.2, the default configuration we use applies SGD optimisation to the feature encoder and classifier layer, and Adam to the normalising flow portion. We also briefly consider three alternatives: training the entire DHM with SGD, training the entire DHM with Adam, and training everything except the final classification layer with Adam (see Table 4.4).

We show the results for CIFAR-100 AUROC scores in Figure 4.3 over the course of a 200 epoch training run for models trained on the CIFAR-10 dataset. Since our main interest is in the DHM's capabilities in identifying near-OOD

DNN	**NF**	**FC**
SGD	SGD	SGD
SGD	**Adam**	**SGD**
Adam	Adam	SGD
Adam	Adam	Adam

Table 4.4: Optimisation configuration options; "dnn" refers to the feature encoder, "nf" refers to the normalising flow component, and "fc" refers to the fully connected layer that maps features to the final logits. In bold is the preferred configuration.

samples (i.e., semantically similar but distinct samples), we focus on CIFAR-100 AUROC scores as the primary measure of the DHM's success. However, we also provide plots for validation accuracy and SVHN AUROC scores (see Appendix A.3.1) as well as a record of final AUROC, AUPR-in, and AUTC scores.

The results show clearly that the preferred configuration of optimisation techniques - Adam for the normalising flow, and SGD for all other components - yields the best results for the DHM architecture. While using SGD for the whole model has no impact on classification accuracy on the CIFAR-10 validation data (see Figure A.1), it clearly falls behind our chosen configuration in terms of CIFAR-100 AUROC scores, likely due to SGD struggling to optimise the normalising flow effectively. Meanwhile, the primarily Adam-based approaches fail completely, falling behind both in terms of validation accuracy and in terms of AUROC scores. Evidently Adam struggles to achieve the same classifier performance on the feature encoder component of the DHM, and this in turn yields lower quality feature representations for the normalising flow to work with. This justifies the use of the chosen hybrid configuration of optimisation algorithms as our standard configuration for training the DHM model.

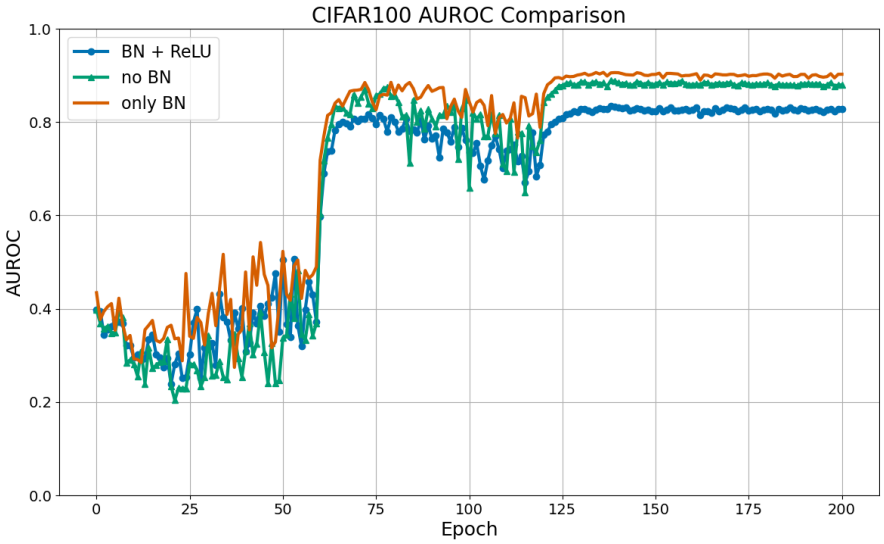


Figure 4.4: CIFAR-100 AUROC under different feature processing configurations.

4.2.2 Feature Processing

As discussed in Section 3.6.1, there are multiple possible configurations for handling the feature vector that is fed into the classifier component and the normalising flow respectively. By default, we apply batch normalisation to the feature vector, but we do not apply ReLU activations, as we view this as an unnecessary restriction on the range of values of the features. However, in this section, we do investigate two alternative configurations: one where ReLU is applied to the feature vector, and one where batch normalisation is left out as well.

The results for CIFAR-100 AUROC scores are shown in figure 4.4. While there is negligible impact on validation accuracy either way - see Figure A.4 - adding ReLU activations to the batch normalised features provides a clear disadvantage in terms of CIFAR-100 AUROC scores, while leaving out batch normalisation also results in a small but clear disadvantage. This indicates that the ReLU activation has no impact on the discriminative power of the feature space representation, but evidently causes the overlap of OOD features with in-distribution samples.

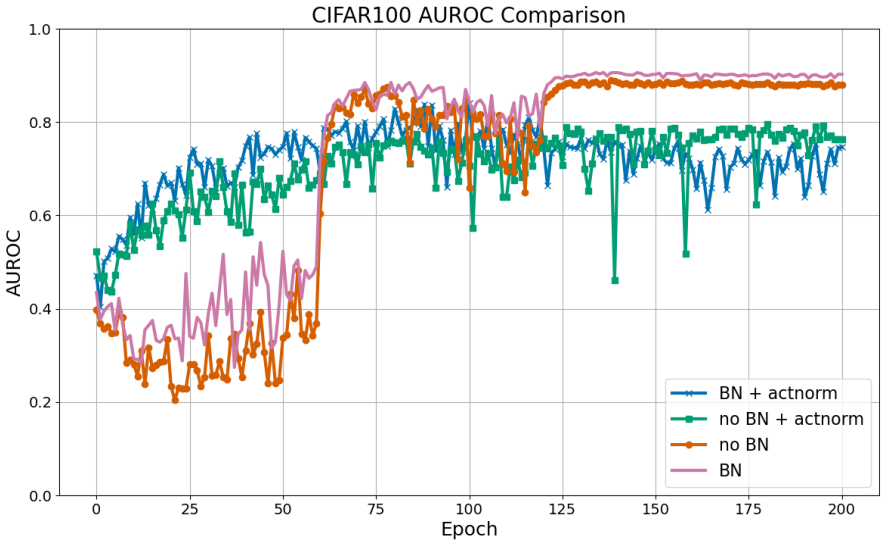


Figure 4.5: CIFAR-100 AUROC with and without activation normalisation, compared with the architectures lacking the final batch normalisation layer.

Interestingly, for final SVHN AUROC scores, there appears to be no difference between the configurations with and without batch normalisation (see Figure A.6), although the initial stages of training seem to be less stable when batch normalisation is included. It is possible that SVHN samples are sufficiently distant in feature space from in-distribution samples that batch normalisation has no effect on their separability from CIFAR-10 samples in terms of log density.

4.2.3 Activation Normalisation

In Section 4.1.1, we observed that activation normalisation drastically improved performance of the normalising flow on the simple synthetic data considered there, resulting in a smoother, more accurate model of the target distribution. However, when applied to the DHM in high-dimensional data, we find that the inclusion of activation normalisation actually reduces OOD detection capabilities considerably; see Figure 4.5.

We note that in the standard configuration, the normalising flow receives the output of the final batch normalisation layer of the feature encoder - that

is, the input will already be normalised. This potentially makes activation normalisation redundant; also, it raises the question of whether it might prove advantageous when given unnormalised features. However, Figure 4.5 shows that removing the batch normalisation layer yields only a minor advantage, and in fact still offers considerably lower performance than when only the batch normalisation layer is used.

4.2.4 Combinations of Configurations

While individually the configurations explored did not offer an improvement over the chosen training configuration, it is possible that some combination of these configurations would yield superior performance overall. An exhaustive search over all possible combinations is impractical, but a large set of promising alternatives was explored, and the most promising alternatives are reported in Appendix A.3.4. However, none were able to beat the chosen training configuration, which is why it remains the default configuration used throughout this thesis unless specified otherwise.

To summarise, we settle upon a training configuration that trains the feature encoder and final classification layer with SGD optimisation, and the normalising flow with Adam optimisation. Features are batch normalised before being passed on to the normalising flow, but ReLU activations are not applied, and activation normalisation is omitted in the normalising flow.

4.2.5 Spectral Normalisation

The feature encoder component of the DHM has the hyperparameter c, the spectral normalisation coefficient, that enforces an upper bound on the spectral norms of the network's layers; see Section 3.5. J. Liu et al. (2020) recommend performing a grid search over some discrete set $c \in \{0.95, 1, 2, ...\}$ and selecting the smallest c that does not impact classification accuracy, and Cao and Z. Zhang (2022a) follow this method as well. Both papers settle on $c = 6$ for a wide Resnet model, although it is worth noting that Amersfoort et al. (2021)

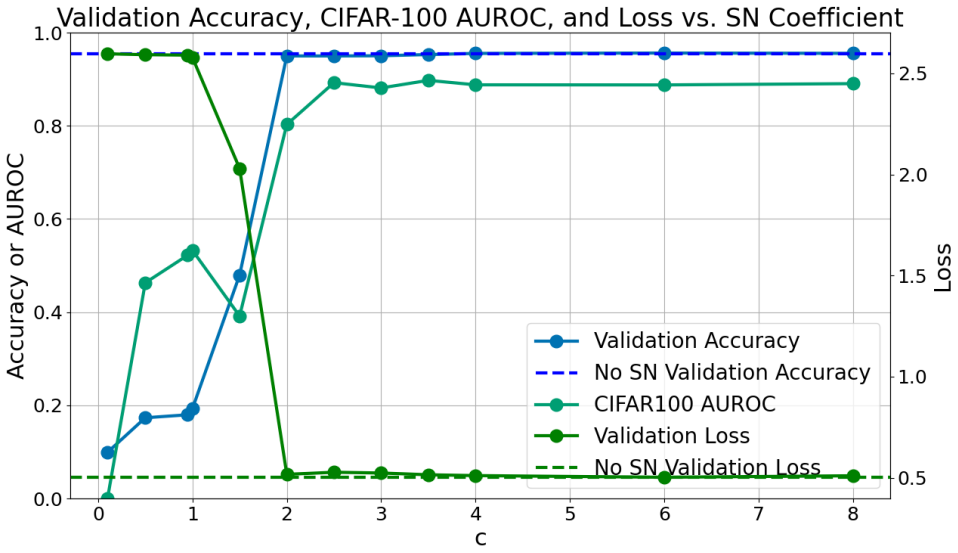


Figure 4.6: Validation accuracy and loss against the DNN spectral normalisation coefficient c, showing a clear cutoff point below $c = 2$.

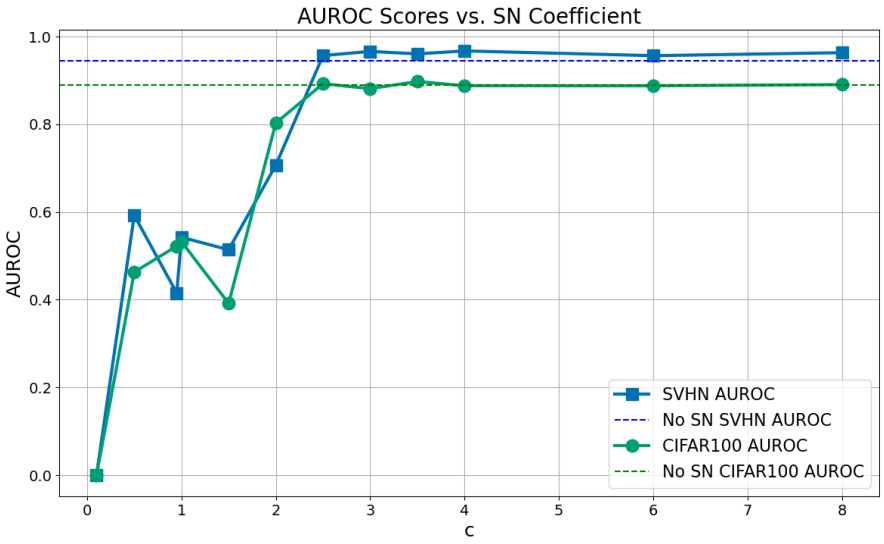


Figure 4.7: CIFAR-100 and SVHN AUROC scores against the DNN spectral normalisation coefficient c.

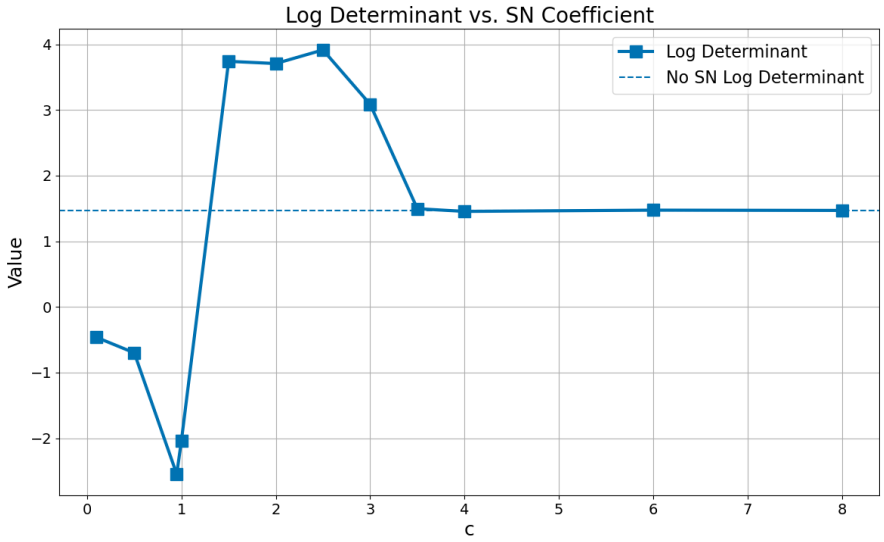


Figure 4.8: Normalising flow log determinant against c, showing a large increase after $c = 1$ and levelling out for $c \geq 4$.

similarly use a spectrally normalised wide Resnet and choose $c = 3$ instead.

As seen in Figure 4.6, for the DHM configuration used as the standard configuration in this thesis, validation accuracy rapidly drops off below $c = 2$, and CIFAR-100 and SVHN AUROC scores follow the same pattern (see Figure 4.7). Meanwhile, the normalising flow's log determinant (Figure 4.8) shows an interesting pattern of remaining negative for $c \leq 1$, before jumping up and then decreasing, levelling out at around 1.47 for $c \geq 4$. In other words, for $c \leq 1$, as the rate of change of the latent variable z with respect to the feature x is < 1, the density for z must therefore be scaled by a small value to conserve probability in feature space; see Section 2.2.5.2 for an explanation of the relationship between x and z in terms of the log determinant.

For $c < 1$, the data exhibits a wider distribution in feature space than in latent space. However, both the feature and latent space of the $c = 0.95$ model are far more narrowly distributed than those of the $c = 3$ model; see Table 4.5. Note also that, despite the normalising flow using a standard normal Gaussian for its base distribution, both models exhibit standard deviations less than 1 in the latent space, although the $c = 0.95$ model is an order of magnitude worse in

c	x stdev	z stdev
0.95	0.04241	0.01029
3.0	0.2041	0.2037

Table 4.5: Comparison of feature and latent space standard deviations measured in networks trained with high and low c. Standard deviation is measured across all feature values in the test set after training.

this regard. This provides evidence that the feature encoder is "cheating" by generating features that maximise $p_Z(z)$ when data is passed to the normalising flow: the data is transformed to be more concentrated in feature space, yielding density estimates that are trivially greater. We discuss this phenomenon and its implications in more detail in Section 4.3.1.

Meanwhile, when no spectral normalisation is applied at all, the log determinant ends up being around the same as for $c \geq 4$: at 1.47. This may indicate that spectral normalisation has limited impact on the wide Resnet above a value of 4, rendering it redundant. Interestingly, Cao and Z. Zhang (2022a) claim that without spectral normalisation, the DHM becomes less stable and AUROC scores vary between 0.9 and 1.0. We have not observed such behaviour.

It is also interesting to note that validation accuracy and loss do not seem to be ideal metrics for choosing c, as they suggest that $c = 2$ will perform as well as larger c values, but as the AUROC scores show, OOD performance already begins to drop at $c = 2$; see Figure 4.6. Instead, $c = 2.5$ or 3 appear to be preferable options. This raises the question of how one might pick c in a practical setting where OOD test sets may not be available. We observe that the standard deviation of the ID testing set features correlates strongly with final AUROC scores, and appears to be a more effective measure (see Figure 4.9). Feature standard deviations exhibit a dramatic collapse for $c < 3$, which indicates one should pick the smallest c that does not sacrifice feature variance.

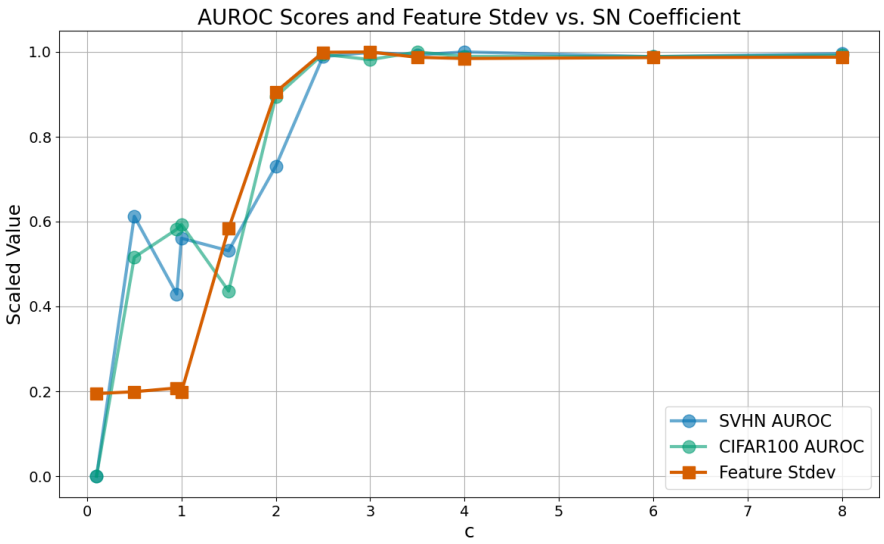


Figure 4.9: Feature standard deviations compared to CIFAR-100 and SVHN AUROC scores. All series have been rescaled to a maximum of 1 for clarity. Feature standard deviation captures the drop in OOD performance more effectively than accuracy or loss.

The results provide evidence that the collapse of features into a small region is closely correlated with a drop in OOD performance.

These observations lead us to conclude that $c = 2.5$ or 3, as used by Amersfoort et al. (2021), is a better choice for the WRN when training on CIFAR-10 data. We observed that while $c = 2.5$ can attain high performance, it also exhibits a higher variance across training runs, i.e., it is less stable. We therefore keep $c = 3$ as the standard.

4.2.6 Logit Transformation

As discussed in Section 3.4, there is ambiguity around whether or not Cao and Z. Zhang (2022a) used a logit transform to preprocess the features for the normalising flow. While we argue that a logit transform is inappropriate for unbounded feature vectors, H. Zhang et al., 2020 - whose architecture Cao and Z. Zhang (2022a)'s DHM model is based on - explicitly mention using it, and Cao and Z. Zhang (2022a) do not contradict this.

Model	CIFAR-100			SVHN		
	AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
DHM	**0.9058**	**0.9154**	**0.3679**	**0.9743**	**0.9597**	**0.2722**
Logit DHM	0.7543	0.6272	0.4888	0.9554	0.7996	0.4273

Table 4.6: DHM with logit transform compared against baseline model in terms of OOD detection capabilities.

Since the logit transform requires all inputs x to be $0 < x < 1$ (see Equation 3.1), they need to be rescaled. While standard intensity values in image data x can simply be scaled by $\frac{1}{255}$, in this case we need to rescale each feature x with

$$x := \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{4.1}$$

where $\min(x)$ is the minimum value element in x and $\max(x)$ is the maximum value element. Note that we need to adjust the log density of each rescaled sample before passing it to the normalising flow. Since all elements of x will be rescaled by $\frac{1}{\max(x)-\min(x)}$, it follows that their densities will therefore be scaled by $(\max(x) - \min(x))$. Since a normalising flow operates by composing the changes in volume of each flow transformation (see Section 2.2.5), which in practice means summing the log determinants of each flow module, we can simply add $\log(\max(x) - \min(x))$ to the log determinant computed for the flow in order to obtain the total change in volume.

The resulting model performs worse on all measures when the logit transform is included (see Table 4.6). This transformation is therefore not used in any further experiments.

4.2.7 Feature Normalisation

The investigation of the logit transform led to the discovery that normalisation, particularly l_∞ normalisation, of the features drastically improved OOD performance. Figure 4.10 plots CIFAR-100 AUROC scores for l_1, l_2, and l_∞ nor-

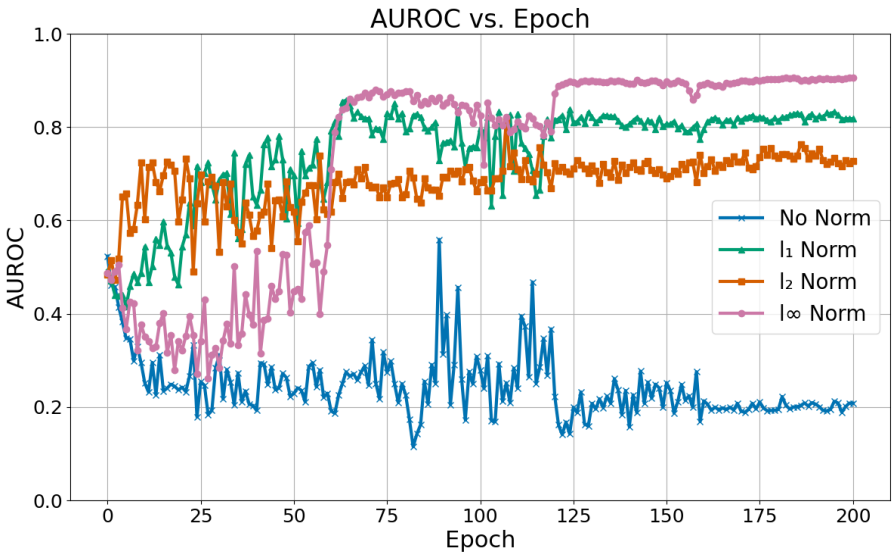


Figure 4.10: CIFAR-100 AUROC scores under different normalisation schemes, showing that l_∞ normalisation yields the best results.

malised features, as well as for un-normalised features. This shows that applying l_∞ normalisation drastically improves performance beyond all others, while not using any normalisation at all leads to the complete failure of the DHM on the OOD task. This ordering of performances, with $l_\infty > l_1 > l_2 > None$, is preserved across all relevant performance metrics; see Appendix A.3.5.

Note that to implement this, we must once again adjust the log densities according to the normalisation scheme. In general terms, a normalisation operation of order p will scale an input x by $\frac{1}{||x||_p}$; therefore, for each normalisation scheme, we adjust the log density by $\log(||x||_p)$.

We discuss the causes for this behaviour in Section 5.1. Given its superior performance, we use l_∞ normalisation for the CIFAR-10 DHM features throughout this work unless specified otherwise.

4.2.8 CIFAR-10 Results of the Replicated DHM

The final configuration of the replicated DHM model does not attain AUROC scores of 100% as claimed in the paper by Cao and Z. Zhang (2022a), but nevertheless performs well in both near-OOD and far-OOD detection, with a

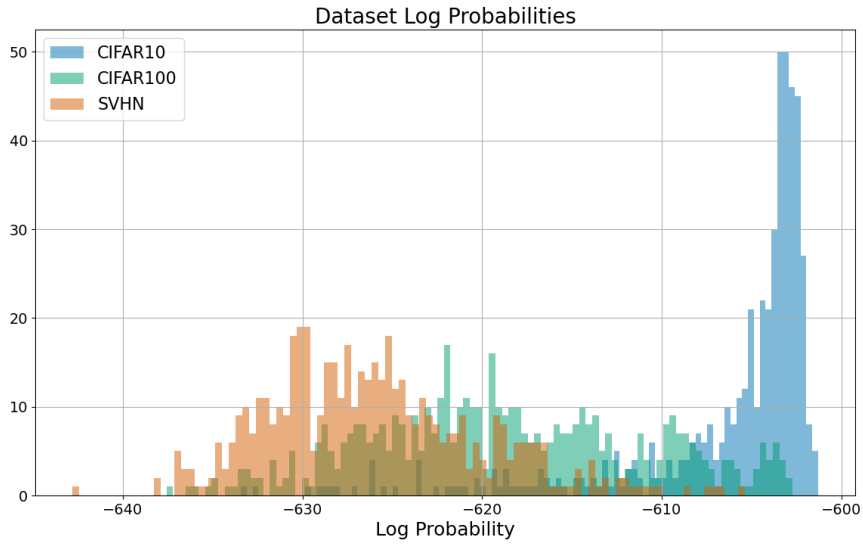


Figure 4.11: Histogram of assigned log probabilities for the DHM trained on CIFAR-10.

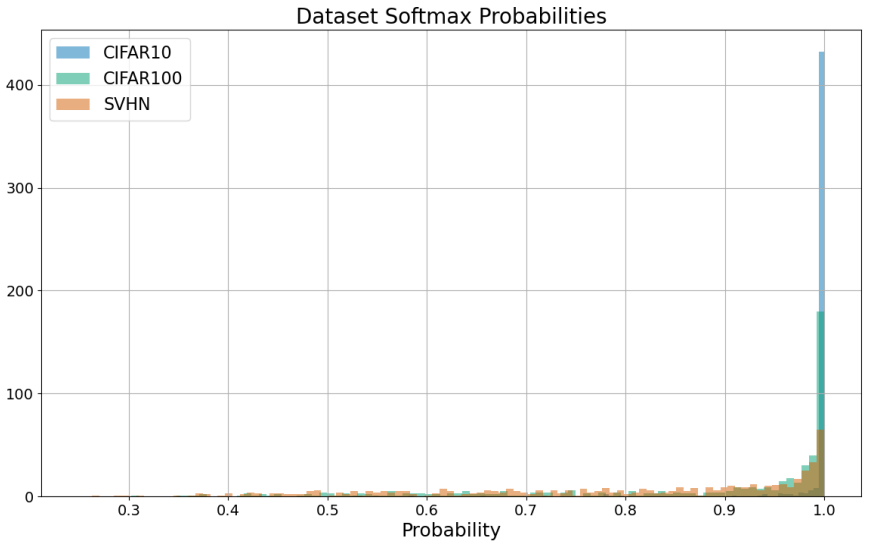


Figure 4.12: Histogram of softmax probabilities for the DHM trained on CIFAR-10, showing a high amount of overlap.

CIFAR-100 AUROC score of 0.9058 and a SVHN AUROC score of 0.9743 (see first row of Table 4.7).

We can compare this behaviour to the OOD method described by Hendrycks and Gimpel (2016), which compares a classifier's maximum softmax score for new samples with the maximum softmax scores attained for correct predictions in the ID test set to detect OOD instances. Using the DHM's softmax scores this way yields a CIFAR-100 AUROC score of 0.8639 and a SVHN AUROC score of 0.9467. An inspection of the log density histograms (which provide a way to visualise the distribution of $\log p_X(x)$) shows the CIFAR-10 samples concentrating into a higher likelihood region than the other samples, although there is still some overlap with CIFAR-100 samples in particular (see Figure 4.11).

However, we note that the AUROC scores for the softmax-based method are perhaps deceptively high. Figure 4.12 shows that despite the apparently high AUROC scores of 0.8639 and 0.9467 for CIFAR-100 and SVHN, respectively, the softmax scores for all datasets are clearly concentrated into the high confidence 0.9 to 1.0 region. Figure 4.15 - where the MDS embedded points from the three datasets are coloured by their softmax score on a range of 0 to 1 - shows clearly that regions well beyond the ID clusters are still assigned high softmax confidence scores.

This is a result of how AUROC scores are calculated: even if the ID and OOD samples mostly overlap in their assigned likelihood ranges, the calculated AUROC score will remain high for as long as OOD samples are more frequently assigned a lower density than ID samples. AUPR scores will yield high values by the same mechanism. While this is strictly speaking an acceptable result, it is counter-intuitive and contradicts our expectations of OOD detection; ideally, one would expect an OOD detector to strongly separate the ID and OOD samples.

This is precisely the issue Humblot-Renaux, Escalera, and Moeslund (2023) raised, leading them to introduce their AUTC score (see Section 2.2.3) with

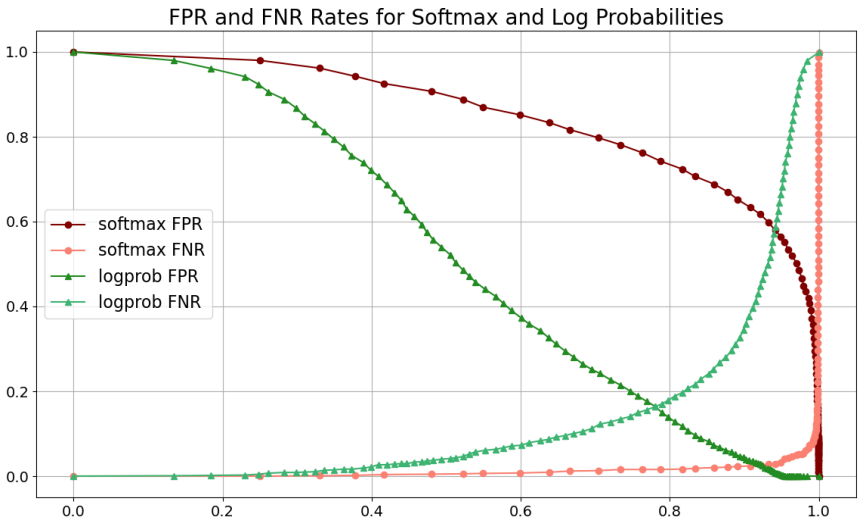


Figure 4.13: FPR and FNR plots for the CIFAR-10/CIFAR-100 DHM softmax and log probability scores, showing a stronger separation between the ID and OOD samples in the DHM log densities than in the classifier softmax scores.

the goal of favouring solutions that can establish clear separations between the ID and OOD datasets. The AUTC score calculates the average between the area under the false positive curve (AUFPR, i.e., the rate at which OOD samples are falsely assigned to the ID set across all thresholds) and the area under the false negative curve (AUFNR), which measures the rate at which ID samples are falsely assigned to the OOD set across all thresholds.

Figure 4.13 shows FNR and FPR plots for the DHM's softmax scores and log densities (with the CIFAR-100 test set as the OOD set), demonstrating exactly why the AUROC scores are misleading: across the range of thresholds (here plotted as a proportion of the real range to show the log density results on the same x scale), the softmax FPR remains above 0.5 until a threshold of 0.97, while the FNR remains almost zero across the entire range of thresholds and has reached 0.05 at this threshold. Only in the final section from 0.97 - 1.0 do they completely invert, and they intersect at a threshold of 0.9979, at which point the FPR and FNR rates are 0.17. In other words, half of all OOD samples are in the top 3% of confidence scores and overlap with 95% of ID

samples, while 17% - almost a fifth - of OOD samples overlap with 83% of ID samples.

In contrast, the DHM log density scores show a much cleaner separation of the ID and OOD datasets, with the FPR dropping below 0.5 at a threshold of 0.51, and the FPR and FNR intersecting at a threshold of 0.78 with rates of 0.16. Furthermore, above a threshold of 0.96 the FPR is exactly 0, while the FNR is at 0.82, i.e., 18% of ID samples are completely above the range of OOD samples. This is reflected by the AUTC scores (see Table 4.7), which show a greater improvement from the softmax to the log density OOD detection method than the AUROC scores do.

We note that an alternative measure for the amount of separation between the ID and OOD samples might be obtained by subtracting the area under the FPR (AUFPR) from the area under the TPR (AUTPR), yielding the difference between the two areas: as long as the ID samples are assigned higher densities than OOD samples, the FPR will be less than the TPR and TPR-FPR will yield a positive result. Ideally, the FPR would decrease at a much greater rate than the TPR, as would be the case with a stronger separation between the ID and OOD samples; this would manifest in a much greater difference in area. In the case of perfect separation, the FPR would drop off immediately to zero while the TPR would still be 1, maximising the difference in area. Under this measure, the DHM receives a score of 0.3366 while the softmax baseline yields a score of 0.137.

However, a certain degree of ID-OOD overlap seems unavoidable. An MDS (multi-dimensional scaling) embedding of the features (Figure 4.14) shows clearly that some CIFAR-100 samples are being placed into the same region as in-distribution classes, making OOD detection impossible. As discussed in Section 3.7.1, there exists semantic overlap between the CIFAR-10 and CIFAR-100 sample images, either due to ambiguity in the classes or due to erroneously labeled images.

Indeed, an inspection of the nearest neighbours among the ID and OOD

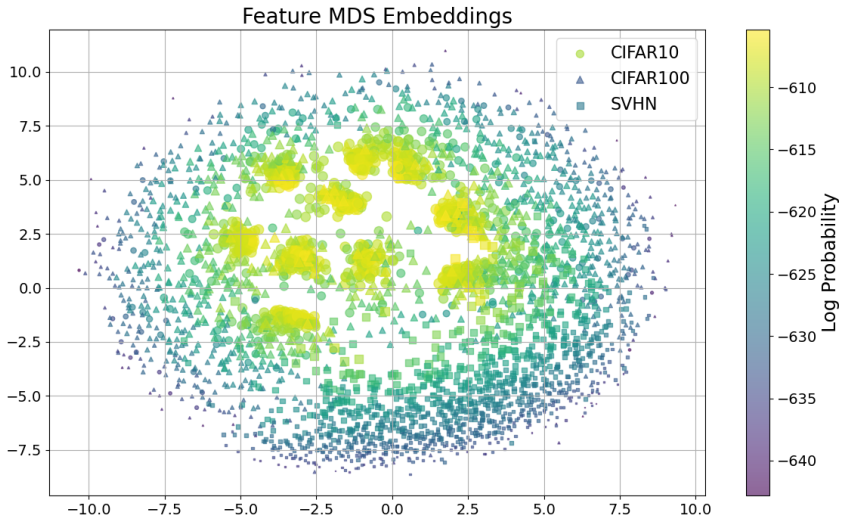


Figure 4.14: Feature MDS embeddings of the DHM trained on CIFAR-10, with corresponding log probabilities showing high likelihoods assigned to ID samples.

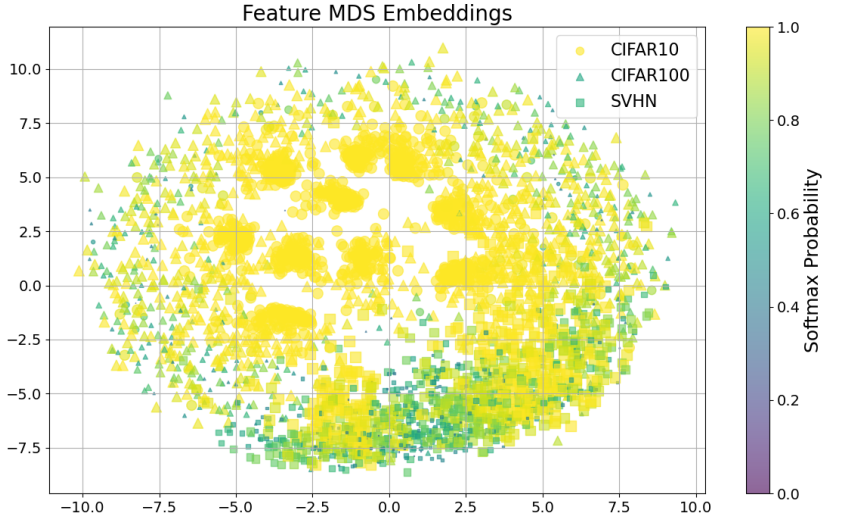


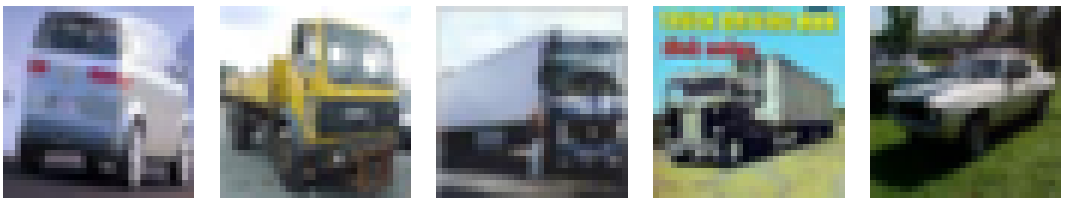
Figure 4.15: Feature MDS embeddings of the DHM trained on CIFAR-10 with corresponding classifier softmax probabilities, showing poorer epistemic uncertainty estimates.

Model	CIFAR-100			SVHN		
	AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
DHM	0.9058	0.9154	0.3679	0.9743	0.9597	0.2722
Softmax Baseline	0.8639	0.8757	0.4319	0.9467	0.9491	0.3759

Table 4.7: DHM compared with softmax baseline OOD detection baseline.

Feature Distance	ID Class	OOD Class	ID Logprob	OOD Logprob
0.0	car	bus	-604.09	-604.09
0.5776	truck	pickup truck	-601.96	**-601.70**
0.5779	truck	bus	**-602.68**	-602.91
0.5801	truck	bus	**-601.91**	-602.80
0.5907	car	pickup truck	**-602.23**	-602.36

Table 4.8: Top 5 nearest neighbour pairs comparing the ID CIFAR-10 samples to OOD CIFAR-100 and SVHN samples, along with the Euclidean distance in feature space, the class labels, and the log probabilities. The higher likelihood value for each pair is in bold, although the differences are minor.



(a) “car” (b) “truck” (c) “truck” (d) “truck” (e) “car”

(f) “bus” (g) “pickup” (h) “bus” (i) “bus” (j) “pickup”

Figure 4.16: Top 5 nearest neighbours (sorted by the distance of the nearest OOD neighbour to each ID sample) between ID and OOD samples for the default DHM model. The first row contains the ID CIFAR-10 samples and the bottom row contains the nearest neighbours from among all CIFAR-100 and SVHN test samples.

classes based on feature vectors extracted from the trained CIFAR-10 DHM reveals that the primary source of confusion are the vehicle classes, particularly trucks, pickup trucks, buses, and cars; see Figure 4.16. The top nearest neighbour is due to both the CIFAR-10 and CIFAR-100 test sets containing the same image, a van labelled (not unreasonably) “car” in CIFAR-10 and “bus” in CIFAR-100. Apart from the top two neighbours, the likelihood assigned to the ID samples was higher than that assigned to the neighbouring OOD sample, although only slightly (see Table 4.8). These results once again call into question whether 100% OOD detection is actually possible for the CIFAR-10 and CIFAR-100 datasets.

4.3 Studying Feature Collapse

In previous sections we observed observed a tendency for the DHM to assign relatively high likelihoods to OOD samples when configured incorrectly. In the following section we analyse this phenomenon, and argue that the collapse of feature values to a tightly concentrated region in the feature space is the cause.

4.3.1 λ Optimisation

As described in Section 3.6.2, the loss function for the DHM is $\log(p(y|x)) + \lambda \log(p(x))$, where the parameter λ determines how much the normalising flow loss contributes to the total loss. As the normalising flow loss is simply the negative log probability density of its training samples (see Section 2.2.5.2), its values are in principle unbounded, and λ must be adjusted for any particular architecture to ensure the flow loss signal does not overpower the effect of the classifier loss.

Adjusting λ is a balancing act: if it is set too low, the normalising flow will not learn anything, and in the extreme case of $\lambda = 0$ one is simply training the classifier by categorical cross-entropy as normal. Meanwhile, if λ is set too high, the classification loss has too little influence on the loss signal and the DHM will arrive at the trivial solution of producing features easily modelled by the normalising flow, but stripped of information useful for classification. We explore this phenomenon in more detail below.

Cao and Z. Zhang (2022a) describe using $\lambda = 0.06$ for experiments using the WRN-28-10 architecture as the encoder. However, despite our best efforts at recreating the DHM architecture exactly as described by Cao and Z. Zhang (2022a), we find this value to be too aggressive: the resulting DHM models have reduced classification accuracy, and AUROC scores of around 0.5, meaning they assign ID and OOD samples roughly equal likelihood on average.

Cao and Z. Zhang (2022a) recommend finding the optimal λ for a DHM

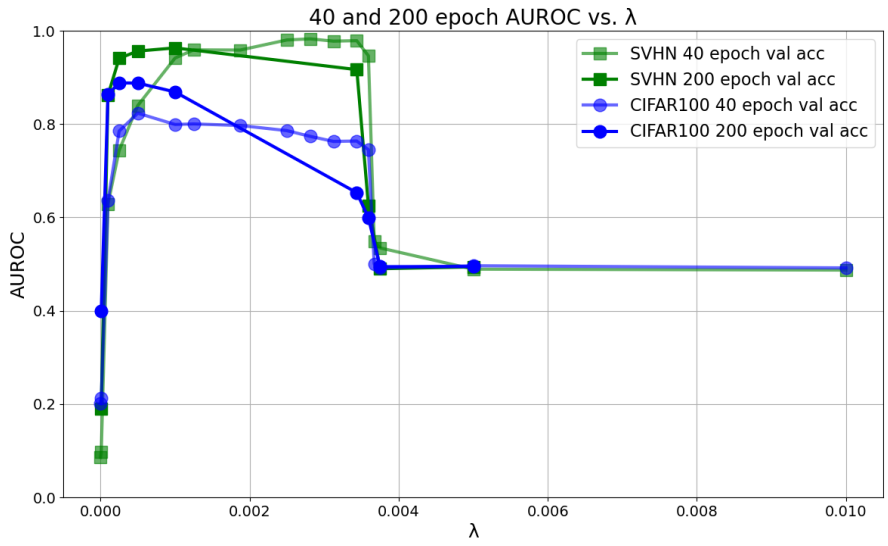


Figure 4.17: Comparison of AUROC scores after 40 and 200 epochs of training.

architecture via a binary search over final classification accuracy. Since the optimal λ value can be fairly small, we begin with a search over a logarithmic scale (i.e., 1, 0.1, 0.01, ...) until a value is reached where performance decreases, to determine the upper and lower bounds for the binary search. We find the search can be sped up by producing each sample with a shortened training run: we generate each sample with a 40 epoch run where the SGD learning rate is dropped by 0.2 after 25 and 35 epochs. While this short training run does not allow the DHM to attain the performance of a 200 epoch run, it is sufficient for identifying the λ threshold at which OOD performance increases. However, the 40 epoch runs are not sufficient for identifying the optimal λ for the final model (see Figure 4.17). In our experiments, we identify an approximate λ using short runs first, before narrowing λ down further using 200 epoch runs.

While the original paper recommends searching over the final validation accuracy, we find that validation accuracy tends to be fairly invariant below a certain λ value (see Figure 4.19), in contrast to the plots Cao and Z. Zhang (2022a) show (Figure 4.18). Hence, another measure would be preferred. An OOD dataset is not available in a deployment setting, so we compared training metrics with CIFAR-100 and SVHN AUROC scores to identify one strongly

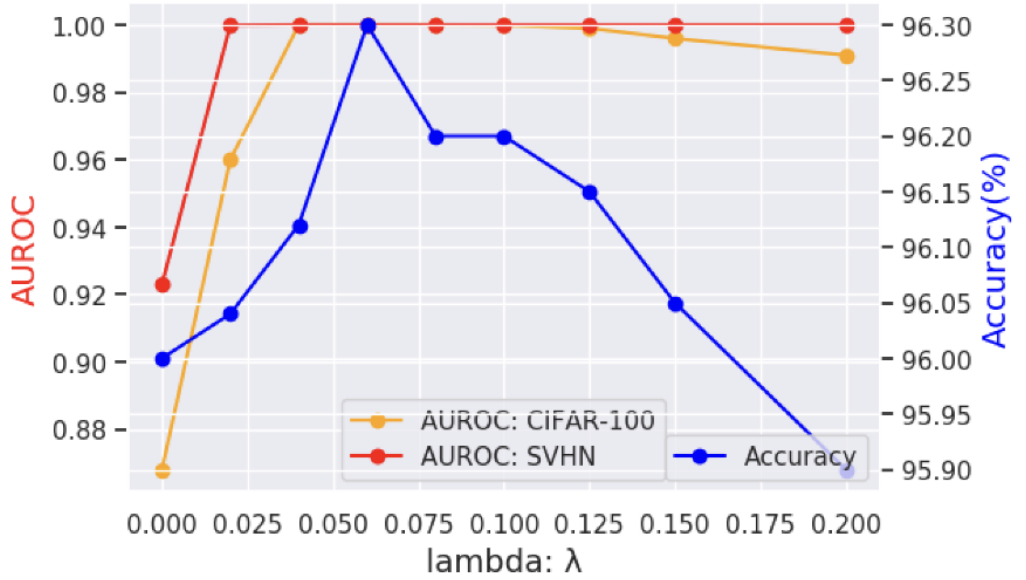


Figure 4.18: Plots of CIFAR-100 and SVHN AUROC scores and validation accuracy published by Cao and Z. Zhang (2022a) ©2011 IEEE (permission to reuse granted by IEEE), indicating that validation accuracy can be used for selecting λ.

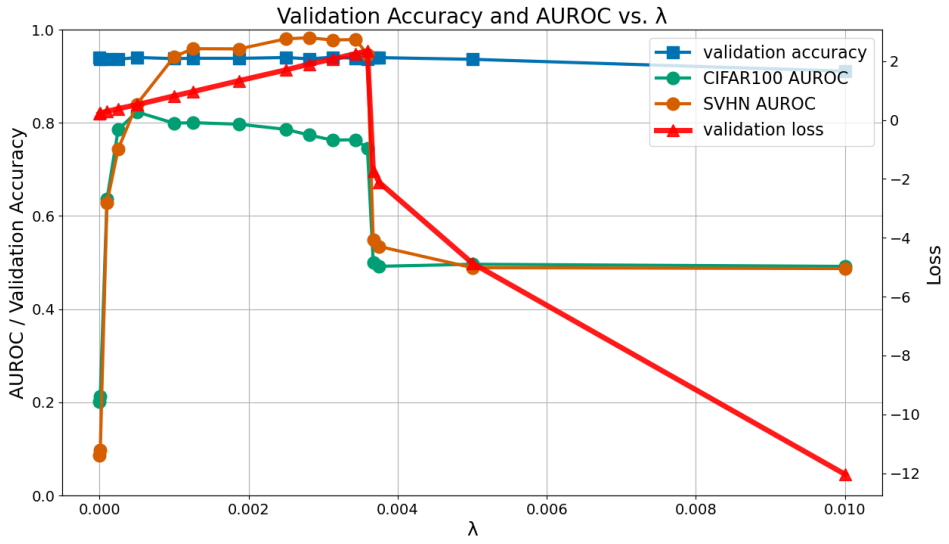


Figure 4.19: A plot of validation accuracy, validation loss, CIFAR-100 AUROC, and SVHN AUROC scores against λ. In contrast to what Cao and Z. Zhang (2022a) report (see Figure 4.18), we find DHM validation loss to be a more effective indicator of AUROC performance.

correlated with final OOD detection performance and found that the DHM loss appears to be a better indicator for OOD performance in terms of λ. Counter-intuitively, the optimal λ is where the final DHM validation loss is *maximised*; see Figure 4.19. This is likely because, at this point, the contributions of the classifier and normalising flow losses to the combined DHM loss are being balanced effectively, thus ensuring that both components contribute meaningfully to model updates. The classifier loss follows the same pattern as the total loss, while the flow loss stays roughly constant for λ values below maximum validation loss. The flow loss is dependent on the log determinant, which is a strong alternative candidate for selecting λ, as is discussed further below.

Figure 4.19 illustrates clearly that there is some threshold λ value, τ, below which OOD detection instantly improves from AUROC scores of 0.5 - corresponding to random guessing - to the highest observed AUROC scores. We also note that far-OOD detection scores (SVHN AUROC) are at their highest immediately below τ, and decrease as λ drops further, while near-OOD scores (CIFAR-100 AUROC) increase still further before dropping off near $\lambda = 0$.

For $\lambda < \tau$, validation loss decreases linearly with λ. The explanation for this is simple: since the loss for the normalising flow, L_{nf}, is scaled to a very small value, it can be seen as approximately constant in this region, and the loss for the classifier, L_{dnn}, is independent of λ and will therefore be effectively constant for a constant number of training epochs (note that increasing the influence of L_{nf} on the final loss can decrease the rate at which L_{dnn} is reduced, but the effect is negligible for $\lambda << 1$). Thus the full loss is determined by two effectively constant losses (for a constant number of training epochs) and λ, and therefore decreases linearly with λ.

For $\lambda > \tau$, we attribute the dramatic change in OOD performance to a more complex system of interactions. To explain the observed behaviour, recall first that the loss for the normalising flow is $-\log p_X(x)$, where

$$\log p_X(x) = \log p_Z(f(x)) + \log(det(J_f)), \tag{4.2}$$

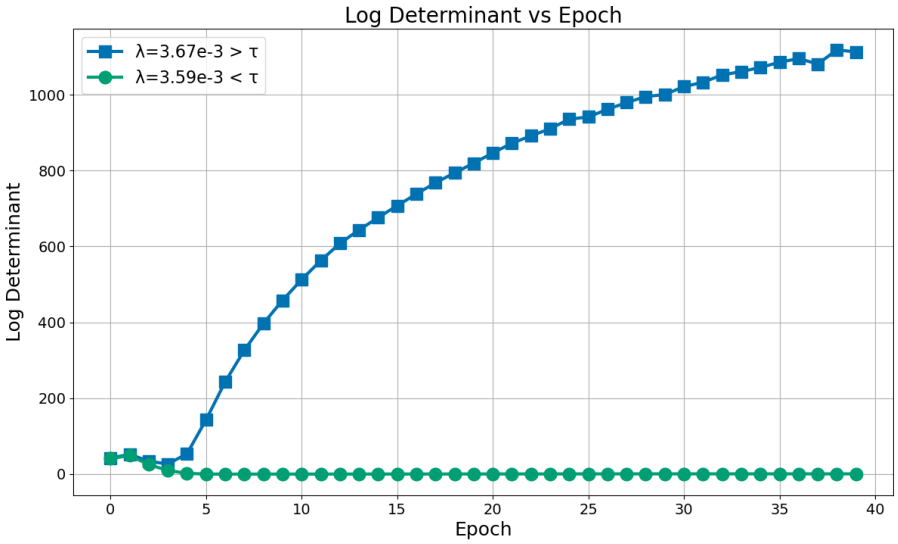


Figure 4.20: Log determinant plots showing the difference between a DHM with $\lambda < \tau$ and one with $\lambda > \tau$ (note that $3.59e-3 < \tau < 3.67e-3$).

and the determinant of the Jacobian of the flow f, J_f, tracks the change in volume of the flow transformation between x and z (see Section 2.2.5.2). We now consider observations of the log determinant made for two 40 epoch training runs made on either side of τ: one with $\lambda = 0.003594$, and one with $\lambda = 0.003672$; see Figure 4.20. The $\lambda < \tau$ model achieves CIFAR-100 and SVHN AUROC scores of 0.745 and 0.946, respectively, while the $\lambda > \tau$ model achieves CIFAR-100 and SVHN AUROC scores of 0.500 and 0.549, respectively. We observe that in the $\lambda > \tau$ model, in contrast to the successful model, $\log det(J_f))$ increases rapidly and considerably during training, up to a final value of 1113, while for the $\lambda < \tau$ model $\log(det(J_f))$ decreases and then remains steady at around 0, with a final value of 0.1209.

This indicates that the rate of change of z is very large with respect to x, as a density for z must be scaled by a large value to conserve total probability when transformed to x. This is reflected clearly in the distributions of the feature vectors learnt by the two models: in the $\lambda < \tau$ case, the validation sample features have a mean of 0.008 with a standard deviation of 0.136, while for the $\lambda > \tau$ model, final sample features have a mean of 0.332 ± 0.184. The effect

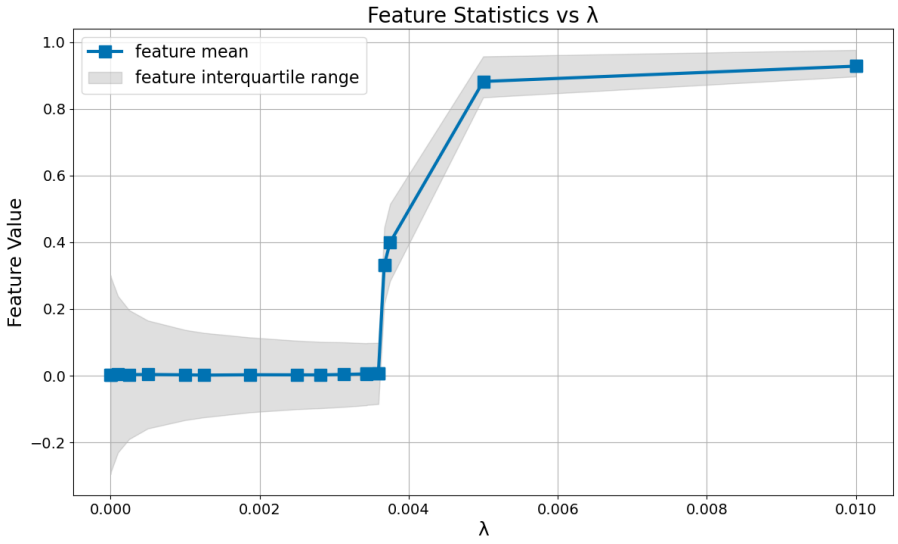


Figure 4.21: Feature mean and interquartile range against λ shows the feature values collapsing for $\lambda > \tau$.

becomes even more pronounced as λ is increased; for example, for $\lambda = 0.1$, we get a sample feature mean of 0.963 ± 0.064. See Figure 4.21 for the distributions of features across λ values. We argue that this occurs because, for $\lambda > \tau$, the classifier component of the DHM is focusing increasingly on concentrating feature vectors into one point in order to maximise $p_Z(z)$; the log determinant must necessarily increase to compensate. OOD detection capabilities collapse as a result, since all inputs are being mapped to the same point and therefore being assigned the same probabilities on average. This is an example of feature collapse, which is discussed in more detail in Chapter 5.

We can demonstrate this phenomenon by directly manipulating feature values. In Figure 4.22, we plot a histogram of all feature values across the entire CIFAR-10 testing set (i.e., for every sample in the testing set, we plot every feature element value in the histogram). Next, we "compress" these features by reducing the feature variance around the mean, which is achieved by setting each feature $x \sim D$ to:

$$x' = \frac{1}{2}(x - \mu_D) + \mu_D, \tag{4.3}$$

where μ_D is the mean value of all feature elements in D. In other words, we

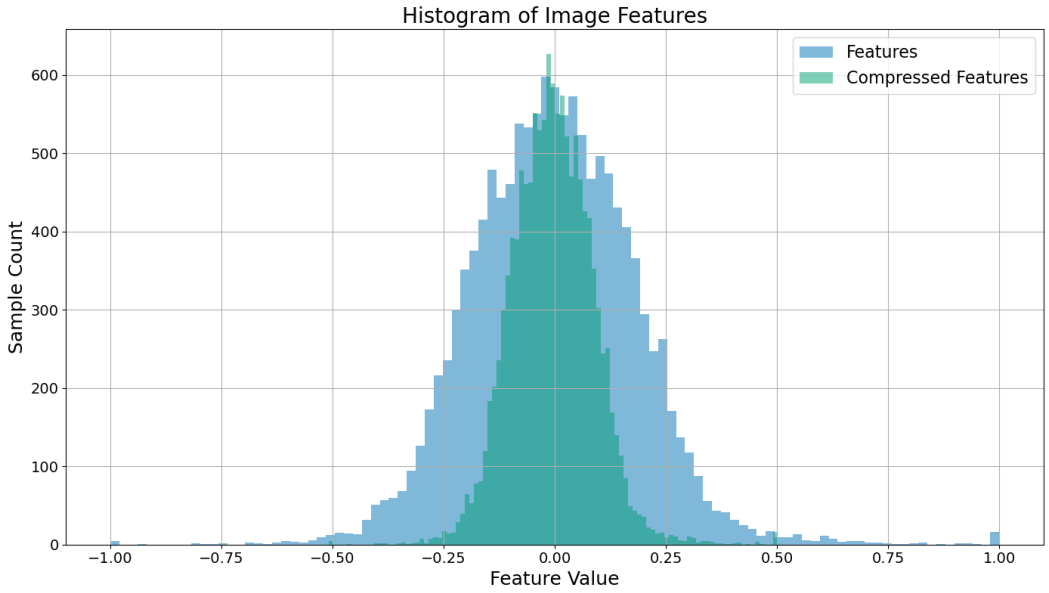


Figure 4.22: Histogram of feature values and compressed feature values for the CIFAR-10 test set in a high-performance DHM.

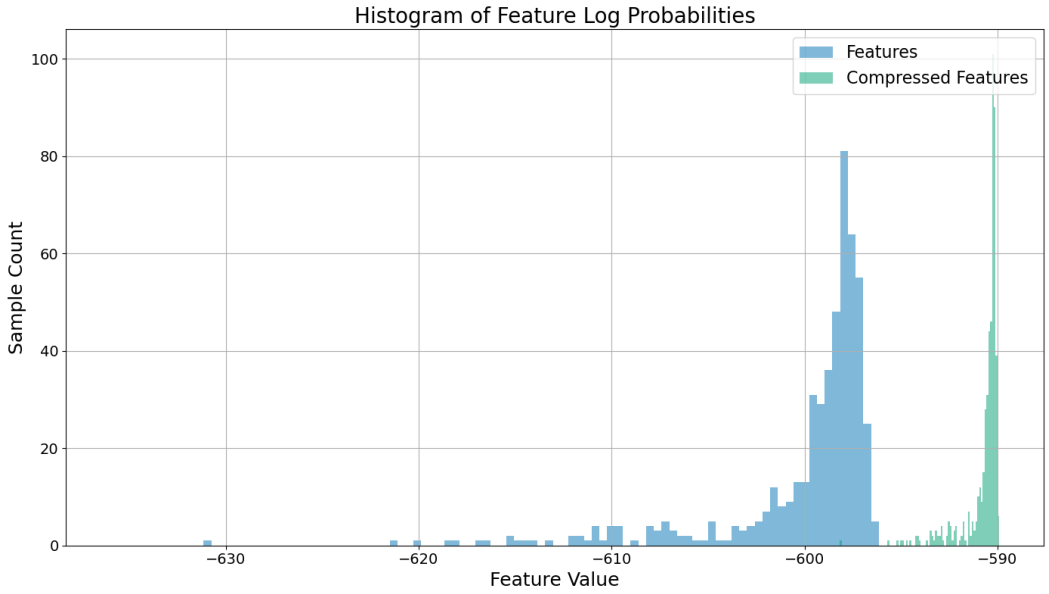


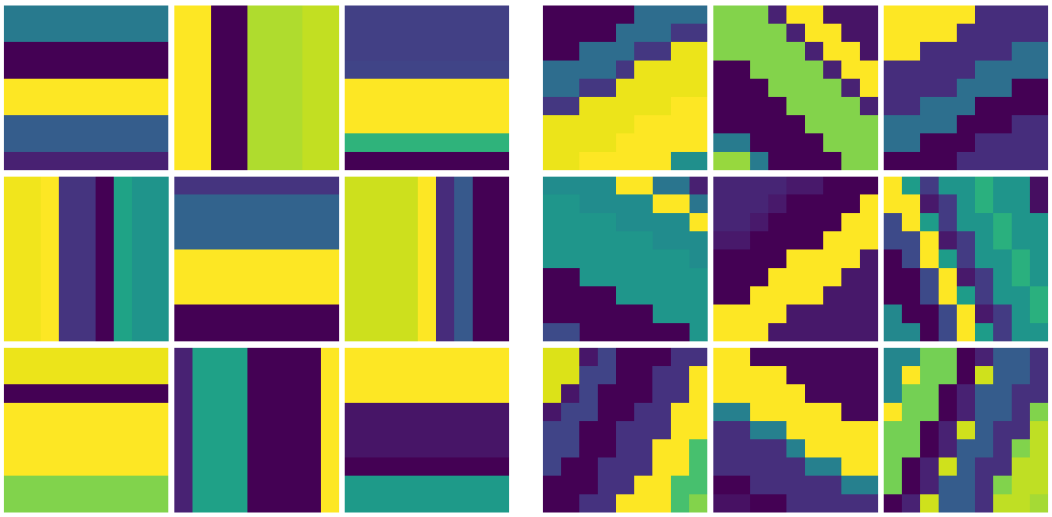
Figure 4.23: Histogram of feature log probabilities for the CIFAR-10 test set features and compressed test set features.

simply contract feature values to be closer to their global mean. The log probabilities resulting from passing these features to the DHM's normalising flow are shown in Figure 4.23, where the compressed features have been assigned distinctly higher probabilities. Intuitively this makes sense: if the features are distributed around some mean, when every feature is closer to the mean then each feature has a higher probability density (assuming a reasonably normal density), which leads to an overall increase in the probability density of the sample.

When fitting a normalising flow directly to real data, this kind of behaviour is not a problem, as the data remains fixed and will not be arbitrarily compressed. However, in the DHM architecture, the feature encoder is capable of doing this, and can achieve a considerable decrease in sample log probability - hence loss - by doing so. By constraining λ to be sufficiently small, we ensure that the final loss decreases at a greater rate in the direction of optimising the feature encoder for classification, but this comes at the simultaneous cost of reducing the strength of the loss signal with which the normalising flow is trained.

Above τ, the change in the loss function from concentrating features into a narrow point - thus maximising $p_Z(z)$ - is greater than the change in loss from training the classifier and fitting the flow model to the resulting features. Furthermore, as we increase λ to $\lambda >> \tau$, even the change in loss from fitting the classifier becomes marginal, at which point the DHM focuses purely on collapsing the features to a point and validation accuracy drops off as well.

We argue that the relatively narrow band of high-performance λ values observed in Figure 4.19 therefore represents the region where the normalising flow still receives sufficient feedback to model the features effectively, but not enough that the normalising flow loss overpowers the classifier loss. This is an undesirable, inherently adversarial trade-off, which suggests that some alternative method of constraining the feature representations might improve the DHM's capability and stability.



(a) (b)

Figure 4.24: Examples of the "stripes" dataset, with horizontal/vertical in-distribution samples and diagonal out-of-distribution samples.

4.3.2 DHM on Stripes Dataset

To investigate the DHM's behaviour, we now consider a synthetic "stripes" image dataset, which allows us to observe how the data is processed at a lower-dimensional, and therefore more comprehensible, scale. Additionally, due to the simple nature of the image dataset, we can construct a manual feature extractor that can clearly distinguish the ID and OOD samples. We find that even in this much simplified scenario, the DHM architecture often struggles to model the in-distribution samples effectively.

4.3.2.1 Stripes Dataset

The purpose of the DHM architecture is to distinguish between high-level semantic concepts, while being insensitive to changes in low-level feature distributions, and the learning scenario should exhibit corresponding properties. At the same time, we would like to have a dataset where the solution to the training objective is intuitively understandable to a human, such that the results can be easily interpreted. Finally, we also need a way to generate out-

-1	0	1
-1	0	1
-1	0	1

Table 4.9: Prewitt kernel for vertical edge detection. The transpose of this kernel gives a horizontal edge detector.

of-distribution samples that have the same low-level pixel statistics as the ID samples.

We therefore created a method for generating 9×9 images of stripes, where each stripe is between 1 and 4 pixels wide, with the widths sampled from a uniform distribution. Each stripe is assigned a value sampled uniformly from a 0-1 range. We define two ID classes: horizontal and vertical stripes. For the OOD samples, we generate diagonal stripes, that are defined in terms of the angle at which the stripes extend from the left side of the image. The angle is sampled uniformly from between 15° to 75° (i.e., $45 \pm 15°$). With a likelihood of 50%, the gradient is inverted to generate stripes going down instead of up. See Figure 4.27 for examples of the generated ID and OOD images.

In this dataset, the two classes - vertical or horizontal stripes - are predicted by local pixel patterns in the images, i.e., the value of any particular pixel is not correlated with the class. At the same time, a convolutional network can be hand-crafted to solve the classification problem: one simply requires two Prewitt kernels (Prewitt et al., 1970), one for vertical and one for horizontal edges. Table 4.9 depicts a vertical edge detecting Prewitt kernel.

The handcrafted feature extractor consists of a single convolutional layer with these two kernels set as the filters. As an activation function, we use the absolute function, since we know that horizontal or vertical edges will create either strongly negative or strongly positive responses when convolved by the appropriate kernel; that is, we only care about the magnitude of the kernel response. We pass the resulting feature map through a max pooling layer to

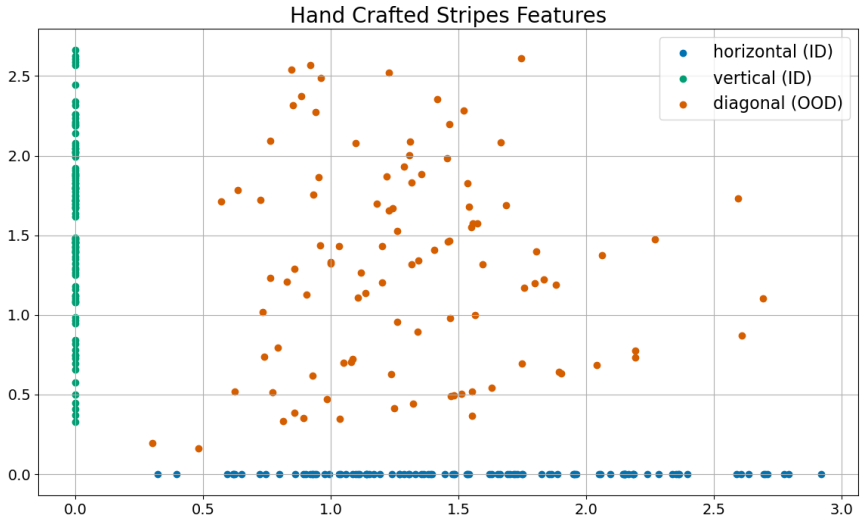


Figure 4.25: Handcrafted features can establish a clear separation between ID and OOD samples.

get the maximum activation of each kernel's feature map and generate a 2D feature in the same manner as the wide Resnet does. A softmax activation over this feature lets us interpret the prediction as a one-hot encoding just as with any typical classifier.

The resulting classifier is perfect on our dataset, and generates a feature space where the OOD samples are clearly separable from the ID samples; see Figure 4.25. In principle, a DHM model should therefore be able to arrive at a similar solution.

4.3.2.2 Results

Hyperparameter Tuning: To construct the DHM we use the previously described handcrafted feature extractor as the DHM's feature encoder, except this time the parameters are randomly initialised and trained as normal. We attach a residual flow with 50 residual flow blocks and a hidden dimension of 32 directly to the feature vector after max pooling, without any further preprocessing steps; note that while reducing the number of residual blocks still allows the normalising flow to attain high performance, it requires more

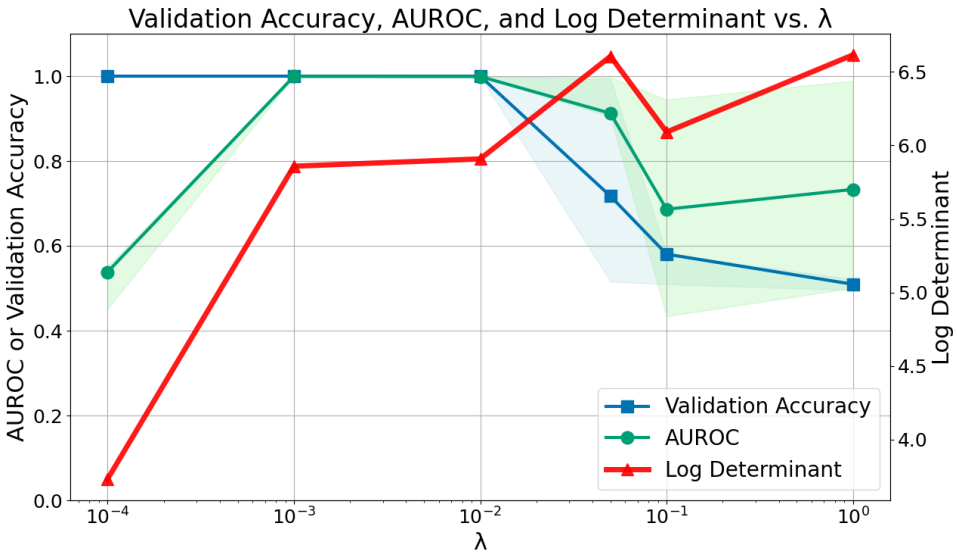


Figure 4.26: Stripes dataset validation accuracy, log determinant, and reference OOD AUROC scores against λ.

epochs of training to do so, and we choose the larger number of residual flows for convenience. We train the flow component with Adam optimisation using a learning rate of $1e-3$ and a weight decay of $1e-5$. The feature encoder is trained with vanilla SGD using a learning rate of 0.01, and a spectral normalisation coefficient of 0.9. We generated 2500 samples from the "stripes" generator for use as the training set consisting of horizontal and vertical stripes only, as well as another 500 images for testing. We also generated 500 diagonal images for the OOD testing set. In these demonstrations, we train the DHM with a batch size of 256 over 160 epochs. We then use the lessons learnt in previous sections to find the optimal λ for training the DHM.

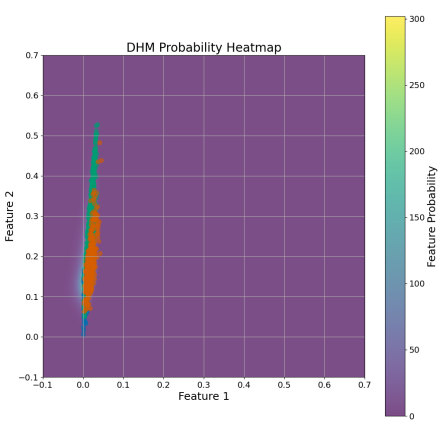
In Section 4.3.1, we argued that above some threshold τ, the feature encoder learns to collapse features to maximise $p_Z(z)$, and that a sudden increase in the final log determinant is a strong indicator for this behaviour. We therefore argued for using the log determinant to identify τ and choose the optimal λ value in the absence of an OOD testing set. Here, we put the theory into practice and performed a search over λ using the log determinant as a measure of success, with the aim of finding the largest λ before the log determinant

explodes.

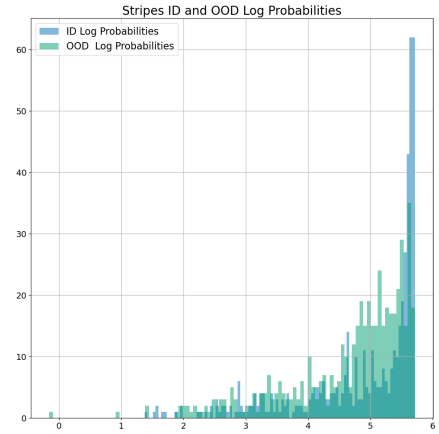
Results: Unlike with the CIFAR-10 dataset, a logarithmic search over λ does not reveal a stark drop in the log determinant, although an overall change in behaviour is still noticeable. Figure 4.26 plots the mean validation accuracy, log determinant, and AUROC scores against λ over 10 runs each. The error bands display the interquartile range. This reveals a marginal drop in the log determinant, as well as a sudden increase in validation accuracy and a reduction in model variance to virtually nothing, at $\lambda \leq 0.01$, with the AUROC scores confirming that high OOD performance begins at this point. AUROC scores vary considerably for $\lambda > \tau$ (in some runs attaining perfect performance) while the loss of validation accuracy is guaranteed. OOD performance also drops off for $\lambda << \tau$, as the flow component of the loss signal becomes too weak for the normalising flow to learn the distribution of the features within 160 epochs. This behaviour matches our observations made in Section 4.3.1.

An inspection of the test set features confirms that the $\lambda > \tau$ model begins to collapse features - at the expense of discriminative capabilities - to maximise the likelihood densities of the samples; see Figures 4.27a and 4.27b. The $\lambda < \tau$ model (we display $\lambda = 0.01$), meanwhile, is able to independently replicate the embeddings of the handcrafted model. This makes modelling the feature space a fairly straightforward task for the normalising flow, which assigns high densities to the two ID classes and distinctly lower likelihoods to the OOD samples, as can be seen in Figure 4.27c.

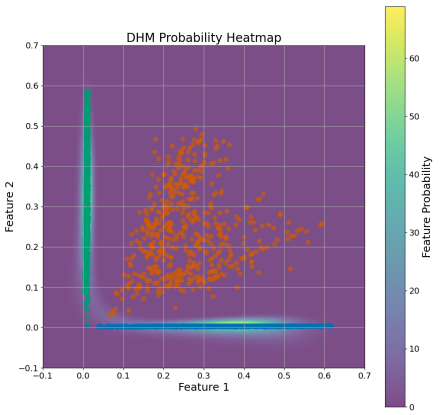
We observe, however, that even in the case of the DHM successfully modelling the ID features, there is noticeable variance between the probability densities assigned to the two classes. For example, at $\lambda = 0.01$, we find there to be an average difference between the mean vertical feature log probability and the mean horizontal feature log probability of 0.2633. Although both features have the same mean log probability of 3.211 over multiple runs *on average*, in any particular run one will have a considerably higher mean log



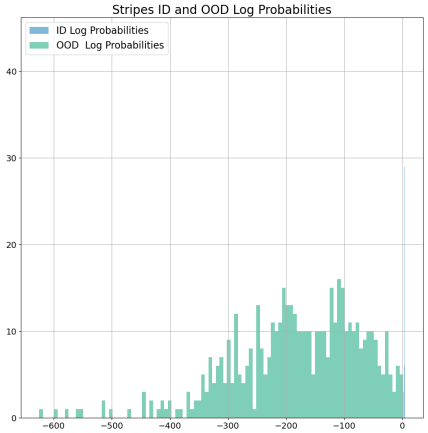
(a) $\lambda = 1$ features



(b) $\lambda = 1$ logprob histogram



(c) $\lambda = 0.01$ features



(d) $\lambda = 0.01$ logprob histogram

Figure 4.27: Examples of the "stripes" dataset, with horizontal/vertical in-distribution samples and diagonal out-of-distribution samples. At $\lambda \leq 0.01$, OOD performance is near perfect. Note that in figure d), the ID histogram is extremely narrow compared to the OOD histogram, as the OOD samples have a far greater range of log density values than the ID samples.

probability than the other. We explore this phenomenon in more detail in Section 5.2.1.

Chapter 5

Bi-Lipschitz Continuity

5.1 Sensitivity and Smoothness

In the previous chapter we identified a fundamental issue with feature extraction in DHMs: the tendency for feature collapse, particularly when the parameter λ that trades off between classifier loss and normalising flow loss is too large, and we showed how the structure of the DHM can in fact incentivise this behaviour. Similar behaviour has been reported in the OOD detection literature as well: Postels et al. (2020) and Van Amersfoort et al. (2020), who investigate uncertainty estimation in feature space and describe a similar phenomenon. Postels et al. (2020) note that, in a standard classifier that must learn to filter out features that are irrelevant to the task, OOD samples that share features with the ID data may end up being indistinguishable in feature space; however they do not train their classifier jointly with their density estimator. Van Amersfoort et al. (2020) train a radial basis function network on the features of a classifier, which relies on distance measures rather than density estimates for uncertainty measurements, but shares a conceptual similarity with the DHM in that the objective function incentivises the accumulation of feature points around the centroids.

They argue that, to avoid this behaviour, one must enforce *sensitivity* on the feature encoder; that is, a change in input produces a change in the feature.

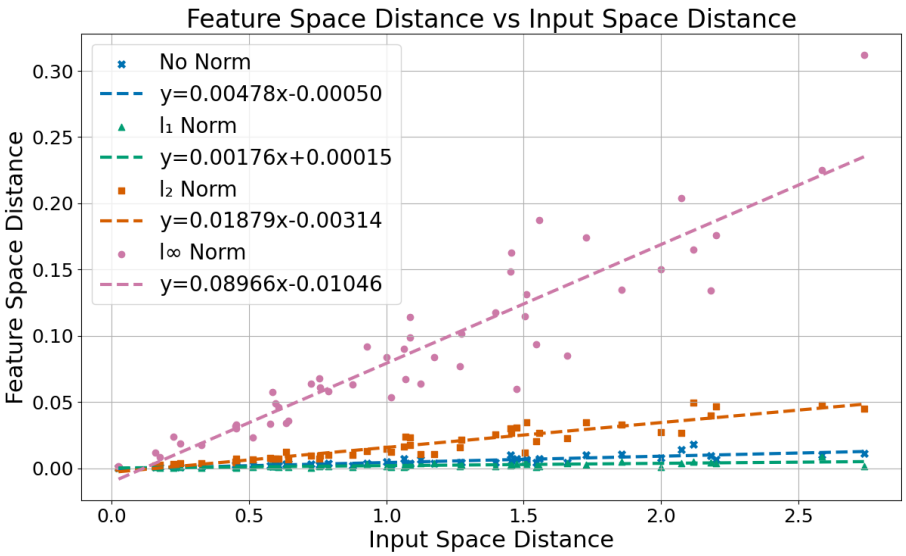


Figure 5.1: Deviations in feature space against deviations in input space, showing that the l_∞ normalised DHM is the most sensitive configuration.

At the same time, to encode information that is necessary to estimate epistemic uncertainty (i.e., uncertainty due to the lack of data in the relevant region of the feature space), it is also important for the feature encoder to roughly preserve distances in the input space. That is, distances in input space should be roughly proportional to distances in feature space. Van Amersfoort et al. (2020) refer to this property as *smoothness*, while J. Liu et al. (2020) describe this property as *distance awareness*.

This view allows us to explain the behaviour observed in Section 4.2.7, for example: by applying random deviations to a test input and measuring the resulting distance deviations in feature space, we can measure the sensitivity of a model directly, which reveals that the l_∞ normalised configuration of the DHM is by far the most sensitive one (see Figure 5.1). The l_∞ normalised DHM therefore performs better because it is more sensitive to changes in the input, thus increasing the separation of OOD samples from ID samples and making their identification a simpler problem.

Formally, then, a feature encoder f that avoids feature collapse and generalises sufficiently well to enable out-of-distribution detection must, first, incur

some minimum rate of change in feature space such that $||f(x+\epsilon)-f(x)|| > 0$ for some very small vector ϵ. In other words, we must have:

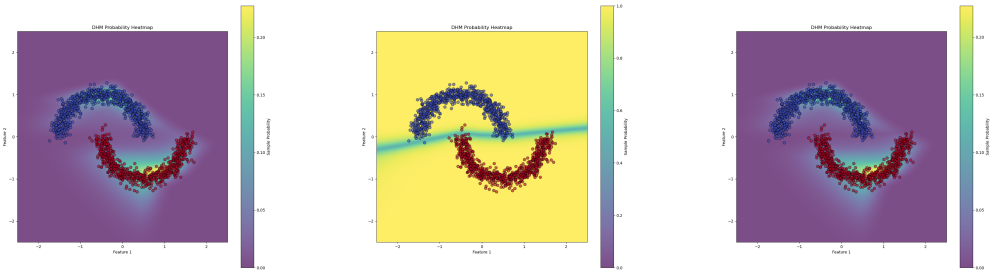
$$\frac{\delta f(x)}{\delta x} > 0 \tag{5.1}$$

This condition enforces sensitivity. Second, there must also be an upper bound to the rate of change, preferably as close as possible to 1. Successfully enforcing these two conditions on a function is referred to as enforcing bi-Lipschitz continuity, as these constraints amount to setting the upper and lower bound of the feature encoder's Lipschitz constant (see Section 2.1.5.2 for a description of the Lipschitz constant).

J. Liu et al. (2020) show that a spectrally normalised residual network will be bi-Lipschitz continuous. Spectral normalisation establishes an upper bound on the Lipschitz constant of a network layer (thus enforcing smoothness), and by applying spectral normalisation to the residual connection of a residual block, one also enforces a lower bound on the residual block's Lipschitz constant (thus enforcing sensitivity); see Appendix A.1 for the proof, which shows that ensuring some upper bound α for the residual connection's Lipschitz constant will constrain the residual block's Lipschitz constant, L_{resblock}, to $(1-\alpha) \leq L_{\text{resblock}} \leq (1+\alpha)$.

Interestingly, although this regularisation technique is applied in DHMs, Cao and Z. Zhang (2022a) justify spectral normalisation via a different approach: noting that the feature vector $h = f(x)$ should approximately reflect the likelihood of x, they argue that by restricting the residual network's Lipschitz constant as much as possible with $\alpha << 1$, we are able to restrict the residual network's Lipschitz constant to be approximately 1, thus preserving the exact distances in input space and presumably therefore also enabling preservation of probability densities.

However, this interpretation does not seem to be reflected in empirical parameter settings, as even Cao and Z. Zhang (2022a) report setting the spectral normalisation coefficient to 6.0 for the wide Resnet architecture, a setting that clearly does not restrict the network's Lipschitz constant to approximately 1.



(a) $p(x)$ (b) $p(y|x)$ (c) $p(y|x)p(x)$

Figure 5.2: Heatmaps of the DHM's epistemic uncertainty $p(x)$, aleatoric uncertainty $p(y|x)$, and full uncertainty $p(y|x)p(x)$ as described by Cao and Z. Zhang (2022a)

In this context, it is worth noting that Smith et al. (2021) argue that a network that is not dimension preserving cannot be distance preserving. The question arises, therefore, whether the DHM really works as intended.

5.2 Non-Volume Preserving Feature Spaces

The limitations on enforcing bi-Lipschitz continuity call into question how effectively the proposed aim of enforcing $p(x) \approx p(h)$ can be achieved. In the next section, we argue that successfully enforcing this condition is at odds with achieving good performance using DHMs.

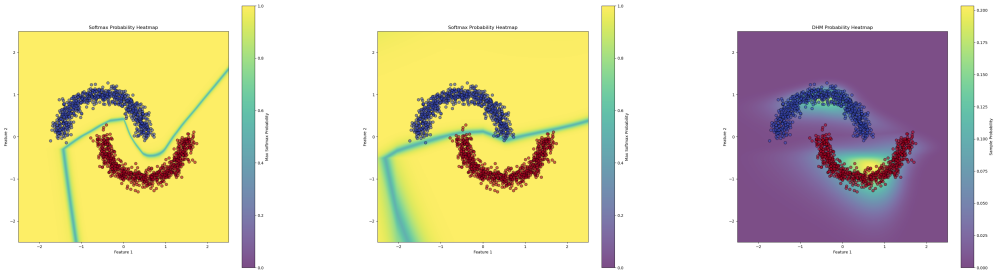
5.2.1 Discrepancy Observed for the Moons Dataset

Training the DHM model on the toy "moons" dataset demonstrates its in-distribution modelling capabilities, but also reveals an informative flaw in the architecture. We design a simple residual network with 6 residual blocks, where each residual connection consists of two fully connected layers, each preceded by a ReLU activation. The first has a hidden dimension of 128 (i.e., maps 2D inputs to 128D activations) and the second layer maps the activations back to a 2D vector. We pick this architecture as it is the one

used by J. Z. Liu et al. (2023) to generate a similar set of plots as depicted in Figure 5.3, discussed below. When spectral normalisation is applied, we apply it to the fully connected layers. We consider both a regular classification model trained using cross-entropy and a DHM, investigating plots of predictive uncertainty for both. In the case of the DHM, we attach a residual flow with 5 residual blocks and a single 128-dimensional layer per residual connection to the penultimate layer of the Resnet.

For the Resnet, we plot a heatmap of the maximum softmax value of the output logits to measure confidence, see Figure 5.3a and 5.3b. For the DHM, we plot the uncertainty as described by Cao and Z. Zhang (2022a) (Figure 5.3c), who factorise the uncertainty as $p(y|x)p(x)$ (see Section 3.6.2.1), where $p(y|x)$ is the softmax score of the maximum logit and $p(x)$ is the probability assigned to the sample by the normalising flow using Equation 2.54. As discussed in Section 3.6.2.2, $p(y|x)$ represents the aleatoric uncertainty of the model, or the uncertainty inherent in the data itself. Meanwhile, $p(x)$ measures the epistemic uncertainty, i.e., the uncertainty due to the lack of data. Our primary interest is in $p(x)$, since this pertains to the model's OOD detection capability, but we also plot the full uncertainty as it might be used in a deployed model. Figure 5.2 shows the resulting heatmaps of a DHM for $p(x)$, $p(y|x)$, and $p(y|x)p(x)$.

The results for the basic Resnets are as expected: the pure Resnet's softmax score heatmap depicts clearly the decision boundary drawn by the network between the two classes (Figure 5.3a), while applying spectral normalisation to the network makes the decision boundary slightly fuzzier (Figure 5.3b). The DHM, meanwhile, assigns high density where the data is located, while assigning low density to the input space both between and beyond the data point regions (see Figure 5.3c). This is desirable behaviour; however, upon closer inspection, it is found that the densities assigned to the two classes are uneven. In this case, samples from the blue (upper moon) class were found to have an average log density (i.e., $\log p(x)$) of -3.06, while the lower moon class



(a) DNN (b) DNN + SN (c) DHM

Figure 5.3: Demonstration of a DHM on the toy "moons" dataset. A simple 6-layer residual net only draws a decision boundary (5.3a); spectral normalisation makes the boundary slightly more conservative (5.3b), while the DHM model assigns high density only to the data itself (5.3c), although unevenly.

had an average log density of -2.45.

The reason for this turns out to be that the Resnet is not sufficiently volume-preserving. Given an input x, let us denote the intermediate feature vector that is passed to the flow as h. The DHM architecture relies on the assumption that $p(x) \approx p(h)$ (see Section 3.2); however, this condition is not necessarily upheld in practice, as can be checked by empirically measuring the probability volume described by a density model, which should be 1.

An approximation of the volume can be obtained by using the method described in Section 4.1.3 to approximate the volume modelled by the DHM. In the case above, we approximate the volume of the probability distributions in the input and feature spaces by submitting the grid of samples to either the DHM or the DHM's normalising flow, respectively. In this instance, on the half-moon data, we find the input space to have a volume of around 0.4 (although the exact value varies widely throughout training and under different configurations), while the feature space maintains a volume of around 1 always, as expected for a normalising flow. This shows that the densities in the original input space do not necessarily correspond to densities in the feature space, and

in modelling the feature space density, the normalising flow does not necessarily capture the densities of the actual inputs. This can be clearly visualised by superimposing on top of the data points their corresponding transformation in the feature space; see Figure 5.4a.

This can be corrected by setting a stricter bound for the spectral normalisation coefficient c; indeed, this is what Cao and Z. Zhang (2022a) recommend to achieve $p(x) \approx p(h)$ (see Section 3.2). Ideally, since the Lipschitz constant of a residual network has a lower bound of 1 (see Section 2.1.5.2), enforcing $c << 1$ for the residual connections will ensure a tight Lipschitz bound, which can preserve the densities of the input space, but in the degenerate case, this can be achieved through a simple identity mapping. For this example problem, setting $c = 0.2$ yields an approximate volume of 0.995 for the input space, but classification accuracy is reduced to 0.91 on the test set, from 1.0.

5.2.2 Demonstration on the Rings Dataset

For more complex problems, the sacrifice in classification accuracy is considerably greater: we demonstrate this with a synthetic dataset consisting of 5 concentric rings, where each ring represents a separate class. Training the DHM on this dataset with $c = 0.9$ yields good classification performance, with a final testing accuracy of 0.94. However, $p(x)$ has clearly been concentrated into the centre ring, while the outer classes are neglected (see Figure 5.5a). Constraining c ensures a stricter correspondence between $p(x)$ and $p(h)$ - resulting in more evenly distributed class probability densities and an input space probability volume of ~ 1 - but at a considerable cost in testing accuracy (see Figure 5.6). The DHM therefore seems to require an undesirable tradeoff between the discriminative power of the feature encoder and the modelling capabilities of the normalising flow.

Note that the condition $p(x) \approx p(h)$ is crucial; without this condition, we cannot enforce the volume preserving property of a normalising flow, and without this restriction, the normalising flow can assign arbitrarily high probability

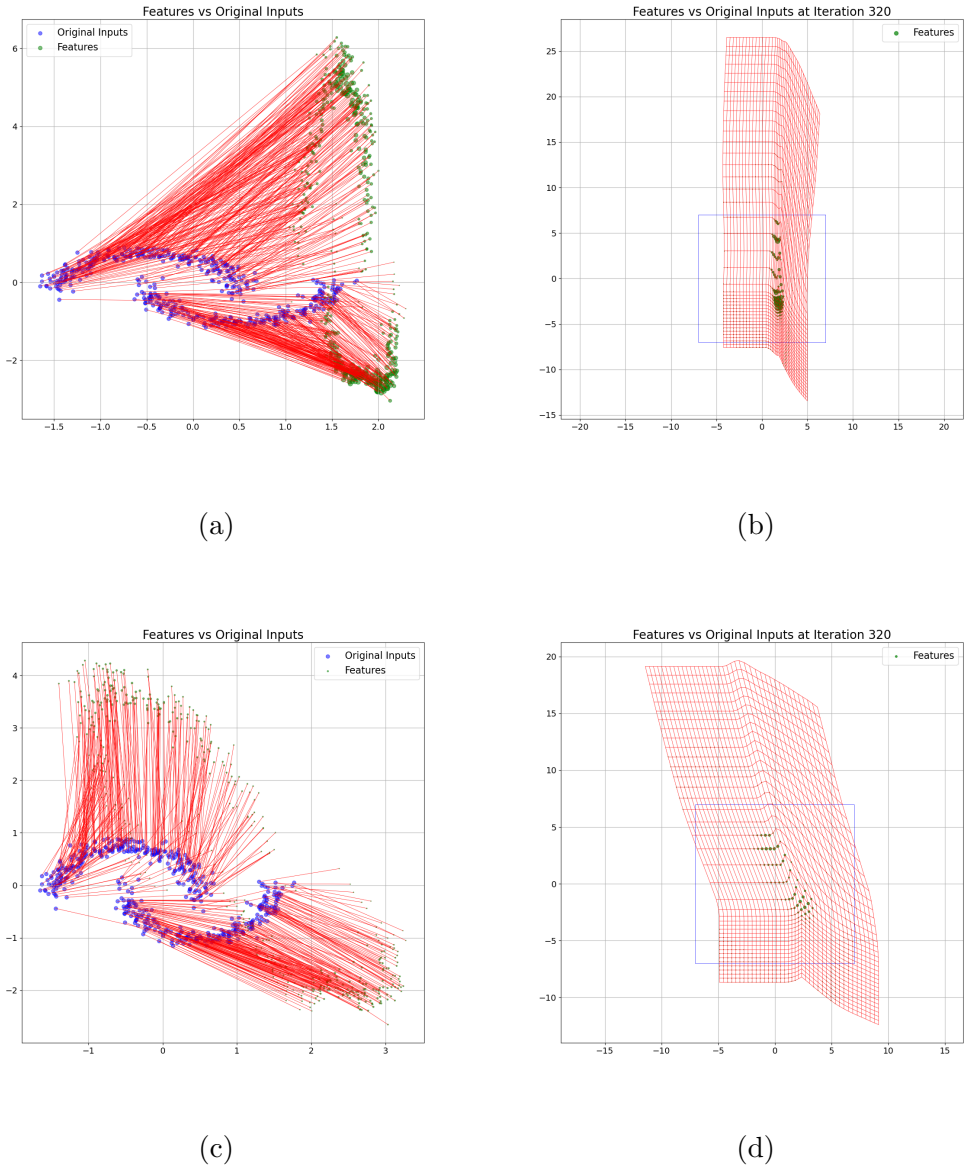
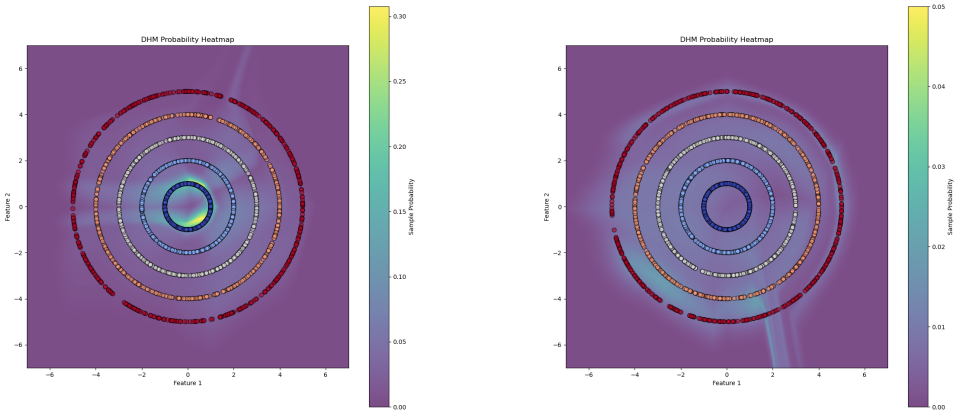


Figure 5.4: In 5.4a, comparing the datapoints with their corresponding positions in the feature space shows that the feature space does not preserve the original density. 5.4b shows how a grid in the original input space (blue square) is mapped into the feature space, which also depicts the densities assigned to each point in the grid; this shows clearly the difference in density in the feature space between the two moons. With the penalty term applied, the points are more evenly spaced out (5.4c), and the transformed space exhibits less extreme contractions (5.4d).



(a) Pure DHM (b) Penalised DHM

Figure 5.5: DHM learnt $p(x)$ for the ring dataset with (5.5a) and without (5.5b) the class deviation penalty term.

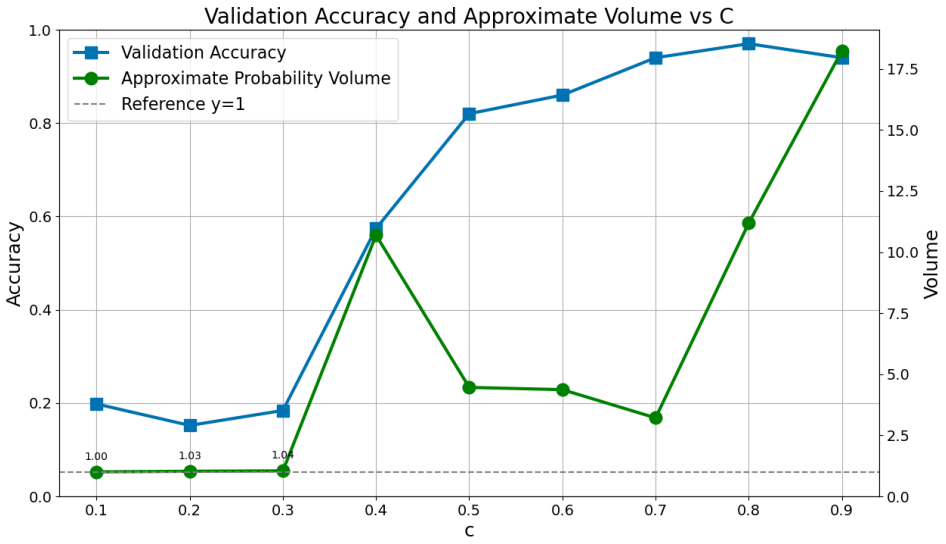


Figure 5.6: Testing accuracy and approximate probability volume on the synthetic rings dataset against the spectral normalisation c coefficient. Volume = 1 is indicated by the dotted line, and the volumes of the first three points are displayed.

densities to the in-distribution features. Additionally, this issue is challenging from a practical standpoint because it is a reasonably "silent" failure; the classifier component's testing accuracy and loss are not affected, since effective classification only requires the classes to be separate in the feature space, and the normalising flow loss is not affected either, since the normalising flow only models the distribution of the feature space. OOD performance, however, may be affected, since the log probabilities assigned to samples do not necessarily correspond to the data density. This motivates the development of an alternative intervention to ensure the feature space densities correspond approximately to the input space densities.

We observe that, in the classification problem, we have some prior knowledge about the densities of the classes; namely, that they should be roughly the same (in the case of a balanced dataset). Even in the case of an unbalanced dataset, we may assume that each class should have an average density in feature space that is proportional to the rate at which it appears in the dataset. This informs the development of a simple penalty term that can added to the existing DHM loss function to encourage more evenly distributed densities among the classes in feature space.

The term involves calculating, for each class k of K classes, the average log density $P_k = \frac{1}{N_k} \log p(x_k)$ of all N_k samples of k in the current mini-batch used for optimisation. Then, we subtract from the overall log density of the batch, $P_{\text{batch}} = \frac{1}{N} \sum_{i=0}^{N} \log p(x_i)$, the difference between the smallest per-class average log density $P_{\min}$ and the largest per-class average log density $P_{\max}$:

$$P_{\text{adjusted}} = P_{\text{batch}} - (P_{\max} - P_{\min}) \tag{5.2}$$

As the loss for the normalising flow is the negative of the training log density, i.e. $L_{\text{nf}} = -P_{\text{batch}}$, we can simply update the normalising flow component of the DHM loss function to $L_{\text{nf}} = -P_{\text{adjusted}}$.

We note that this penalty is a *regularising* term, rather than a strict way to enforce the constraint on the estimated density in input space, i.e., the penalty term does not in any way enforce a probability volume of 1. However, it does

DHM Type	c	Testing Accuracy	$P_{\max} - P_{\min}$
Default	0.1	0.20	1.73
Default	0.9	0.94	2.69
Penalised	0.9	0.90	0.19

Table 5.1: Default DHM vs penalised DHM performances.

assign a cost to the difference in densities between the classes in feature space, and encourages a feature space mapping where the class regions have similar densities, without applying any constraints on the feature space between class regions. We argue that this serves as a potential work-around for the volume-preserving/discriminative power tradeoff observed in the DHM architecture: because the different classes are now modelled with similar densities in the feature space, the feature encoder is less constrained in how it models the rest of the space, allowing it to preserve its discriminative power.

On the rings dataset, this results in a visibly more evenly distributed model for $p(x)$ (see Figure 5.5b). Table 5.1 records the key metrics comparing the new penalisation term against reducing c as a tool for reducing $P_{\max} - P_{\min}$: where reducing c from 0.9 to 0.1 reduces the difference in maximum and minimum per-class average log probability density from 2.69 to 1.73, while also reducing testing accuracy from 0.94 to 0.20. Meanwhile, introducing the penalty term during training has only a minor effect on testing accuracy - a reduction from 0.94 to 0.90 - while reducing per-class average log density difference from 2.69 to 0.19.

5.2.3 Stripes Dataset Revisited

We now apply the penalised loss to a DHM trained on the "stripes" dataset introduced in Section 4.3.2. In that section, we observed that even a well adjusted DHM will learn to assign highly varying densities to the different ID classes, similar to the behaviour we noted in the previous section on the rings

DHM Type	Horizontal Logprob	Vertical Logprob	Mean Absolute Difference
Default	3.211	3.211	0.263
Penalised	1.98	2.01	0.224

Table 5.2: Mean horizontal log probability, vertical log probability, and absolute difference between the standard and penalised DHM, averaged over 10 runs.

dataset.

We observe that applying the penalised loss to the stripes DHM during training only marginally reduces the difference between class log probability densities; see Table 5.2. It is possible that, due to the relatively simple solution to modelling the feature space - since a single convolutional filter is sufficient for extracting one class independently of the other feature - the tendency for the DHM to model unevenly distributed features is relatively minor.

In other words, the standard solution will already map features with roughly similar densities, making the problem simpler than the moons dataset. We suspect that as a result the effect of deviating class densities is relatively minor, and the penalty term seems to have a larger effect on inhibiting class probability densities than on significantly reducing deviation between the classes. This leaves the question of whether the penalty term might be more beneficial for more complex datasets.

5.2.4 Results for CIFAR-10

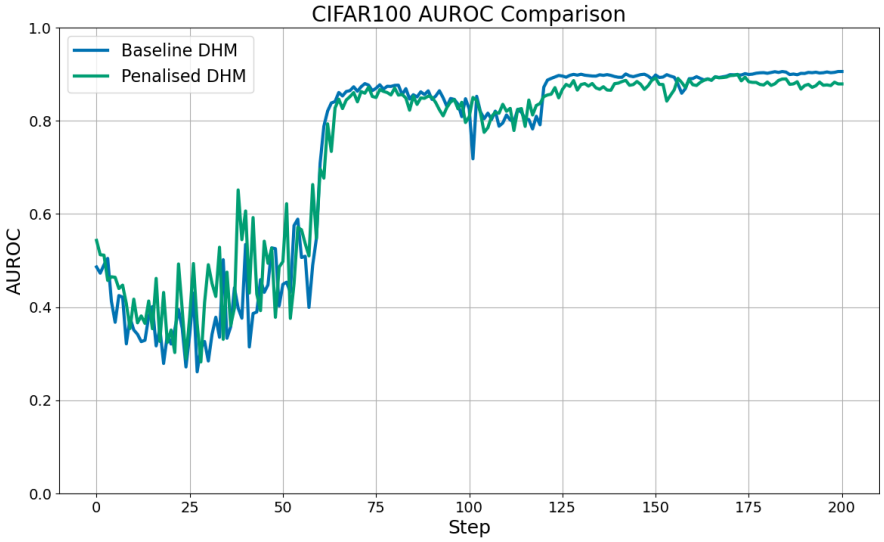
Applying the loss penalty to the DHM when training on CIFAR-10 yields mixed results. Table 5.3 shows that the penalty term operates as intended: after a full training run, the maximum difference between average per-class log densities is 0.424 instead of 3.856, as is the case for the standard DHM (we compare against the DHM results reported in Section 4.2.8). The classes also end up with a slightly higher log density on average, with an average log

Class	**DHM Logprob**	**Penalised DHM Logprob**
Bird	-606.520	-604.537
Car	-604.542	-604.420
Cat	-608.398	-604.365
Deer	-605.314	-604.453
Dog	-607.060	-604.171
Frog	-605.175	-604.214
Horse	-604.922	-604.301
Plane	-607.106	-604.405
Ship	-605.328	-604.175
Truck	-605.620	-604.114
max - min	3.856	0.424

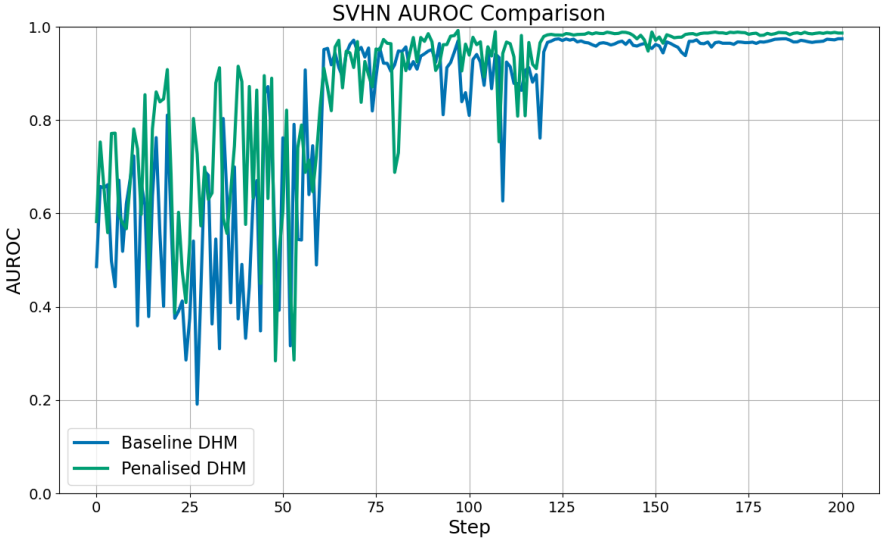
Table 5.3: Comparison of pure and penalised DHM log probability densities for CIFAR-10 (in distribution) classes.

density of -604.316 compared to the -605.999 of the standard model, although the difference is unlikely to be of practical consequence.

In terms of practical results, Figure 5.7 shows that the penalty simultaneously improves SVHN AUROC scores and reduces CIFAR-100 AUROC scores (the exact scores are recorded in Table 5.5), with a CIFAR-100 AUROC score of 0.879 (compared to 0.9058 for the standard DHM) and an SVHN AUROC score of 0.9864 (compared to 0.9743). This suggests that the penalty term sacrifices some near-OOD detection capabilities for a boost to far-OOD detection capabilities. This could be due to the penalty term having a "squashing" effect on in-distribution densities. The effect can be seen in Figure 5.5b for the rings dataset: by enforcing a more even distribution of densities across the class clusters, the density model also becomes more spread out overall. This is unsurprising: since the penalty term is based on the average log density value per class, it will be skewed towards the extreme density values we have

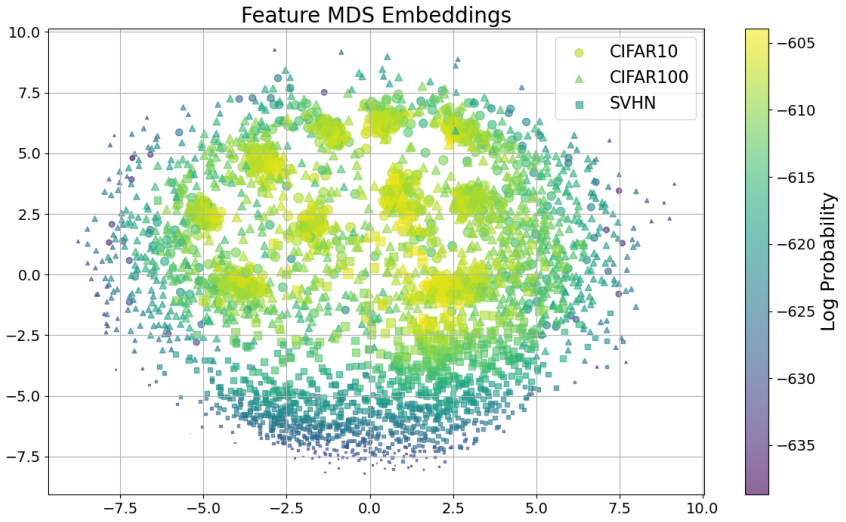


(a) CIFAR-100 AUROC

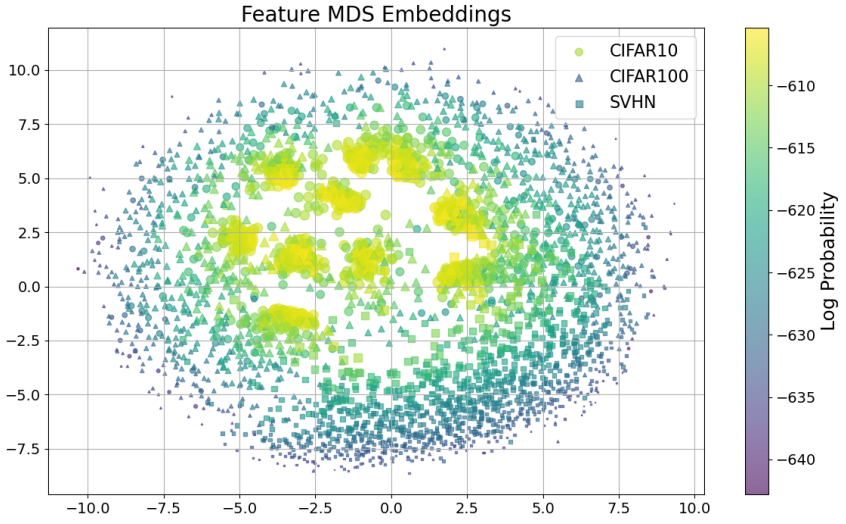


(b) SVHN AUROC

Figure 5.7: AUROC scores vs training epoch of the DHM with and without the loss penalty applied.



(a) Feature space MDS embeddings for the penalised DHM.



(b) Feature space MDS embeddings for the standard DHM.

Figure 5.8

observed the standard DHM to model. The penalty term therefore punishes the formation of these extreme densities, as intended - and it seems to do so by spreading the points in feature space out more evenly across the class cluster. This can be observed in Figure 5.4c, as well as in the MDS embeddings of the penalised DHM (see Figure 5.8a) which - compared to the standard DHM feature embeddings (see Figure 5.8b) - are noticeably more spread out.

The result is a more spread out feature space that is more inclusive of near-OOD samples. Since the penalty term only responds to the in-distribution samples, there is no incentive to adjust the feature space on the immediate boundary of the in-distribution cluster where the near-OOD samples would arise; pushing down the density of the cluster center simultaneously raises the densities slightly around the boundaries, thus bringing a few more near-OOD samples into the in-distribution region at test time.

At the same time, the effect on far-OOD detection appears to be beneficial. In terms of AUTC scores (Table 5.5) and TPR-FPR scores (Table 5.6), the effect of loss penalisation is reduced for SVHN compared to CIFAR-100. In terms of AUTC, SVHN sees an increase by 0.023 compared to the 0.053 increase for the CIFAR-100 dataset. On the TPR-FPR metric, SVHN sees a reduction by 0.051 compared to the reduction by 0.157 recorded for the CIFAR-100 dataset. This indicates that while loss penalisation still draws the SVHN log probabilties closer to the ID probabilities, the effect is considerably reduced compared to the effect on near-OOD CIFAR-10 datapoints.

We expect that the same mechanism that reduced near-OOD scores is at play here as well: by penalising extreme density values and ensuring a more even assignment of probabilities to the ID samples, the spread of ID probabilities is reduced to a narrower band, thus decreasing the amount of overlap with far-OOD probabilities and improving OOD scores. At the same time, the "squashing" effect of the penalty term reduces the relative differences in probability densities between the ID clusters and the surrounding space, thus reducing slightly the separation between the ID and far-OOD set. This

Dataset	**Lower Quartile**	**Upper Quartile**	**Range**
DHM			
CIFAR-10	-606.594	-602.937	3.657
CIFAR-100	-624.297	-612.51	11.787
SVHN	-630.362	-622.545	7.817
Penalised DHM			
CIFAR-10	-604.59	-602.891	1.699
CIFAR-100	-613.163	-605.843	7.32
SVHN	-625.388	-615.339	10.049

Table 5.4: Comparison of ID (CIFAR-10), near-OOD (CIFAR-100), and far-OOD (SVHN) probability inter-quartile ranges.

explains the results we see in Table 5.5, where for the SVHN dataset, both AUROC and AUPR-in scores improve (the narrowing of probability density spread reduces overlap between ID and far-OOD samples) while the AUTC score becomes slightly worse (the separation between ID and far-OOD samples is reduced).

Table 5.4 shows the interquartile ranges of the log probabilities assigned to the datasets by the standard and penalised DHMs, and shows exactly this pattern; the spread of the ID dataset (CIFAR-10) becomes narrower, while the near-OOD dataset (CIFAR-100) gains not only a narrower spread, but moves much closer to the ID dataset in terms of log probabilities. The far-OOD dataset (SVHN), which we would expect to be least affected by the changes to the feature mappings caused by the penalty, actually shows an increased spread of probabilities as the lower quartile fails to shift toward the ID probabilities as much as the upper quartile does.

Overall, the loss penalty appears to regularise the DHM and support the formation of a more even feature space. This yields slight benefits for far-OOD detection at the cost of a slight reduction in near-OOD detection. However,

Model	CIFAR-100			SVHN		
	AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
DHM	**0.9058**	**0.9154**	**0.3679**	0.9743	0.9597	**0.2722**
Penalised DHM	0.879	0.856	0.421	**0.9864**	**0.9695**	0.2954

Table 5.5: Model performance on CIFAR-100 and SVHN datasets.

Model Type	CIFAR-100 TPR-FPR	SVHN TPR-FPR
DHM	0.3366	0.4825
Penalised DHM	0.1793	0.4316

Table 5.6: TPR-FPR scores comparing the penalised DHM with the standard DHM.

given the high ambiguity of some near-OOD samples we observed in Section 4.2.8, this might not necessarily be a downside, depending on the application. We propose that the loss penalty is beneficial for general purpose OOD detection, but might not be suitable for applications where fine semantic distinctions separate the ID and OOD data.

5.3 Using CIFAR-100 and SVHN as In-Distribution Data

In this section, the DHM is demonstrated on CIFAR-100 and SVHN as in-distribution datasets as well, in each case using the remaining two datasets as OOD sets. We use the same architecture and training configuration as was used for the CIFAR-10 dataset for the sake of comparison, with minor adjustments made in the case of CIFAR-100 to account for the larger number of classes.

Dataset	Metric	DHM	Penalised DHM	Softmax Baseline	SNGP
CIFAR-100	Validation Accuracy	0.8065	0.8024	0.8065	0.799
CIFAR-10	AUROC ↑	0.7068	0.7085	**0.8332**	0.798
	AUPR-in ↑	0.7356	0.7105	0.7512	**0.923**
	AUTC ↓	0.4396	0.4436	**0.3794**	-
SVHN	AUROC ↑	0.6226	0.7293	0.8008	**0.846**
	AUPR-in ↑	0.4729	0.5986	0.4935	**0.801**
	AUTC ↓	0.4724	0.4389	**0.4038**	-

Table 5.7: Results of the CIFAR-100 models; three DHM variants are compared against the SNGP architecture described by J. Liu et al. (2020).

5.3.1 CIFAR-100:

For the CIFAR-100 dataset some adjustments were made to the model architecture and training regime. The DHM now has a final classifier layer of size 100 instead of 10 to account for the larger number of classes, leading to a slight increase in parameters. The training regime is modified slightly to adopt methods commonly described in the literature for training WRN models on CIFAR-100, which are mostly based on the training regime described by Zagoruyko and Komodakis (2016) in the original WRN paper. This entails setting the WRN learning rate to 0.1 (instead of 0.05 described by Cao and Z. Zhang (2022a)), and the batch size is reduced to 128, although we found that keeping the initial learning rate at 0.05 actually improved final validation accuracy marginally.

When applying the penalty loss to the CIFAR-100 DHM, we observed a potential shortcoming with the measure: since it requires the mean log density to be measured for every class each batch, for small batches with a large number of classes it is possible that one or several classes have no samples at all. In this case we simply ignore the missing class, and calculate the difference from the counted classes. Assuming the classes are equally likely to be present, the difference calculated on average per batch will still correspond to the true difference between class densities.

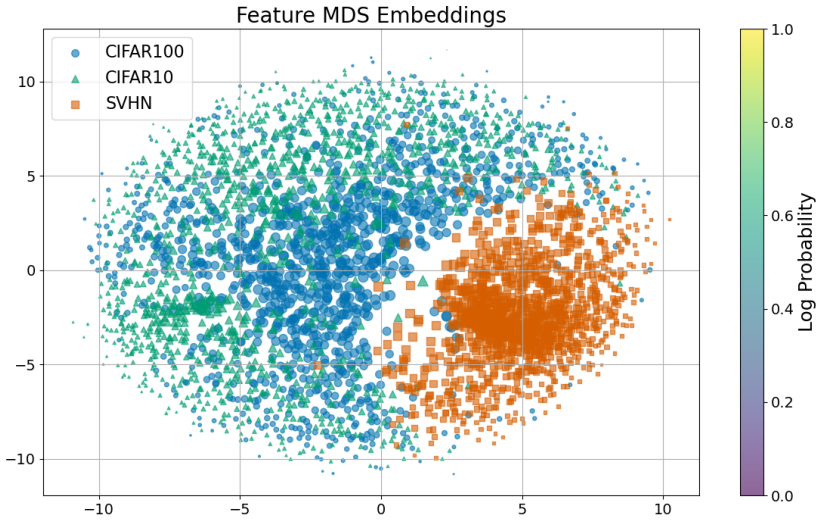


Figure 5.9: CIFAR-100 trained DHM feature embeddings, showing a clear separation of SVHN as an OOD dataset but a noticeably poor density model that often fails to recognise it as such.

Furthermore, we found that λ had to be increased to 0.001 from the default 0.000375 found for the CIFAR-10 dataset, otherwise the normalising flow failed to model the feature space. Performance was overall poorer; see Table 5.7. While the validation accuracy of 0.8065 is commonly reported in the literature for WRN-28-10 models (this result is also reported by Zagoruyko and Komodakis (2016)), OOD performance falls behind even the DHM's own softmax scores. J. Liu et al. (2020) report results for a comparable model, a spectrally normalised WRN-28-10 model that uses a Gaussian process to model feature densities, showing that the DHM model seems to be falling well behind what ought to be possible.

An inspection of the feature space (see Figure 5.9) shows that the SVHN and to a lesser extent the CIFAR-10 features are indeed being separated quite effectively from the CIFAR-100 features. This indicates that the feature mapping is quite well-behaved, that is, sensitivity is being enforced by spectral normalisation and feature collapse due to the feature encoder is not an issue. Instead, it appears the normalising flow is failing to model the feature space, as

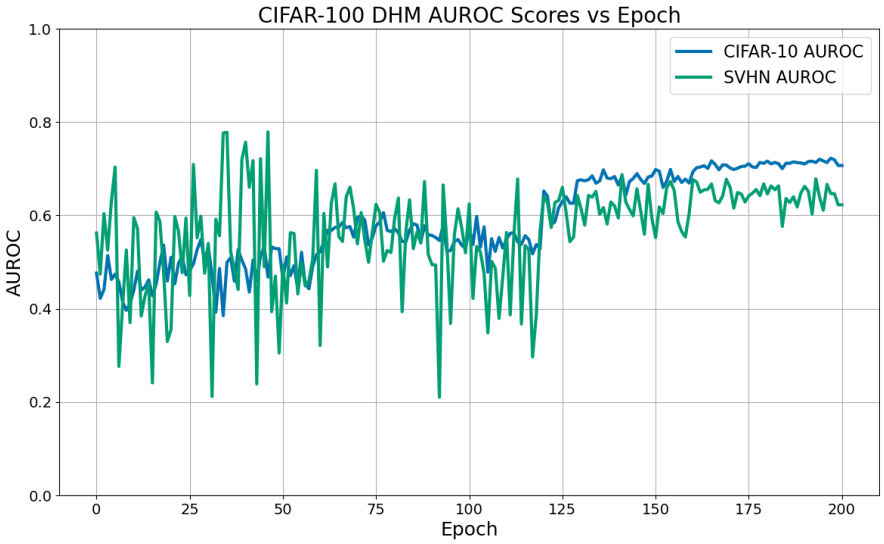


Figure 5.10: CIFAR-10 and SVHN AUROC scores for the DHM trained on CIFAR-100 show limited improvement.

evidenced in part by the need to increase λ. Above 0.001, however, the DHM exceeds τ (the λ threshold described in Section 4.3.1) and begins collapsing features for the normalising flow again. At the same time, AUROC scores do not seem to be increasing further, suggesting limited benefits to longer training; see Figure 5.10.

We suspect that the increased complexity of the feature space - that is, 100 small clusters as opposed to 10 large clusters - contributes to the difficulty the normalising flow is having, especially considering the high dimensionality of the space and the relatively low number of samples per class. It is possible that a considerably longer training regime might yield further improvements. The exact reason for why the DHM fares poorly on the CIFAR-100 dataset remains unclear however, and could be a fruitful avenue for future investigation.

5.3.2 SVHN

For the SVHN dataset, we initially adopted the training regime described by Zagoruyko and Komodakis (2016) for training their WRN-28-10 architecture on SVHN, as Cao and Z. Zhang (2022a) seem to directly adopt the training

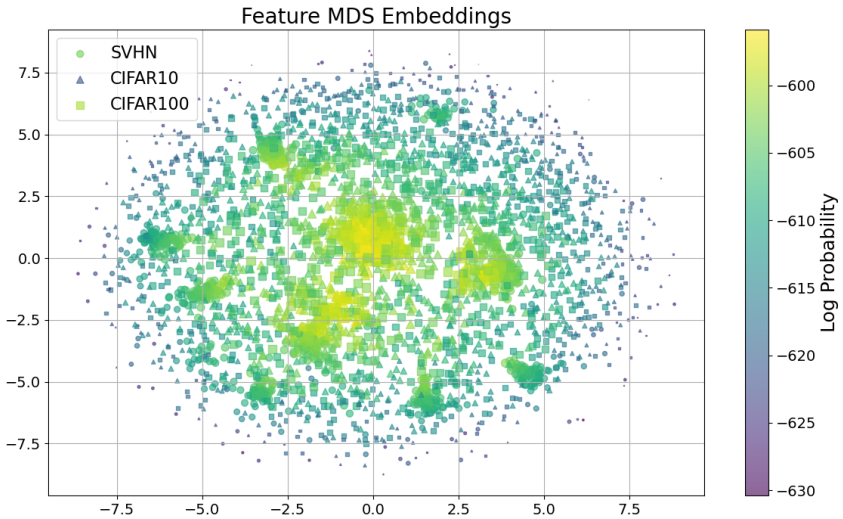
Model	Validation Accuracy	CIFAR-10			CIFAR-100		
		AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
DHM	**0.969**	**0.9331**	**0.9709**	**0.3582**	0.9273	**0.9676**	**0.3614**
Penalised DHM	**0.970**	0.8749	0.8902	0.4203	0.8672	0.8851	0.4292
Softmax Baseline	**0.969**	**0.9315**	0.9430	0.4147	**0.9282**	0.9408	0.4146

Table 5.8: Results of the SVHN models.

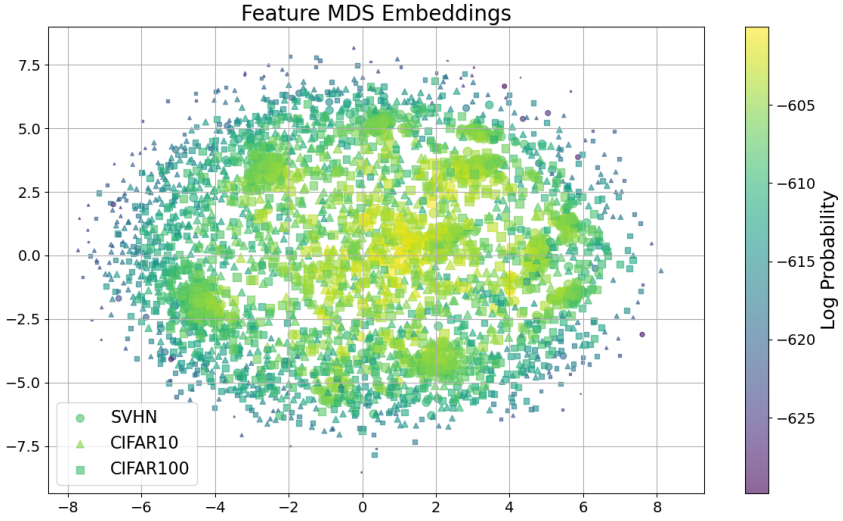
regimes described there for CIFAR-10 training, and do not report results for using SVHN as the in-distribution set. SVHN images are not preprocessed, following Zagoruyko and Komodakis (2016), aside from being scaled from a 0-1 range. We still include the addition of uniform noise as described in Section 3.6.2 to de-quantise the data.

The training regime described by Zagoruyko and Komodakis (2016) for SVHN involved reduced the WRN learning rate to 0.01, training for 160 epochs, and dropping the WRN learning rate by 0.1 at 80 and 120 epochs. However, we found that training for longer did not have any negative effects on final validation accuracy, and improved OOD performance slightly, likely due to the normalising flow having more time to fit the features. We therefore kept the default CIFAR-10 training regime of training for 200 epochs, with a learning rate drop of 0.2 at 60, 120, and 160 epochs, and an initial learning rate of 0.005. The parameters for normalising flow training were left unchanged, and the DHM architecture itself was kept the same for the benefit of comparison.

The results are shown in Table 5.8, which shows results for the DHM, the penalised DHM, and the softmax baseline. Compared to the softmax baseline, the DHM does not confer a meaningful advantage in terms of AUROC scores, although a reduced AUTC score for the DHM indicates that overall the ID and OOD sets have been slightly more effectively separated by the DHM. Penalisation confers no significant advantage in this case, and in fact reduces performance across the board; the advantage gained in far-OOD detection does not compensate for the reduction in ID class densities in this case. Figure 5.11



(a)



(b)

Figure 5.11: Embeddings of the default (a) and penalised (b) DHM on SVHN, showing more discriminative densities for the default model.

clearly shows more ambiguous, spread out class clusters under the penalised loss.

We take this to indicate that the feature mapping learnt by the feature encoder has limited distance-preserving properties, resulting in far-OOD samples still being mapped fairly closely to the ID samples. As a result, the penalty loss draws far-OOD samples into the ID clusters as well and measured OOD performance is overall low. In principle, this should be solvable by means of a stricter spectral normalisation upper bound c, although we found that stricter c constraints began reducing discriminative performance, as has been shown to be typical when c is restricted too aggressively.

5.4 Case Study: Stink Bugs

In this section, we look at applying the DHM architecture to a real-world problem, the identification of out-of-distribution stink bug species, as outlined in Section 3.7.3. In that section, we described the collection of a dataset of images of various stink bugs, with 4 species endemic to New Zealand and 2 potential pests that are treated as OOD classes. We also noted that the classes were fairly unbalanced due to the different rates at which these species were photographed and submitted to iNaturalist, and begin by considering different strategies for handling the data imbalance.

5.4.1 Data Imbalance

We consider two approaches to handling the class imbalance. The first is to keep the dataset as is, and compensate for the class imbalance by weighing the categorical cross-entropy loss function appropriately to emphasise the less frequent classes. Note that the most populous in-distribution class, *Nezara viridula*, has 14550 samples, compared to the 754 samples of the smallest class, *Dictyotus caenosus*; the smallest class makes up only 3.9% of the dataset, a considerable imbalance.

The alternative approach is to restrict the maximum number of samples per class, thus ensuring a more even distribution between classes at the cost of reducing the total number of training samples. Since the majority class is so much larger than the others (see Table 3.4 for the instance counts per class), making up 75% of the entire dataset, we found that a considerable amount could be left out without adversely affecting the performance of the classifier. We expect that this is because the benefit gained from more even class representation does not incur a significant cost to the majority class classification rate until the data restriction is considerable. We settled upon taking a maximum of 5000 samples per class, which in this case meant only reducing the majority class. Loss weighting was not found to be beneficial for final performance in this case, and was left out.

5.4.2 Training Regime

The training regime was adjusted from the CIFAR-10 training regime with the goal of maximising validation accuracy. We adopted the same learning rates and weight decay parameters, and trained as per usual with SGD optimisation for the feature extractor and Adam optimisation for the normalising flow. The Adam parameters were left unchanged. However, as the network took much longer to converge - there appears to be more aleatoric uncertainty between the classes - we pushed the learning rate drops back to later epochs.

We found the model to have effectively converged after 140 epochs of training, with learning rate drops at 80 and 130 epochs, and a batch size of 256. Further, though marginal, improvements could be gained by training longer for 200 epochs, which we did to obtain the final model. As has been extensively explored in Sections 4.2.5 and 4.3.1, the DHM proves to be quite sensitive to the loss parameter λ and the spectral normalisation coefficient c, which must be searched for. We have previously suggested using shorter training runs for carrying out these searches, in the interests of saving time, as we found these shorter runs to be effective for approximately identifying the correct regions

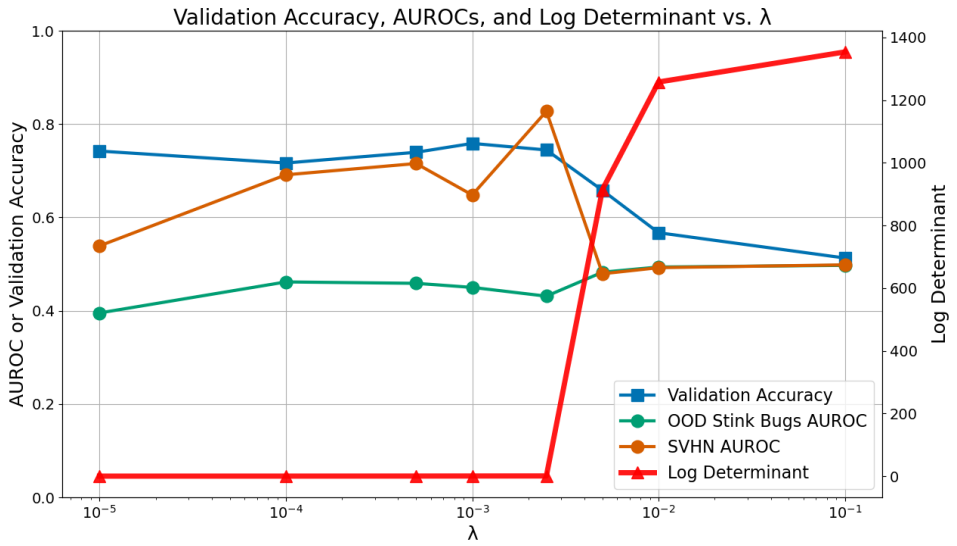


Figure 5.12: Results of the λ search on the stink bugs dataset reveals $0.0025 < \tau < 0.005$.

for the target parameters (see Section 4.3.1).

In the case of the stink bugs dataset, we found that training for 60 epochs with a learning rate drop after 40 epochs was sufficient for approximating final performance, and this training regime was used for initial broad searches.

5.4.3 λ Search

We began by carrying out a search over λ. Starting with a logarithmic search from $\lambda = 0.1$ to $\lambda = 1e-5$, the final log determinant value suggested that the optimal λ sits between 0.001 and 0.01. Two further samples indicated that the threshold τ sits between 0.0025 and 0.005; see Figure 5.12. We generally found the model to struggle to achieve high OOD performance, so for the shorter 60 epoch training runs the stink bug OOD AUROC scores do not reveal any increase in performance; however, using SVHN as a far-OOD dataset, we can see that OOD performance is boosted along with the drop in the log determinant and the considerable increase in validation accuracy, all indications that for $\lambda \leq 0.0025$ the DHM is forming a well-behaved feature space. Empirically, over longer runs we found $\lambda = 0.001$ to yield slightly higher near-OOD

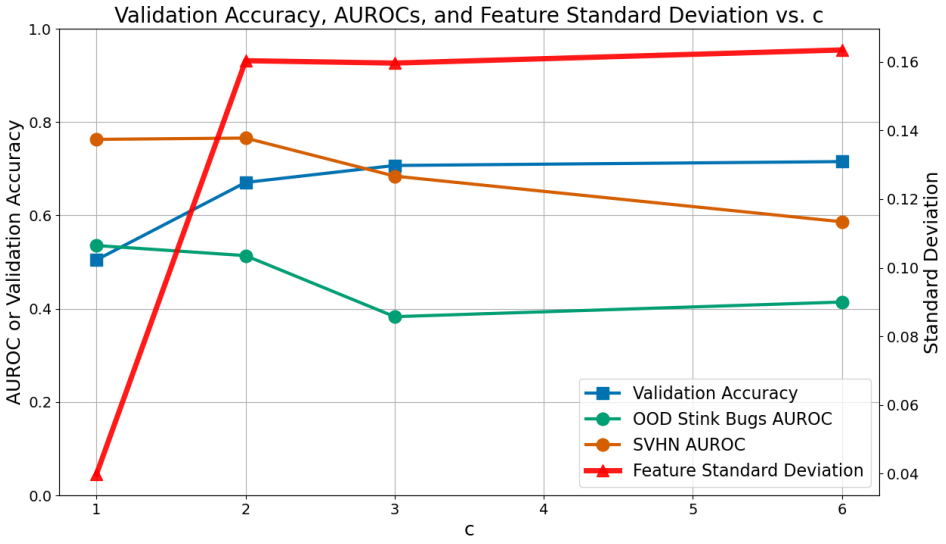


Figure 5.13: Results of the c search on the stink bugs shows $c \geq 2$ is ideal.

performance than $\lambda = 0.0025$, which produces more variable performance.

5.4.4 c Search

Previously we found the DHM to be fairly insensitive to c provided it is above some threshold below which the feature encoder loses discriminative power; see Section 4.2.5. There, we found feature standard deviation to be indicative of OOD performance, in that the collapse of feature variance below some threshold coefficient indicates that the feature encoder is no longer modelling a descriptive feature space. A brief search over c revealed that the encoder loses its power at $c < 2$ (see Figure 5.13), while AUROC scores improve from $c < 3$. We therefore pick $c = 2$ for the spectral normalisation coefficient.

5.4.5 Final Training Configuration and Results

For the full training run, we increased the number of training epochs to 200, with the learning rate dropped by 0.2 at 80 and 130 epochs of training. The increased number of epochs has no impact on final validation accuracy, but slightly improves OOD scores; presumably the extra training time allows the normalising flow to model the feature space more accurately. The results are

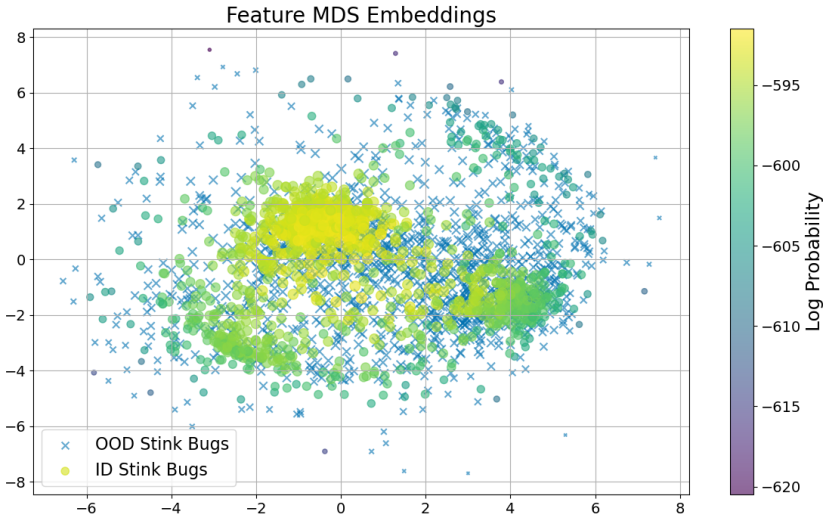
Dataset	Metric	DHM	Penalised DHM	Softmax Baseline
ID Stink Bugs	Accuracy	0.8108	0.7962	0.8108
OOD Stink Bugs	AUROC	0.6031	0.4719	**0.7602**
	AUPR-in	0.2171	0.1468	**0.3243**
	AUTC	0.4836	0.5005	**0.4567**
	TPR-FPR	0.0477	-0.0027	**0.0888**
CIFAR-100	AUROC	0.7534	**0.784**	0.7625
	AUPR-in	**0.5113**	0.408	0.2939
	AUTC	**0.4467**	0.4689	0.4529
	TPR-FPR	**0.1552**	0.0797	0.0994
SVHN	AUROC	0.9038	**0.9207**	0.8285
	AUPR-in	**0.6094**	0.6021	0.1967
	AUTC	**0.3884**	0.4437	0.4223
	TPR-FPR	**0.2766**	0.1253	0.1492

Table 5.9: Comparison of different models on three datasets across four metrics.

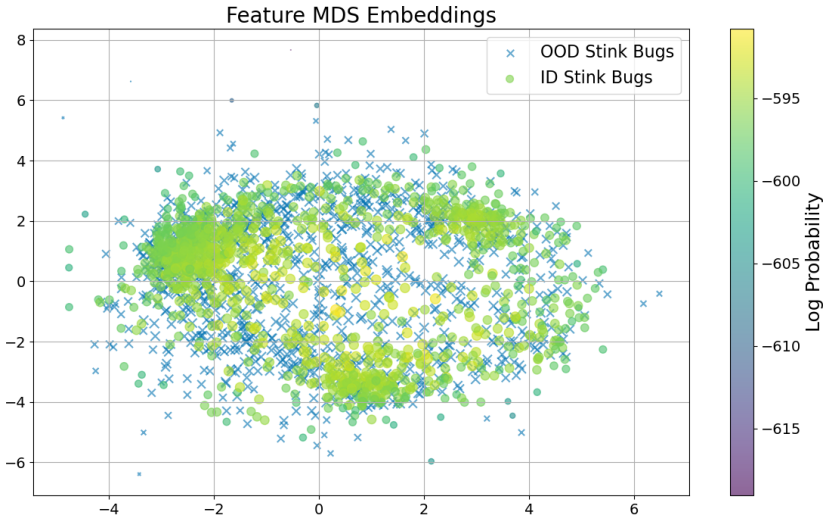
recorded in Table 5.9.

We attain a maximum validation accuracy of 0.8108, with an AUROC score of 0.6031 for the OOD stink bugs dataset. The DHM's softmax scores beat this with an AUROC of 0.7602. Applying the loss penalty dramatically reduces OOD stink bug AUROC scores, although once again it improves slightly AUROC scores for far-OOD datasets.

Inspections of the feature space (see Figure 5.14) show that the OOD samples heavily overlap with the ID stink bug species, which could be explained by a poorly regularised feature space with limited distance-preserving properties. However, the relatively high performance on far-OOD datasets, and the fact that the penalty term still succeeded in improving far-OOD detection performance (outperforming the softmax baseline as well), indicates that the



(a)



(b)

Figure 5.14: Embeddings of the default (a) and penalised (b) DHM on the stink bugs dataset. Far-OOD datasets (CIFAR-100 and SVHN) have been omitted for clarity; the model struggled to develop a feature map that adequately separates ID from OOD stink bugs.

feature extractor is pulling the far-OOD samples into new regions with reasonable success. Furthermore, the considerable drop in near-OOD detection when the loss penalty is applied indicates that a very large portion of the OOD stink bugs are included in the ID class clusters once the densities are evened out. Based on the arguments we made in Section 5.2.4 for why the penalty term reduces near-OOD detection performance, this suggests that most OOD stink bug samples are very close to the ID feature space clusters, but not in the same region, as would be the case in the event of feature collapse.

This suggests that the cause for poor performance may be due to aleatoric uncertainty, i.e., the OOD stink bugs share most of the features that the classifier learnt to discriminate between the in-distribution classes, making the distinction of OOD bugs a difficult task given only the ID samples. Nevertheless, the higher AUROC scores obtained by the softmax probabilities indicates there is still room for improvement in the normalising flow's modelling of the feature space.

5.4.6 Conclusions

The limited performance attained on the stink bugs dataset shows the DHM to struggle with fine-grained near-OOD detection. While the stink bugs problem is inherently more challenging, the comparatively high performance attained by the DHM on CIFAR-10 vs CIFAR-100 and SVHN datasets makes the case for using more real-world datasets in evaluating OOD detection methods. We note that while the principle behind OOD detection is to ignore low-level covariate shifts in the data in favour of focusing purely on semantic shifts, those same covariate shifts may be indicative of fine-grained semantic shifts.

Again we return to the fact that an effective OOD detector *must* be sensitive to shifts in x, while at the same time reducing the data to lower dimensional abstractions to distinguish semantic concepts. Satisfying both of these conditions with a straightforward dimension-reducing feature encoder could be seen as a fundamentally contradictory requirement, forcing the practitioner to

constantly balance the trade-off between sensitivity and smoothness; between an awareness of covariate shifts, and of semantic shifts. Pushing the boundaries of OOD detection to greater levels of performance might benefit from the development of novel architectures that side-step this trade-off.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this work we have explored in depth the deep hybrid model framework proposed by Cao and Z. Zhang (2022a) in an effort to replicate their remarkable findings. While we were not able to replicate their 100% OOD scores for CIFAR-10 against CIFAR-100 and SVHN, and in fact cast doubt on the feasibility of such performance, the DHM nevertheless shows itself to be a capable out-of-distribution detector given careful fine-tuning. The requirements behind such finetuning we explored extensively, and we present an experimental methodology that enables fairly streamlined training of the DHM.

These results are particularly interesting given the findings by Nalisnick et al. (2018) that normalising flows fail completely on OOD tasks when trained directly on images. This shows there is promise in the concept of training normalising flows on higher-level features.

At the same time, the sensitivity of the DHM to how finely-tuned the hyperparameters are can make it a challenging architecture to work with. We investigated the source of this sensitivity, and found that the definition of the DHM fundamentally puts the classifier and the normalising flow in an adversarial relationship.

We also found that one of the key conditions Cao and Z. Zhang (2022a)

argued for in ensuring the functionality of the DHM, the assumption that $p(x) \approx p(h)$, is not effectively upheld. On the one hand, the loosening of this restriction enables the collapse of features, resulting in high density scores at the cost of classification accuracy. On the other hand, attempting to enforce this condition more strictly reduces the representational capacity of the classifier, once again reducing performance.

We argued for a penalty term that assumes $p(x) \neq p(h)$ and discourages feature collapse through direct intervention on the normalising flow densities, which works as expected but yields only limited benefits for far-OOD detection. Fundamentally, the DHM has an unstable feature space by construction. However, with careful finetuning, we were able to show competitive results on the CIFAR-10 dataset when using CIFAR-100 and SVHN as OOD datasets. These results beat standard softmax approaches to OOD detection, showing that there is some promise to the underlying principle of modelling high-level features with a normalising flow.

Finally, we demonstrated the DHM on a real-world dataset with a particularly challenging OOD problem, where performance was generally poor. This suggests in part that real-world datasets are an important component of evaluation OOD detection systems, as standard benchmarks may be too clean for reliable indications of real-world performance.

6.2 Future Work

Fundamentally, the key issue we found with the DHM was with the formation of the feature space. The joint training configuration, while bringing some benefits, also makes the DHM unstable. This suggests that alternative structural approaches, which do not put classification and density estimation in an adversarial position, might yield more stable results.

The tendency for feature collapse also suggests that a better method is needed for forming the feature space. The DHM architecture was motivated

by a need for high level features that effectively represent semantic concepts we wish to distinguish, and ultimately ran into issues with the model resorting to trivial collapsed representations. We note that this reflects a similar issue common to unsupervised learning in general, where the need to prevent feature collapse requires interventions that interfere with downstream tasks, as described by Assran et al. (2023) for example. Assran et al. (2023) and a conceptually similar work by Caron et al. (2021) recently achieved considerable success in training self-supervised embeddings, by implementing methods that prevent feature collapse. Future work may look at adopting some of the techniques developed in this space to produce more well-behaved feature spaces for the normalising flow to model.

Overall, the results show that while there is still considerable room for improvement in the area of OOD detection, there is also much promise in using normalising flows to model feature spaces.

Bibliography

Agarwal, Praveen, Mohamed Jleli, and Bessem Samet (2018). "Fixed point theory in metric spaces". In: *Recent Advances and Applications* 10, pp. 978–981.

Ahmed, Faruk and Aaron Courville (2020). "Detecting semantic anomalies". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04, pp. 3154–3162.

Amersfoort, Joost van et al. (Feb. 2021). "On Feature Collapse and Deep Kernel Learning for Single Forward Pass Uncertainty". In: arXiv: 2102.11409 [cs.LG]. URL: http://arxiv.org/abs/2102.11409.

Amodei, Dario et al. (2016). "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565*.

Assran, Mahmoud et al. (2023). "Self-supervised learning from images with a joint-embedding predictive architecture". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15619–15629.

Balduzzi, David et al. (2017). "The shattered gradients problem: If resnets are the answer, then what is the question?" In: *International Conference on Machine Learning*. PMLR, pp. 342–350.

Behrmann, Jens et al. (2019). "Invertible residual networks". In: *International conference on machine learning*. PMLR, pp. 573–582.

Bitterwolf, Julian, Maximilian Mueller, and Matthias Hein (2023). "In or out? fixing imagenet out-of-distribution detection evaluation". In: *arXiv preprint arXiv:2306.00826*.

Cao, Senqi and Zhongfei Zhang (2022a). "Deep hybrid models for out-of-distribution detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4733–4743.

— (2022b). "Deep hybrid models for out-of-distribution detection supplementary material". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. URL: https://openaccess.thecvf.com/content/CVPR2022/supplemental/Cao_Deep_Hybrid_Models_CVPR_2022_supplemental.pdf.

Caron, Mathilde et al. (2021). "Emerging properties in self-supervised vision transformers". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9650–9660.

Chen, Ricky TQ et al. (2019). "Residual flows for invertible generative modeling". In: *Advances in Neural Information Processing Systems* 32.

Davis, Jesse and Mark Goadrich (2006). "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240.

Dinh, Laurent, David Krueger, and Yoshua Bengio (2014). "Nice: Non-linear independent components estimation". In: *arXiv preprint arXiv:1410.8516*.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2016). "Density estimation using real nvp". In: *arXiv preprint arXiv:1605.08803*.

Fang, Zhen et al. (2021). "Learning bounds for open-set learning". In: *International conference on machine learning*. PMLR, pp. 3122–3132.

Fawcett, Tom (2006). "An introduction to ROC analysis". In: *Pattern recognition letters* 27.8, pp. 861–874.

Feinman, Reuben et al. (2017). "Detecting adversarial samples from artifacts". In: *arXiv preprint arXiv:1703.00410*.

Fort, Stanislav, Jie Ren, and Balaji Lakshminarayanan (2021). "Exploring the limits of out-of-distribution detection". In: *Advances in Neural Information Processing Systems* 34, pp. 7068–7081.

Germain, Mathieu et al. (2015). "Made: Masked autoencoder for distribution estimation". In: *International conference on machine learning*. PMLR, pp. 881–889.

Gouk, Henry et al. (2021). "Regularisation of neural networks by enforcing lipschitz continuity". In: *Machine Learning* 110, pp. 393–416.

Guo, Chuan et al. (2017). "On calibration of modern neural networks". In: *International conference on machine learning*. PMLR, pp. 1321–1330.

Hall, Brian C (2013). *Lie Groups, Lie Algebras, and Representations*. Springer International Publishing. DOI: 10.1007/978-3-319-13467-3. URL: https://link.springer.com/book/10.1007/978-3-319-13467-3.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

— (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hendrycks, Dan and Kevin Gimpel (2016). "A baseline for detecting misclassified and out-of-distribution examples in neural networks". In: *arXiv preprint arXiv:1610.02136*.

Humblot-Renaux, Galadrielle, Sergio Escalera, and Thomas B Moeslund (2023). "Beyond AUROC & co. for evaluating out-of-distribution detection performance". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3880–3889.

Hutchinson, Michael F (1989). "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines". In: *Communications in Statistics-Simulation and Computation* 18.3, pp. 1059–1076.

iNaturalist.org (2023). *iNaturalist.org*. https://www.inaturalist.org. Accessed: 2023-12-07.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kingma, Durk P and Prafulla Dhariwal (2018). "Glow: Generative flow with invertible 1x1 convolutions". In: *Advances in neural information processing systems* 31.

Kingma, Durk P, Tim Salimans, et al. (2016). "Improved variational inference with inverse autoregressive flow". In: *Advances in neural information processing systems* 29.

Kirichenko, Polina, Pavel Izmailov, and Andrew G Wilson (2020). "Why normalizing flows fail to detect out-of-distribution data". In: *Advances in neural information processing systems* 33, pp. 20578–20589.

Kobyzev, Ivan, Simon JD Prince, and Marcus A Brubaker (2020). "Normalizing flows: An introduction and review of current methods". In: *IEEE transactions on pattern analysis and machine intelligence* 43.11, pp. 3964–3979.

Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). "Learning multiple layers of features from tiny images". In.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2017). "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6, pp. 84–90.

Kuleshov, Volodymyr and Stefano Ermon (2017). "Deep hybrid models: Bridging discriminative and generative approaches". In: *Proceedings of the Conference on Uncertainty in AI (UAI)*.

LeCun, Yann et al. (1995). "Learning algorithms for classification: A comparison on handwritten digit recognition". In: *Neural networks: the statistical mechanics perspective* 261.276, p. 2.

Lee, Kimin et al. (2018). "A simple unified framework for detecting out-of-distribution samples and adversarial attacks". In: *Advances in neural information processing systems* 31.

Li, Hao et al. (2018). "Visualizing the loss landscape of neural nets". In: *Advances in neural information processing systems* 31.

Lin, Min, Qiang Chen, and Shuicheng Yan (2013). "Network in network". In: *arXiv preprint arXiv:1312.4400*.

Liu, Jeremiah et al. (2020). "Simple and principled uncertainty estimation with deterministic deep learning via distance awareness". In: *Advances in neural information processing systems* 33, pp. 7498–7512.

Liu, Jeremiah Zhe et al. (2023). "A simple approach to improve single-model deep uncertainty via distance-awareness". In: *Journal of Machine Learning Research* 24.42, pp. 1–63.

Liu, Weitang et al. (2020). "Energy-based out-of-distribution detection". In: *Advances in neural information processing systems* 33, pp. 21464–21475.

Ming, Yifei, Hang Yin, and Yixuan Li (2022). "On the impact of spurious correlation for out-of-distribution detection". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. 9, pp. 10051–10059.

Ministry for Primary Industries (Mar. 2023). *Brown marmorated stink bug: threat to NZ and identification*. `https://www.mpi.govt.nz/biosecurity/pests-and-diseases-not-in-new-zealand/horticultural-pests-and-diseases-not-in-nz/brown-marmorated-stink-bug-threat-to-nz-and-identification/`. Accessed: 2024-2-4. URL: `https://www.mpi.govt.nz/biosecurity/pests-and-diseases-not-in-new-zealand/horticultural-pests-and-diseases-not-in-nz/brown-marmorated-stink-bug-threat-to-nz-and-identification/`.

Miyato, Takeru et al. (2018). "Spectral normalization for generative adversarial networks". In: *arXiv preprint arXiv:1802.05957*.

Nalisnick, Eric et al. (2018). "Do deep generative models know what they don't know?" In: *arXiv preprint arXiv:1810.09136*.

— (2019). "Hybrid models with deep and invertible features". In: *International Conference on Machine Learning*. PMLR, pp. 4723–4732.

Netzer, Yuval et al. (2011). "Reading digits in natural images with unsupervised feature learning". In: *NIPS workshop on deep learning and unsupervised feature learning*. Vol. 2011. 5. Granada, Spain, p. 7.

Papamakarios, George et al. (2021). "Normalizing flows for probabilistic modeling and inference". In: *Journal of Machine Learning Research* 22.57, pp. 1–64.

Postels, Janis et al. (2020). "Quantifying aleatoric and epistemic uncertainty using density estimation in latent space". In: *arXiv preprint arXiv:2012.03082* 1.

Prewitt, Judith MS et al. (1970). "Object enhancement and extraction". In: *Picture processing and Psychopictorics* 10.1, pp. 15–19.

Prince, Simon J.D. (2023). *Understanding Deep Learning*. MIT Press. URL: http://udlbook.com.

Raina, Rajat et al. (2003). "Classification with hybrid generative/discriminative models". In: *Advances in neural information processing systems* 16.

Ramachandran, Prajit, Barret Zoph, and Quoc V Le (2017). "Searching for activation functions". In: *arXiv preprint arXiv:1710.05941*.

Ren, Jie et al. (2019). "Likelihood ratios for out-of-distribution detection". In: *Advances in neural information processing systems* 32.

Ruff, Lukas et al. (2021). "A unifying review of deep and shallow anomaly detection". In: *Proceedings of the IEEE* 109.5, pp. 756–795.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Smith, Lewis et al. (2021). "Can convolutional ResNets approximately preserve input distances? A frequency analysis perspective". In: *arXiv preprint arXiv:2106.02469*.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Stimper, Vincent et al. (June 2023). *normalizing-flows: PyTorch implementation of normalizing flow models*. en. DOI: 10.5281/zenodo.7565800. URL: https://github.com/VincentStimper/normalizing-flows.

Strang, Gilbert (Feb. 2009). *Introduction to Linear Algebra*. en. Wellesley-Cambridge Press. ISBN: 9780980232714.

Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Torralba, Antonio, Rob Fergus, and William T Freeman (2008). "80 million tiny images: A large data set for nonparametric object and scene recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 30.11, pp. 1958–1970.

Tsuzuku, Yusuke, Issei Sato, and Masashi Sugiyama (2018). "Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks". In: *Advances in neural information processing systems* 31.

Uria, Benigno, Iain Murray, and Hugo Larochelle (2013). "RNADE: The real-valued neural autoregressive density-estimator". In: *Advances in Neural Information Processing Systems* 26.

Van Amersfoort, Joost et al. (2020). "Uncertainty estimation using a single deep deterministic neural network". In: *International conference on machine learning*. PMLR, pp. 9690–9700.

Veit, Andreas, Michael J Wilber, and Serge Belongie (2016). "Residual networks behave like ensembles of relatively shallow networks". In: *Advances in neural information processing systems* 29.

Yang, Jingkang et al. (2021). "Generalized out-of-distribution detection: A survey". In: *arXiv preprint arXiv:2110.11334*.

Yoshida, Yuichi and Takeru Miyato (2017). "Spectral norm regularization for improving the generalizability of deep learning". In: *arXiv preprint arXiv:1705.10941*.

Zagoruyko, Sergey and Nikos Komodakis (2016). "Wide residual networks". In: *arXiv preprint arXiv:1605.07146*.

Zhang, Hongjie et al. (2020). “Hybrid models for open set recognition”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, pp. 102–117.

Appendix A

Appendix

A.1 Bi-Lipschitz Continuity in Residual Blocks

Proof that residual block $g(x) = x + f(x)$ is bi-Lipschitz continuous, based on the result given by J. Liu et al. (2020). Let the Lipschitz constant of g be L_g and the Lipschitz constant of f be L_f. It follows that $L_g = 1 + L_f$. When constraining L_f such that $L_f = L_g - 1 \leq \alpha \leq 1$, we show that:

$$(1-\alpha)||x - x'|| \leq ||g(x) - g(x')|| \leq (1+\alpha)||x - x'|| \tag{A.1}$$

where for simplicity of notation we assume Euclidean distance metrics in input and output spaces. The inequality can be shown in two stages, where we show first the left-hand side, and then the right-hand side, are true. On the left-hand side:

$$\begin{aligned} ||x - x'|| &\leq ||x - x' - (g(x) - g(x')) + (g(x) - g(x'))|| \\ &\leq ||x - x' - g(x) + g(x') + (g(x) - g(x'))|| \\ &\leq ||(g(x') - x') - (g(x) - x) + g(x) - g(x')|| \\ &\leq ||f(x') - f(x)|| + ||g(x) - g(x')|| \\ &\leq \alpha||x' - x|| + ||g(x) - g(x')|| \end{aligned} \tag{A.2}$$

Note that $||f(x') - f(x)|| \leq \alpha||x' - x||$ because $L_f \leq \alpha$. Finally, we rearrange to get:

$$\begin{aligned} ||x - x'|| - (\alpha||x' - x||) &\leq ||g(x) - g(x')|| \\ (1 - \alpha)||x - x'|| &\leq ||g(x) - g(x')|| \end{aligned} \tag{A.3}$$

For the right-hand side:

$$\begin{aligned} ||g(x) - g(x')|| &= ||x + f(x) - (x' + f(x'))|| \\ &\leq ||x - x'|| + ||f(x) - f(x')|| \\ &\leq ||x - x'|| + \alpha||x' - x|| \\ &\leq (1 + \alpha)||x - x'|| \end{aligned} \tag{A.4}$$

Put together, this gives us the inequality:

$$(1 - \alpha)||x - x'|| \leq ||g(x) - g(x')|| \leq (1 + \alpha)||x - x'|| \tag{A.5}$$

Q.E.D.

A.2 Simultaneous Optimiser Updates

```
dhm = DHM(...)


# define the optimisers
flow_optimizer = optim.Adam(
    dhm.flow.parameters(),
    lr=1e-4,
    weight_decay=16e-4
)
dnn_optimizer = optim.SGD(
    dhm.parameters(),
    [
        {'params': dhm.dnn.parameters()},
        {'params': dhm.fc.parameters()}
    ],
```

```
        nesterov=True,
        lr=0.05,
        momentum=0.9,
        weight_decay=5e-4
    )

    # update both during the training loop
    for epoch in range(args.epochs):
        for i, batch in enumerate(trainloader):
            # forward pass
            ...
            # backward pass and optimisation
            flow_optimiser.zero_grad()
            dnn_optimiser.zero_grad()
            loss.backward()
            flow_optimiser.step()
            dnn_optimiser.step()
```

A.3 Hyperparameter Exploration Results

A.3.1 Optimisation Configurations

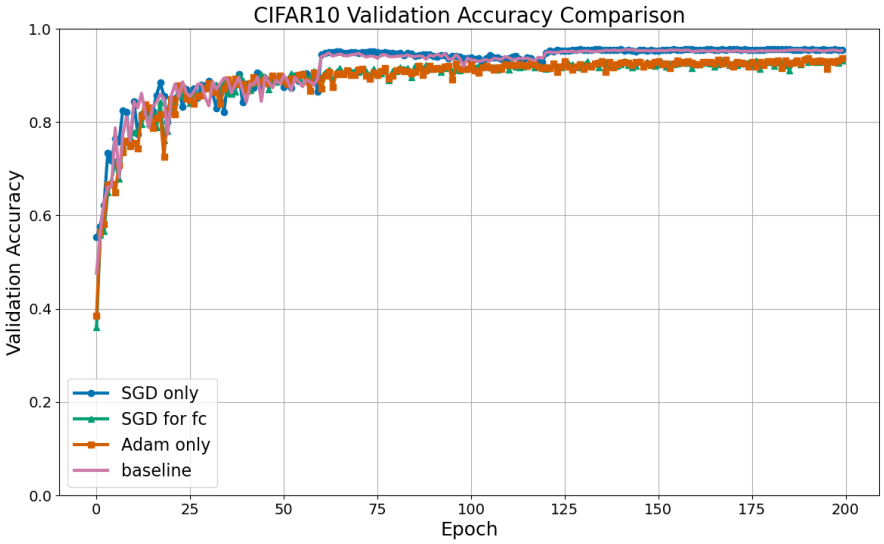


Figure A.1: CIFAR-10 validation accuracy

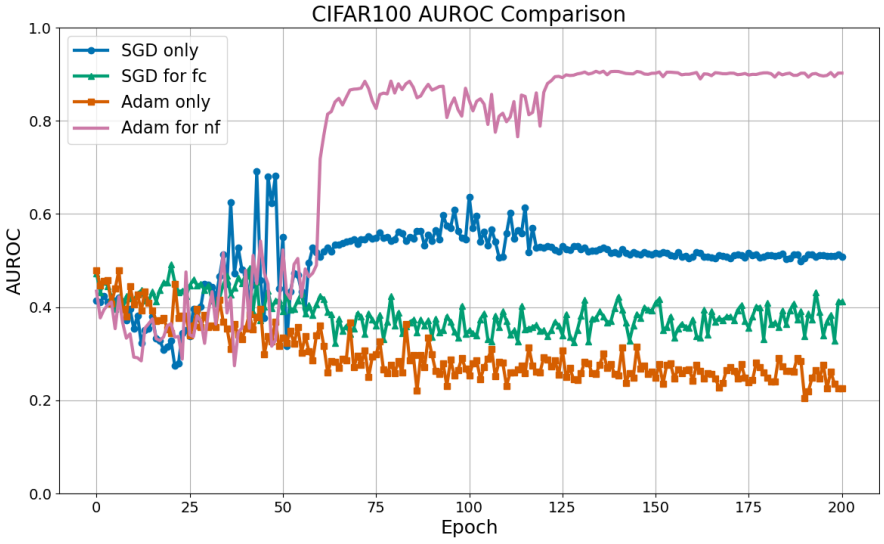


Figure A.2: CIFAR-100 AUROC

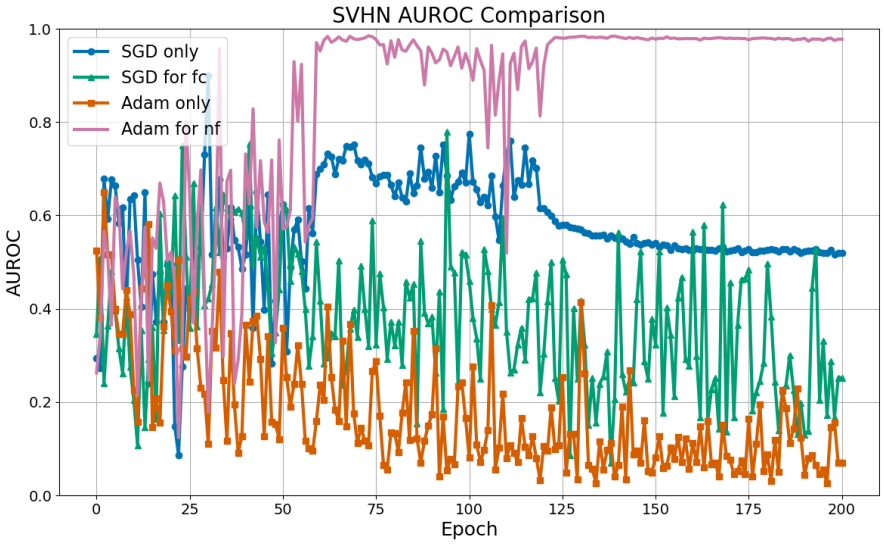


Figure A.3: SVHN AUROC

Table A.1: Summary of performances for optimiser configurations.

Model	CIFAR-100			SVHN		
	AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
Adam for nf (default)	**0.9058**	**0.9154**	**0.3679**	**0.9743**	**0.9597**	**0.2722**
Adam only	0.2257	0.3493	0.5698	0.0697	0.1581	0.6486
SGD for fc	0.4118	0.4146	0.5178	0.2520	0.1823	0.5524
SGD only	0.5153	0.5098	0.4961	0.5396	0.5286	0.4900

A.3.2 Feature Processing Configurations

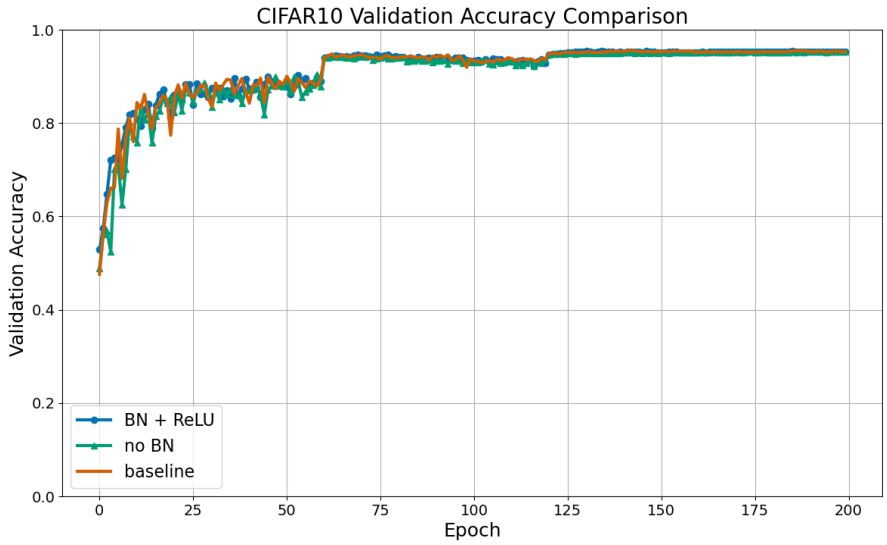


Figure A.4: CIFAR-10 validation accuracy under different feature processing configurations

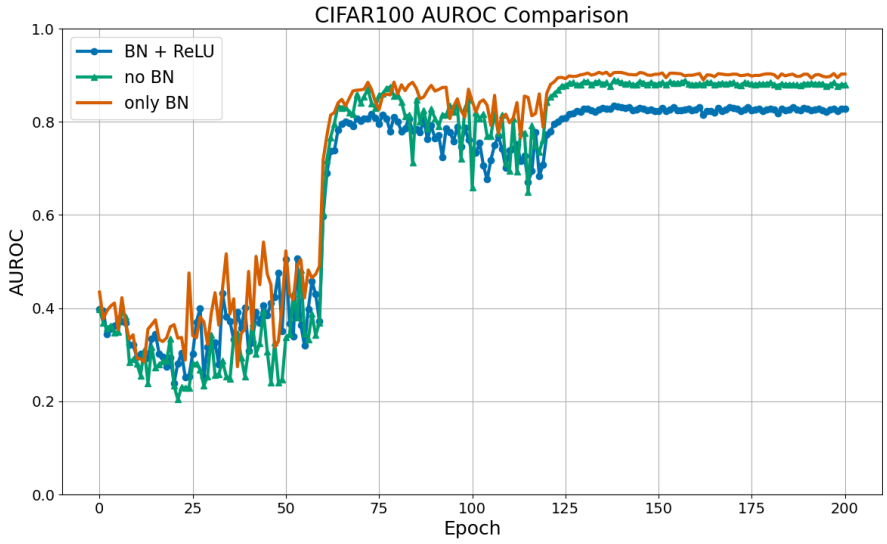


Figure A.5: CIFAR-100 AUROC under different feature processing configurations

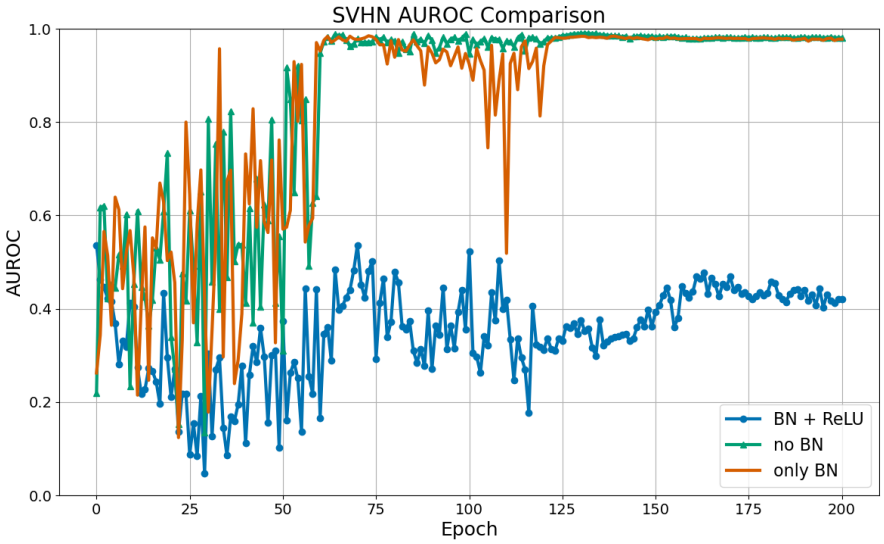


Figure A.6: SVHN AUROC under different feature processing configurations

Table A.2: Summary of performances for feature processing configurations.

Model	CIFAR-100			SVHN		
	AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
only BN (default)	**0.9058**	**0.9154**	**0.3679**	0.9743	0.9597	**0.2722**
BN + ReLU	0.8279	0.7448	0.4490	0.4196	0.2289	0.5131
no BN	0.8825	0.8409	0.4246	**0.9784**	**0.9719**	0.3395

A.3.3 Activation Normalisation

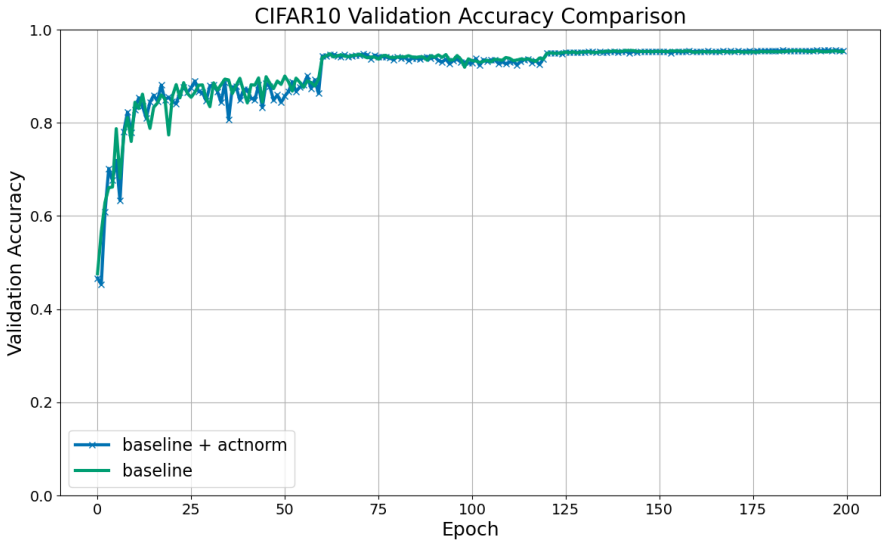


Figure A.7: CIFAR-10 validation accuracy with and without activation normalisation

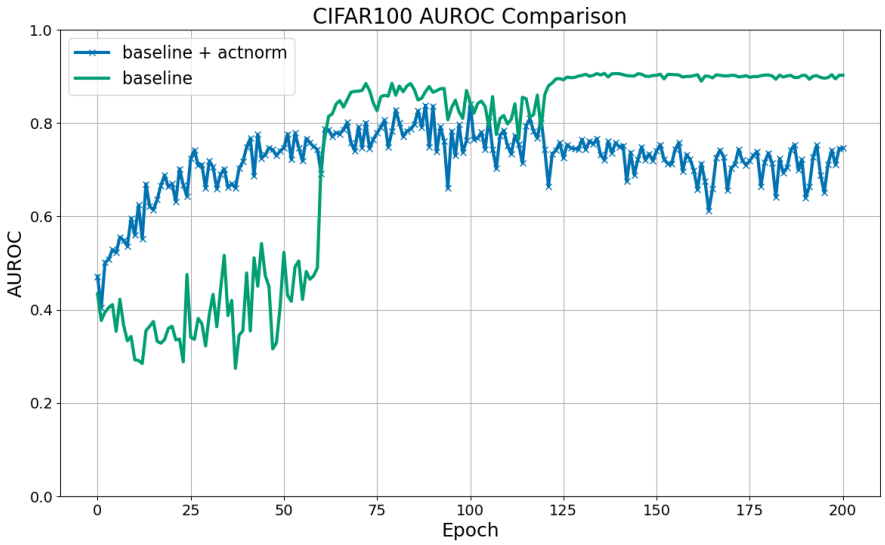


Figure A.8: CIFAR-100 AUROC with and without activation normalisation

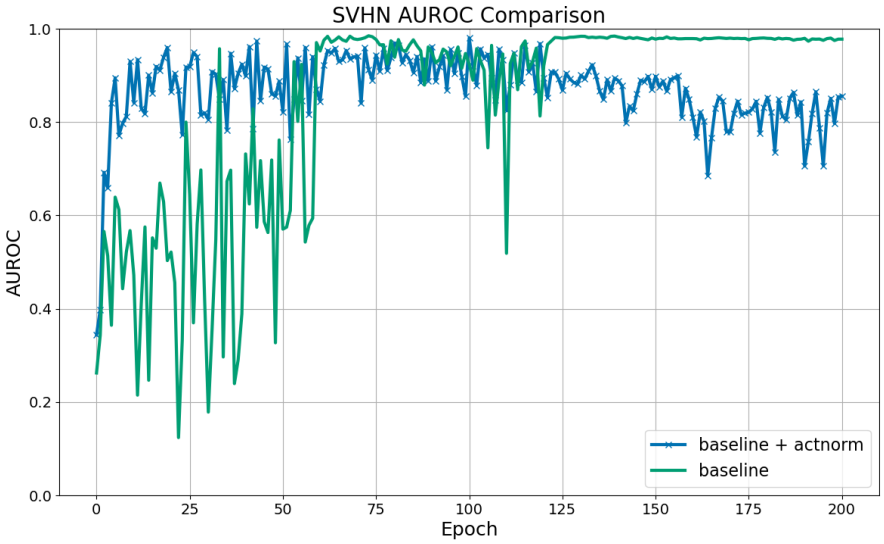


Figure A.9: SVHN AUROC with and without activation normalisation

Table A.3: Summary of performances for activation normalisation configurations.

Model	CIFAR-100			SVHN		
	AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
no actnorm (default)	**0.9058**	**0.9154**	**0.3679**	**0.9743**	**0.9597**	**0.2722**
Actnorm	0.7487	0.6993	0.4890	0.8462	0.8396	0.4854

A.3.4 Combinations of Interest

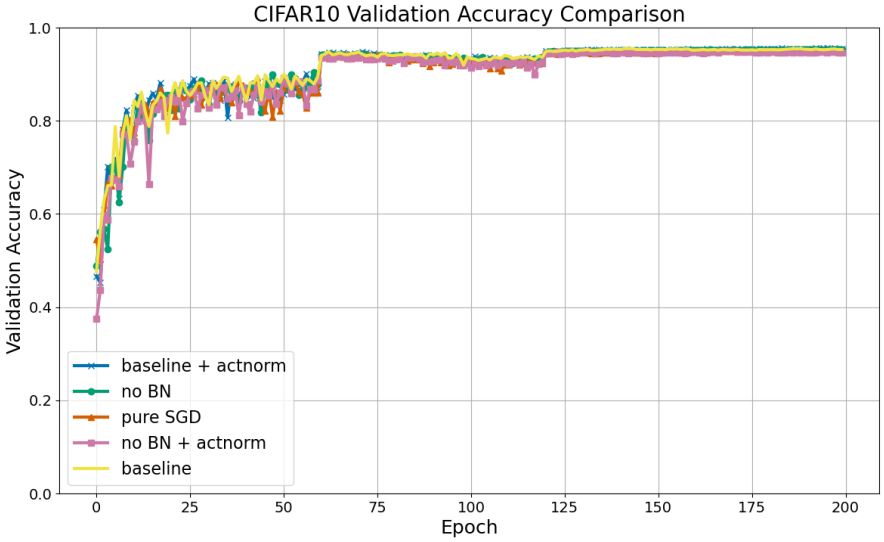


Figure A.10: CIFAR-10 validation accuracy under high performing combinations

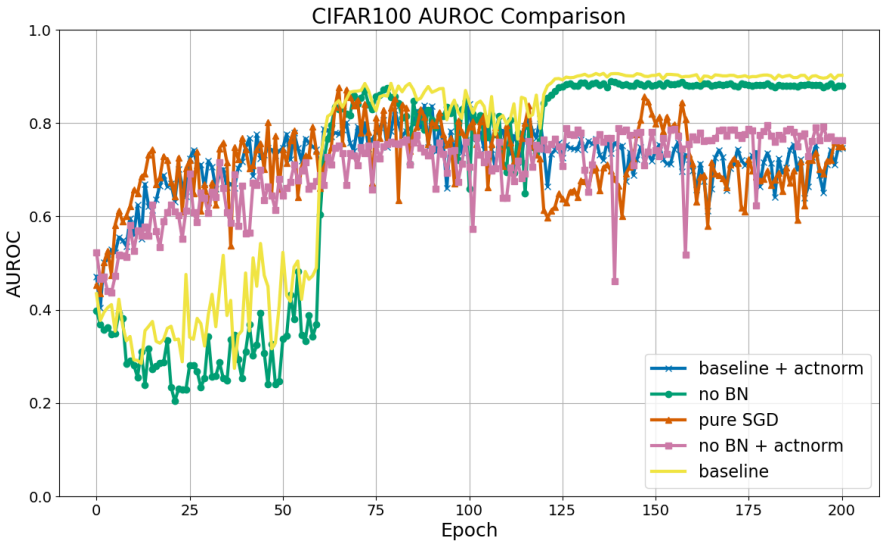


Figure A.11: CIFAR-100 AUROC under high performing combinations

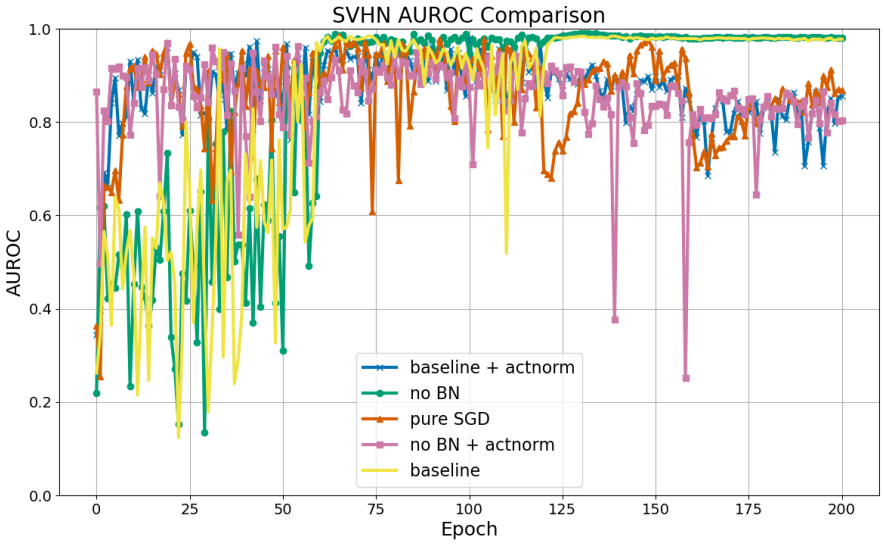


Figure A.12: SVHN AUROC under high performing combinations

Table A.4: Summary of performances for configuration combinations of interest.

Model	CIFAR-100			SVHN		
	AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
Default DHM	**0.9058**	**0.9154**	**0.3679**	0.9743	0.9597	**0.2722**
Actnorm	0.7487	0.6993	0.4890	0.8462	0.8396	0.4854
no BN	0.8825	0.8409	0.4246	**0.9784**	**0.9719**	0.3395
SGD only	0.5153	0.5098	0.4961	0.5396	0.5286	0.4900
No BN + actnorm	0.7681	0.7291	0.4925	0.8076	0.7788	0.4768

A.3.5 Normalisation Results

Norm	Validation Accuracy	CIFAR-100			SVHN		
		AUROC ↑	AUPR-in ↑	AUTC ↓	AUROC ↑	AUPR-in ↑	AUTC ↓
None	0.9565	0.2083	0.3453	0.5964	0.0884	0.1613	0.6452
l_1	0.9442	0.8179	0.8000	0.4181	0.9201	0.8299	0.3777
l_2	0.9490	0.7272	0.6862	0.4669	0.8741	0.7151	0.4319
l_∞	0.9506	**0.9058**	**0.9154**	**0.3679**	**0.9743**	**0.9597**	**0.2722**

Table A.5: Performance scores for the DHM models under different feature normalisation schemes, showing that l_∞ dramatically improves performance across the board.