

Relevance Detection of Unknown Classes through Cluster Distances

Based on Statistical Distance Measures in Feature Space

MSc Thesis

Dewwret Sitaldin

Relevance Detection of Unknown Classes through Cluster Distances

Based on Statistical Distance Measures in Feature Space

A thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in Applied Mathematics by

Dewindre Dewwret Dhartishwer
Trishul Doebe Sitaldin

born in Paramaribo, Suriname.

To be defended publicly on Tuesday the 25th of October 2022, at 10:00.

Student number: 4429648
Specialisation: Computational Science & Engineering
Research Group: Numerical Analysis
Institution: Delft University of Technology
Faculty: Electrical Engineering, Mathematics and Computer Science

Thesis Committee:	Prof.dr.ir. Kees Vuik	TU Delft,	supervisor
	Dr.ir. Gertjan Burghouts	TNO,	supervisor
	Prof.dr.ir. Arnold Heemink	TU Delft,	committee member

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

ॐ गं गणपतये नमः

*My salutations to God, Who has many names,
and Whom I lovingly invoke as Ganesh.*

सर्वं जगदिदं त्वत्तो जायते ।

The Entire Universe has Manifested from You.

सर्वं जगदिदं त्वत्तस्तिष्ठति ।

The Entire Universe is Sustained by Your Power.

सर्वं जगदिदं त्वयि लयमेष्यति ।

The Entire Universe will Dissolve in You.



Acknowledgements

This thesis is written to obtain the degree of Master of Science in Applied Mathematics at the Delft University of Technology. It marks the end of my time as a student; a transformational period that has truly changed me as an individual, on many levels. I am overjoyed and grateful to arrive at this milestone. But this very milestone, and everything else that I have ever achieved, could not have been possible without the support of others.

I would like to start with my supervisors prof. Kees Vuik and dr. Gertjan Burghouts, who have guided me during the entirety of this project. Thank you so much for your guidance, knowledge, and most of all, your understanding. I truly appreciate how both of you have been so considerate of me when COVID affected my family; your support made it bearable and helped me to stay on track. Thank you again.

My student life would have been so much more difficult, and so much less fun, without my friends at the TU Delft. I would like to thank all of you, especially Ferhat Sindy, Kasper Uleman, Zakaria Abdellaoui, Tarik Benaich, Jochem Hoogendijk, Mirza Mhravorovic, and Jan Bosma. I shall never forget our adventures, 14-hour work days, and late-night study sessions in the 'EWI-studieruimte'. During exam weeks, we owned the place.

Lastly, and above all, I would like to express my gratitude to my family; Mijn ouders (met name mijn lieve moeder), broertje, nana, nani, en vriendin. Jullie zijn de drijfveer achter al het goede wat ik doe, de essentie van al het goede in mij, en degenen zonder wie ik niet kan. Prachtig dat we dit samen mogen vieren. Ontzettend bedankt voor jullie bijzondere liefde en steun.

*Dewwret Sitaldin
Zoetermeer, 18th of October 2022*

Abstract

In the open world, machine learning (ML) models can encounter a multitude of unknown or novel classes. In a surveillance, safety, or security use case, unknown samples can pose potential threats that are hard to detect since those samples have never been trained on. At the same time, most of the unknowns that will be encountered by a surveillance ML model will be harmless. This results in too many unwanted alerts and manual analyses, of harmless unknowns that have been flagged.

Through this thesis, for the first time (to the best of our knowledge), a method is developed that can automatically assess the relevance of unknown classes, by modelling their image features as clusters (or distributions) and comparing them using statistical distance measures. Our use case lies in computer vision for military applications, where based on the user input, relevance is defined. We define road vehicles as relevant classes and use those for our training set. Our aim is to build a model that can successfully classify new unseen road vehicles as 'relevant unknowns', while also successfully classifying harmless unknown birds that are not part of the training set, as 'irrelevant unknowns'. On the DomainNet data-set, we demonstrate that our novel method can very accurately determine the relevance of unknown classes at test time for both low and high-dimensional data, with AUC scores ranging from 0.99 to a perfect 1.00.

Contents

Acknowledgements	ii
Abstract	iii
1 Introduction	1
1.1 Problem & Motivation	1
1.1.1 TNO's case & Societal Impact	2
1.1.2 Scientific Goal vs. State-of-the-Art	2
1.2 Main Contributions & Methodology	4
1.2.1 Proposed solution (simplified)	5
1.3 Research Questions	5
1.4 Thesis Outline & Reader's Guide	6
2 Theoretical Background Computer Vision	7
2.1 Image Processing & High-dimensional data	7
2.1.1 Representing our input data: an image as a matrix	7
2.1.2 High-dimensional input	8
2.2 Deep Feedforward Neural Network	9
2.2.1 Network Structure	9
2.2.2 Mappings & Network Parameters	10
2.2.3 Activation Functions	11
2.3 Supervised Learning & Classification	11
2.3.1 Classification framework	12
2.3.2 Softmax	12
2.4 Training Neural Networks	13
2.4.1 Feature generation	13
2.4.2 Loss function & Backpropagation	13
2.4.3 Optimisation	15
2.4.4 Regularisation	16
3 Related Work: OOD detection	17
3.1 Techniques for detecting unknowns	17
3.1.1 Taxonomy	18
3.1.2 Distribution Shift	18
3.1.3 Comparison & Selection of Detection Technique	20
3.2 Exploring OOD Detection	22
3.2.1 Insight 1: Formalised OOD detection	23
3.2.2 Insight 2: Comparison of scoring functions	24
3.2.3 Insight 3: OOD detection performance changes under certain parameters	25
3.2.4 Insight 4: Feeding scoring functions explicit OOD data through Outlier Exposure	26
3.2.5 Insight 5: Prior knowledge of classes related to our OOD classes might improve detection	27
3.3 Research Gaps & Possible Niche	28
3.3.1 Research Gaps	28
3.3.2 Axioms	29
4 Methodology & Design Challenges	30
4.1 Methodology Overview	30
4.2 Neural Network Feature Generation	31
4.2.1 Motivation for the CLIP Neural Network	31
4.2.2 Post-processing: Dimensionality Reduction of CLIP Vectors	33

4.3	OOD Detection Module	37
4.3.1	Motivation of OOD Scoring module	37
4.3.2	Challenges & Requirements	37
4.3.3	GEM Score Calculation	40
4.4	Clustering & Sampling Distributions	41
4.4.1	Motivation of the Clustering module	41
4.4.2	Challenges & Requirements	41
4.5	Relevance Detection	42
4.5.1	Conceptual Overview of the Module	42
4.5.2	Overview of Candidate Distance Measures	43
4.5.3	Motivation & Challenges	50
5	Experimental Setup & Results	51
5.1	Hypotheses	51
5.2	Data-set	52
5.3	Design Adaptation	53
5.4	Main Setup	53
5.4.1	Step 1: Increasing prior domain knowledge through training	53
5.4.2	Step 2: Calculating a relevance score	55
5.4.3	Step 3: Threshold determination and measuring performance	56
5.5	Experiment 1: Increasing Prior Domain Knowledge	57
5.5.1	Point-based detection models	58
5.5.2	Distribution-based detection models	59
5.6	Experiment 2: Ablation Study for Removing K-means	62
5.7	Experiment 3: Varying K in K-means	64
5.7.1	Determining the optimal K for clustering	64
5.7.2	Determining the optimal K for relevance detection	65
5.8	Evaluation & Discussion	66
5.8.1	Going beyond the AUC score: which classes are detected the best/worst?	66
5.8.2	Why the Euclidean distance performs so well	71
5.8.3	Aligning results with hypotheses	72
6	Conclusions & Recommendations	73
6.1	Conclusions	73
6.2	Recommendations for Future Work	74
	References	78
A	Background on CLIP	79
A.1	CLIP Neural Network Architecture	80

Introduction

'Real knowledge is to know the extent of one's ignorance.' - 孔夫子 (Kǒng Fūzǐ)

We humans learn largely through our senses; our eyes, ears, nose, hands, and mouth are our tools for discovery. Whenever we venture into new environments with new objects, we are able to recognise and classify these new instances based on what we have learned so far. Even when we observe something that is unfamiliar to us and we have never seen before, the human mind is aware that this is something we do not know. To showcase this, we request you to identify each of the following animals in the images in [Fig. 1.1](#):

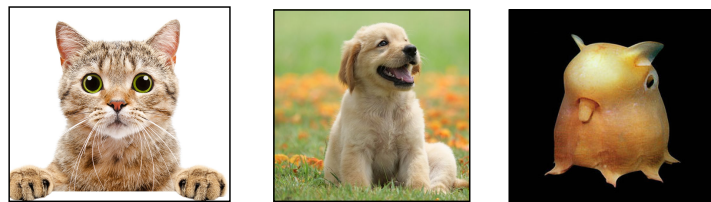


Figure 1.1: A cat, a dog, and an unknown animal.

You can tell that the first two animals are a cat and a dog, because you are likely to have seen these animals before and *learnt* what they are. Conversely, because you have probably *not learnt nor seen* the third creature, you would not be able to identify it and tell us that you simply 'do not know' what it is. Computers, however, do not inherently have access to this level of intelligence when encountering unknown objects. An average neural network, using e.g. a standard softmax classifier, will falsely classify an unknown class as something it does know, resulting in a wrong label [1]. A deep learning model trained on cats and dogs only, would correctly classify the first two images as 'cat', and 'dog' respectively, but also forcefully and falsely classify the last as either 'cat' or 'dog'. This occurs as the unknown class was not part of the training set; either because it would be too expensive to keep training the model on more and more classes, or for simply the fact that one did not have access to its training data (the new creature has never been discovered before, making any data about it nonexistent). It is precisely this concept that led to this thesis in computer vision.

This chapter further introduces the problem above by discussing its motivation and a realistic variant found in the industry. Furthermore, it presents the goals and structure of this thesis.

1.1. Problem & Motivation

The relevance of research on this specific classification topic is two-fold; it is useful to both the industry and academia. From an industrial perspective, the problem of detecting unknown classes or instances can be extended to various sectors, ranging from credit card fraud detection in finance, to anomalous gene expression signatures in cancer treatment [2]. From an academic perspective, the field of detecting unknown classes is still under-explored and becoming increasingly popular. Its problems challenge state-of-the-art (SOTA)

algorithms from academic literature to find solutions; therefore a novel approach with mathematical rigour could generate refreshing insights to the same problem while encouraging new research directions.

1.1.1. TNO's case & Societal Impact

TNO (Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek, i.e. the Dutch Organisation for Applied Scientific Research), is an independent research institute. It was founded by law in 1932 to make optimum use of scientific knowledge in solving high-priority societal and economic themes faced by the Dutch government and the industry.

Today TNO's reach has extended beyond the Netherlands and its research portfolio encompasses a wide variety of scientific fields, ranging from sustainable energy to maritime technology and defence. Their operations within those fields are categorised into 'focus areas'; this thesis was carried out at the 'Intelligent Imaging' department which is an expertise group within the focus area of 'Defense, Safety & Security'.



Figure 1.2: Visual overview of TNO's Defense, Safety & Security focus area, one where TNO often partners with military and security professionals of the Dutch Ministry of Defense and governments abroad.

The Intelligent Imaging research group is engaged in applying state-of-the-art image processing, visual pattern recognition, and artificial intelligence (e.g. computer vision) to themes which directly contribute to national security and safe society. Recall the case of the unknown animal of Fig. 1.1, its more realistic and industrial equivalent within the safety & security domain would be the identification/detection of unknown vehicles. Take a surveillance drone/robot for example, armed with a computer vision model trained on cars and trucks. Once it ventures into some area of interest at test time, it can encounter things that it has never seen before:

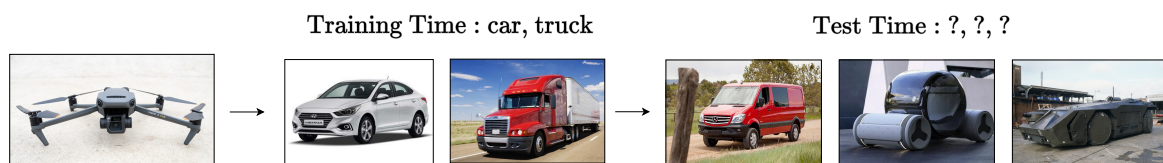


Figure 1.3: A surveillance drone, trained on cars and trucks, encounters three unknown vehicles at test time.

This model, when encountering vans, might classify some vans as 'car' and some vans as 'truck', as a van is similar to both, but not the same. While a regular van might not pose any significant danger, a strange unidentified vehicle that looks like a car, or an unknown military vehicle that has similar features to a truck, definitely could. It is in these use cases where detection of unknowns can mean threat prevention; and why TNO allocates resources to study this problem from different angles at the Intelligent Imaging department.

1.1.2. Scientific Goal vs. State-of-the-Art

The problem described above has become increasingly popular since various modern day use cases can depend on the detection of unknowns [2] and since especially neural networks, the most principal and ubiquitous type of model in computer vision, are vulnerable to unknown samples [1, 3, 4]. As a result, many attempts and techniques to solve this were made in related and recent work (Chapter 3). Therefore, our goal was to find a scientific niche in order to contribute. As we were given a lot of freedom at TNO, we allocated our literature study to formulate and design our own thesis topic: Distinguishing relevant classes from irrelevant classes in out-of-distribution detection (OOD). OOD (explained in Chapter 3) has become an umbrella term of a way to detect unknown classes vs. known classes (in-distribution or 'ID'). After exploring recent literature

we noticed that while many works present new OOD techniques that detect whether a class is 'known vs unknown', there were no works that, to the best of our knowledge, analyse whether the detection of such an OOD class was actually relevant or helpful to the use case. The example below in Fig. 1.4 illustrates this.

In the surveillance or safety use case described above, only knowing whether a class is ID or OOD is not enough as it does not take safety risks into account. For example, the drone might encounter many things that it has not been trained on like harmless birds, animals, or a lost screwdriver; all rightfully flagged as OOD but nothing worthy of raising the alarm. An unidentified vehicle, however, could potentially pose risks and is therefore more interesting/relevant.





	Training	Testing
State-of-the-art		 <ul style="list-style-type: none"> → ID → OOD
This thesis		 <ul style="list-style-type: none"> → ID_{Relevant} → ID_{Irrelevant} → OOD_{Irrelevant} → OOD_{Relevant}

Figure 1.4: SOTA models train on low-resolution data, and work well when OOD classes are substantially different from training classes. It is much easier to detect that images of landscapes and wood textures are OOD, when training on vehicles. Our model can do OOD detection when classes are very close to our training set (testing on unknown vehicles, when training on vehicles) and also tell whether a sample is relevant to our use case or not.

Based on what a user defines as relevant, our model can detect which classes at test time are unknown and will separate those classes into relevant and irrelevant classes. Fig. 1.5 serves as a demo of our work, showcasing actual model output: our model is trained on images of the buses and encounters thousands of images at test time. Based on what it has learned, it will classify all road vehicle classes as relevant and all other classes as irrelevant.



Figure 1.5: Based on examples of a relevant, known class (e.g., 'bus' - top row), our method can classify test images as either known or unknown, where the unknowns are divided into relevant (other vehicles, middle row) and irrelevant (non-vehicles, bottom row).

1.2. Main Contributions & Methodology

Below are listed the main contributions of this thesis, for which a visual overview is provided in [Section 3.3.2](#).

1. **Relevance Detection:** Our most significant contribution is detecting whether a class is relevant or irrelevant, after it has been determined whether it is ID or OOD. In this contribution we propose a way to pre-specify a sense/measure of relevance that can be tailored to the specific the use case, such that this solution can be generalised to different domains beyond surveillance & security.
2. **Distribution approach:** Until now, only single points/samples of each class were used at test time in analyses, i.e. detecting a single OOD van, or single OOD bird at test time. It is quite plausible that one would encounter many instances of OOD birds and OOD vehicle classes, with varying characteristics. Therefore, we propose to leverage statistical properties of those groups of OOD classes by treating them as distributions. When treated as distributions (either point clouds, probability densities, etc.), a whole arsenal of mathematical techniques from analysis and probability theory are at one's disposal.
3. **Using realistic data:** Contemporary literature mostly employs simple low-resolution data-sets (which does not align with reality, and often tests to detect OOD classes that are substantially different from their training data. Detecting that an unknown frog is OOD when trained on cars is far easier than detecting that vans are OOD, as vans and car look alike.
4. **Varying prior information (domain knowledge):** Through this we showcase and test how well OOD relevance detection works as we know more, or less of the domain we are operating in. For example, given that vehicle classes are of interest, we aim to test what happens to relevance detection; can the model better detect unknown vehicles as relevant and unknown birds as irrelevant, if we train on 4 vehicles (car, bus, truck, van), instead of only training on 1 class (car)?

'Knowing the extent of your ignorance is true knowledge', paraphrased from the Chinese philosopher Kǒng Fūzǐ, is what we aim to do with our contributions. By adding relevance detection and prior knowledge, we are playing around with the extent of a model's ignorance; knowing that a class is unknown is a great start, but knowing that a class is unknown and (ir)relevant can be considered true knowledge in some use cases. While all four are main contributions, the first three are also true novelties in the field as we have, to the best of our knowledge, not encountered anything similar in previous studies and literature.

1.2.1. Proposed solution (simplified)

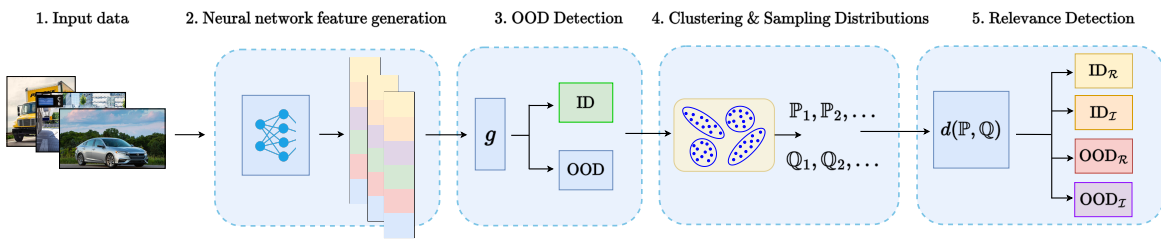


Figure 1.6: Simplified pipeline of our solution at test time, allowing for the 4 outputs shown in Fig. 1.4 .

The details and design choices of the pipeline above are thoroughly discussed in Chapter 4, so below follows a concise description only.

1. After the test input images are loaded, they are sent to a neural network of choice to generate image features (Section 2.4.1). These features are meaningful vector representations of each image that can be used for further computations and analyses. The neural network is trained beforehand.
2. We use a SOTA OOD detection algorithm of choice (Chapter 3) to separate the classes of test data into two groups. These groups contain the exact same image features, separated into an OOD batch and ID batch. The OOD algorithm is trained beforehand on ID classes (e.g. car & truck), to detect OOD classes at test time (e.g. van, hammer).
3. We then cluster within both these ID and the OOD batches to get ID class clusters $\{Q_1^{\text{ID}}, Q_2^{\text{ID}}, \dots\}$, and OOD class clusters $\{Q_1^{\text{OOD}}, Q_2^{\text{OOD}}, \dots\}$, which we treat/sample as distributions. In addition, we also sample distributions $\{P_1, P_2, \dots\}$, of our training classes (the ones we have seen before testing).
4. In the last phase of the pipeline, various (statistical) distance measures d are used to determine whether an unknown distribution Q_1^{OOD} at test time is close to our learned distribution of e.g. a car P_{car} by calculating $d(P_{\text{car}}, Q_1^{\text{OOD}})$. If d is small, then Q_1^{OOD} is likely relevant and vice versa.

Example: Let $Q_1^{\text{OOD}}, Q_2^{\text{OOD}}$ be unknown data clusters of a hammer and a van respectively a test time, and P_{car} a training clusters sampled during training. One can expect that $d(P_{\text{car}}, Q_1^{\text{OOD}}) > d(P_{\text{car}}, Q_2^{\text{OOD}})$ because vans are closer to cars, than hammers. This means that even without knowing what $Q_1^{\text{OOD}}, Q_2^{\text{OOD}}$ are, we can tell that one is more relevant than the other.

1.3. Research Questions

The main objective of this thesis is to answer the following question:

(How) Can relevant unknown classes be distinguished from irrelevant unknown classes?

Sub-questions

SRQ1: Which technique to detect unknown classes is most suitable for such a task?

- (a) Which field of techniques: Anomaly Detection, Novelty Detection, Open-Set Recognition, OOD?
- (b) Which detection models?

SRQ2: How can we leverage the statistical properties of OOD classes to distinguish relevant from irrelevant?

- (a) Parametric or non-parametric approach?
- (b) What kind of distance measures work best? f -divergences, L_2 , statistical distances?

SRQ3: What role does the dimensionality of the data play in relevance detection?

SRQ4: How can the performance of our solution pipeline be measured?

- (a) What experimental setup best measures this performance?
- (b) What performance metrics should be used?
- (c) When is this solution poor/unusable?

SRQ5: How does prior information about the domain, and related classes to the OOD classes of interest help detection?

1.4. Thesis Outline & Reader's Guide

The first two chapters from now on provide theoretical background from the literature.

- [Chapter 2](#) provides general theoretical background on deep learning and computer vision; a basic understanding on how image processing, and how neural networks learn and work with image features.
- [Chapter 3](#) focuses on the specific problem at hand and aims to find a direction of the thesis through a literature review. We explore what research has thus far been conducted and compile results of a select number of papers whose insights we can use. The end of this chapter will provide an overview of insights and research gaps that we can use to start working on a solution.

The next two chapters are concerned with the design and implementation/testing of our solution. In these chapters one will see that the solution is inspired by the theoretical background found in earlier chapters, but also adapted to better fit our problem specifically.

- [Chapter 4](#) builds our solution (and pipeline) from the ground up and is derived from the techniques found in literature. All design choices and explanations of the approach are treated here and will be supported by its corresponding mathematical foundations taken from measure theory, vector norms/metrics, and (high-dimensional) statistical distances.
- [Chapter 5](#) starts off with our data-set and the pre-processing that will be done prior to experimentation. Next, various configurations of our model are discussed and a setup for the main experiments is laid out. Preliminary tests and adjustments to our design choices are also treated here.

At last, [Chapter 6](#) draws conclusions and recommends topics for future research.

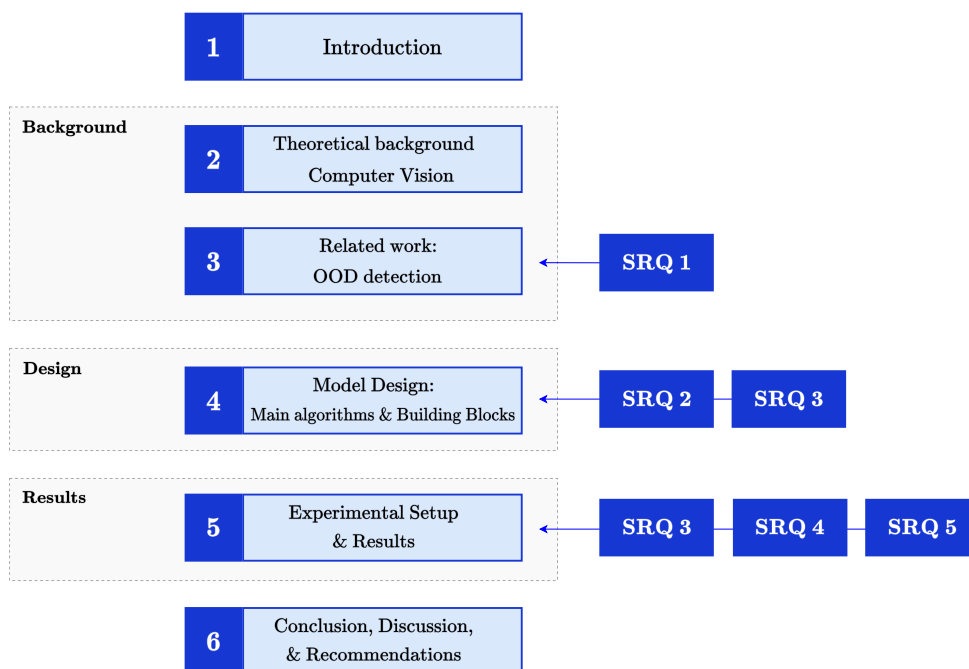


Figure 1.7: A visual of the thesis outline, denoting each chapter and the corresponding research question that is answered.

2

Theoretical Background Computer Vision

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from visual input like digital images and videos. One can say that if AI allows computers to think, computer vision allows computers to see, observe, and understand our world. It is a multidisciplinary field concerned with the image processing, classification, and object detection among other things, and used to visually analyse and detect real-world objects like humans and cars. This thesis is mainly centred around the *image processing* and *classification* aspects of computer vision as we use deep learning to perform these tasks. In today's world, deep learning (neural networks) is an indispensable tool in image related tasks; the advent of deep learning has disrupted the field of image processing altogether by introducing new methods based on neural networks [5], which could handle vast amounts of data and do good *feature extraction*.

Chapter goals

1. **Providing a brief introduction of computer vision & deep learning** - [Sections 2.1, 2.2 and 2.4](#)
Since a neural network will form a main building block in our solution, the structure, training, and operational dynamics of a neural network shall be treated with mathematical rigour.
2. **Preparing the reader for Related Work (next chapter)** - [Section 2.3](#)
We briefly treat essentials of how a network is used to classify data, in order to understand how this concept will be extended to classifying unknown classes in the next chapter.

2.1. Image Processing & High-dimensional data

Before a computer vision algorithm like our solution can make sense of what object, creature, or class is inside an image, we need to establish the notion of an image first. For the mathematical modelling of images, we distinguish the following two cases:

Definition 2.1.1 (Continuous (ideal) Image) *A continuous planar image can be represented as a mapping $f : \Omega \rightarrow \mathbb{R}$, where $f(x, y)$ is the value (intensity) at the spatial coordinates $(x, y) \in \Omega$, and $\Omega \subset \mathbb{R}^2$ denotes the domain of the image (rectangle).*

Definition 2.1.2 (Discrete (digital) Image) *Given an image $f : \Omega \rightarrow \mathbb{R}$. When x and y , and the intensity values $f(x, y)$ are all finite discrete quantities, the image is referred to as a digital image.*

The formalisation above allows one to easily perform mathematical transformations and operations like filtering, smoothing, compression etc.

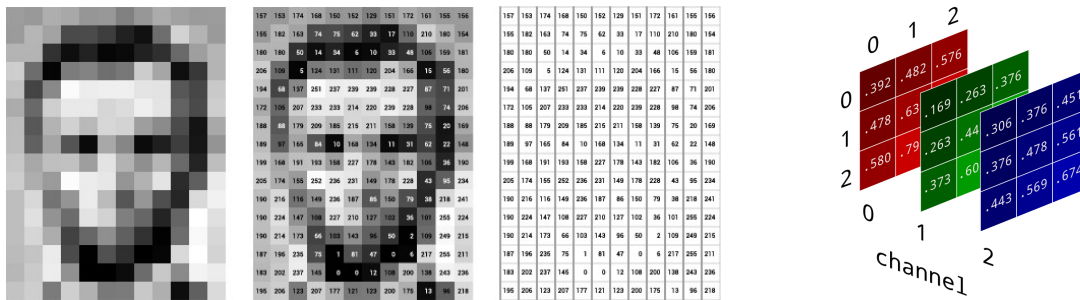
2.1.1. Representing our input data: an image as a matrix

The obvious connection between the ideal images (with infinite value) and the discrete image (with finite values) is the fact that the domain Ω into $N_1 \times N_2 \times \dots \times N_d$ small rectangles called *pixels*. Discretising coordinates (x, y) is called *sampling* and discretising intensity values $f(x, y)$ is called *quantisation*. Applying

both sampling and quantisation results exactly in what we will feed our model: a digital image represented as an $M \times N$ matrix:

$$f = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & \dots & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}_{M \times N}$$

where each entry's value corresponds to the intensity of a pixel. Take a grayscale image, the intensity of each pixel falls into the interval $[L_{\min}, L_{\max}]$, i.e. $L_{\min} \leq f(x, y) \leq L_{\max}$. The number of gray levels is defined by the bit depth $k > 0$ of an image: $L = 2^k$, 8-bit images are common and make $2^8 = 256$ distinct gray levels. This means that images take values in $\{0, 1, \dots, 255\}$ as depicted in Fig. 2.1a.



(a) Representation of a grayscale image where a value of 0 corresponds to 'black' and a value of 255 to 'white' [6]. (b) Three 'grayscale' copies of a colour image, per channel/primary colour [7].

Figure 2.1: Matrix representations of an image.

This works similarly for 8-bit RGB colour images which we will be using; such an image is now defined as a 2D vector-valued function $f : \Omega \rightarrow \mathbb{R}^3$, with

$$f(x, y) = (f_R(x, y), f_G(x, y), f_B(x, y)),$$

where each colour R,G,B, has intensities from 0 to 255. As a consequence, RGB colour images are composed of three channels where each channel is the 'grayscale' copy of the image (Fig. 2.1b), but in one of the primary colours. All three of these channels loaded into a neural network.

2.1.2. High-dimensional input

It is important to note that images are not read as a matrix by neural neural networks, but as a vector. As an example, take a grayscale digital image of the number '9', $f_{\text{nine}} : \Omega \rightarrow \mathbb{R}$, which is a low resolution (28×28 pixels) image from the MNIST data-set. When read, all pixels of each row are collected into a column vector; 28 rows with 28 pixels per row make a vector that is 784 entries long, i.e. $x \in \mathbb{R}^{784}$.



(a) Representation of the number "9" of the MNIST data-set [8]. (b) Feeding the 784-pixel vector to a neural network [9].

Figure 2.2: Image pixels flattened into vector before it is fed to a neural network.

This showcases that even the most crudest of images results into data of a substantial dimension (784D). Any realistic use case will deal with images of at least hundreds by hundreds pixels $\Rightarrow x \in \mathbb{R}^{>10^4}$. Fortunately, it is not the resolution, but the amount of the extracted features (Section 2.4.1) that can cause complexities for when we will be doing analyses using distance metrics (Chapter 4).

2.2. Deep Feedforward Neural Network

(Deep) feedforward neural networks, also called multilayer perceptrons (MLP), are the most elementary neural networks from which more advanced and contemporary structures were later derived [5, 10, 11]. They are called 'deep' because of the multiple layers in the network, that are stacked on top of each other which can result into a structure of substantial *depth*. Feed-forward is used to denote that there are no connections in which outputs of the model are fed back into itself. This section and the sections that will follow, treat the fundamentals of the MLP only, as most of it still holds true for more complex architectures and because this material suffices to understand and appreciate the rest of the thesis.

2.2.1. Network Structure

As shown in Fig. 2.3, a neural network is essentially a computation graph comprised of 3 parts (input, hidden, output) where information flows from the input layer, through the hidden layer, to arrive at the output (prediction) layer without any recurrences or feedback. Each node in the network is called a neuron (inspired from the neurons/nerve cells that constitute the human brain), hence its name.

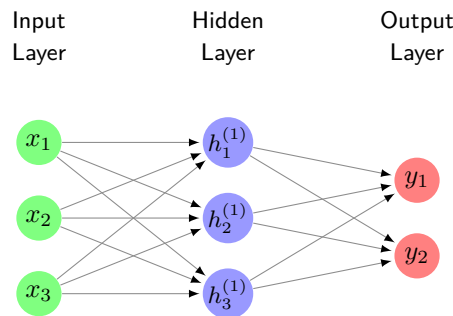


Figure 2.3: A single (hidden) layer feedforward neural network with input $\mathbf{x} \in \mathbb{R}^{n=3}$ and output $\mathbf{y} \in \mathbb{R}^{m=2}$. $L = 1$, and $(k^0, k^1, k^2) = (3, 3, 2)$

The main goal of a deep feedforward neural network is to approximate some mapping f , given some input vector (e.g. an image) $\mathbf{x} \in \mathbb{R}^n$, to arrive at an output $\mathbf{y} \in \mathbb{R}^m$ [12]. Before we go through the learning/approximation process in the next sections, let us first mathematically formalise the operations within a neural network more rigorously. For each (hidden) node, information is passed as follows.

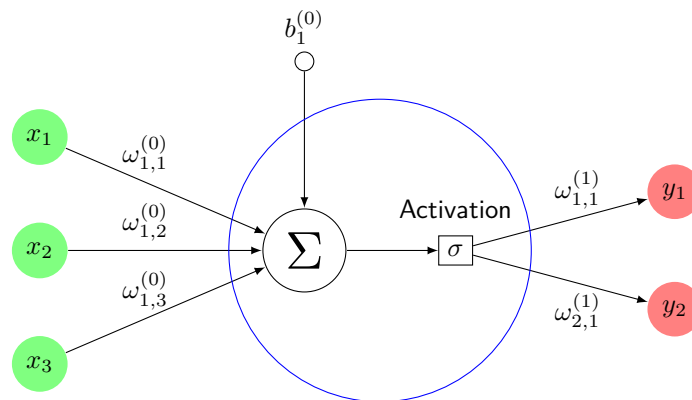


Figure 2.4: Mathematical operation in the MLP in Fig. 2.3, from input ($\ell = 0$) through a single hidden neuron (blue) ($\ell = 1$) to output ($\ell = 2$).

Let $\ell \in \{0, 1, \dots, L, L + 1\}$ denote the layers of the network, where $\ell = 0$ is used for the input layer, $\{1, \dots, L\}$ represent the hidden layers, and $\ell = L + 1$ is the output layer. Within the network, information is passed from the previous layer $\ell - 1$, through a weighted connection before it reaches a node in the next layer ℓ . For each connection with weight w_{ij}^ℓ , i indicates the index of the neurons of the next layer, j the index of the neurons of the preceding layer, and the subscript ℓ denotes, in this case, the layer the connection originates from. For each layer we also define the node width as k^ℓ , see Fig. 2.3 for a visualisation of this notation.

For an MLP to be able to approximate both linear and nonlinear mappings, a combination of an affine mapping and a nonlinear activation function is used above (more on this in [Section 2.2.3](#)). After a small bias b_i^ℓ is added to the sum, it is fed a nonlinear activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$, and sent to the neuron in the next layer. This process is repeated as an MLP is comprised of multiple layers.

2.2.2. Mappings & Network Parameters

Each operation between the layers can be formulated as a mapping $f_i^\ell(\mathbf{x}) : \mathbb{R}^{k^{\ell-1}} \rightarrow \mathbb{R}^{k^\ell}$ [13] as follows:

$$f_i^\ell(\mathbf{x}) = \sigma \left(\sum_{j=1}^{k^{\ell-1}} w_{ij}^{\ell-1} f_j^{\ell-1}(\mathbf{x}) + b_i^{\ell-1} \right) \quad (2.1)$$

$$f_j^0(\mathbf{x}) = x_j, \quad f(\mathbf{x}) = \sum_{j=1}^{k^L} w_{1,j}^L f_j^L(\mathbf{x}) + b^L$$

Where $f_j^0(\mathbf{x})$ and $f(\mathbf{x})$ are the mappings, defined at the input and output respectively.

As mentioned before, the above mapping is a composition of an affine mapping and a nonlinear activation function. To treat them separately, let us collect the parameters weights and biases in a matrix/vector:

$$\mathbf{W}^\ell = (w_{ij}^\ell) \in \mathbb{R}^{k_{\ell+1} \times k_\ell}, \quad \mathbf{b}^\ell = (b_i^\ell)$$

We can now define affine mappings $\psi_{\mathbf{W}^\ell, \mathbf{b}^\ell} : \mathbb{R}^{k^{\ell-1}} \rightarrow \mathbb{R}^{k^\ell}$, dependent on these parameters, as follows:

$$\psi_{\mathbf{W}^\ell, \mathbf{b}^\ell}(\mathbf{y}^\ell) = \mathbf{W}^\ell \mathbf{y}^\ell + \mathbf{b}^\ell, \quad \text{where } \mathbf{y}^\ell = f(\mathbf{x}; \mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1}) := \sigma(\mathbf{W}^{\ell-1} \mathbf{x} + \mathbf{b}^{\ell-1}) \quad (2.2)$$

The notation of [Eq. \(2.2\)](#) might seem rather superfluous and confusing, but in reality, it simplifies the notation of [Eq. \(2.1\)](#) and directly portrays the computation of a neural network; the input at layer l is a matrix multiplication and addition, applied point wise by a nonlinear function, of the output at layer $l - 1$. Using this, we can now define a neural network as a composition of functions:

$$\mathcal{NN}(\mathbf{x} \mid \Theta) := f(\mathbf{x}) = \psi_{\mathbf{W}^L, \mathbf{b}^L} \circ \sigma \circ \dots \circ \sigma \circ \psi_{\mathbf{w}^0, \mathbf{b}^0} \quad (2.3)$$

With $\Theta := (\theta^0, \dots, \theta^L) \in \mathbb{R}^p$ now denoted as the *network parameters* per layer and p as the total parameters, where the parameter vector per layer is defined as $\theta^\ell = \{\mathbf{W}^\ell, \mathbf{b}^\ell\}$. It is precisely these parameters that the network will attempt to *learn* by changing and tweaking its values in order to arrive at a suitable approximation (see [Section 2.4](#) for this learning process).

To illustrate this notation's simplicity, take the following neural network with fixed activation functions,

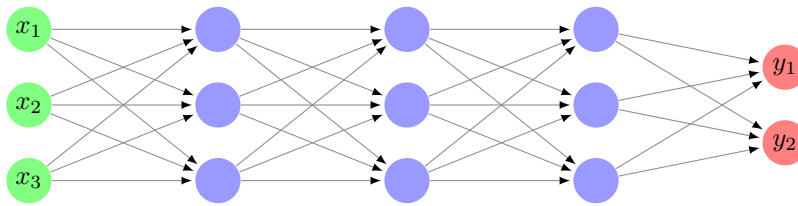


Figure 2.5: An MLP that passes input vector $\mathbf{x} \in \mathbb{R}^3$ through three hidden layers: $\mathbf{h}_1 = f_1(\mathbf{x}), \mathbf{h}_2 = f_2(\mathbf{h}_1), \mathbf{h}_3 = f_3(\mathbf{h}_2)$, to arrive at a vector-valued output $\hat{\mathbf{y}} \in \mathbb{R}^2$

its output can be decomposed in the following hierarchical manner:

$$\begin{aligned} \hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{W}^{\ell=4}, \mathbf{b}^{\ell=4}) &= \mathbf{W}^4 \mathbf{h}_3 + \mathbf{b}^4 \\ \mathbf{h}_3 &= \sigma(\mathbf{W}^3 \mathbf{h}_2 + \mathbf{b}^3) \\ \mathbf{h}_2 &= \sigma(\mathbf{W}^2 \mathbf{h}_1 + \mathbf{b}^2) \\ \mathbf{h}_1 &= \sigma(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1) \quad \text{where } \mathbf{x} \text{ is the input vector.} \end{aligned} \quad (2.4)$$

This very decomposition showcases deep learning's core strength; through a deep sequence of a simple composition functions, highly complex patterns can be approximated with arbitrary precision. In fact, the Universal Approximation Theorem (UAT) [14] is a (set of) theorem(s) that proves the class of feedforward neural networks are capable of approximating any real valued continuous function defined on compact subsets of \mathbb{R}^d , with arbitrary precision. That is to say; for any non-affine continuous function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$, there is a sum $\hat{f}(\mathbf{x})$, for which

$$|f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \varepsilon \quad \text{for all } \mathbf{x} \in \mathbb{R}^d$$

where $\varepsilon > 0$ and $\hat{f}(\mathbf{x})$ is defined as in Eq. (2.1). See the next section for a brief treatment of these non-affine functions σ .

2.2.3. Activation Functions

Activation functions are present in the neural networks to add non-linearity to their architecture. As a consequence, neural networks are able to extract non-linear patterns in the data. Without these non-linear functions, a neural network's utility is restricted to a great extent as it would be nothing more than an exalted linear transformation of its input.

Definition 2.2.1 (Activation function) *An activation function at a node in a computational network is a non-linear mapping $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ that defines the output of that node given an input or set of inputs. It determines the relationship between the input x and output $\sigma(x)$ of a node.*

The main idea behind the activation function besides adding non-linearity, is to decide whether a neuron 'fires up' or not. Meaning that an activation function suppresses neurons whose inputs are of no significance to the overall application of the neural network and 'activates' those that are not. It does this by using a threshold; if the weighted input from the previous node reaches a certain value it fires up, and vice versa if not. This response depends on the function as we can see in Fig. 2.6.

The most commonly used activation functions are the sigmoid function, tanh function and the ReLU (Rectified Linear Unit). The function share similarities as well as differences; the sigmoid $\left((1 + e^{-x})^{-1}\right)$ and tanh function have properties of having a smooth derivative and normalised outputs, while the ReLU $(\max\{0, x\})$ has unbounded output and is computationally faster [15].

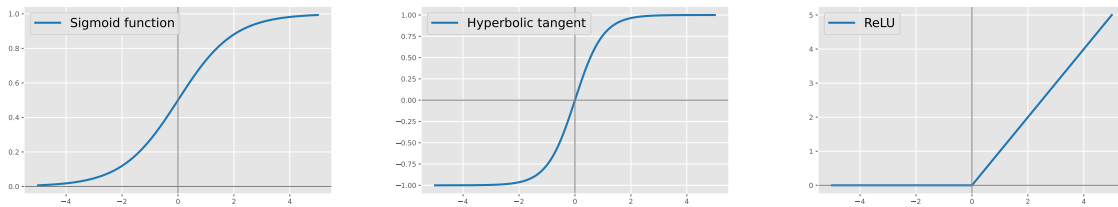


Figure 2.6: The sigmoid, tanh, and ReLU activation functions (from left to right).

While selecting the optimal activation function depends on the use case, there are quite some more notable differences between the ones we mentioned but analysing those is beyond the scope of this thesis as we shall see in Section 4.2. We therefore would like to refer to the works of Nwankpa et al. [16] and Shaw [17] for an extensive comparison.

2.3. Supervised Learning & Classification

Recall that in Section 2.2.2, we pointed out that a neural network can be used to approximate functions. The reason why we are in need of a such a function approximator is because the main problem of this thesis, presented in Chapter 1, is characterised as a *supervised learning* task.

Definition 2.3.1 (Supervised learning) *A machine learning task of learning a function that maps an input (some input image) to an output (label/name of the class in the image), based on example input-output pairs (training data).*

What this boils down to is that given some input image, a neural network needs to tell us what it is by assigning a label. We proceed to lay out the notations and fundamentals of this learning setting, as it will be used extensively throughout this thesis.

2.3.1. Classification framework

In supervised machine learning, the task constitutes an input space $\mathcal{X} = \mathbb{R}^d$, a label (output) space e.g. $\mathcal{Y} = \{\text{car, truck, } \dots, \text{bus}\}$, and a labelling function that relates the two. The goal is to find a labelling function, that is either modelled by some mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$, or by a probability distribution $P(y | \mathbf{x})$, for all $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$, such that it outputs the learning task's labelling of any input vector \mathbf{x} with high probability. In our case, we have selected a neural network as our learner f (since that is state-of-art in computer vision). Thus, the task is to train a neural network f_θ , parameterised by θ , which computes the posterior probabilities over $\mathcal{Y} : f_\theta(\mathbf{x}) = \{p(y | \mathbf{x}); y \in \mathcal{Y}\}$. That is, given an observation/image $\mathbf{x} \in \mathcal{X}$, what is the probability that the image belongs to the label $y \in \mathcal{Y}$?

Our learner is given access to a set of pre-labelled data (training data-set) $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ which is sampled independent and identically distributed (i.i.d) from a data-generating joint probability distribution $P_{\mathcal{X}\mathcal{Y}}$ defined over $\mathcal{X} \times \mathcal{Y}$. In short: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \stackrel{i.i.d}{\sim} P_{\mathcal{X}\mathcal{Y}}$, where each \mathbf{x}_i represents the image data, and each y_i the corresponding label.

Both regression and classification exist in the supervised learning setting, and our case is clearly a classification problem. The only implication this has is that all $y \in \mathcal{Y}$ assume discrete values, known as class labels.

2.3.2. Softmax

More activation functions exist than we covered in the previous section, but there is one more that plays an especially prominent role in deep learning as it is used to classify the input at the end of the neural network; the softmax activation function.

Definition 2.3.2 (Softmax function) *The softmax is an activation function $\sigma : \mathbb{R}^k \rightarrow [0, 1]^k$ that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the input vector.*

$$\sigma(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}} \quad \text{for } i \in \{1, \dots, k\} \text{ and } \mathbf{x} = (x_1, \dots, x_k) \in \mathbb{R}^k \quad (2.5)$$

The softmax function, first used in statistical mechanics as the Boltzmann distribution used to characterise state of a system of particles with respect to temperature and energy [18], has found its way into machine learning through statistical decision theory [19] and is now frequently used in multi-class classification. It is mainly used as the activation function at the output layer of neural networks. The other activation functions like the ones in the previous section are used in the middle (hidden) layers at each node, and while they can be used at the output layer as well, they are not fit for applications beyond binary classification. In softmax, the exponential function is applied to each element x_i of the input vector \mathbf{x} and then normalised by dividing by the sum of all the exponentials. This process guarantees that the elements in the output vector $\sigma(\mathbf{x})$ sum up to 1, giving a clear probability of input belonging to any particular class.

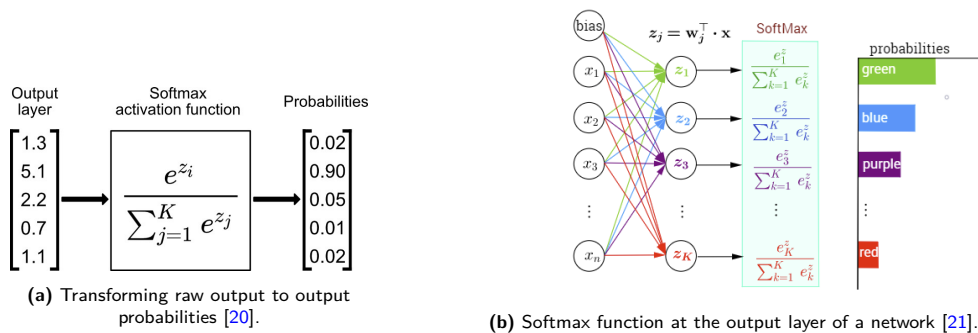


Figure 2.7: An example of softmax used to classify a colour out of multiple options (multiple classes) [21]. Since the probability that the observation is green is highest, the classification given by the softmax will be 'green'.

2.4. Training Neural Networks

No matter how advanced and sophisticated a machine model might be at its core, when poorly trained it will not amount to much. In this section, we will treat how a neural network is trained, and based on what the output probabilities that determine which image belongs to which class label $y \in \mathcal{Y}$ are calculated.

2.4.1. Feature generation

At test time (after training), before the softmax can even make a prediction \hat{y} , the image vector x of an input (which are just pixel values) needs to be transformed into meaningful information; a *feature vector*. Features are certain properties of data that have meaning; take an image of a horse for instance, image features that one can think of for a feature vector are $[\text{colours}, \text{shape of the animal}, \text{amount of legs}, \dots, \text{fur texture}]^T$. Neural networks are trained to extract these features from an image vector x and transform it into a more meaningful feature vector $h(x; \theta)$. The amount and kind of features that are learnt differ per network. Most neural networks learn more complex features as the data progresses through the layers; the first layer learns edges, the second layer learns shapes, the third layer learns objects, the fourth layer learns eyes, and so on.

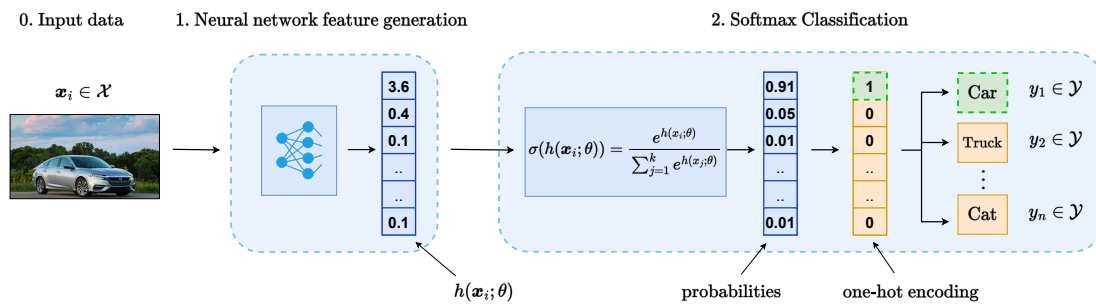


Figure 2.8: Adapted version of our pipeline with feature vector $h(x; \theta)$.

Ideally, after a neural network has been trained well enough, it will likely generate a feature vector $h(x)$ that captures the semantic information of an image accurately. That is, when seeing a car, the values in $h(x)$ for the hypothetical features 'fur', 'ears', 'legs' should be low, while they should be high for features like 'metal', 'wheels', etc. Therefore the output probabilities $\sigma(h(x))$ calculated by the softmax will likely be highest for the class label $y = \text{'car'}$. High quality feature generation will be of paramount importance to our pipeline and will further be discussed in [Section 4.2](#).

2.4.2. Loss function & Backpropagation

Training a neural networks starts with feeding the network with example input-output pairs (training data). The network's initial predictions will contain errors at first, therefore optimisation is required before the network can be deployed. To check whether a prediction \hat{y} made by the softmax is actually equal to the true label $y^* \in \mathcal{Y}$, loss functions \mathcal{L} are used.

Such a loss function is used to train the model through an optimisation algorithm ([Section 2.4.3](#)) and quantifies the model's loss; how close the model's outputs are from the true/desired values. Various loss functions exist [22] and they can be as simple as the MSE ($\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i^*)^2$) but the selection of such a function depends on the use case. Since our use case corresponds to a multi-class classification problem we use the (categorical) cross-entropy as a loss function which is a common and SOTA choice [23, 24] it is defined as:

$$\mathcal{L}_{\text{CE}}(y^*, \hat{y}) = - \sum_{i=1}^N y_i^* \ln(f(\hat{y}_i)) = - \sum_{i=1}^n y_i^* \ln \left(\frac{e^{\hat{y}_i}}{\sum_{j=1}^n e^{\hat{y}_j}} \right), \quad (2.6)$$

where y_i^* are the true labels (ground truths) and \hat{y}_i are the predicted scores for each class i in the n possible classes. The function f refers in our case to the softmax activation function that is applied to the predicted scores at the final layer of the model. The cross entropy loss is derived from the KL-divergence¹ and shows how far away a prediction is from the ground truth. Since the ground truth labels $y^* \in \mathcal{Y}$ are usually one-hot

¹See [Section 4.2.2](#) for more information.

encoded, $y_i^* \in \mathcal{Y}$ contains only one non-zero element and the loss function therefore reduces to:

$$\mathcal{L}_{\text{CE}}(y^*, \hat{y}) = -\ln \left(\frac{e^{\hat{y}_i}}{\sum_{j=1}^n e^{\hat{y}_j}} \right) \quad (2.7)$$

The goal now is to minimise this loss before this network is deployed at test time, e.g. for what values of the network parameter (weight and biases, [Section 2.2.2](#)) does the network make a prediction for which the loss is the smallest? To find out, we need derivatives of the loss function with respect to these parameters, which are then *backpropagated* through the network in order to adjust these weight and biases.

The process above is called backpropagation because one moves "from back to front"; we use the chain rule to calculate the derivative from the last layer which is the one directly connected to the loss function, all the way to the first layer, which is the one that takes the input data. For the (categorical) cross-entropy loss the derivative with respect to the outputs \hat{y}_n is as follows:

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{y}_n} &= \frac{\partial}{\partial \hat{y}_n} \left[-\ln \left(\frac{e^{\hat{y}_i}}{\sum_{j=1}^n e^{\hat{y}_j}} \right) \right] \\ &= -\frac{\partial}{\partial \hat{y}_n} [\hat{y}_i] + \frac{\partial}{\partial \hat{y}_n} \left[\ln \left(\sum_{j=1}^n e^{\hat{y}_j} \right) \right] \end{aligned}$$

The first term of the derivative is

$$\frac{\partial \hat{y}_i}{\partial \hat{y}_n} = \begin{cases} 1 & \text{if } i = n \\ 0 & \text{otherwise} \end{cases} = \mathbb{1}_{\{i=n\}}$$

For the second part,

$$\begin{aligned} \frac{\partial}{\partial \hat{y}_n} \left[\ln \left(\sum_{j=1}^n e^{\hat{y}_j} \right) \right] &= \frac{1}{\sum_{j=1}^n e^{\hat{y}_j}} \cdot \frac{\partial}{\partial \hat{y}_n} \left[\sum_{j=1}^n e^{\hat{y}_j} \right] \\ &= \frac{e^{\hat{y}_n}}{\sum_{j=1}^n e^{\hat{y}_j}} \end{aligned}$$

Combining both, results in

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{y}_n} &= \mathbb{1}_{\{i=n\}} + \frac{e^{\hat{y}_n}}{\sum_{j=1}^n e^{\hat{y}_j}} \\ &= \mathbb{1}_{\{i=n\}} + f(\hat{y}_n) \end{aligned} \quad (2.8)$$

Now that we have an expression of the derivative of \mathcal{L}_{CE} with respect to any of the outputs \hat{y}_i of the network, one can obtain the derivative of \mathcal{L}_{CE} with respect to any particular weight $w_{i,j}^\ell$ by using the chain rule of differentiation:

$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_{i,j}^\ell} = \sum_{i=1}^n \frac{\partial \mathcal{L}_{\text{CE}}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial w_{i,j}^\ell} \quad (2.9)$$

Recall from [Eq. \(2.4\)](#) that at each layer we essentially compute a function of the form $y = \sigma(\mathbf{W}^\ell \mathbf{x} + \mathbf{b}^\ell)$, the inner derivatives per layer (second product term in [Eq. \(2.9\)](#)) are therefore given by

$$\frac{\partial y}{\partial w_{i,j}^\ell} = \sigma'(\mathbf{W}^\ell \mathbf{x} + \mathbf{b}^\ell) \mathbf{x}$$

In short, backpropagation is an automatic differentiation algorithm [\[25\]](#) that calculates the derivatives of \mathcal{L} w.r.t. the weights in the network. It does not minimise \mathcal{L} yet, that is why we need an optimisation algorithm which will be treated in the next subsection.

2.4.3. Optimisation

Even simple MLPs, that classify low-resolution 28x28 images like handwritten digits (Fig. 2.2) will have 784 input neurons. As an example, take the amount of hidden layers to be 2 with only 16 neurons each, and 10 output neurons (as there are 10 digits to be classified), already leads to $784 \times 16 + 16 \times 16 + 16 \times 10 = 12960$ weights, and $16 + 16 + 10 = 42$ bias terms, making a loss function of a total of 13,002 parameters. For simplicity we collect all parameters per layer in $\Theta := (\theta^0, \dots, \theta^L)$ so we can denote the loss as $\mathcal{L}(\Theta)$.

Gradient Descent: How Neural Networks Learn

Finding the minimum of the loss function explicitly by direct calculation the derivatives and setting those to zero is a nontrivial task for a function with such a vast amount of parameters. In a typical loss landscape Fig. 2.9, it is evident that there are multiple local minima one can find depending on the starting point. The gradient descent algorithm is an iterative method that finds the global minimum by checking when \mathcal{L} decreases the most quickly. Note that since the gradient vector ∇ of a multi-variable function gives the direction of steepest increase, the negative gradient $-\nabla\mathcal{L}(\Theta)$ gives the steepest descent for our loss.

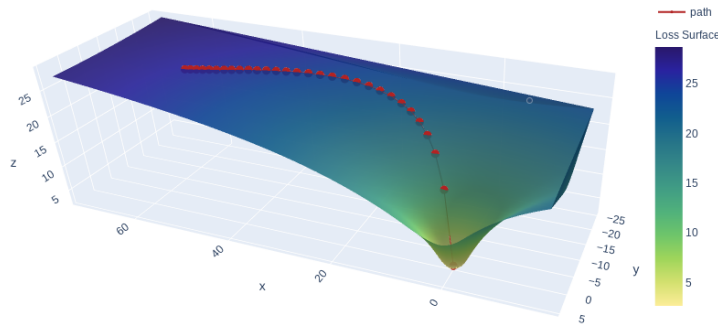


Figure 2.9: Gradient descent landscape where the loss is graphed and traversed from a random starting point $\mathcal{L}(\Theta^0)$ to find a minimum, taken from [26].

Keep in mind that we are only interested in minimising the loss, because we want to find a set of weights & biases collected in $\Theta := (\theta^0, \dots, \theta^L)$ that makes the network perform best. So given a randomly selected starting point Θ^0 , we want to 'descend' from $\mathcal{L}(\Theta^0)$ to a position $\mathcal{L}(\Theta^1)$ that is lower. The gradient descent algorithm iteratively calculates the next point using the gradient vector at the current position. The obtained value $\nabla\mathcal{L}(\Theta)$ is subtracted from the current position and therefore makes a step into the direction where $\mathcal{L}(\Theta)$ decreases fastest. The whole process is captured by the following:

$$\Theta^{(i+1)} = \Theta^{(i)} - \eta \nabla_{\Theta} \mathcal{L}(\Theta^{(i)}) \quad (2.10)$$

As opposed to other methods like Newton-Raphson (which requires the Hessian matrix and therefore ∇^2), gradient descent only requires the gradient, making it a first-order method that optimises $\Theta^{(i)}$ at each step. Note that, for $\Theta = (\theta^0, \dots, \theta^L)$, we update according to

$$\theta_L^{(i+1)} = \theta_L^{(i)} - \eta \nabla_{\theta_L} \mathcal{L}(\Theta^{(i)})$$

The parameter $\eta > 0$, known as the *learning rate*, scales the gradient and thus controls the step size. Choosing η too small causes slower convergence or may reach the maximum iteration before reaching the optimal point. Conversely, if η is too large the algorithm may not converge to the optimal point as it would 'jump' around near it, or even to diverge completely. This makes η a major component in the algorithm.

Stochastic Gradient Descent: How Neural Networks Learn Faster

In practice it takes computers a tremendous amount of time to calculate the influence of every single training example (which easily can be in the thousands of images) in every single gradient descent step. While true gradient descent involves backpropagation for all training examples and averaging them out, stochastic gradient descent (SGD) does this only for one random point on each iteration causing the updates to have a

higher variance, but also faster convergence. As a result, from a computational perspective, it performs much better in large-scale data-sets and is therefore widely used as the optimisation algorithm in deep learning. Suppose, for each iteration i , we have an unbiased estimator $\mathbf{G}^{(i)}$ of $\nabla \mathcal{L}(\Theta^{(i)})$, i.e.,

$$\mathbb{E}[\mathbf{G}^{(i)} \mid \theta^0, \dots, \theta^L] = \nabla_{\theta_L} \mathcal{L}(\Theta^{(i)})$$

Each stochastic gradient descent iteration is then give by

$$\theta_L^{(i+1)} = \theta_L^{(i)} - \eta \mathbf{G}^{(i)}$$

2.4.4. Regularisation

Like any machine learning model, neural networks too are prone to overfitting. Overfitting happens when the network optimises very well on the training set, to such an extent that it does not learn right features and does not generalise on new data at test time. For this reason, common practice in neural networks is the incorporation of regularisation of the weight parameters. The regularised loss function will then be

$$\tilde{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \lambda \Gamma(\theta) \quad (2.11)$$

where λ is a positive scalar and $\Gamma(\theta)$ is a function penalising the value of the weights. The biases should not be penalised because that can lead to underfitting [12]; biases typically require less data to fit accurately than the weights. As each bias controls only a single variable, we do not induce too much variance by leaving the biases unregularised. It is common to penalise the values of the weights by the L_1 or L_2 -norm, adding a small term induces bias which in turn decreases variance which is called the bias-variance trade-off.

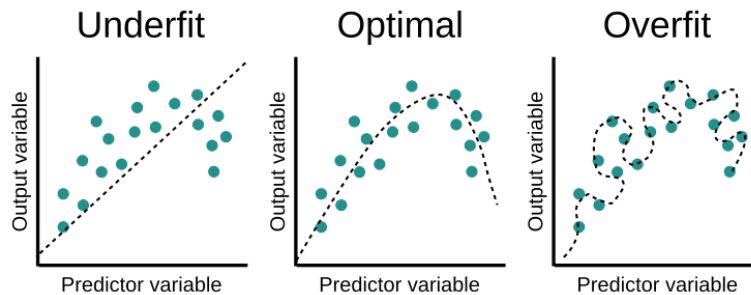


Figure 2.10: Three cases of model behaviour that can occur after training [27].

The unbiasedness of the network is traded to reduce the variance, by adding parameters. One should be careful though, as adding more network parameters can causes another descent in the loss after the bias-variance trade-off has already been taken into account, known as double descent [28].

In summary:

- Backpropagation is an automatic differentiation algorithm that calculates the derivatives of the loss w.r.t. the weights in a neural network. Its goal is to obtain what relative proportions in a change in weights and biases, causes the most rapid decrease in loss.
- Stochastic gradient descent is an optimisation algorithm for minimising the loss by using the gradients calculated by backpropagation to optimise the network parameters Θ .
- Backpropagation and SGD are used together to train neural network models.

Now that we have established the elementary building blocks and operational dynamics of computer vision/neural networks from literature, we can explore the related work done on use case where an image x belongs to a label $\bar{y} \notin \mathcal{Y}$, i.e. images of classes that are not part of the training set and therefore unknown. The next chapter deals with the examination of related work and provides an insight into how machine learning models like neural networks are used to classify classes that are not part of the training set.

3

Related Work: OOD detection

The starting point of this thesis was to study a problem in which unknown classes in images could be detected by a robot. We were given the freedom to find our own topic and build the thesis based on that, meaning that the specific problem and direction were not defined before starting the study on related work. This chapter is therefore also on finding a thesis topic before studying the work related to that topic, and for that reason it is structured into three sections with the following goals.

Chapter goals

1. **Finding a topic by exploring techniques to detect unknowns** - [Section 3.1](#)
We answer the first research question by achieving this as we look for the most suitable category of techniques that detect unknowns. After studying a taxonomy of detection techniques we narrow down our focus to OOD detection.
2. **Exploring the selected topic (OOD detection)** - [Section 3.2](#)
OOD detection has been around for a few years and new techniques are being developed as we speak. Through this section we delve into the technical details of OOD detection and find the most promising results from existing work that can help solve our problem.
3. **Analysing Research Gaps & Formulating Scientific Niche** - [Section 3.3](#)
After determining that OOD detection will be used in this thesis and studying a selection of works in OOD, we look for novel ways in which we can solve our problem while contributing to this field.

3.1. Techniques for detecting unknowns

Numerous methods to detect unknown samples exist in machine learning and engineering [29, 30, 31]. Depending on the use case, the unknown sample is at times called an *anomaly*, an *abnormality* or a *novelty*, but both are often used interchangeably. In fact, even the names of the various techniques to detect unknowns are used interchangeably, causing confusion among researchers [32]. After a quick exploration of the field of detection for unknown classes, we have found that all methods to tackle problems like our use case in [Section 1.1.1](#) can be categorised into four main approaches:

Table 3.1: The 4 main categories for detecting unknowns.

Anomaly Detection (AD)
Novelty Detection (ND)
Open-Set Recognition (OSR)
Out-of-Distribution Detection (OOD)

What causes further complications is that the techniques in [Table 3.1](#) have shared concepts, with similar goals, but have been investigated independently. This independent research has led researchers to use the

names of these techniques interchangeably despite subtle and/or fundamental differences between them, e.g. *technique A* in *paper X* is called *novelty detection*, but is in fact the same as *technique B* in *paper Y* which is called *anomaly detection*. This has caused a great deal of confusion among practitioners, thus a proper taxonomy has to be developed in order to select the method that suits our purpose best.

3.1.1. Taxonomy

Previous literature studies [30, 31, 32, 33] have attempted to clear the confusion on the techniques in Table 3.1. After carefully studying these works, we have tried to reach or create some general consensus between them, and based on that we have found that these techniques are distinguished by two principal factors:

- the occurrence of **distribution shifts**;
- the capability of classifying **single** or **multiple** classes.

We will treat distribution shifts in detail in Section 3.1.2 before covering the properties of the detection technique themselves in Section 3.1.3 and making a final selection. This subsection serves as a point of reference for the reader as it contains a taxonomy of the whole field. We have decided to present this taxonomy prior to the analysis of all techniques, as it will support the reader in recognising the topics and therefore also in getting a solid grasp of the material.

Table 3.2: Taxonomy of AD/ND/OSR/OOD based on [32].

Covariate Shift <i>Single class</i>	Semantic Shift	
	<i>Single class</i>	<i>Multi-class</i>
Covariate Anomaly Detection	Semantic Anomaly Detection	
	One-class Novelty Detection	Multi-class Novelty Detection
		Open-Set Recognition
		Out-of-Distribution Detection

Table 3.2 helps us realise immediately, even before analysing any of the techniques in Table 3.1, that only the techniques under semantic shift will be of significance to our thesis since our case is a multi-class problem. In the next section we treat the distinguishing factors in this taxonomy so that Section 3.1.3 can expand further on the techniques themselves.

3.1.2. Distribution Shift

Distribution shift is the phenomenon when the joint distribution $P_{\mathcal{X}\mathcal{Y}}$ of the input set \mathcal{X} and output set \mathcal{Y} , differs between training and test stages. As it is also one of the determining factors that distinguish the techniques above from each other [32], it is helpful to cover this first. Within the scope of machine learning, there are two general types of distribution shifts that impact the above techniques:

1. *covariate* shift (also called *sensory* shift);
2. *semantic* shift (also called *label* shift).

To illustrate this, let \mathcal{X} be the input (covariate /sensory) and \mathcal{Y} the output (semantic/label) spaces. Also let the input data be realisations of the joint distribution $P_{\mathcal{X}\mathcal{Y}}$. A distribution shift, like in Fig. 3.1, occurs either due to a change in the marginal distributions $P_{\mathcal{X}}, P_{\mathcal{Y}}$, or both.

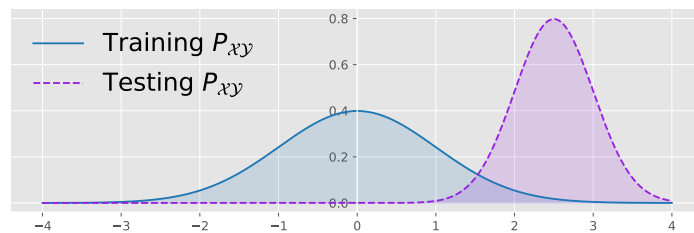


Figure 3.1: Shift in joint probability distribution of the data, due to a change in $P_{\mathcal{X}}$ and/or $P_{\mathcal{Y}}$ at test time.

Let the distribution of inputs (covariates) be denoted by $P_{\mathcal{X}}$ during training time, and $P_{\mathcal{X}}$, and $P'_{\mathcal{X}}$ during testing. The notation is equivalent for the distribution of outputs (labels) $P_{\mathcal{Y}}, P'_{\mathcal{Y}}$.

Definition 3.1.1 (Covariate shift) We speak of covariate shift when the distribution of inputs $P_{\mathcal{X}}$ can change over time, while the labelling function $P_{\mathcal{Y}|\mathcal{X}}$ does not change. In that case, the 'normalities' (known classes) are from $P_{\mathcal{X}}$ and the 'abnormalities' (unknowns) from $P'_{\mathcal{X}}$. It is called covariate shift because it is caused by a shift in the distribution of the covariates (inputs/features): $P_{\mathcal{X}} \neq P'_{\mathcal{X}}$.

Among distribution shifts, covariate shift is the most widely studied and Fig. 3.2 demonstrates why this phenomenon can be so significant.

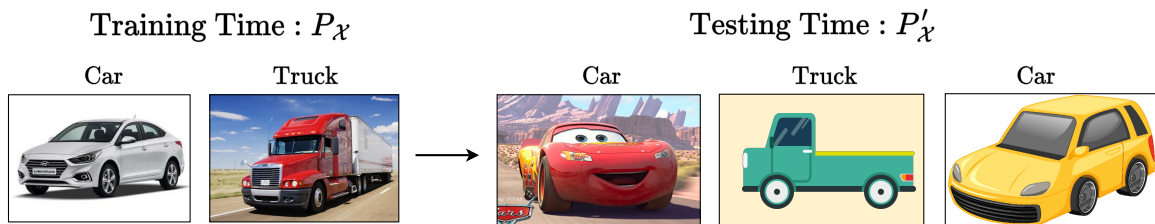


Figure 3.2: Shift in marginal distribution $P_{\mathcal{X}}$, for a classifier trained on cars & trucks. Classes stay the same so $P_{\mathcal{Y}}$ does not change. Input samples at training time (left), input samples at testing time (right).

While the training data is comprised of photos, the test data now contains only cartoons. This implies that training is done on a data-set with substantially different characteristics/features from the test set, leading to higher errors as the label space \mathcal{Y} has not changed. This is an example of domain and style shift within covariate shift, another case of covariate shift is *adversarial examples*, where training data is augmented with noise [1], also leading to the same classes at testing time, but with a difference in appearance.

Definition 3.1.2 (Semantic shift) We speak of semantic shift when the distribution of outputs $P_{\mathcal{Y}}$ can change over time. The labels \bar{y} of classes at testing time are now not in the label space \mathcal{Y} , but in \mathcal{Y}' as they have not been trained on. In that case, the 'normalities' (known classes) are from $P_{\mathcal{Y}}$ and the 'abnormalities' (unknowns) from $P'_{\mathcal{Y}}$. It is called semantic shift because it is caused by a shift in the distribution of the semantic labels (outputs): $P_{\mathcal{Y}} \neq P'_{\mathcal{Y}}$.

Within the setting of the previous example, this means that the classifier attempts to go beyond only classifying cars and trucks and aims to detect new classes (Fig. 3.3). It is precisely this shift that is of utmost relevance to our research since we do not expect to see cartoons in a surveillance/security setting (so no change in $P_{\mathcal{X}}$), while we do expect to see unknown vehicle classes (change in $P_{\mathcal{Y}}$).

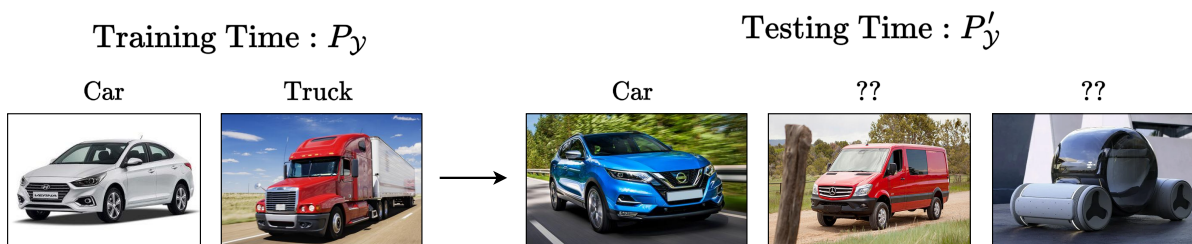


Figure 3.3: Shift in marginal distribution $P_{\mathcal{Y}}$, for a classifier trained on cars & trucks. Classes at testing time are now different; the van (second image) and futuristic vehicle (third image) belong to a label $y \notin \mathcal{Y} \Rightarrow$ change in $P_{\mathcal{Y}}$.

In Fig. 3.3, the distribution of our label space \mathcal{Y} , now changes when we are presented an unfamiliar/unknown class (van and futuristic vehicle), as our model does not have a label for it yet.

3.1.3. Comparison & Selection of Detection Technique

The techniques in our taxonomy all detect an unknown sample in one way or another. The notion on what is 'unknown' is based on some 'predefined normal behaviour' that a new sample deviates from, but its formal definition would depend on the metric that is used to measure it:

- **Density-Based:** model the normal/training data with probabilistic models and flag test data as an 'unknown' when it is in low-density regions of this model.
- **Distance-Based:** calculate the distance (e.g. L_p -norm) between training (normal) data, and testing data. If the distance exceeds a threshold, the sample is 'unknown'.

Definition 3.1.3 (Density-based 'unknown') An 'unknown' is an observation $x \in \mathcal{X}$ that deviates substantially from the predefined normal behaviour/ground truth P_N on \mathcal{X} , i.e. lies in a low probability region under P_N . Let P_N have a corresponding probability density function $p_N(x)$, the set of 'unknowns' can then be defined as follows

$$\mathcal{U}_p = \{x \in \mathcal{X} \mid p_N(x) \leq \tau\}, \quad \tau \geq 0, \quad (3.1)$$

where the probability $p_N(x)$ of \mathcal{U} under P_N , can be arbitrarily small.

Definition 3.1.4 (Distance-based 'unknown') An 'unknown' is an observation $x \in \mathcal{X}$ that deviates substantially from the predefined normal behaviour/ground truth set of training samples $N \subset \mathcal{X}$, i.e. lies far away from the centroid of N : μ_N . Let d be a distance metric defined on \mathcal{X} the set of 'unknowns' can then be defined as follows

$$\mathcal{U}_d = \{x \in \mathcal{X} \mid d(x, \mu_N) \leq \tau\}, \quad \tau \geq 0, \quad (3.2)$$

where the distance $d(x, \mu_N)$ can be arbitrarily small.

Based on the taxonomy, it is clear which techniques fall under semantic shift multi-class classification and are therefore qualified for our research: **Semantic Anomaly Detection**, **Multi-class novelty detection**, **Open-Set Recognition**, and **OOD detection**. Fig. 3.4 shows that for the multi-class case, the four techniques are essentially equivalent, so why should one favour one over the other? We will very briefly go over each technique and finally conclude why we select OOD for our research.

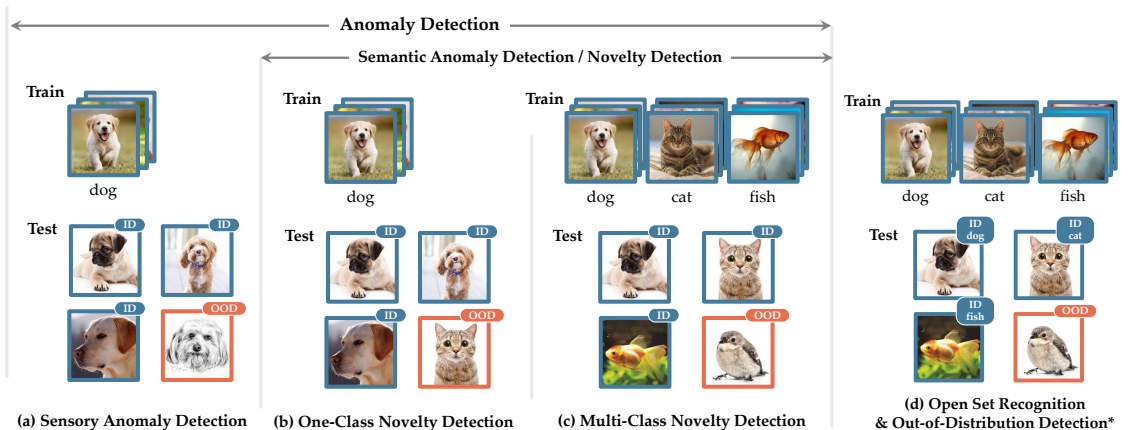


Figure 3.4: Each detection technique with image examples, adapted from [32]. For the multi-class case we see that all four techniques are (almost) equivalent.

Anomaly & Novelty Detection

Anomaly Detection (AD) and Novelty Detection (ND) are identical. Researchers tend to use the word 'anomaly' whenever the sample that should be detected is 'unwanted' or 'contrasting', like noisy images in surveillance, or strange behaviour in fraud detection. The term 'novelty' is generally used for detecting something unknown, new, and *interesting*.

Recent works in AD [30, 31] have shown that their focus is mostly on covariate shift problems using Variational Auto-encoders (VAE) [34, 35] or generative models like Generative Adversarial Networks (GANs) [36, 37]. The main issue with these techniques is that they work well for pixel-level detection of anomalies (differences

in patterns/pixels), rather than semantic level (understanding what the anomaly is) [31], while works in novelty detection focus on “novel class detection” [38, 39], indicating that it is focusing on detecting semantic shift. Both are interchangeable, but novelty detection appreciates new classes as resources for potential future use with a positive learning attitude, which aligns with our work.

Open-Set Recognition

Most machine learning models are trained with the assumption that the data at test time is assumed to be drawn i.i.d. from the same distribution as the training data, this is called the *closed-world assumption*. Open-Set Recognition (OSR), first coined by Scheirer et al. [40], acknowledges that models can encounter classes outside of their training set. It is almost equivalent to ND, except for the fact that OSR has increased supervision, categorised as follows:

- **Known Known Classes (KKC):** Training samples that we know they are known. They are already given and labeled.
- **Known Unknown Classes (KUC):** Training samples that we know they are not known: they do not belong to the known categories. E.g. background images, or any image that we know is not categorized into the known classes are in this group. They are already given and labeled.
- **Unknown Known Classes (UKC):** Training samples that we do not know they are known. For example, known test time samples are in this group. These are not given at the training phase.
- **Unknown Unknown Classes (UUC):** Training samples that we do not know they are not known: unknown test time samples are in this group. These are not given at the training phase.

If a sample is flagged as 'known', OSR formally requires further correct classification of it, e.g. after a car image is flagged as 'known', it should subsequently be further classified as 'car'. This is a stronger condition that is not present in the other techniques, and also not a requirement for our use case.

Out-of-distribution detection

While the related and sometimes equivalent counterparts of OOD detection like AD, ND and OSR have been around longer, the term of 'out-of-distribution detection' (OOD) first formally emerged in 2017 for a solution specifically made for neural networks [41]. Nguyen, Yosinski, and Clune pointed out in the paper "Deep Neural Networks are Easily Fooled" [4], that neural networks make notoriously high confidence predictions when presented OOD input.

Nguyen, Yosinski, and Clune, realised that the false predictions (with high confidence) of anomalous data by neural networks was caused by the softmax function of the neural network [4]. As shown in Fig. 2.7, when presented a new test sample x , the softmax computes a probability of how likely it is that the sample belongs to each class $y_1, \dots, y_k \in \mathcal{Y}$.

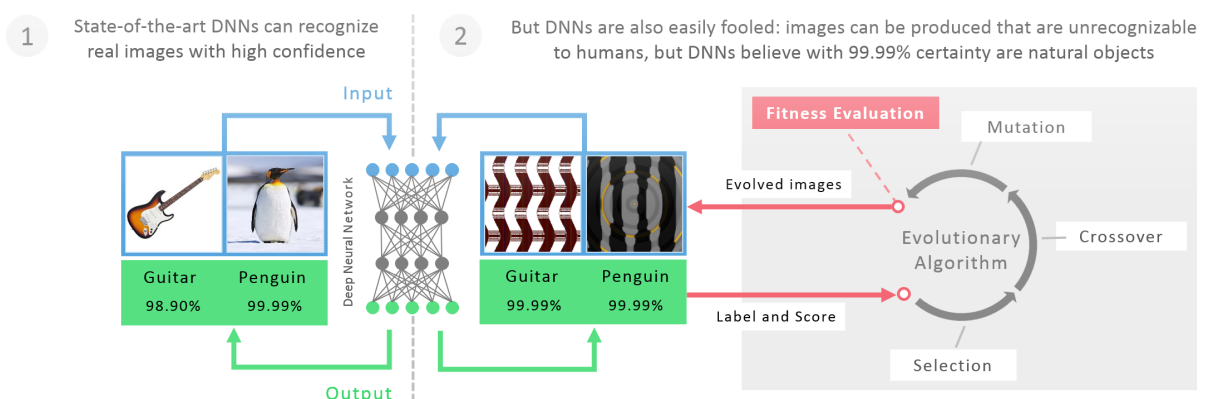


Figure 3.5: Neural networks can be fooled by anomalous data; making them believe that noise is a penguin. Taken from [4].

Since these probabilities are computed with the fast-growing exponential function, minor additions to the softmax inputs, can lead to considerable changes in the output distribution, hence the overconfident predictions of neural networks. Hendrycks and Gimpel pursue this issue [41] and conclude that softmax classifier probabilities are not directly useful as confidence estimates, but also show that simple statistics derived from

softmax distributions can still provide an effective way to determine whether an example is misclassified, or from a different distribution. Their proposed solution to this, among other OOD techniques, will be discussed in [Section 3.2](#), where we delve into the technical details and previous studies on OOD.

Rationale for OOD.

For our use case we need semantic shift and multi-class classification, but we have seen in the previous sections that multi-class AD/ND, or OSR in some way, already suits our purpose. So does OOD. But a reason to favour OOD above the others is that OOD detection is a more generalised category, encompassing both AD, ND, and OSR according to Yang et al. in [32], giving us an extra degree of freedom. Additional and more specific motivations for selecting OOD for this thesis are:

- OOD techniques in the papers we will see in the next section are specifically designed to work well with neural network architectures. Since we will definitely leverage the power of deep learning in this thesis, rather than rely on classical statistical models, this is useful to us.
- OOD is relatively new, expanding as we speak, and full of potential. This makes it an inherently more exciting topic for us.

Having said this, the first direction of this thesis has been decided and all formalisms and techniques will be based on that direction. But funnily enough, since we are genuinely interested in the new classes we will encounter, we are in fact also doing a form of novelty detection. Furthermore, in our use case we go beyond the closed-world assumption and acknowledge that not all information about the world is known, which implies that we research an open-world problem. And finally, our theoretical framework will be formally based on the theory of OOD detection, specifically for neural networks. All of this combined implies that we will be researching: **Novelty detection**, in an **open world setting**, using **OOD** techniques.

3.2. Exploring OOD Detection

In spite of being a relatively new subject (under this name), much work has already been and is being done on OOD detection. Especially considering the fact that techniques from ND,AD, and OSR, also fall within the conceptual framework of OOD detection and have been around much longer. In this section we concisely present 5 insights, divided over 5 subsections, from recent work that is relevant for this thesis. These results serve either as a starting point, or inspiration for our scientific niche.

All results will be applied to our use case and setting, where we use neural networks to classify image data. The main idea behind OOD for neural networks, as denoted in the previous subsection, is to find an alternative for the softmax function such that OOD samples will not be classified as 'known' with high confidence.

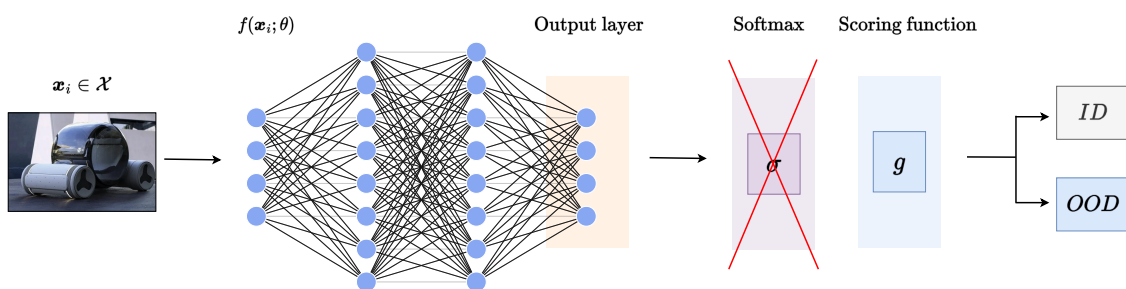


Figure 3.6: OOD detection for neural networks; replacing the softmax function σ with a scoring function g to classify input as 'ID' or 'OOD'.

The next subsections will deal with various options for g and things we need to take into account when building our solution.

3.2.1. Insight 1: Formalised OOD detection

Paper	Author & Reference	Year
Provable Guarantees for Understanding Out-of-distribution Detection	Morteza and Li [42]	2021

While Hendrycks and Gimpel were the first to set up a direct solution for OOD detection with neural networks in 2016, and many works have been done even before that under a different name than OOD detection, Morteza and Li were the first to formalise the theory of OOD in [42].

Recall the classification framework of Section 2.3 where \mathcal{X} is the input space and \mathcal{Y} is the label space. Let $P_{\mathcal{X}\mathcal{Y}}^{\text{in}}$ the data-generating distribution defined over $\mathcal{X} \times \mathcal{Y}$ of the in-distribution (ID) data, and $P_{\mathcal{X}}^{\text{in}}, P_{\mathcal{Y}}^{\text{in}}$ the joint and marginal distributions. Given a classifier $f: \mathcal{X} \rightarrow \mathcal{Y}$ that learns to map a given input to the label space, the goal is to determine whether a new sample $\mathbf{x} \in \mathcal{X}$ at test-time, is generated from $P_{\mathcal{X}}^{\text{in}}$ or not.

I.e. given a classifier f learned on training samples from in-distribution $P_{\mathcal{X}\mathcal{Y}}^{\text{in}}$, the goal is to design a binary function estimator

$$g: \mathcal{X} \rightarrow \{ \text{in}, \text{out} \}$$

that tells us whether a new sample is in-distribution (ID) or out-of-distribution (OOD) [42].

To determine the nature of a new test sample \mathbf{x} , both the marginal distribution $P_{\mathcal{X}}^{\text{in}}$ and/or $P_{\mathcal{X}}^{\text{out}}$ can theoretically speaking be used. However, due to the lack of knowledge on the OOD data, and therefore $P_{\mathcal{X}}^{\text{out}}$, we are naturally compelled to base our scoring function g on the probability density $P_{\mathcal{X}}^{\text{in}}$.

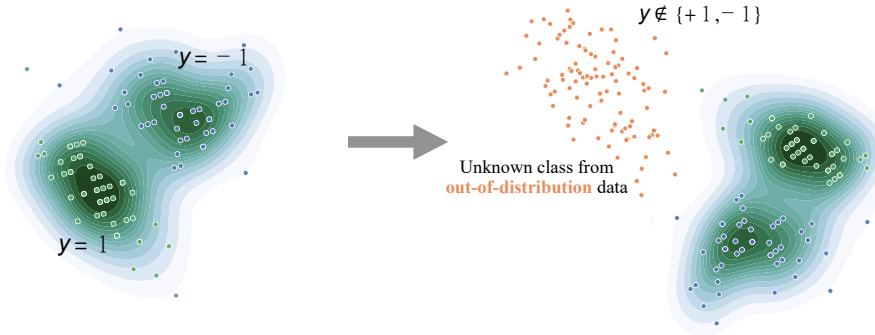


Figure 3.7: On the left, the ID data distribution $P_{\mathcal{X}}^{\text{in}}$ comprises of two classes $\mathcal{Y} = \{-1, +1\}$, indicated by green and blue dots respectively. On the right, new unknown data is presented. In OOD detection, the model should label these instances (orange dots) as 'OOD', rather than misclassifying them into known classes (blue and green dots). [42]

Definition 3.2.1 (Ideal Classifier) An ideal classifier for OOD detection based on the data density $P_{\mathcal{X}}^{\text{in}}$ is defined as follows

$$g_{\lambda}^{\text{ideal}}(\mathbf{x}) = \begin{cases} \text{in} & p_{\mathcal{X}}^{\text{in}}(\mathbf{x}) \geq \lambda \\ \text{out} & p_{\mathcal{X}}^{\text{in}}(\mathbf{x}) < \lambda \end{cases}$$

where $p_{\mathcal{X}}^{\text{in}}$ is the density function of $P_{\mathcal{X}}^{\text{in}}$ and λ is the threshold, which is chosen so that a high fraction (e.g., 95%) of in-distribution data is correctly classified. [42]

Note that this is exactly in line with our definition (Definition 3.1.3) for an OOD sample, where the density of predefined normality $p_N(\mathbf{x})$ is now defined by $p_{\mathcal{X}}^{\text{in}}(\mathbf{x})$ with threshold λ . The 95% here is due to convention; 95% is two standard deviations away from the mean and therefore often chosen, but it can be any number that suits the use case. The explicit use of the term 'ideal' is because the classifier f is trained on samples from $P_{\mathcal{X}\mathcal{Y}}^{\text{in}}(\mathbf{x})$, and it is generally speaking hard to directly obtain $p_{\mathcal{X}}^{\text{in}}(\mathbf{x})$ due to possible intractability:

$$p_{\mathcal{X}}^{\text{in}}(\mathbf{x}) = \sum_j p_{\mathcal{X}\mathcal{Y}}^{\text{in}}(\mathbf{x}, y_j) \quad (3.3)$$

$$= \sum_{j=1}^k p_{\mathcal{X}|\mathcal{Y}}^{\text{in}}(\mathbf{x} | y_j) \cdot p_{\mathcal{Y}}^{\text{in}}(y_j) \quad (3.4)$$

In practice, calculating the exact marginal distribution above becomes intractable in high dimensions, making it computationally expensive. Therefore, with [Definition 3.2.1](#) as our starting point, the main challenge in designing g lies in finding alternative scoring function that is close, or if possible, proportional to $p_{\mathcal{X}}^{\text{in}}(\mathbf{x})$.

3.2.2. Insight 2: Comparison of scoring functions

Scoring function g	Paper	Author & Reference	Year
1. MSP(f, \mathbf{x})	A Baseline for Detection Misclassified and OOD examples in Neural Networks	Hendrycks and Gimpel [41]	2016
2. M(f, \mathbf{x})	A Simple Unified Framework for Detecting OOD Samples and Adversarial Attacks	Lee et al. [43]	2018
3. E(f, \mathbf{x})	Energy-based Out-of-distribution Detection	Liu et al. [44]	2020
4. GEM(f, \mathbf{x})	Provable Guarantees for Understanding Out-of-distribution Detection	Morteza and Li [42]	2021

Ever since Hendrycks and Gimpel have paved the way to find an alternative to the softmax, various of scoring functions for neural networks have been sought and developed to improve OOD detection. With [Definition 3.2.1](#) in mind, we need to find a function that is proportional to $p_{\mathcal{X}}^{\text{in}}(\mathbf{x})$, given the output $f(\mathbf{x})$ of a classifier f . We briefly compare 4 functions that have been proposed in recent years, but note that more scoring functions are developed continuously.

1. Maximum Softmax Probability

Hendrycks and Gimpel noticed that while the softmax does falsely misclassify OOD samples as known with high probability, the probability of incorrect and OOD samples tends to be lower than the prediction probability for correct (ID) samples. Therefore, they propose to take the maximum of the softmax function [41], which in general and short notation would be $\max_i p_{\mathcal{Y}|\mathcal{X}}(y_i | \mathbf{x})$, as the softmax computes the probability of a label y_i , given an input \mathbf{x} .

$$g_{\lambda}^{\text{MSP}}(\mathbf{x}) = \begin{cases} \text{in} & \text{MSP}(f, \mathbf{x}) \geq \lambda \\ \text{out} & \text{MSP}(f, \mathbf{x}) < \lambda \end{cases} \quad \text{where } \text{MSP}(f, \mathbf{x}) = \max_i p_{\mathcal{Y}|\mathcal{X}}(y_i | \mathbf{x}) \quad (3.5)$$

A variation on the MSP called ODIN [45] was introduced later where *temperature scaling* was added, which is a method for recalibrating prediction probabilities [46].

2. Maximum Mahalanobis Distance

In [43], Lee et al. assumed (and later verified) that the classes of their ID data, and the features of those classes generated by a neural network, follow class-conditional Gaussian distributions. Thus, sample class means $\hat{\mu}_i$ and covariances $\hat{\Sigma}$ were computed, such that a new sample \mathbf{x} could be compared to it:

$$g_{\lambda}^{\text{M}}(\mathbf{x}) = \begin{cases} \text{in} & \text{M}(f, \mathbf{x}) \geq \lambda \\ \text{out} & \text{M}(f, \mathbf{x}) < \lambda \end{cases} \quad \text{where } \text{M}(f, \mathbf{x}) = \max_i \sqrt{(f(\mathbf{x}) - \mu_i)^{\top} \Sigma^{-1} (f(\mathbf{x}) - \mu_i)} \quad (3.6)$$

The Mahalanobis distance $\sqrt{(f(\mathbf{x}) - \mu_i)^{\top} \Sigma^{-1} (f(\mathbf{x}) - \mu_i)}$ is a measure that compares a sample to a known distribution, more on this will be treated in [Chapter 4](#).

3. Energy Score

The energy score [44] was derived from Energy Based Modelling (EBM), which is a field of its own and beyond the scope of this thesis. For now, it suffices to know that EBM uses an energy function $E(\cdot)$ from statistical mechanics, that assigns high values (energies) to unobserved input, and low energies to observed input, making it a contender for OOD detection. More details on EBM and the choice of $E(\cdot)$ in [47].

$$g_{\lambda}^{\text{E}}(\mathbf{x}) = \begin{cases} \text{in} & \text{E}(f, \mathbf{x}) \geq \lambda \\ \text{out} & \text{E}(f, \mathbf{x}) < \lambda \end{cases} \quad \text{where } \text{E}(f, \mathbf{x}) = -\log \sum_{j=1}^k \exp(f_j(\mathbf{x})) \quad (3.7)$$

4. GEM Score

Gaussian mixture based Energy Measurement (GEM) [42], combines the Mahalanobis distance with the Energy score as follows:

$$g_{\lambda}^{\text{GEM}}(\mathbf{x}) = \begin{cases} \text{in} & \text{GEM}(f, \mathbf{x}) \geq \lambda \\ \text{out} & \text{GEM}(f, \mathbf{x}) < \lambda \end{cases} \quad \text{where } \text{GEM}(f, \mathbf{x}) = -\text{E}(\text{M}(f, \mathbf{x})) \quad (3.8)$$

$$\text{That is, } \text{GEM}(f, \mathbf{x}) = \log \sum_{i=1}^k \left[\exp \left(\sqrt{(f(\mathbf{x}) - \boldsymbol{\mu}_i)^\top \Sigma^{-1} (f(\mathbf{x}) - \boldsymbol{\mu}_i)} \right) \right].$$

The scoring functions above can appear to be very abstract and non-intuitive, so one should keep in mind that, ultimately, they are used to get a score for every sample that decides whether the sample is ID or OOD. Morteza and Li have shown in [42], that the GEM function is the only one that is, to a particular degree, proportional to the ideal classifier. It is log-proportional to be precise, i.e. $\propto \log(p_{\lambda}^{\text{in}}(\mathbf{x}))$, while the other scoring functions are not proportional at all. This might lead one to think that it is superior as well, but the results say otherwise. In Table 3.3 we find the performance of all scoring functions on the CIFAR data-set, we see that the Mahalanobis and the GEM score are on par with each other.

Table 3.3: Performance of scoring functions trained on the CIFAR data-sets. All scores are averaged over multiple tests, each containing different OOD data. Test results are courtesy of Liu et al. and Morteza and Li, details can be found in [42, 44]. If one is not familiar with the performance metrics FPR, AUROC, AUPR, please see Chapter 5.

ID data-set	Method	FPR [%] ↓	AUROC [%] ↑	AUPR [%] ↑
CIFAR-10	Softmax score [Hendrycks and Gimpel, 2016]	51.04	90.90	97.92
	ODIN [Liang et al., 2018]	35.71	91.09	97.62
	Mahalanobis [Lee et al., 2018]	36.96	93.24	98.47
	Energy score [Liu et al., 2020]	33.01	91.88	97.83
	GEM [Morteza and Li, 2021]	37.21	93.23	98.47
CIFAR-100	Softmax score [Hendrycks and Gimpel, 2016]	80.41	75.53	93.93
	ODIN [Liang et al., 2018]	74.64	77.43	94.23
	Mahalanobis [Lee et al., 2018]	57.01	82.70	95.68
	Energy score [Liu et al., 2020]	73.60	79.56	94.87
	GEM [Morteza and Li, 2021]	57.03	82.67	95.66

We would like to point out that the GEM and Mahalanobis scoring functions rely heavily on the Gaussian mixture model assumptions of the data as it only needs the sample mean and variance (which is sufficient to describe a Gaussian, but not other distributions). While this is the case for the CIFAR data-set, this does not hold true for all computer vision data-sets as we shall see in Chapter 4.

3.2.3. Insight 3: OOD detection performance changes under certain parameters

Paper	Author & Reference	Year
Provable Guarantees for Understanding Out-of-distribution Detection	Morteza and Li [42]	2021
MOS: Towards Scaling Out-of-distribution Detection for Large Semantic Space	Huang and Li [48]	2021
OOD Detection in High-Dimensional Data Using Mahalanobis Distance	Maciejewski, Walkowiak, and Szyk [49]	2022

Recent work has found that the scoring functions, and therefore OOD detection, behave differently under the change of the following parameters:

1. The number of ID classes

OOD detection algorithms are challenged when the label space of the ID data increase, i.e. when the ID data-set contains a large amount of classes. The explanation is as follows: A model is trained on a certain set of ID data, samples that deviate enough from them are labelled unknown. It is impossible to precisely define and anticipate those OOD classes in advance; you often do not know which classes are unknown, otherwise you would have trained your model on it. This results in a large space of uncertainty. As the number of ID classes increases, the amount of ways that OOD data may occur increases correspondingly. In short, more ID classes trained, means more ways a new sample can be unknown, increasing the space of uncertainty and making detection more difficult.

2. The dimension of the data

A classical problem within statistics; when the amount of features (dimension of the data) increases to such an extent that it is of comparable size or larger than the amount of observations, estimations of statistical models become either unstable or do not exist [49, 50].

3. How close the ID and OOD data are from each other

This is straightforward and to be expected as classes that are semantically more different are easier to detect, e.g. if ID = car, then it is easier to detect that a frog is OOD, than a bus.

Fig. 3.9 depicts how the performance of the MSP scoring function drops rapidly as the amount of ID classes increases from 50 to 1000 [48], tested on 4 OOD data-sets: iNaturalist, SUN, Places, and Textures.

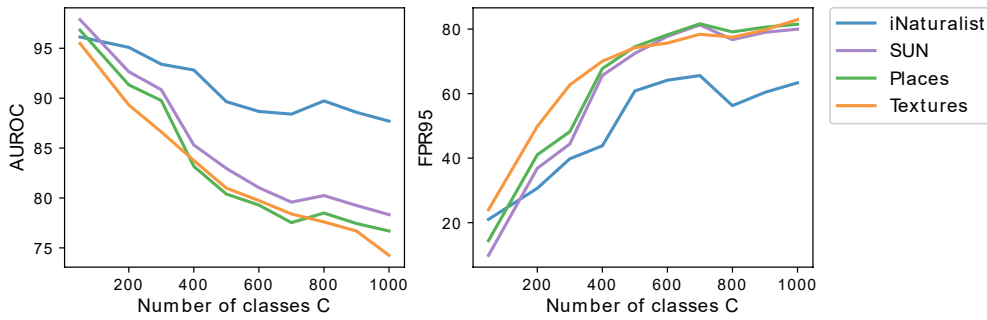


Figure 3.8: Performance of OOD detection using MSP scoring function when trained on the ImageNet-1k data-set, taken from [48]. Here iNaturalist, SUN, Places, and Textures indicate the OOD data-sets that have been used, see Fig. 3.12 for a visual overview of what these data-sets are like. High AUROC (low FPR) indicates high performance.

Fig. 3.9 showcases the performance of the GEM scoring function when all three parameters are increased incrementally. Increasing the number of classes yields to a rate of decrease in performance that is similar to the MSP: an increase of 0→100 classes leads to an increase of 0.25-0.30 in FPR.

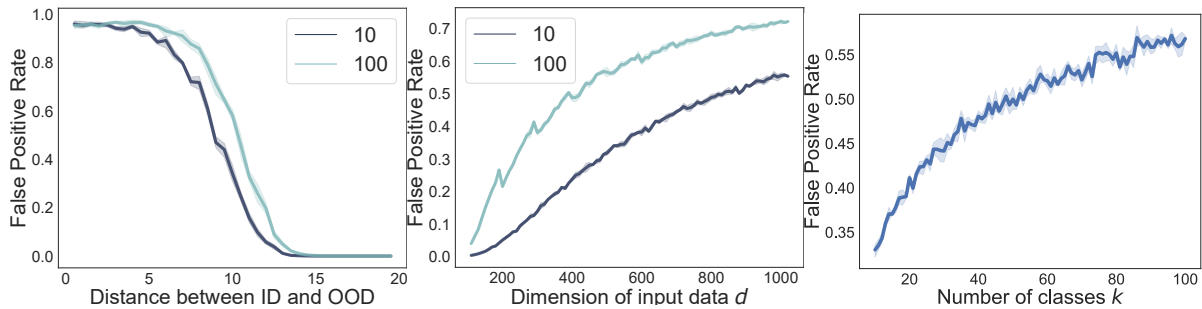


Figure 3.9: Testing has been done on the CIFAR-10 and CIFAR-100, indicated by the two lines in the first two figures [42]. Each curve is averaged over 5 different runs, with similar OOD data-sets as in Fig. 3.8.

The tests done on GEM furthermore show that as the distance between ID and OOD grows, detection performance increase, while growth in dimension does not. This behaviour is supported by the work of Maciejewski, Walkowiak, and Szyk [49] for the Mahalanobis scoring function.

These are all notable insights for choosing the amount of classes and data-set in our own experimental setup in Chapter 5.

3.2.4. Insight 4: Feeding scoring functions explicit OOD data through Outlier Exposure

Paper	Author & Reference	Year
Deep Anomaly Detection with Outlier Exposure	Hendrycks, Mazeika, and Dietterich [51]	2018

Take p_{in} to be the distribution of ID data \mathcal{D}_{in} , and p_{out} to be the distribution of OOD data \mathcal{D}_{out} . It is difficult, if not impossible in most cases, to even anticipate or successfully guess what p_{out} is in advance. The idea of Outlier Exposure (OE), is to *expose* the model to explicit OOD data from an auxiliary data-set \mathcal{D}^{OE} such that it has a better understanding of what OODs might look like and can generalise better when being presented an actual OOD sample. Hendrycks, Mazeika, and Dietterich in [51] train a model through regularisation on a small set of OOD data that one has access to beforehand, to help it learn heuristics to detect whether a sample is ID or OOD. They have found that heuristics generalise to new and unseen OOD samples at test time as well.

Below we present the main insights of this work that can be of relevance to our thesis:

1. Synthetic OODs in \mathcal{D}_{out}^{OE} do not increase performance:

A common method to quickly create an OE data-set of OOD data is to distort ID data with noise (covariate shift). Hendrycks, Mazeika, and Dietterich have attempted to do this, but it did not amount to much as the classifier f recognised a pattern. According to their experiments, it is better to use realistic data (see insight 3 below).

2. Data in the OE set should not be similar to training data

Whenever this is the case, performance is improved with marginal results only. In other words, if the ID set only contains vehicles (car, truck), exposing the model to unknown vehicles (e.g. vans), will not help the model detect an unknown military vehicle (e.g. tank).

3. \mathcal{D}_{out}^{test} and \mathcal{D}_{out}^{OE} need not be close to improve performance:

This means that the classes in our OE set do not have to be similar to the OOD classes we encounter at test time, which is great as one often does not have the slightest clue of what one will see at test time. In Hendrycks, Mazeika, and Dietterich, tests were done where the ID data consisted of house numbers, and the OOD data at test time of emojis. In the OE set, images of natural landscapes were used, which are not at all close to house numbers nor emojis.

Could this potentially mean that we can detect strange/alien vehicles by exposing our model to images of French cheese platters? Likely not, as the OOD classes we are interested in are close to our ID data, while the OE method trains the model to spot large deviations.

3.2.5. Insight 5: Prior knowledge of classes related to our OOD classes might improve detection

Paper	Author & Reference	Year
Few-Shot Learning with Intra-Class Knowledge Transfer	Roy et al. [52]	2020

The work of Roy et al. [52] is not a method we will use as it is based on GANs and covariate shift (while we look for semantic shift), but it led to an inspiring insight that we could use to develop our scientific niche. They propose a method that leverages statistical properties of a few-shot class (a class that it only seen a few times) and trained ID classes (classes it has seen) to detect OOD classes (classes it has never seen).

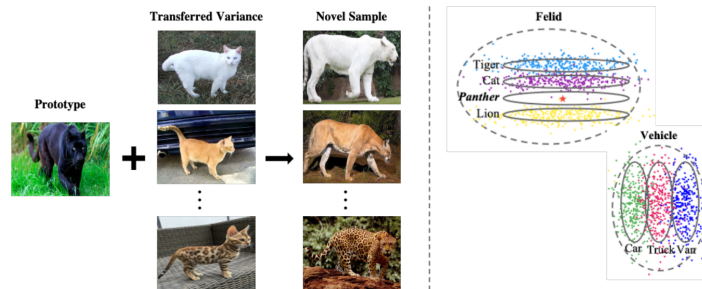


Figure 3.10: Left: Leveraging the properties of a panther (few shot class), and different types of house cats, to better detect unknowns of the felid animal class (lions, tigers, cheetahs etc.). Right: clusters of samples of each animal in feature space.[52]

Roy et al. point out that when classes are close in feature space, their statistical mean and variance are similar as well [52]. Conversely, when the classes are far away, their statistical mean and variance are different too. This means that one can leverage intra-class knowledge of neighbour classes to detect new classes.

This idea leads to two thoughts that should be explored for this thesis:

1. If we have prior information of the domain and situation our robot/drone will be operating in, e.g. detection of unknown vehicles, can we use statistical information of a prototype/few-shot vehicle to spot new unknown vehicles?
2. Can it help us distinguish OOD classes from each other? Given a prior information vehicles, can we use that to distinguish irrelevant OODs (hammers, birds) from relevant OODs (vehicles)?

3.3. Research Gaps & Possible Niche

Starting out from a real-world case, we have ventured into the world detection techniques for unknown classes (specifically OOD literature) and made a selection of papers whose contents align with our purpose. In this section we present the research gaps we have found through this study, and a scientific niche.

3.3.1. Research Gaps

1. Realistic Data-sets

First and foremost, we point out that contemporary literature mostly employs simple and crude data-sets, both for classification and for OOD detection. The CIFAR data-set (32×32 pixel images) is ubiquitous in OOD papers and is often used as the main data-set. These images are crude and contain therefore far less information resulting into less rich features. A far more realistic choice would for instance be using images of the domainnet data-set (varying resolutions; around 300×400) [53].

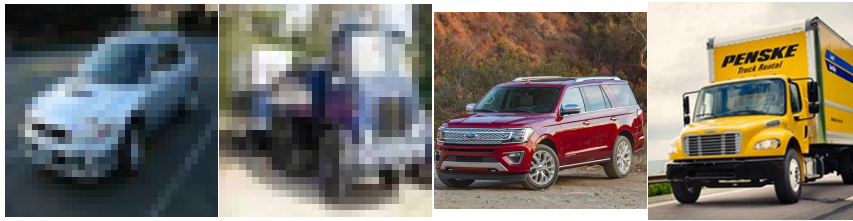


Figure 3.11: Images of cars and trucks take from the CIFAR and domainnet data-set. Left: CIFAR, right: domainnet.

In addition, OOD scoring functions/techniques are, to the best of our knowledge, always tested on OOD data-sets that are semantically different in a substantial way. The detection of plants or even textures as OOD is interesting to showcase the theoretical performance of a novel scoring function, but is not at all relevant nor realistic from an industry perspective. In our case, it would be much interesting how well OOD techniques can spot unknown vehicles, rather than an image of the Grand Canyon.



Figure 3.12: OOD data-sets used in contemporary literature, taken from [48].

2. Prior Knowledge & Relevance Detection

In real-world applications, knowledge of the problem setting and scope is known. Research on OOD detection is by definition general and therefore does not exploit this knowledge to improve detection. An OOD model could flag a 1000 samples a day as OOD, but which of these samples were actually relevant to our use case? One can imagine that an OOD plant, or OOD sky image is not as alarming as an OOD tank. In our case, a major contribution would therefore be the capability to distinguish relevant OOD from irrelevant OODs.

3. Distribution Approach

So far, OOD detection has been done on single input samples at test time. Say that a drone has ventured into new territory for a full day and has encountered a great deal of OOD samples, many belonging to the same class. It would be interesting to see if having access to multiple samples of an OOD class (multiple samples of a strange vehicle) can yield more distinguishing power between relevant and irrelevant classes. If one can treat the collection of samples per OOD class as a distribution, one can use statistical distance measures to perform inference.

3.3.2. Axioms

In order to build a proper pipeline (in the next chapter) for relevance detection, we define the following sets as a starting point.

Axiom 1 All classes c_i that will be trained and/or tested with our pipeline on come from 2 main (disjoint) sets: ID & OOD.

Axiom 2 The ID and OOD sets consist of 2 disjoint subsets, based on whether a class in this set is interesting to our use case or not.

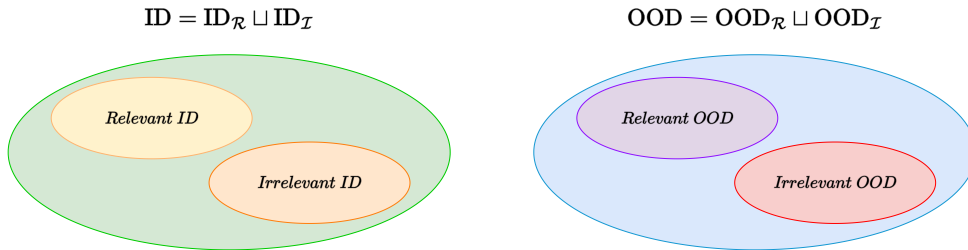


Figure 3.13

The $OOD_{\mathcal{R}}$ class is of most interest to us. Below we sketch a toy problem that is in line with our domain-specific use case, to visualise the 4 sets we will be classifying the output of our pipeline to:

- $ID_{\mathcal{R}}$: relevant classes that we have trained on.
- $ID_{\mathcal{I}}$: irrelevant classes that we have trained on.
- $OOD_{\mathcal{R}}$: relevant classes that we have not trained on.
- $OOD_{\mathcal{I}}$: irrelevant classes that we have not trained on.

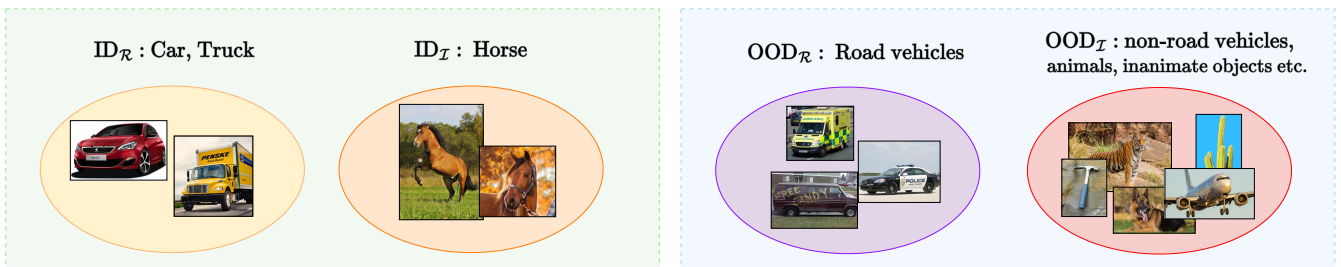


Figure 3.14

With these axioms in mind, we design and present our methodology in [Chapter 4](#).

4

Methodology & Design Challenges

Based on the research gaps and scientific niche presented in [Chapter 3](#), we construct a pipeline of the solution of thesis from scratch and we thoroughly discuss its design choices and challenges.

Chapter goals

1. **Presenting/motivating the design choices of each step of the pipeline** - [Section 4.1](#) to [Section 4.5](#)
Here we zoom in on each step of the solution pipeline to treat its design challenges/choices.
2. **Answering the second (sub) research question: statistical properties of the data** - [Section 4.5](#)
In the fifth building block we address how we can leverage the statistical properties of the data in feature space, to determine relevance.
3. **(Partly) answering the third (sub) research question: dimensionality of the data** - [Section 4.5](#)
Dimensionality of the data has an effect on the performance of distance measures, we treat these effects of high dimensions in the fifth building block as well. The full answer to this question shall be provided in [Chapter 5](#), through experimental results.

4.1. Methodology Overview

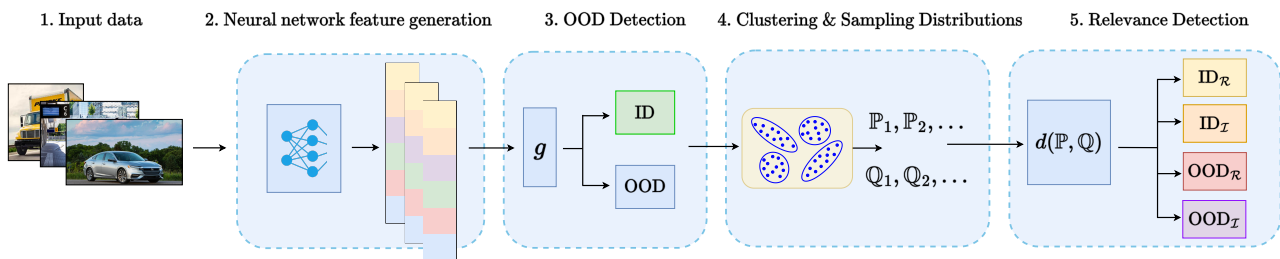


Figure 4.1: The simplified pipeline of our solution at test time, allowing for the 4 outputs shown in [Fig. 1.4](#).

Before we dive into the details of each step, we present a high-level look at the proposed solution again, as a point of reference. The first three blocks require the background knowledge that is provided in [Chapters 2](#) and [3](#), while the fourth and the fifth blocks are more self-contained.

Each step/module of the pipeline is treated in its own section, which are all structured as follows:

- Motivation of that module/used technique
- Challenges
- Explanation of its operational dynamics
- (optional) Link to research question

4.2. Neural Network Feature Generation

The neural network plays a principal role in most contemporary computer vision pipelines. As explained in [Section 2.4.1](#), a neural network learns the information (features) from the image data as the image passes through the layers and the information of each feature is stored in the entries of a *feature vector* $h(\mathbf{x}_i; \theta)$.

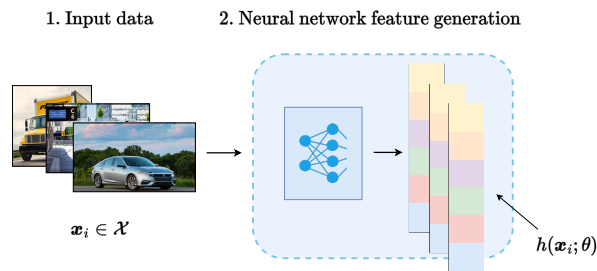


Figure 4.2: Input of this block: image data \mathbf{x}_i . Output of this block: feature vector $h(\mathbf{x}_i; \theta)$.

Without features, one does not have the information of the data that will be used to teach a classification model which class belongs to which label, which class is OOD/ID, which class is relevant/irrelevant etc. Therefore we need a feature generator at the start of our pipeline, but which one? Countless networks exist.

4.2.1. Motivation for the CLIP Neural Network

For the neural network in our pipeline, we have selected CLIP: a pre-trained neural network developed by OpenAI, trained on 400 million images [54]. Details on the CLIP network are treated in [Appendix A.1](#). Our rationale behind CLIP is directly linked to the challenges of this building block and is three-fold:

1. **Saving a substantial amount of time/resources** → Since CLIP is pre-trained, we do not (necessarily) need to (re)train the network ourselves which allows us to focus on our niche.
2. **High performance** → CLIP has proven, for data-sets of common and everyday classes it has never seen before, that it is competitive with fully supervised neural network architectures like ResNet50 [55]. This means that we are likely to be better off to use CLIP without re-training, than to train a fully supervised network. That is, as long as we are not using it for abstract data-sets/tasks like lymph node detection, or satellite images [54].
3. **Rich feature space** → CLIP is trained on a vast amount of classes, and has learnt a substantial variety of features, resulting in a rich feature space, i.e. the vectors $h(\mathbf{x}_i; \theta)$ are very rich and meaningful. This is such a key attribute that we will further discuss this below.

Imagine that each image feature vector $h(\mathbf{x}_i; \theta)$, per class, is plotted in a 2D feature space ([Fig. 4.3](#)). Depending on the class, the feature vector will have different values and therefore a different location in feature space. Let us inspect the case where a given feature space is 'not rich', i.e. a feature space of a simple neural network that has only ever been trained on 3 vehicles: car, truck, and bicycle (green), and encounters at test time the classes pickup truck, bus, motorcycle, airplane, and bird (blue).

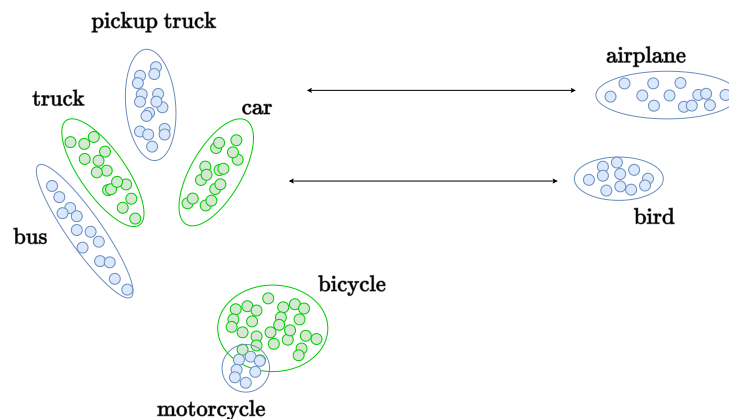


Figure 4.3: Due to a lack of learnt features, OOD classes that are totally different, can be equally far away.

A military ship would (for us) be semantically closer to a car/truck than a bird would be, for instance because military ships are made of metal (like cars and trucks) and birds have wings and feathers. Fig. 4.3 showcases that when some features are not learnt (e.g. the features 'metal', 'wings', 'feathers'), 2 (OOD) classes can be equally 'different' or 'far' from ID classes (e.g. because both ships and birds do not have wheels) even though one is more related than the other. This makes relevance detection in the final step of our pipeline incredibly hard, if not impossible.

While the example in Fig. 4.3 might be a simplified version of reality, the idea is legitimate [52]. Therefore, to diminish the effects of a poor feature space as much as possible, we need a pre-trained network that is trained on a wide variety of classes and able to generate high quality features. This makes CLIP a suitable candidate, as it satisfies the criteria and outperforms its alternatives [54].

Is a pre-trained neural network, trained on a multitude of classes not counter-intuitive and against this thesis? In other words, how can we detect OOD classes if our model has already been trained on so many classes? Fortunately, this does not lead to any issues. The reason why, is because we are *not* using the pre-trained neural network for classification, but for feature generation only. That means that we deploy CLIP as a fancy 'data-processor' to obtain high-quality features $h(x_i; \theta)$ and use a separate classification head g (like softmax or GEM) to classify to which class $h(x_i; \theta)$ belongs (ID or OOD). As a slightly strange but clear analogy, suppose we have 2 printers symbolising our neural networks:

1. Low resolution, black-white printer → simple neural network, almost no pre-training
2. High resolution, multi-colour printer → CLIP neural network, heavy pre-training

Now suppose we have a human 'classifier' (analogous to our scoring function g) who has only ever seen cars, truck and bicycles, and has to determine which images printed by a printer are unknown classes (OOD) and which are known (ID). Since the more advanced printer adds more features to its output (feature 1: colour, feature 2: high resolution), it will be easier for the human classifier to distinguish unknown classes from known classes.

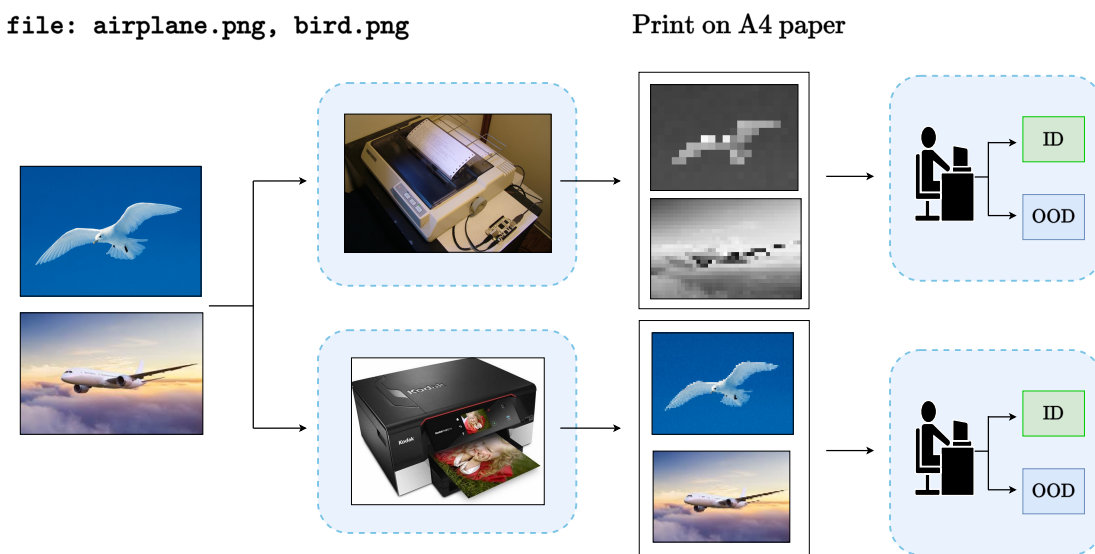


Figure 4.4: Printer analogy with neural network: at test time we see two new classes, With a black & white printer (non-pre-trained network), 2 distinct classes can look similar (or equally far in feature space).

Note here that we are not changing anything about the human's inherent ability to detect ID vs OOD by swapping printers: pre-training makes sure the network generates more diverse features, it is not teaching the classifier anything. What does happen, is that a human's performance will be better given more high quality photos, simply because a higher quality photo contains more relevant information. In the next building block of the pipeline (Section 4.3), the inherent ability of the classifier itself is treated; we teach an OOD scoring function g which classes are ID by training it on the feature vectors of the ID classes. Prior to that we briefly treat how these feature vectors from CLIP are reduced to a dimension that we can work with.

4.2.2. Post-processing: Dimensionality Reduction of CLIP Vectors

The feature vectors obtained from CLIP contain 512 features, meaning that each image is represented as a point in a 512-dimensional feature space, through $h(\mathbf{x}_i; \theta) \in \mathbb{R}^{512}$. Such high-dimensional vectors are rich in information but can be undesirable to work with in its current form, in our pipeline, for a variety of reasons:

1. Points in dimensions higher than 3D are hard, if not impossible, to visualise.
2. These vectors need to be fed to our scoring function g in the next block, which is known to be significantly less accurate for high dimensions (see Fig. 3.9).
3. Distance measures between points can often struggle in high dimensions (see Section 4.5).

It is therefore useful to reduce the dimension d of $h(\mathbf{x}_i; \theta)$ such that further analysis will improve, without losing too much information. Let $f_\varphi^{\rightsquigarrow}$ denote a general function that reduces d of $h(\mathbf{x}_i; \theta)$ to an arbitrary dimension $n < d$, then, a more complete depiction of the second module in our pipeline is given by:

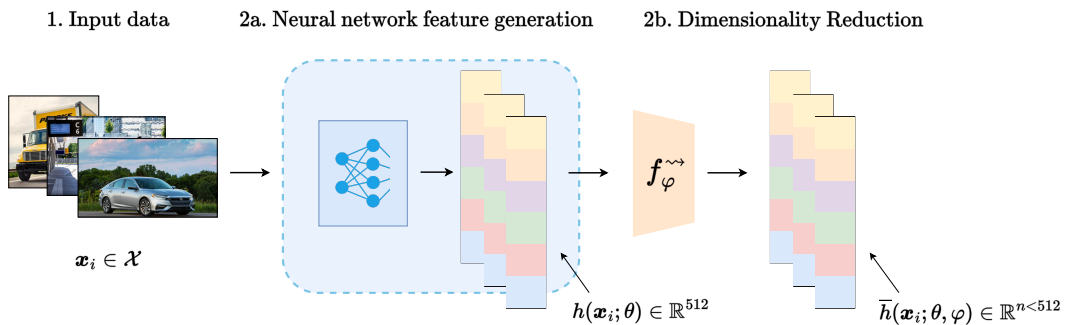


Figure 4.5: Input of $f_\varphi^{\rightsquigarrow}$: feature vector in 512D. Output of $f_\varphi^{\rightsquigarrow}$: feature vector in a lower dimension.

For simplicity, we shall stick to the general notation $h(\mathbf{x}_i; \theta)$ for the reduced feature vectors as well, throughout this thesis.

Dimensionality reduction is a commonly used technique in mathematics, statistics, and machine learning where a dimensionality reducer $f_\varphi^{\rightsquigarrow}$, governed by some built-in parameter φ , helps the visualisation and understanding of high-dimensional data by transforming its statistical and/or topological properties. Examples of $f_\varphi^{\rightsquigarrow}$ are Principal Component Analysis (PCA) [56], Linear Discriminant Analysis (LDA) [57], but also *feature extraction* by the neural network is in itself a form of dimensionality reduction as we reduce an image \mathbf{x}_i of e.g. 200x200 pixels ($\in \mathbb{R}^{4 \times 10^4}$) to $h(\mathbf{x}_i; \theta) \in \mathbb{R}^{512}$.

Within contemporary machine learning, t-SNE (2008, [58]) and UMAP (2018, [59]) are especially popular and the most widely used choices for $f_\varphi^{\rightsquigarrow}$ [60]. Below we briefly compare both techniques, provide our motivation for UMAP, and end with a short exposition of UMAP's inner workings.

t-SNE vs UMAP

Both t-Distributed Stochastic Neighbour Embedding (t-SNE) and Uniform Manifold Approximation and Projection (UMAP) are non-linear reducers and essentially 'graph layout' or 'graph drawing' algorithms that arrange high-dimensional data in a low-dimensional space. They follow two broad steps:

1. Construct graphs of local relationships between data points in the high and low-dimensional spaces.
2. Optimise the low-dimensional graph such that the structure of the original graph is preserved.

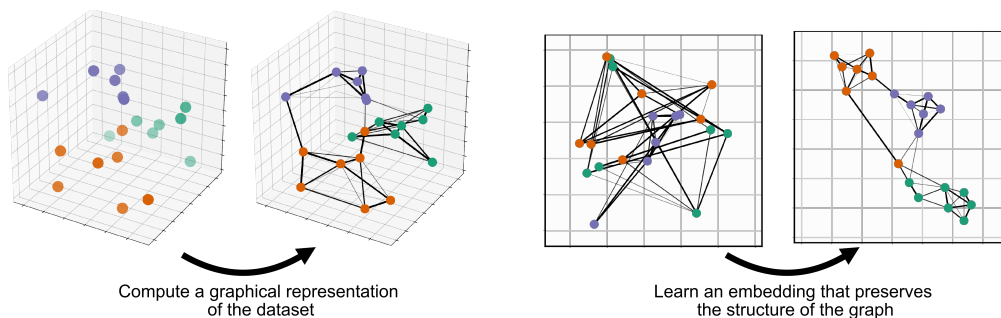


Figure 4.6: 3D points (left) are transformed to a 2D representation (right) through a graph, adapted from [61].

In short, the techniques construct high-dimensional graph representations of the data and then optimise a low-dimensional embedding of that graph to be as structurally similar as possible. In Fig. 4.6 we can see that the optimisation process is essential, otherwise little to no structure will be preserved (third plot) and the embedding will visually not make any sense. Before we delve into the specifics of the optimisation process (for UMAP) we motivate our choice for UMAP by setting out the most important shortcomings of t-SNE that are relevant for this thesis below:

1. **t-SNE cannot work with high-dimensional data directly**, PCA is often used as a pre-reducer before feeding the reduced vectors to t-SNE for visualisation.
2. **t-SNE can in practice only embed into 2 or 3 dimensions** making it useful for visualisation, but hard to use it as a general dimension reduction technique that can reduce a high-dimensional vector to any n dimensions. This is especially important for our research where we want to reduce our CLIP vectors to a variety of dimensions that can still be high (e.g. 10D) in order to test the performance of various distance measures in multiple dimensions.
3. **t-SNE does not preserve global data structure**, this means that within clusters (local structure), the distances between points are meaningful, but in-between clusters (global structure) they are not. This implies that both similar and dissimilar clusters can be close to each other with t-SNE, making its output unsuitable for clustering. This is by far the most important characteristic of t-SNE that makes it undesirable for our work as we will need clustering in the further stages of the pipeline, and base our hypotheses on the notion that related classes are closer in feature space while unrelated classes are not. Not being able to preserve global structures also defeats the purpose of using CLIP: since CLIP generates rich features that will distinguish classes in feature space, not having a good dimensionality reducer f_{φ^*} will not do the CLIP vectors justice. All the above implies that using t-SNE will make it very hard, if not impossible to determine relevant from irrelevant OOD classes using distance measures.

Furthermore, t-SNE is also slower and more computationally expensive than UMAP, but we are willing to trade resources and time for better performance. UMAP however, is simply far superior for our purpose, which is preserving the global structure of the data in feature space.

To showcase the preservation of local vs global structure we compare the outputs of PCA, t-SNE and UMAP.

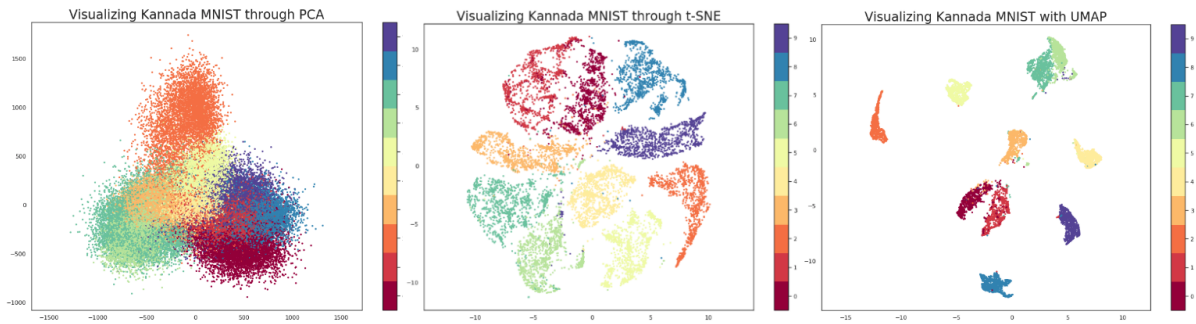


Figure 4.7: 28x28 pixel images ($\in \mathbb{R}^{784}$) of handwritten digits of the MNIST data-set (written in the Indian Kannada script), reduced to 2D by PCA, t-SNE, and UMAP [62]. Each colour corresponds to a different class (which is in this case a handwritten digit).

While t-SNE correctly maps the points of each class close to each other (purple points are close to purple points, yellow points are close to yellow points etc.), the distance between clusters of these classes are not meaningful; it is hard to tell which clusters are more, and which are less similar to each other. Unlike t-SNE, UMAP preserves global structure much better and maps similar clusters close to each other and vice versa. We will need this preservation of global structure because our classes of relevance (road vehicles) should be close in feature space, while aircrafts and animals should be relatively farther.

Without even having treated the underlying formalisms of t-SNE, the list of its shortcomings and the output in Fig. 4.7 clearly justify the use of UMAP. Therefore it suffices to solely focus on the treatment UMAP.

General Exposition of UMAP

UMAP has a rigorous mathematical foundation based on algebraic topology and topological data analysis which are thoroughly described in the work of McInnes, Healy, and Melville (2018, [59]), and beyond the scope of this thesis. Therefore, we only treat the main ideas in a general fashion. Recall from Fig. 4.6 that there are 2 main steps the algorithm goes through, here we treat these steps specific to UMAP:

1. Construct (probabilistic) graphs of the local relationships in the high-dimensional data.
2. Optimise the embedding of this graph through cross entropy (CE) into a low dimensional space in order to preserve its structure.

Step 1: Construction of the graph.

To construct a graph we need a few elementary definitions from geometry and algebraic topology:

Definition 4.2.1 (k -simplex σ_k) A simplex σ_k is geometric object that is a generalisation of the notion of a triangle or tetrahedron to arbitrary dimensions.

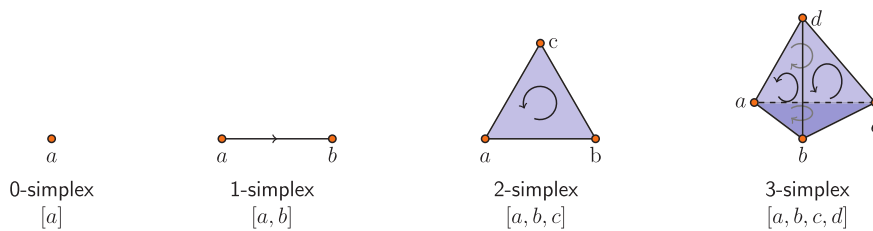


Figure 4.8: k -simplices with vertices $V = [a, b, c, d]$, taken from [63].

Definition 4.2.2 (Simplicial complex \mathcal{K}) A simplicial complex \mathcal{K} in \mathbb{R}^n is a collection of simplices σ_k . It is a space in \mathbb{R}^n such that

1. Every face of a simplex of \mathcal{K} is in \mathcal{K} .
2. The intersection of any two simplices of \mathcal{K} is a face of each of them.

One of the main assumptions of UMAP is that data points are uniformly distributed on a manifold \mathcal{M} ¹ supported by a Riemannian metric tensor g_p (to allow for distances and angles). This uniform distribution rarely holds in practice but is allowed when the manifold has a Riemannian metric [59]. When the data points lie in such a metric space (\mathcal{M}, g_p) , we can construct a graph using the notion of distances to find nearest neighbours. The authors of UMAP propose to do this by first forming Čech complexes, which are a type of simplicial complex where one connects points by drawing ε -balls around each point and connecting points whose radii intersect, and subsequently turning those intersecting points into collection of 0-simplices and 1-simplices as they are computationally less expensive (as it is simply a graph of nodes and vertices):

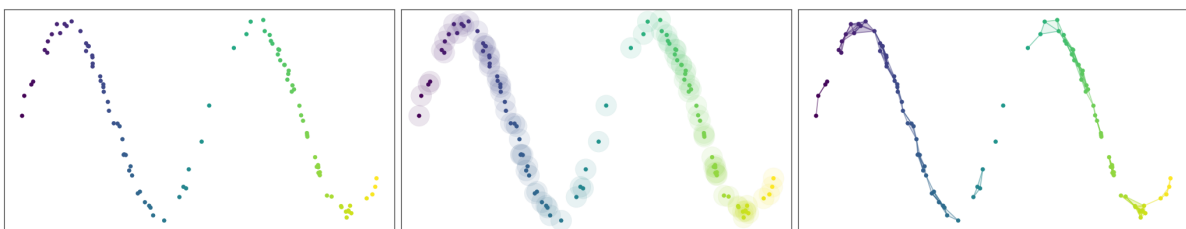


Figure 4.9: Constructing a graph of data points through simplicial complexes, taken from [64]. At the end we end up with a simplicial complex, also called a Vietoris-Rips complex.

Fig. 4.9 showcases how UMAP constructs Vietoris-Rips graphs in a visual manner, the main implication that follows from the topological analysis above, is that in practice each data point has a fixed k number of

¹A manifold \mathcal{M} is a topological space where each point $x \in \mathcal{M}$ has a neighbourhood homeomorphic to Euclidean space. A neighbourhood N of $x \in \mathcal{M}$ is a subset of \mathcal{M} , that contains x and an open set U such that $x \in U \subset N \subset \mathcal{M}$. Within N one can move away from x to some extent in any direction without leaving \mathcal{M} . A homeomorphism is a continuous and bijective mapping between two topological spaces.

neighbours for the construction of the graph [59]. To construct this graph, UMAP uses the kNNG (k-Nearest Neighbour Graph) algorithm developed by Dong, Charikar, and Li [65] that can efficiently and smoothly approximate the kNN graphs of the data by using probabilistic similarity measures. UMAP uses kNNG to create a probability distribution that represents 'similarities' between neighbours and the notion of similarity used by the authors is as follows: the 'similarity' of data point x_j to data point x_i is the conditional probability $p_{j|i}$, that x_i would pick x_j as its neighbour.

In the high-dimensional space, UMAP models the similarity per point (locally) through the distribution:

$$p_{j|i} = \exp(- (d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i) / \sigma_i) \quad (\text{see [59] for the derivation}) \quad (4.1)$$

Where d is the distance metric, σ_i is a local connectivity parameter to ensure smooth approximation of the kNN graph, and $\rho_i = d(\mathbf{x}_i, \text{nearest_neighbour}(\mathbf{x}_i))$ is a local connectivity parameter set to the distance from a point x_i to its nearest neighbour. ρ_i is an important parameter as it ensures the local connectivity of the manifold, i.e. for each data point, ρ_i is an offset that causes the distance metric to **vary from point to point**. The result of varying distance metrics though, is that when gluing all similar points found above together, it is possible that the weight of the graph between nodes 1 and 2 is not equal to the weight between 2 and 1. To resolve this, UMAP smoothes out those effects by symmetrising the high-dimensional probability distribution:

$$p_{ij} = (p_{j|i} + p_{i|j}) - p_{j|i}p_{i|j} \quad (4.2)$$

After computing the graph in the high-dimensional space, UMAP will model the similarity for points $(\mathbf{y}_i, \mathbf{y}_j)$ in the low-dimensional space, following the same notion as before but through a different family of functions:

$$q_{ij} = \left(1 + a \|\mathbf{y}_i - \mathbf{y}_j\|^{2b}\right)^{-1} \quad (\text{see [59] for the derivation}) \quad (4.3)$$

Where a and b are hyperparameters, based on a desired minimum distance between points in the low-dimensional embedding space. This minimum distance controls how tightly points are allowed to be packed together and is set by the parameter `min_dist`. If one is interested in clustering for instance, then `min_dist` should be chosen small.

Step 2: Optimising the low-dimensional graph.

Now that graphs in both the high-dimensional and low-dimensions space are computed, UMAP needs to make sure that the structure is preserved in the low-dimensional space as well. Therefore, it uses the cross entropy as a measure to compute how much the probability distributions p_{ij} and q_{ij} diverge.

$$CE(p_{ij}, q_{ij}) = \sum_{i \neq j} \left[p_{ij} \log \left(\frac{p_{ij}}{q_{ij}} \right) + (1 - p_{ij}) \log \left(\frac{1 - p_{ij}}{1 - q_{ij}} \right) \right] \quad (4.4)$$

The higher the cross entropy, the more these graphs differ from each other. Therefore, the low-dimensional representation is optimised by minimising the cross entropy, in this case through gradient descent (see [Section 2.4.3](#)).

Below we list the most important UMAP parameters that can be tweaked when using the model in practice. We further discuss them in [Chapter 5](#) when treating our experimental setup as it will make more sense.

- `n_neighbours`: the amount of neighbours k , each data point can have. This parameter controls how UMAP balances local versus global structure in the data.
- `min_dist`: the minimum distance between points in the low-dimensional representation.
- `n_components`: the dimensionality of the target dimension space we embed our low-dimensional representation of the data in.
- `metric`: the distance metric d to compute distances between points.
- `output_metric`: the space UMAP will map the reduced data to, be it on a sphere, plane, or custom metric space. This parameter choice largely depends on the input data ([Chapter 5](#)), but for the coming sections one can assume we map to Euclidean space.

In the next section we show how this UMAP data will be loaded in the OOD detection module of our pipeline.

4.3. OOD Detection Module

After generating the features with CLIP and reducing their dimensionality through $f_{\varphi}^{\rightsquigarrow} = \text{UMAP}$, we need a function to perform OOD detection on them. The OOD detection module does precisely that and is essentially a scoring function g that separates the features of the network into two batches: ID & OOD. The OOD scoring function is trained on ID classes and learns to assign a high score for ID samples, such that it will assign a low score to OOD samples at test time (or vice versa depending on the design choice). The training of g shows that this layer of the pipeline does not know anything of the data beforehand, even though the neural network is heavily pre-trained.

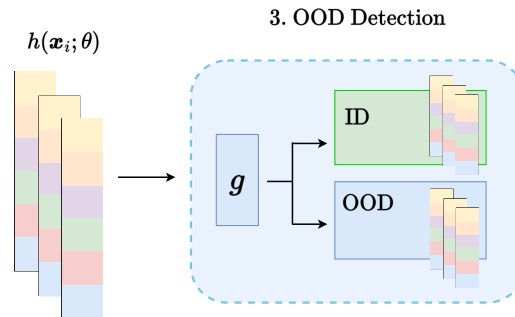


Figure 4.10: Input of this block: (reduced) feature vectors $h(\mathbf{x}_i; \theta)$. Output of this block: the same (reduced) feature vectors $h(\mathbf{x}_i; \theta)$, but now separated into two batches: ID & OOD.

4.3.1. Motivation of OOD Scoring module

The motivation, use, and variants of an OOD scoring function in general were treated in [Chapter 3](#), so it is clear why our pipeline will need such a module. Out of all scoring functions, we selected the GEM function $\text{GEM}(f, \mathbf{x})$ even though it was not the 'best' performer:

1. **GEM is log-proportional** to the distribution of our ID data ($p_{\mathcal{X}}^{\text{ID}}(\mathbf{x})$), meaning that its behaviour and rate of change of the data distribution are best captured by GEM, while the other functions are not proportional at all.
2. **GEM is mathematically sound** as its mathematically proven guarantees of its performance under various changes in the data. That is, its behaviour when varying the dimension of ID data, distance between ID and OOD data, and amount of ID classes, has been mathematically proven, providing us with a point of reference when running our own experiments.
3. **GEM is almost the best performer**, losing to Mahalanobis with only fractions of a percentage point on average ([Table 3.3](#)).

The GEM score is based on the Mahalanobis distance which is a centroid-based distance that 'assumes' Gaussianity. In practice, not all data follows a Gaussian distribution, so should or can one still use techniques like these for non-Gaussian data? This will be treated in [Section 4.5](#) where we study various distance measures between point clouds/distributions. It should be noted that like any scoring function, GEM is not perfect. On the contrary, GEM makes mistakes for both simple and difficult tasks. The performance of scoring functions therefore leads to some challenges.

4.3.2. Challenges & Requirements

The authors of the GEM and EBM papers [[42](#), [44](#)], have tested multiple scoring functions on various OOD data-sets to discover where the challenges lie. Those tests were performed with the following data-sets:

- **ID data-set:** CIFAR-10, which is a mix of 10 classes consisting of vehicles like cars & trucks, and animals like cats & horses.
- **OOD data-set 1:** SVHN, which are images of house number plates.
- **OOD data-set 2:** Places365, which are images of scenes (natural landscapes, living rooms, cafeteria, forests etc.)

We have already shown the average performance of the tests mentioned above in [Table 3.3](#), so below in [Table 4.1](#) we present the two tests where the GEM performance has been both at its **best** and at its **worst**.

Table 4.1: Performance of scoring functions trained on the CIFAR-10 data-sets, with SVHN and Places365 as the OOD data-sets. Test results are courtesy of Liu et al. and Morteza and Li, details can be found in [42, 44]

OOD data-set	Scoring Function	FPR [%] ↓
SVHN	Softmax score [Hendrycks and Gimpel, 2016]	48.49
	ODIN [Liang et al., 2018]	33.55
	Mahalanobis [Lee et al., 2018]	12.86
	Energy score [Liu et al., 2020]	33.01
	GEM [Morteza and Li, 2021]	13.42
Places365	Softmax score [Hendrycks and Gimpel, 2016]	59.48
	ODIN [Liang et al., 2018]	57.40
	Mahalanobis [Lee et al., 2018]	68.42
	Energy score [Liu et al., 2020]	40.14
	GEM [Morteza and Li, 2021]	68.03

One would think that a SOTA scoring function like GEM, trained on vehicles and animals, would be able to easily tell that an image of natural scenery would be so different from its ID data-set that it must be an OOD class. Unfortunately, the results say otherwise: with a False Positive Rate (FPR) of 68.03%, GEM falsely assigns 68.03% of all scenery images as ID, e.g. think they are not that different from vehicles and animals. From Table 4.1 it is also clear that the scoring functions perform differently depending on the OOD data-set, making it even harder to select one as none of the OOD data-set used align with our use case (we want to test on realistic OOD classes like vehicles, not scenery, number plates or white noise).

The main question that arises now is: given the relatively high FPR, how much of a bottleneck will the scoring function present in our pipeline? Is it something we can work with, correct, and ultimately accept? To find out, we need to set some expectations and requirements of this part of the pipeline. Let's consider the toy problem setting introduced in Section 3.3.2 again. We use the following classes to visualise the output of this module, in order to easily define our expectations and requirements.

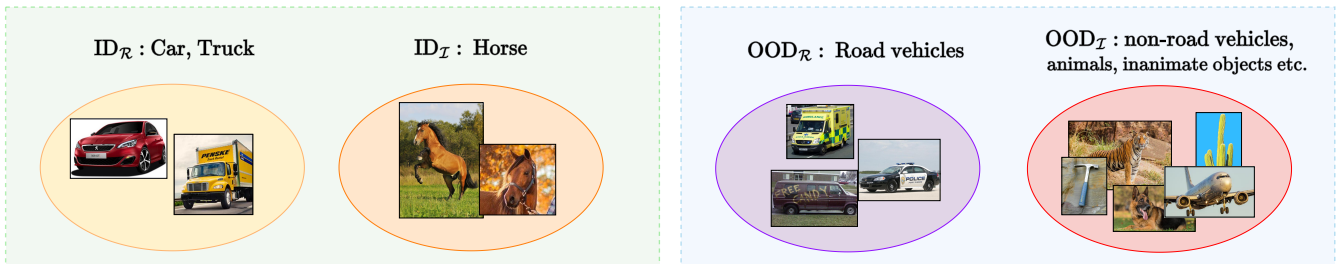


Figure 4.11: OOD_R contains the relevant OOD classes that we want to detect and is therefore the target set.

Given the fact that scoring functions make mistakes, we zoom in on our scoring module and add the following sub-sets within the ID and OOD batches, that come out of the scoring function:

- OOD_{false}: OOD classes that are falsely labelled as ID.
- ID_{false}: ID classes that are falsely labelled as OOD.

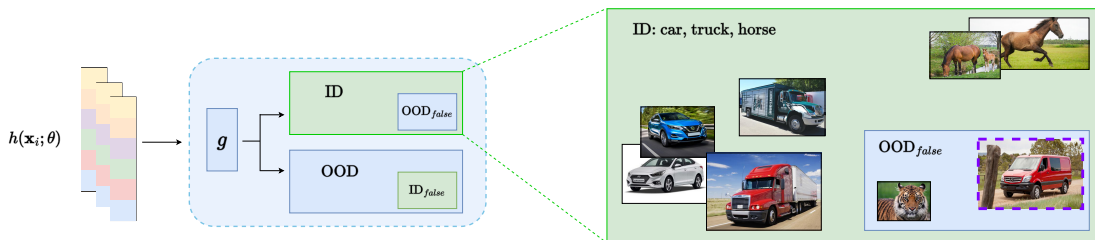


Figure 4.12: As an example we zoom in on the ID batch. Out of the classes presented, we expect cars, trucks, and horses to be in the ID set and OOD classes in OOD_{false}, where a relevant OOD class has a purple dashed border.

Below we present three possible scenarios that can occur, based on the performance of the scoring function. In each scenario, the features of the classes of the toy problem are used as input and go through two steps: I. OOD detection step, II. Relevance detection step (which is in the final part of the pipeline). The figures below are flowcharts, where each step acts like a filter, arranging all classes into the four output buckets: $ID_{\mathcal{R}}$, $ID_{\mathcal{I}}$, $OOD_{\mathcal{R}}$, $OOD_{\mathcal{I}}$. Through these flowchart we aim to showcase two things:

1. In which way can poor performance of OOD detection influence relevance detection?
2. When do the errors of OOD detection form too much of a bottleneck?

In the ideal scenario we assume that both OOD detection and relevance detection are perfect, this leads to perfect results at the end.

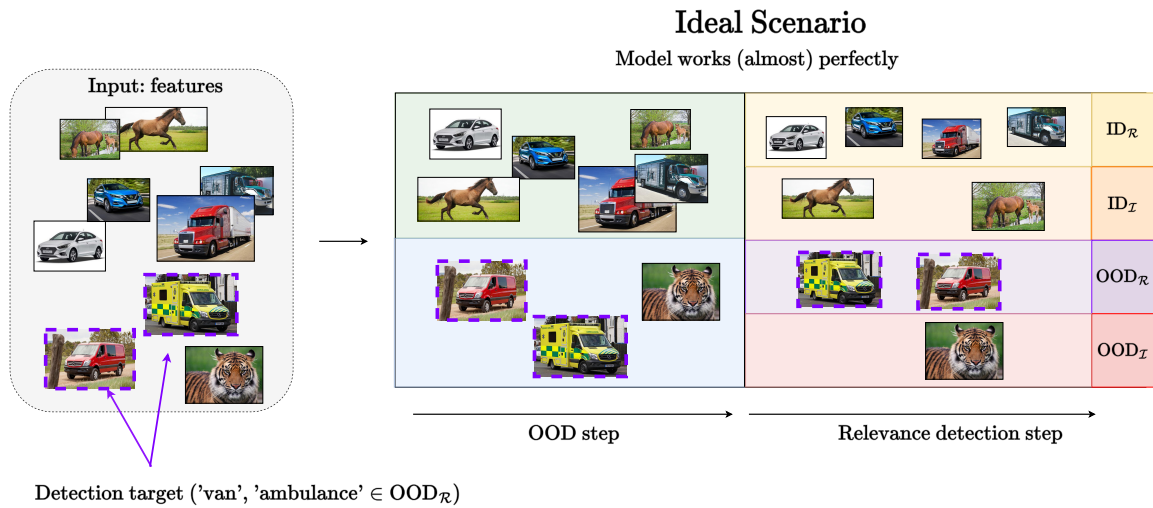


Figure 4.13: Ideal case: no mistakes in OOD and relevance detection. Target classes (purple dashed border) are correctly labelled.

In the worst case scenario, we assume that both OOD and relevance detection are poor. We want to find out whether it would have mattered if relevance detection was accurate or not when the OOD step produces bad results; can we still fix the mistakes in the OOD step? The answer is no. Mistakes made in a cascaded structure like this are carried to the next step:

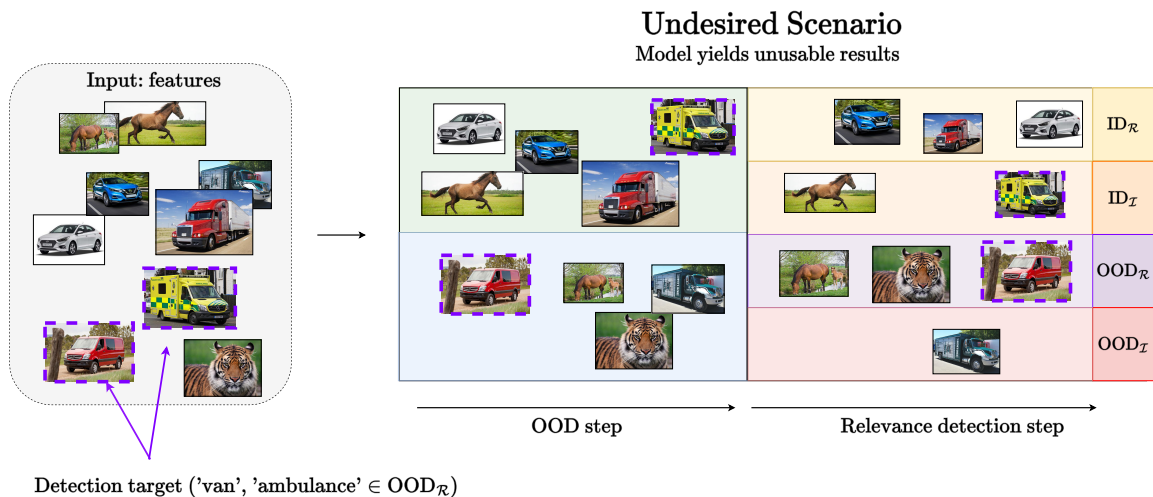


Figure 4.14

Fig. 4.14 showcases the irreversible mistake of the OOD detector; once a sample has been falsely classified as OOD or ID, there is no return, making it an error one has to accept. But what happens to the relevance

detection once we accept these errors? In the scenario above we see that the model makes absolutely no sense when both OOD and relevance detection sequentially make mistakes: e.g. trucks are seen as OOD (first mistake) and flagged as irrelevant (second mistake), horses as OOD (first mistake) and flagged as relevant (second mistake).

It would much more acceptable if, whenever OOD mistakes are made, the samples are at least correctly classified as relevant/irrelevant:

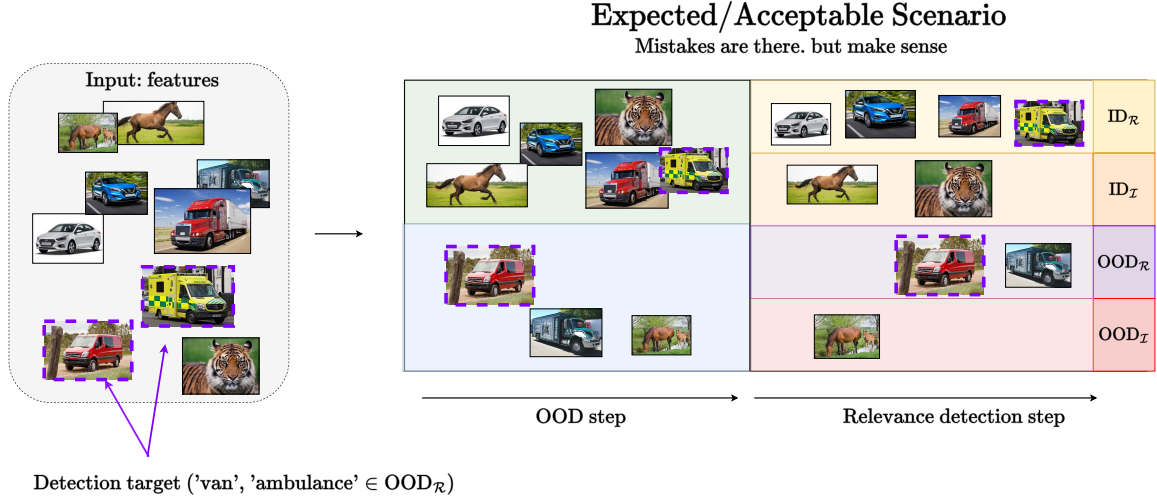


Figure 4.15

In the last scenario, at least all relevance detection is done without mistakes. This is important because now, even when OOD mistakes are made, we might be able to still find our OOD_R samples in the yellow and purple buckets through post-hoc techniques. With these scenarios, we have guidelines on how to assess our pipeline's performance in the experiments that will be conducted in [Chapter 5](#).

4.3.3. GEM Score Calculation

The GEM scoring function models the feature vectors of the data through a Gaussian distribution. It assumes that our batch of ID data (training data), consists of k classes, where each class is normally distributed, i.e. given their corresponding label y_i , the features $h(\mathbf{x}; \theta)$ each class in the training data are modelled by:

$$h(\mathbf{x}; \theta) | y_i \sim \mathcal{N}(\mathbf{u}_i, \bar{\Sigma})$$

where $\mathbf{u}_i \in \mathbb{R}^m$ is the mean of each class y_i and $\bar{\Sigma} \in \mathbb{R}^{m \times m}$ is the covariance matrix. Since these are Gaussians, to estimate each class distribution $\mathcal{N}(\mathbf{u}_i, \bar{\Sigma})$, it suffices to estimate its parameters by computing the empirical class mean and covariance given training samples $\{(\mathbf{x}_1, \bar{y}_1), (\mathbf{x}_2, \bar{y}_2), \dots, (\mathbf{x}_N, \bar{y}_N)\}$:

$$\hat{\mathbf{u}}_i = \frac{1}{N_i} \sum_{j: \bar{y}_j = y_i} h(\mathbf{x}_j; \theta), \quad \hat{\Sigma} = \frac{1}{N} \sum_{i=1}^k \sum_{j: \bar{y}_j = y_i} (h(\mathbf{x}_j; \theta) - \hat{\mathbf{u}}_i)(h(\mathbf{x}_j; \theta) - \hat{\mathbf{u}}_i)^T,$$

Once the Gaussian mixtures of the training data (ID batch) is computed, at test time when observing a new image \mathbf{x}_τ , the GEM score is computed as follows:

$$\text{GEM}(\mathbf{x}_\tau; \theta) = \log \sum_{j=1}^k \exp(M_j(\mathbf{x}_\tau; \theta)), \quad \text{where} \quad M_j(\mathbf{x}_\tau; \theta) = -\frac{1}{2} (h(\mathbf{x}_\tau; \theta) - \hat{\mathbf{u}}_j)^T \hat{\Sigma}^{-1} (h(\mathbf{x}_\tau; \theta) - \hat{\mathbf{u}}_j) \quad (4.5)$$

M_j is the Mahalanobis distance that is computed by comparing the feature vector $h(\mathbf{x}_\tau; \theta)$ of each new sample to the k Gaussians of the ID data at every iteration j . If \mathbf{x}_τ belongs to one of the ID classes, M_j will be small \Rightarrow small GEM score \Rightarrow ID, if not, then the GEM score will be high \Rightarrow OOD.

4.4. Clustering & Sampling Distributions

This module will organise the still 'raw' data of the previous steps and is fairly straightforward, yet essential.

4.4.1. Motivation of the Clustering module

After OOD detection has split our input data at test time into two groups, the features are still 'unsorted' in the sense that all classes are mixed in both batches. We need them to be organised per class in order to determine which classes are relevant and which are not. The idea is to organise the mixed $h(x_i; \theta)$ into groups per class, e.g. all features belonging to the class car should be grouped together $\{h(x_1^{\text{car}}; \theta), \dots, h(x_N^{\text{car}}; \theta)\}$ the same goes for the horse class $h(x_1^{\text{horse}}; \theta), \dots, h(x_N^{\text{horse}}; \theta)$ and so on. This process is executed by the clustering module in our pipeline:

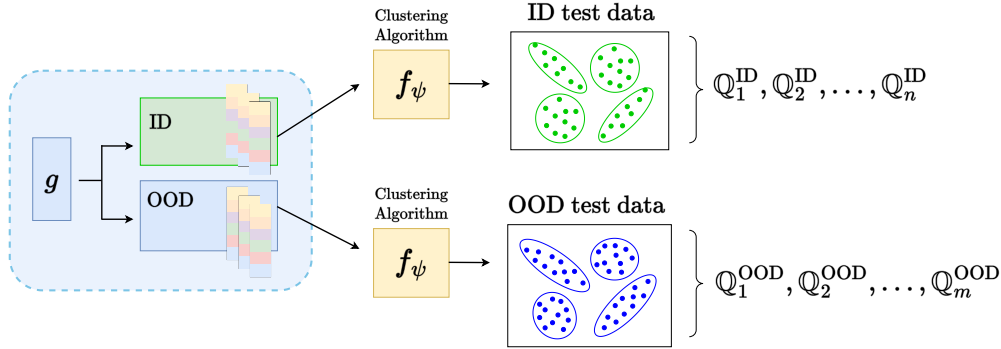


Figure 4.16: Input of this block: feature vectors $h(x_i; \theta)$, reduced by UMAP and separated by the previous module into ID and OOD batches. Output of this block: groups/clusters Q_j per class of $h(x_i; \theta)$, where n denotes the amount of ID classes detected, and m the amount of OOD classes.

At test time we do not know the labels of our input features/data, therefore, rather than ' $h(x_i^{\text{car}}; \theta)$ ', the clusters are denoted with numerical indices $Q_1 = \{h(x_1^1; \theta), \dots, h(x_N^1; \theta)\}$. Fortunately, we work with rich features from CLIP, whose structure in feature space is preserved quite well with UMAP, therefore the data per batch is quite organised which benefits clustering. What the clustering algorithm f_ψ needs to do is essentially label and group all data points in the UMAP feature space into separate clusters/distributions Q_1, \dots, Q_n . This internal process is depicted in Fig. 4.17.

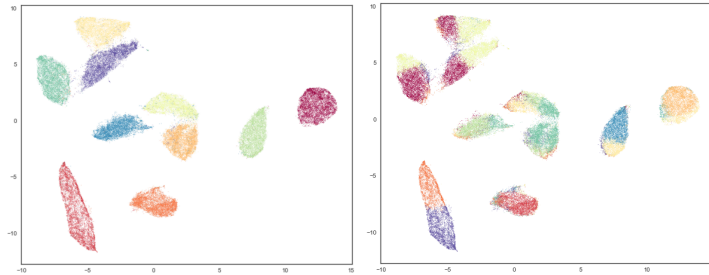


Figure 4.17: UMAP feature space of test data (left), formed clusters of the UMAP data (right). [66]

4.4.2. Challenges & Requirements

As one can see in Fig. 4.17, UMAP does not necessarily produce clean spherical clusters, making algorithms like k -means or Gaussian Mixture Models (GMM) sub-optimal choices for f_ψ since these are centroid-based models with an assumption of largely spherical clusters. A more suitable alternative would be a density-based algorithm like HDBSCAN which can, unlike k -means, discover clusters of arbitrary shape. Regardless of the choice for f_ψ , the clusters are required to contain mostly samples of the same class, otherwise relevance detection in the next section will be difficult: the comparison between Q_{car} to Q_{bus} will not be fair if the car cluster is comprised of truck data points for 20%, and the bus cluster of tiger data points for 18%. The exact requirement for minimal cluster purity is not clear at this point, but we will set it at $>90\%$ for now and evaluate this, and the choices for f_ψ in the experiments in Chapter 5.

4.5. Relevance Detection

The clustering step in the previous step of the pipeline yields the final representation of our input image files: data points, clustered per class, in feature space. In order to determine which classes are relevant and which not, we can compare clusters at test time, to cluster of classes from which we have trained on and therefore know that they are relevant. We propose to accomplish this through this module by leveraging the statistical properties of the clusters such as shape, distribution and relative distances between each other.

4.5.1. Conceptual Overview of the Module

The main idea of this module is to use a distance function $d(\cdot, \cdot)$ to determine how 'far' away two or more clusters are from each other. We have seen in Fig. 4.17 that our classes of interest (road vehicles) are close to each other in feature space, while other (irrelevant) classes are usually significantly further away. In order to visualise this concept, we clarify some variables, parameters, and notation below:

Symbol	Description
\mathbb{Q}_j	Clusters at test time, whose specific class is unknown.
\mathbb{P}_i	Clusters at training time, whose specific class relevant, known, and trained on.
j	Index of the test clusters, denoting the class. Since we do not know the test classes, $j \in \{1, \dots, m\}$.
i	Index of the test clusters, denoting the class. Since we do know the training classes, $i \in \{1: \text{'car'}, \dots, k: \text{'truck'}\}$.
m	Total amount of test clusters.
k	Total amount of train clusters.
$q_t^{(j)}$	A single point in each test cluster, where $t \in \{1, \dots, N\}$ denotes the index of each point $q_t^{(j)} : \mathbb{Q}_j = \{q_1^{(j)}, \dots, q_N^{(j)}\}$.
$p_s^{(i)}$	A single point in each train cluster, where $s \in \{1, \dots, N\}$ denotes the index of each point $p_s^{(i)} : \mathbb{P}_i = \{p_1^{(i)}, \dots, p_N^{(i)}\}$.
N	Sample/cluster size, both training and testing clusters always contain the same N amount of points (see next sub-section).

The relevance detection module $d(\mathbb{P}, \mathbb{Q})$ is equivalent for both the ID and OOD data batches, so its internal process is depicted in Fig. 4.18 for the OOD batch only, where we train on two relevant clusters ($k = 2$).

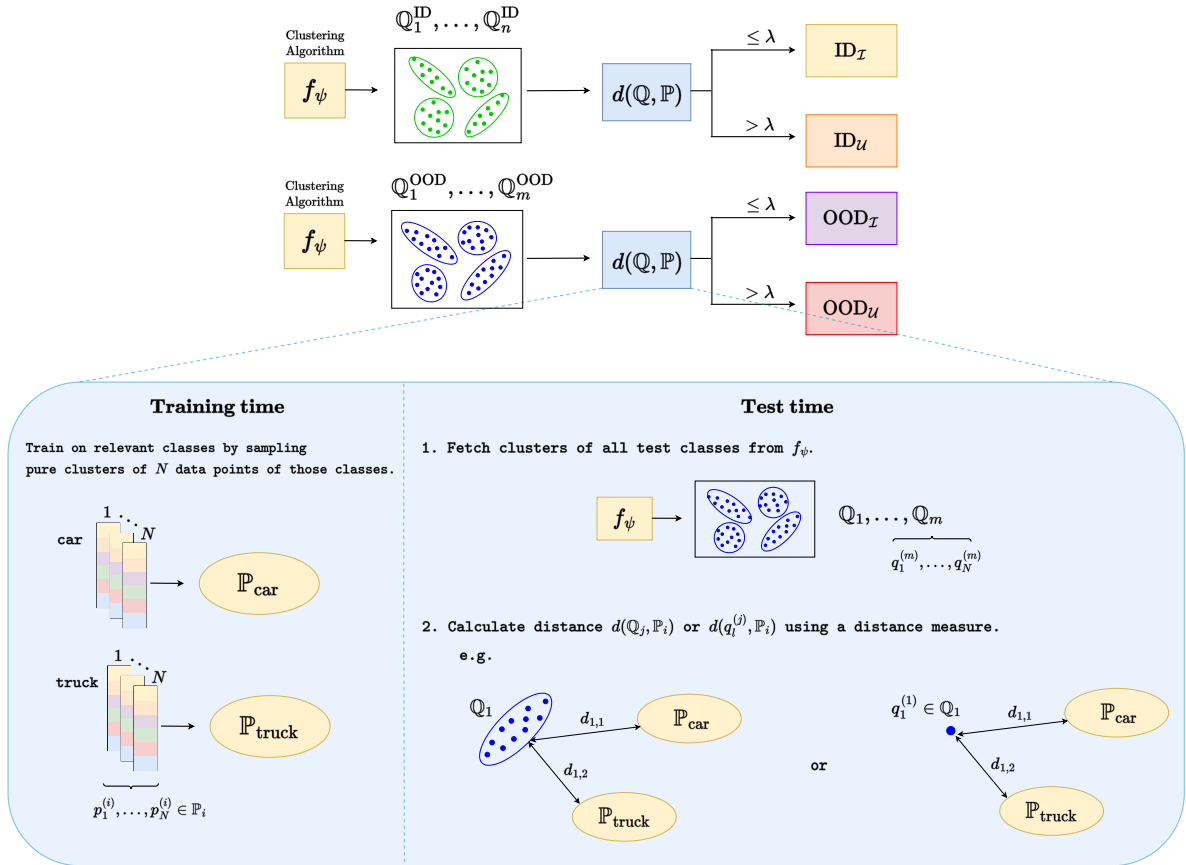


Figure 4.18: Input of this block: ID/OOD clusters \mathbb{Q}_i . Output of this block: Indices of the clusters, separated into 4 groups.

In Fig. 4.18, two cases are portrayed,

- **case 1:** point clouds or 'distributions' $\mathbb{Q}_j, \mathbb{P}_i$ are compared.
- **case 2:** each single test point of $\mathbf{q}_t^{(j)} \in \mathbb{Q}_j$ is compared to a train cluster \mathbb{P}_i individually.

When $k > 1$, we train on more than 1 relevant class \mathbb{P}_i so multiple distances are calculated and averaged.

For **case 1**, the distances between a single test cluster \mathbb{Q}_i , and all training clusters $\{\mathbb{P}_1, \dots, \mathbb{P}_k\}$ are computed, then averaged, and finally repeated for all m test clusters:

$$\frac{1}{k} \sum_{i=1}^k d(\mathbb{Q}_j, \mathbb{P}_i) \rightarrow \text{repeat for each test cluster } \mathbb{Q}_j: j \in \{1, \dots, m\} \quad (4.6)$$

For **case 2**, the distances between a single test point $\mathbf{q}_t^{(j)} \in \mathbb{Q}_j$ against all training clusters $\{\mathbb{P}_1, \dots, \mathbb{P}_k\}$, are computed and averaged as in case 1, and then repeated for all N points in all m test clusters:

$$\frac{1}{k} \sum_{i=1}^k d(\mathbf{q}_t^{(j)}, \mathbb{P}_i) \rightarrow \text{repeat for all } N \text{ points, in each test cluster } \mathbb{Q}_j: j \in \{1, \dots, m\} \quad (4.7)$$

The two cases above are the most elementary methods of calculating how 'far' test data is from our training data; take the average distance from 1 cluster or test point to the training clusters to determine if it is close or far. But this raises the following concerns:

- **How far/close is far/close enough to be (ir)relevant?**
How does one determine a threshold?
- **When k increases, the average distance can increase as well.**
Let $k = 1$ with \mathbb{Q}_1 and \mathbb{P}_1 very close to each other. Now for $k = 2$, let the new training cluster \mathbb{P}_2 not be close to \mathbb{Q}_1 such that $d(\mathbb{Q}_1, \mathbb{P}_2) > d(\mathbb{Q}_1, \mathbb{P}_1) \Rightarrow$ the average distance (Eq. (4.6)) increases.
Does that actually make \mathbb{Q}_1 less relevant now?
Can we use other variations on case 1 and 2, like the $\arg \min d(\mathbb{Q}_1, \mathbb{P}_i)$ to overcome this?
- **What measure for d works best?**
Based on what do we determine how 'far' test data lies? Absolute distances? Statistical distances?
Do we select a parametric approach (based on e.g. $\{\mu, \sigma^2\}$ of each cluster) or non-parametric?

We thoroughly treat the first two concerns in our experiments (Chapter 5) and discuss the last one below.

4.5.2. Overview of Candidate Distance Measures

The term 'distance' does not fully incorporate what we will measure, as the relevance of classes should be based on more than just absolute distances, like statistical properties for instance. The selection of $d(\cdot, \cdot)$ will therefore be based on candidates from 3 categories of measures which we list and treat below:

Category of d	Description
1. (Statistical) Distance Metrics	Distances between point clouds $\mathbb{Q}_j, \mathbb{P}_i$ or points using metric functions.
2. f -divergences	Modelling the clusters $\mathbb{Q}_j, \mathbb{P}_i$ as probability distributions and calculating their dissimilarity.
3. Anomaly Detection Models	Training machine learning models on \mathbb{P}_i and let them determine (ir)relevance.

1. (Statistical) Distance Metrics

Since our UMAP-reduced data lies in a (Riemannian) metric space (\mathcal{M}, g_p) and will be mapped to Euclidean space, we can define a metric g_p to calculate the how far test data is from training data. On top of that, we can model all clusters \mathbb{Q}_j and \mathbb{P}_i as statistical objects, or empirical distributions per class. Modelling it in such a fashion allows for the use of statistical distance metrics that measure more than absolute distances.

Definition 4.5.1 (Metric Space (M, d)) A metric space is a pair (M, d) , where M is a set and d is a metric defined on M , $d: M \times M \rightarrow \mathbb{R}_{\geq 0}$, if for any $x, y, z \in M$ the following hold

1. $d(x, y) \geq 0$,
2. $d(x, y) = 0 \iff x = y$,
3. $d(x, y) = d(y, x)$ (symmetry),
4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

While a multitude of metrics exist, we select present three potential candidates of this category that are widely used for applications like ours:

A Euclidean Distance

$$d_{L_2}(\mathbf{q}_t^{(j)}, \mathbf{p}_s^{(i)}) = \sqrt{\sum_{h=1}^H (q_h - p_h)^2} \quad (4.8)$$

$\mathbf{q}_t^{(j)}, \mathbf{p}_s^{(i)}$ = two points/feature vectors in H -dimensional Euclidean space belonging to clusters $\mathbb{Q}_j, \mathbb{P}_i$
 q_h, p_h = Euclidean coordinates per point.

B Mahalanobis Distance

The Mahalanobis distance is a statistical distance and metric that can be used to measure how distant a point is from the centre of a multivariate normal distribution \mathbb{P}_i . It is a generalisation of the Euclidean distance above, as it is still a distance between two points, but it takes correlation between multiple variables/features into account.

$$d_M(\mathbf{q}_t^{(j)}, \mathbb{P}_i) = \sqrt{(\mathbf{q}_t^{(j)} - \boldsymbol{\mu}_i)^\top \mathbf{S}_i^{-1} (\mathbf{q}_t^{(j)} - \boldsymbol{\mu}_i)} \quad (4.9)$$

$\boldsymbol{\mu}_i$ = the mean of a cluster/probability distribution \mathbb{P}_i .

\mathbf{S}_i^{-1} = positive-definite covariance matrix of a cluster/sample of a probability distribution \mathbb{P}_i .

The Euclidean distance above works fine in the cases that the dimensions of the space are equally weighted and are independent of each other, but this is typically not the case in realistic data-sets. Within feature space each dimension corresponds to a feature (or principal component of features when dimensionality reduction is used), and those features are usually correlated resulting in misleading results by d_{L_2} . We visualise this through Fig. 4.19, which depicts two test points $\mathbf{q}_1^{(1)} \in \mathbb{Q}_1, \mathbf{q}_1^{(2)} \in \mathbb{Q}_2$ from different clusters in 2D, that are compared to the centre of a training cluster \mathbb{P}_{car} of car image feature vectors. In the first case there is no correlation between the features, and in the second the features are into spread out into a certain direction (principal component):

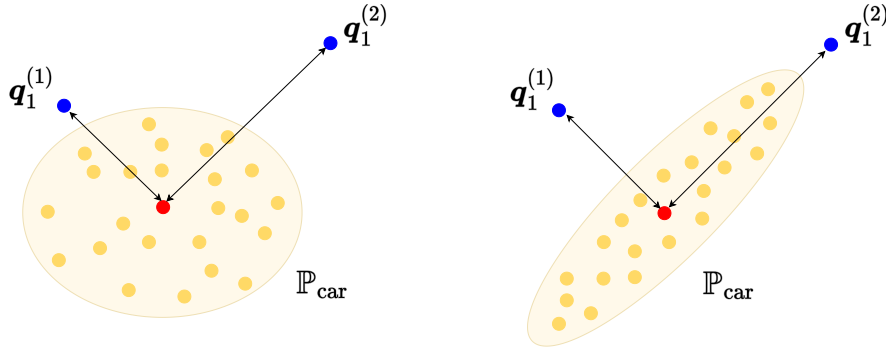


Figure 4.19: Two cases for the distribution of the training cluster \mathbb{P}_{car} , with $\boldsymbol{\mu}_{\text{car}}$ (red).
 Left: features are uncorrelated. Right: Features are correlated in a certain direction.

In both cases the coordinates of $\mathbf{q}_1^{(1)}, \mathbf{q}_1^{(2)}, \boldsymbol{\mu}_{\text{car}}$ are exactly the same, meaning that $\mathbf{q}_1^{(2)}$ is farther away from $\boldsymbol{\mu}_{\text{car}}$ than $\mathbf{q}_1^{(1)}$, but does that always imply that $\mathbf{q}_1^{(2)}$ is also farther away from \mathbb{P}_{car} ? The Euclidean distance would tell us yes, and while this is true for the case where there is no particular correlation, it is clearly not true for the second case above. The Mahalanobis distance takes the covariances of the points of \mathbb{P}_{car} into account, which leads correctly labelling closest point in both cases. The Euclidean distance is essentially a special case of the Mahalanobis distance where the variance of the variables are equal, and covariances zero.

C Wasserstein Distance

The Wasserstein distance is a statistical metric between two probability measures that arised from the field of *Optimal Transport*. We present its definition below and then proceed to briefly treat its derivation, properties, and relevance to our problem, within the setting of Optimal Transport.

We will make use of notions and theorems from measure theory/measure theoretic probability, therefore we refer to [67] as support. In order to ease notation, for each measure or measurable space $(\mathcal{X}, \mathcal{A}, \mu)$ used, we shall not explicitly state the associated σ -algebra \mathcal{A} .

Definition 4.5.2 (Wasserstein Distance) Let (\mathcal{X}, d) be a Polish metric space and let $p \in [1, \infty)$ and $c(x, y)$ a cost function $c : \mathcal{X} \times \mathcal{Y} \rightarrow [0, \infty]$. For any two Borel probability measures μ, ν on \mathcal{X} , the Wasserstein distance of order p between μ and ν is defined by the formula

$$d_{W_p}(\mathbb{Q}_j, \mathbb{P}_i) = W_p(\mu, \nu) := \left(\inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y)^p d\pi(x, y) \right)^{\frac{1}{p}} \quad (4.10)$$

Optimal transport is essentially an optimisation technique used to compare probability measures supported on a metric space, that has blossomed into a separate field finding applications in especially statistical learning, but also optimisation, economics, and pure/applied mathematics [68, 69].

To develop some intuition of the technique, we present the original problem where it was born, studied by Gaspard Monge [70]: mass transportation when moving a pile of sand to a hole with minimal effort. The idea essentially is that one pays a cost for transporting from one measure to another. The pile and hole are modelled by a probability measure μ, ν on spaces \mathcal{X} and \mathcal{Y} respectively, and let $c : \mathcal{X} \times \mathcal{Y} \rightarrow [0, \infty]$ be a cost function where $c(x, y)$ measures the cost (=distance) of transporting one unit of mass $\mu(x)$ from $x \in \mathcal{X}$ to $y \in \mathcal{Y}$. The total effort that should be optimised = mass transported \times distance, given by: $\mu(x)c(x, y)$

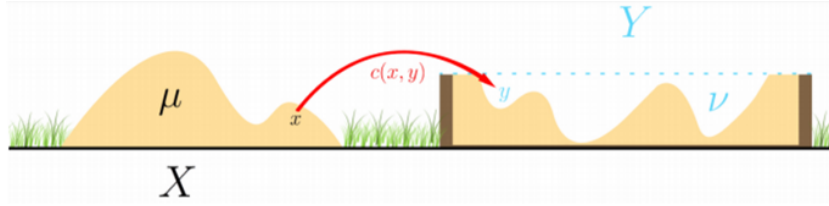


Figure 4.20: Original problem mass transportation studied by Monge, taken from [71].

Both μ and ν can be seen as probability measures, where a cost $c(x, y)$ corresponds to how different they are.

We should define what is meant by transporting one measure to another, i.e. which unit of mass $\mu(x)$ is assigned or transported to which hole $y \in \mathcal{Y}$. In its more general and measure theoretic probabilistic form, this is defined through a *coupling*.

Definition 4.5.3 (Coupling) Let (\mathcal{X}, μ) and (\mathcal{Y}, ν) be two probability spaces. A coupling of μ and ν is a measure π on $\mathcal{X} \times \mathcal{Y}$ (with its tensor-product σ -algebra) such that π admits μ and ν as marginals on \mathcal{X} and \mathcal{Y} respectively, i.e. $\pi(A \times \mathcal{Y}) = \mu(A)$ and $\pi(\mathcal{X} \times B) = \nu(B)$ for all measurable sets $A \subseteq \mathcal{X}$, $B \subseteq \mathcal{Y}$. The set of all couplings of μ and ν is denoted by $\Pi(\mu, \nu)$.

Definition 4.5.4 (Deterministic Coupling or Transport Map) A coupling π is said to be deterministic if there exists a measurable function $T : \mathcal{X} \rightarrow \mathcal{Y}$ such that $\pi = (Id, T)_{\#} \mu$. Here (Id, T) is the map $x \mapsto (x, T(x))$ for $x \in \mathcal{X}$. Therefore, the function T is called the transport map if

$$\nu(B) = \mu(T^{-1}(B)) \quad \forall \nu\text{-measurable sets } B.$$

The $\#$ -notation is widely used in optimal transport and denotes the following: $T_{\#} \mu(A) = \mu(T^{-1}(A))$.

The term $T_{\#}\mu$ is often called the push-forward measure, since T 'pushes forward' the 'mass' of μ to ν , i.e. $T_{\#}\mu = \nu$. Having defined the above, Monge's problem in Fig. 4.20 now can be formalised to

$$\text{What } T \text{ s.t. } T_{\#}\mu = \nu \quad \text{minimizes} \quad \int_{\mathcal{X}} c(x, T(x)) d\mu(x) \quad ?$$

Definition 4.5.5 (Monge problem) Let (\mathcal{X}, μ) and (\mathcal{Y}, ν) be two Polish probability spaces, and $c(x, y)$ a cost function $c: \mathcal{X} \times \mathcal{Y} \rightarrow [0, \infty]$. Monge's problem is then following optimisation problem

$$(MP) := \inf \left\{ \int_{\mathcal{X}} c(x, T(x)) d\mu(x) \mid T: \mathcal{X} \rightarrow \mathcal{Y} \text{ and } T_{\#}\mu = \nu \right\}$$

The Monge problem above is an example of *optimal coupling* or quite literally '*optimal transport*' where one assumes a deterministic coupling T . But such deterministic couplings do not always exist, consider for instance μ equal to the Dirac measure and ν equal to any other measure:

$$\mu = \delta_x \text{ for some } x \in \mathcal{X} \quad \Rightarrow \quad T_{\#}\mu(B) = \mu(T^{-1}(B)) = \delta_{T(x)} \neq \nu(B)$$

By Definition 4.5.4, the above implies that no transport map exists between μ and ν .

The reason why this nonexistence is particularly important for this thesis, is because our data cluster per class is comprised of points (discrete). These clusters or point clouds will be modelled as discrete (probability) measures through Dirac measures.

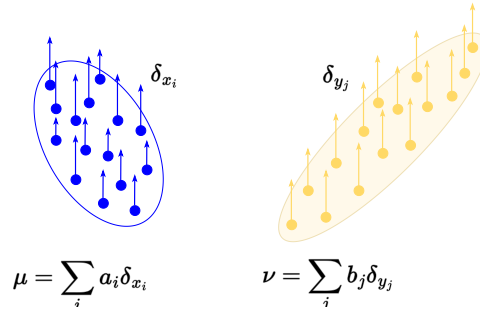


Figure 4.21: Two empirical distributions μ, ν where each point mass is modelled by the Dirac measure with arbitrary weight.

To find a more general form of the Monge problem that suits our needs, we need to find a relaxation of the deterministic nature of transportation; i.e. move away from the notion that a source point x_i can only be assigned to another point or location $y = T(x_i)$. This is known as the *Kantorovich relaxation*, which proposes that the mass $\mu(x_i)$ at any point x_i can be potentially distributed across several locations. Rather than deterministic transport, Kantorovich considers a probabilistic transport or *coupling* instead, which allows what is commonly known now as *mass splitting* from a source toward several targets [72]. This general measure theoretic coupling π was given by Definition 4.5.3, using that to transform the Monge's problem leads to the following formalisation:

Definition 4.5.6 (Kantorovich problem) Let (\mathcal{X}, μ) and (\mathcal{Y}, ν) be two Polish probability spaces, and $c(x, y)$ a cost function $c: \mathcal{X} \times \mathcal{Y} \rightarrow [0, \infty]$. Kantorovich's problem is the following optimisation problem

$$(KP) := \inf_{\pi \in \Pi(\mu, \nu)} \left\{ \int_{\mathcal{X} \times \mathcal{Y}} c(x, y) d\pi(x, y) \mid \pi \in \Pi(\mu, \nu) \right\}$$

In order to further see how the Kantorovich problem links to our problem, suppose that $\mathcal{X} = \mathcal{Y}$, and \mathcal{X} is a metric space with metric d . A natural and intuitive choice for the cost function c would be the distance measured by d . Doing this results in the p th-order Wasserstein distance presented by Definition 4.5.2:

$$W_p(\mu, \nu) := \left(\inf_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{X}} d(x, y)^p d\pi(x, y) \right)^{\frac{1}{p}}.$$

$W_p(\mu, \nu)$ is therefore more than a distance; it is essentially the optimal 'distance' (or 'transport') between two points or distributions, since it minimises the Kantorovich problem. We now treat how the Wasserstein distance can be used for our problem (the discrete case) shown in Fig. 4.21, with added custom notation used for our point clouds $\mathbb{Q}_j, \mathbb{P}_i$.

Suppose \mathcal{M} is the Riemannian manifold on which our data points lie where each cluster of N data points is given by $\mathbb{Q}_j = \{\mathbf{q}_1^{(j)}, \dots, \mathbf{q}_N^{(j)}\} \sim \mu_j$ and $\mathbb{P}_i = \{\mathbf{p}_1^{(i)}, \dots, \mathbf{p}_N^{(i)}\} \sim \nu_i$. All points are treated equally per cluster and therefore have the same mass. The empirical measures are then given by:

$$\hat{\mu}_j = \frac{1}{N} \sum_{i=1}^N \delta_{\mathbf{q}_t}, \quad \hat{\nu}_i = \frac{1}{N} \sum_{j=1}^N \delta_{\mathbf{p}_s}.$$

In this case the Kantorovich problem reduces to

$$\inf_{\pi \in \Pi(\mu, \nu)} \left\{ \frac{1}{N} \sum_s \sum_t \pi_{st} d(\mathbf{p}_s, \mathbf{q}_t) \mid \pi \in \Pi(\mu, \nu) \right\} \quad (4.11)$$

The coupling measures in $\Pi(\mu, \nu)$ can for the discrete case be represented by a bistochastic $N \times N$ matrix $\pi = (\pi_{st})$. This essentially means that all π_{st} are non-negative and sum to 1 across both the rows s and columns t , since each entry of π_{st} is a non-negative real number representing a probability:

$$\Pi(\mu, \nu) := \left\{ \pi \in \mathcal{X} \times \mathcal{X} \mid \pi_{st} \geq 0, \quad \sum_s \pi_{st} = 1; \quad \forall s, \quad \sum_t \pi_{st} = 1; \quad \forall t \right\}$$

This can be compactly written into an expression we will use to *numerically* solve our problem, which is to calculate the distance between two clusters $\mathbb{Q}_j, \mathbb{P}_i$:

$$d_{W_p}(\mathbb{Q}_j, \mathbb{P}_i) = W_p(\mu, \nu) := \min_{\pi \in \Pi(\mu, \nu)} \left(\frac{1}{N} \sum_{s,t} \pi_{st} \|\mathbf{p}_s - \mathbf{q}_t\|_p \right)^{\frac{1}{p}} \quad (4.12)$$

Where we used the Euclidean distance for the metric d , since our UMAP data will be mapped to Euclidean space (Section 4.2.2).

The issue with the optimisation problem in Eq. (4.12), is that it is now a linear programming problem with complexity $\mathcal{O}(N^3 \ln(N))$ arithmetic operations ([68, 73]) meaning that solving it for large sample sizes N will be hard and rendering it impractical or even pointless for many realistic applications. To overcome this drawback, a few numerical techniques have been proposed in contemporary literature to approximate $W_p(\mu, \nu)$, the most seminal and widely used being *Entropic Regularisation* [74] by Cuturi. The theory behind this is extensive and is therefore not treated here. Fortunately, the implementation of this approximated W_p -distance is available through an open source Python package (see Chapter 5).

As a recap, in plain words; the discrete Wasserstein distance in Eq. (4.12) calculates the distance between two clusters $\mathbb{Q}_j, \mathbb{P}_i$ by solving an optimisation problem. This optimisation problem is about minimising the total cost of 'transporting' each point $\mathbf{q}_t^j \in \mathbb{Q}_j$ of one measure μ , to another point $\mathbf{p}_s^i \in \mathbb{P}_i$ of another measure ν , through a transport plan $\pi_{st} \in \Pi(\mu, \nu)$ and a distance metric d .

2. f -Divergences

The f -divergences, denoted by $D_f(P, Q)$, are a statistical distance measure of how dissimilar two probability distributions P, Q are commonly used today in information theory (known as entropy) and statistical learning [74]. Rather than raw points used in the distance measures above, f -divergences expect probability distributions as input (values $\in [0, 1]$). In spite of being a distance measure, an f -divergence does not qualify as a metric, as for instance some $D_f(\cdot, \cdot)$ can be non-symmetric, the triangle inequality does not hold etc.

We start with a general their definition and basic properties, and finally introduce and derive our candidate for D_f in an intuitive fashion.

Definition 4.5.7 (f -Divergence) Let $f : (0, \infty) \rightarrow \mathbb{R}$ be a convex function with $f(1) = 0$. Let P and Q be two probability measures on a measurable space (\mathcal{X}, μ) . If $P \ll Q$ then the f -divergence is defined as

$$D_f(P\|Q) := \mathbb{E}_Q \left[f \left(\frac{dP}{dQ} \right) \right]$$

where $\frac{dP}{dQ}$ is a Radon-Nikodym derivative and $f(0) := f(0+)$. More generally, suppose that $Q(dx) = q(x)\mu(dx)$ and $P(dx) = p(x)\mu(dx)$ for some common dominating measure μ , then we have

$$D_f(P\|Q) = \int_{\mathcal{X}} f \left(\frac{p(x)}{q(x)} \right) q(x) d\mu(x).$$

Remark 4.5.1 For the discrete case, with $Q(x)$ and $P(x)$ being the respective probability mass functions (pmfs) of two discrete clusters $\mathbb{Q}_j, \mathbb{P}_i$, we can write

$$d_{W_p}(\mathbb{Q}_j, \mathbb{P}_i) = D_f(Q\|P) = \sum_{x \in \mathcal{X}} P(x) f \left(\frac{Q(x)}{P(x)} \right).$$

Proposition 4.5.1 (Basic Properties f -Divergences) The following hold for a general f -divergence D_f :

1. $D_{f_1+f_2}(P\|Q) = D_{f_1}(P\|Q) + D_{f_2}(P\|Q)$.
2. $D_f(P\|P) = 0$.
3. $D_f(P\|Q) = 0$ for all $P \neq Q \iff f(x) = c(x-1)$ for some c .
For any other f we have $D_f(P\|Q) = f(0) + f'(\infty) > 0$ for $P \perp Q$.
4. If $P_{X,Y} = P_X P_{Y|X}$ and $Q_{X,Y} = Q_X P_{Y|X}$ then $D_f(P_{X,Y}\|Q_{X,Y}) = D_f(P_X\|Q_X)$.
5. Let $f_1(x) = f(x) + c(x-1)$, then $D_{f_1}(P\|Q) = D_f(P\|Q) \quad \forall P, Q$.

In particular, we can always assume that $f \geq 0$ and (if f is differentiable at 1) that $f'(1) = 0$.

Proof of this proposition can be found in [75].

A KL divergence

Numerous choices for f exist and are used in the field, for instance, setting $f(x) = \frac{1}{2}|x-1|$ is known as the *Total Variation Distance*, and $f(x) = (1 - \sqrt{x})^2$ leads to the squared *Hellinger Distance*. The specific selection of f depends on the use case and the requirements; the Hellinger distance for example, requires third probability measure λ in addition to P and Q for computation (which we do not have). As our candidate, we have selected $f(x) = x \log(x)$, known as the Kullback-Leibler (KL) divergence and the most popular f -divergence as it is a very intuitive way of comparing two distributions. In its discrete form, it is given by:

$$D_{\text{KL}}(Q\|P) = \sum_{x \in \mathcal{X}} Q(x) \log \left(\frac{Q(x)}{P(x)} \right) \quad (4.13)$$

To see the intuition behind the KL divergence from a probabilistic perspective, suppose we want to compare the two distributions $P(x), Q(x)$ above. An intuitive way to start off would be through the likelihood ratio given by $LR = Q(x)/P(x)$ where a ratio smaller or greater than 1 indicates dissimilarity. Suppose that all data-points are identically and independently distributed (i.i.d.), we can

compute this likelihood ratio for the entire set and take the logarithm to ease computation:

$$LR = \prod_{i=0}^n \frac{Q(x_i)}{P(x_i)} \rightarrow \log(LR) = \sum_{i=0}^n \log\left(\frac{Q(x_i)}{P(x_i)}\right) \quad (4.14)$$

where a value of 0 for $\log(LR)$ would indicate perfect similarity since $Q(x)/P(x) = 1$. To calculate this dissimilarity for a sample of N points, we average over N like this: $= 1/N \sum_{i=0}^N \log(Q(x_i)/P(x_i))$. As N increases, the expression of the average gets closer to the expectation of the actual dissimilarity of P, Q :

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=0}^N \log\left(\frac{Q(x)}{P(x)}\right) = \mathbb{E}\left\{\log\left(\frac{Q(x)}{P(x)}\right)\right\} = \sum_{x \in \mathcal{X}} Q(x) \log\left(\frac{Q(x)}{P(x)}\right) \quad (4.15)$$

The above is exactly equal to the KL divergence shown in Eq. (4.13) and concludes its derivation. Given its non-parametric nature, the KL divergence makes an interesting candidate for a distance measure used to compare our train and test clusters.

3. Training machine learning models

As a last option, instead of using statistical metrics/divergences, one can also train machine learning models (classifiers) to discover patterns in the data, in order to determine relevance. The PyOD package of Python contains a great amount of anomaly detection models that can be easily trained and deployed [76]. Most of them are one-class (binary) classifiers that are designed to only classify data into 'inlier' or 'outlier', based on strange or anomalous patterns in the data. This will be used as a baseline to compare the previous two categories to.

Unlike the statistical distance metrics and f -divergences, which are based on comparing two probability measures, these models are entirely point-based. The idea is to train these models on points of relevant classes ($\{\mathbf{p}_1^{(i)}, \dots, \mathbf{p}_N^{(i)}\} \in \mathbb{P}_i$) so that they will be able to tell whether new points $\{\mathbf{q}_1^{(j)}, \dots, \mathbf{q}_N^{(j)}\} = \mathbb{Q}_j$ at test time are inliers (relevant) or outliers (irrelevant).

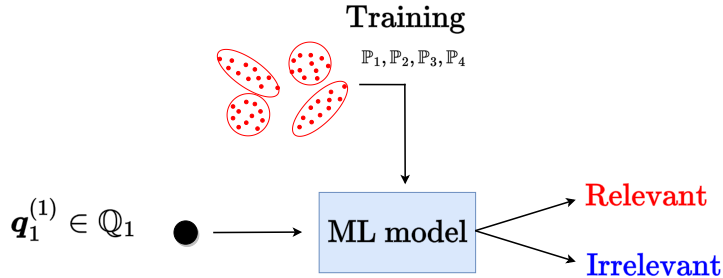


Figure 4.22: A single data point $\mathbf{q}_1^{(1)}$ presented at test time, after the model is trained on points from clusters $\mathbb{P}_1, \dots, \mathbb{P}_4$. No probability distributions are used here during both training and testing.

We have selected a few of the many models available, based on familiarity and popularity:

Anomaly Detection Model
1. Support Vector Machine (SVM)
2. Principal Component Analysis (PCA)
3. Isolation Forest (IF)
4. Isolation using Nearest Neighbour Ensemble (iNNE)
5. Feature Bagging (FB)

The full documentation of how each model works and is trained can be found on the PyOD website [76]. In order to sketch the idea of how the process in Fig. 4.22 works, we treat the general steps of one of the models above, which is the PCA model for anomaly detection.

Consider the PCA detection model for 3D UMAP data. While PCA is usually deployed to perform dimensionality reduction, PyOD uses it to calculate the principal components of all training data points. Based on these principal components, a hyperplane H_{PCA} is drawn that is close to the training data. The model assumes that outliers (irrelevant classes) will be further away from H_{PCA} than inliers (relevant classes) will be. Based on a built-in threshold λ , PCA calculates the distance for each point at test time and determines which ones are inliers, and which ones are outliers.

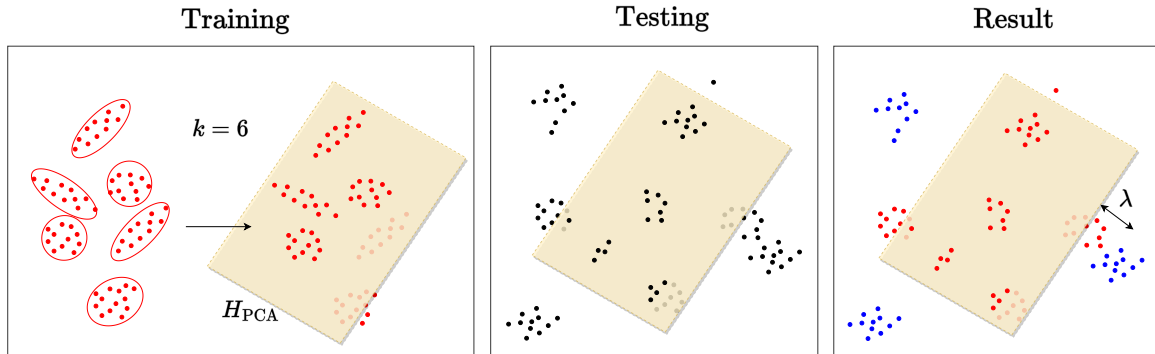


Figure 4.23: Determining relevance through a PCA anomaly detection model, trained on points of $k = 6$ relevant classes: black=points at test time before relevance detection, red=relevant class, blue=irrelevant class.

While the other models in Section 4.5.2 each have their own way of training/testing, they all perform a point-based analysis, as opposed to the distance metrics and f -divergences that assume underlying distributions.

4.5.3. Motivation & Challenges

In this sub-section answer the research questions on the statistical properties (SRQ2) and dimensionality (SRQ3) of the data, combined with the associated challenges for the candidates for $d(Q_j, P_i)$.

1. Statistical Properties:

Both the Euclidean and Mahalanobis distance are parametric methods as they are based on $\{\mu, \sigma^2\}$ only. Such a centroid-based techniques assume the data to be Gaussian (elliptical point clouds/distributions). Fig. 4.17 shows that our data clearly does not exhibit that behaviour. Can one still use these distances then, especially since non-parametric techniques such as Wasserstein, KL-divergence, and ML models are available. The answer is yes. The Euclidean/Mahalanobis are the *natural* distances for elliptical distributions and therefore work optimally for Gaussian data, but ultimately they are still metrics and can therefore calculate distances between non-Gaussian distributions as well. While the Wasserstein distance and KL divergence enjoy several nice properties over parametric methods, including structure preservation, existence in smooth and non-smooth settings, [77] invariance to invertible (linear) transformations, it is worthwhile to use both, in order to see to what extent further statistical information of distributions, e.g. higher moments, matter for relevance detection. Perhaps the centroid of each distribution suffices, and to fully test this out, we shall not use the Mahalanobis distance of our testing. We are curious in finding out how well the more simple parametric distance can perform. To still research the role of cluster shape, we use the non-parametric methods.

2. Dimensionality:

When the dimensionality of the data increases, the volume of the high-dimensional space increases so fast that the available data become sparse; points will concentrate towards the edges of your space which will cause metrics, especially the Euclidean norm, to yield misleading results [78]. A common issue with higher dimensionality within statistics is that the dimension can get close to, or even exceed the amount of observations N . If we have 200 samples per class, we easily exceed that number when using pure CLIP vectors $\in \mathbb{R}^{512}$. This is problematic because it is now impossible to find a model that can describe the relationship between the predictor variables and the response variable and lead to unstable solutions (see Ch.18 of [50]). We should therefore take this into account when reducing our CLIP vectors through UMAP.

5

Experimental Setup & Results

Chapter goals

1. **Presenting our hypotheses, data-set, and experimental setup** - [Section 5.1](#) to [Section 5.4](#)
These sections are essentially explanatory notes on what we aim to learn, and how the pipeline from the previous chapter will be used and tested; it serves as preparation for the experiments that follow.
 2. **Conducting experiments to answer the remaining research questions** - [Section 5.5](#) to [Section 5.7](#)
Each of these sections is centred around a specific research question. SRQ5 is on how our model's performance can be improved by increasing prior information about the domain; it is the main experiment where multiple tests will be conducted. The insights of this experiment led to two additional research questions (SRQ6 & 7) about the effect of clustering, for which we did separate tests.
 3. **Evaluating & analysing the results** - [Section 5.8](#)
In the final section, we evaluate and check whether the results are in line with our hypotheses, to what extent the research questions have been answered, and where there is room for improvement.
-

5.1. Hypotheses

- H1: Relevance detection can be done in a more robust way when using a cluster-based approach.**
Every class, both relevant (e.g. bus), or irrelevant (e.g. dog), has its own natural spectrum to which their instances belong; buses and dogs come in all shapes, sizes, colours, and models/species. We hypothesise that using statistical distances between clusters or 'distributions' better take atypical points (strange buses and/or dogs) into account, making it a more robust relevance detector than individual point-to-point distances.
- H2: Statistical properties, such as cluster shape, play an important role in relevance detection.**
Apart from absolute distance in feature space, we hypothesise that the actual shape of each cluster matters as well. We hope to discover that certain classes that are related to each other have similar cluster shapes as well. To capture this we will be using non-parametric methods like the Wasserstein distance and the KL-divergence.
- H3: Relevant OOD classes are better discovered when we have more prior domain information.**
Given some user input, e.g. the vehicle domain is of interest and all other classes are not, we aim to test whether relevance detection performance will rise when increasing the number of vehicles in the training set. We hypothesise that increasing prior domain knowledge, for instance, training on 4 vehicle classes (car, bus, truck, van) rather than training on 1 class (car), will improve the detection of relevant and irrelevant classes at test time.
- H4: Our pipeline will perform better on low-dimensional data.**
As stated before, when the dimensionality of the data increases, the volume of the high-dimensional space increases so fast that the available data become sparse; points will concentrate towards the edges of your space which will cause metrics to yield misleading results [78]. We hypothesise that (ir)relevant distributions will be more challenging to detect in higher dimensions.

5.2. Data-set

The DomainNet data-set [53] has been selected for our testing as it contains a wide variety of common objects, ranging from animals, fruits, and nature landscapes, all the way to images of screwdrivers, vehicles, and shoes. In total the DomainNet data-set contains 345 of such classes, over 6 *domains* (hence its name): clipart, painting, drawing, and real photographs. Given our problem, we are in need of a realistic data-set and therefore select the DomainNet-real portion for our testing. Within DomainNet-real, the 345 classes each have hundreds of images with resolutions in the range of 200×200 to 600×600 pixels. Fig. 5.1 presents an overview of the complete data-set over the 6 domains, of which the real segment is of interest.

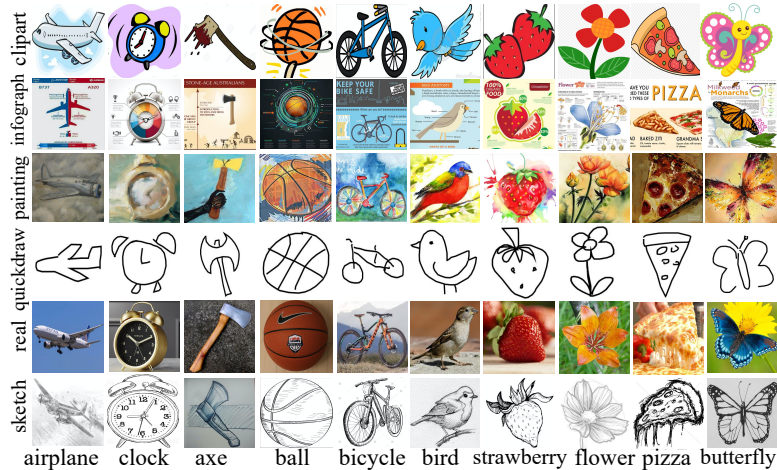


Figure 5.1: Overview of the DomainNet data-set where the second row from below is the real domain. Taken from [53].

We have manually searched this data-set for classes that align with our use case. Out of all 345 classes, 21 classes are vehicles, i.e. are of interest. For all following experiments, we have narrowed down our interest from vehicles to road vehicles only, of which there are 11 classes (Fig. 5.2). By narrowing down our interest to road vehicles only, we expect the problem to become more challenging and realistic: our model has to learn that airplanes, in spite of being a vehicle, are not relevant anymore since they are not *road* vehicles.

All vehicle classes count = 21	Road vehicles count = 11	Bikes count = 2	Aircrafts count = 3	Watercrafts count = 5
aircraft_carrier	ambulance	bicycle	airplane	aircraft_carrier
airplane	bulldozer	motorbike	airplane	cruise_ship
ambulance	bus		airplane	sail_boat
bicycle	car		airplane	speedboat
bulldozer	firetruck		airplane	submarine
bus	pickup_truck		airplane	
car	police_car		airplane	
cruise_ship	school_bus		airplane	
firetruck	tractor		airplane	
flying_saucer	train		airplane	
helicopter	truck		airplane	
motorbike			airplane	
pickup_truck			airplane	
police_car			airplane	
sail_boat			airplane	
school_bus			airplane	
speedboat			airplane	
submarine			airplane	
tractor			airplane	
train			airplane	
truck			airplane	

Figure 5.2: Vehicle classes of the DomainNet data-set, where the green classes correspond with the classes of interest.

The images of all 345 are loaded into the CLIP neural network to generate feature vectors. Although all images come from the same data-set, each DomainNet image has its own distinct resolution, while CLIP only accepts images of a fixed resolution of 224×224 pixels. In order to ensure compatibility, all images are cropped with `trn.transform.crop` of the PyTorch Python framework, to fix the resolution.

5.3. Design Adaptation

In order to fully test the potential and validity of the relevance detection module (Section 4.5), we wanted to diminish, if not remove, any bottlenecks in the pipeline. As a consequence, it was decided to remove the OOD detection module (Section 4.3) from the experimental setup. We would like to stress that the OOD detection module is still a valid and important part of the pipeline, and should therefore still be a part of future extensions of this research, or future implementations that will be used in practice. The rationale behind its removal though, is that the GEM scoring function’s false positive rate (FPR) is quite high (Section 3.2.2), resulting in many known images being falsely flagged as OOD, and vice versa. These error margins were known to us from the start and we anticipated on these effects accordingly by setting out various scenarios (Section 4.3.2). We wanted to ensure that, at least for our study on relevance detection which is a novel application to the best of our knowledge, no *unnecessary* bottlenecks are present, in order to prove whether our idea is a valid method. We have adapted and developed our detection module setup in such a way now that it is independent of a scoring function and can be used separately. This is treated in step 2 of the next section.

5.4. Main Setup

SRQ4: *How can the performance of our solution pipeline be measured?*

We treat the experimental setup with the same notation laid out in Section 4.5. The following sub-sections build our experimental setup in three steps, for the 345 classes of the DomainNet data-set that we test on.

5.4.1. Step 1: Increasing prior domain knowledge through training

To perform relevance detection, one needs to establish what relevant means first; a starting point. After a user provides such a starting punt, i.e. input on what kind of classes are relevant, these classes are selected as training classes and are taken through the pipeline to obtain training clusters $\mathbb{P}_1, \dots, \mathbb{P}_k$. In our case, \mathbb{P}_i are all road vehicles (see Fig. 5.2).

We define the information on the k relevant classes specified by the user as domain knowledge, and in realistic use cases, this information is known beforehand. We aim to test the impact of this prior domain knowledge by using varying amounts of it for relevance detection. We will run our relevance detection module for 10 iterations, where we vary k from 1 to 10. Not increasing k to 11 ensures that in the last iteration there will still be an unknown relevant class left on which we can test. We are interested in seeing how much the performance of relevance detection increases after each iteration, that is if it increases at all.

Given the 11 road vehicles, Table 5.1 depicts one possible sequence of increasing the prior domain knowledge, where in each iteration, more knowledge is provided to our model by training it on an extra class.

Table 5.1: One possible way of incrementally increasing training classes.

Iteration										
1	ambulance									
2	ambulance	bulldozer								
3	ambulance	bulldozer	bus							
4	ambulance	bulldozer	bus	car						
5	ambulance	bulldozer	bus	car	firetruck					
6	ambulance	bulldozer	bus	car	firetruck	pickup_truck				
7	ambulance	bulldozer	bus	car	firetruck	pickup_truck	police_car			
8	ambulance	bulldozer	bus	car	firetruck	pickup_truck	police_car	school_bus		
9	ambulance	bulldozer	bus	car	firetruck	pickup_truck	police_car	school_bus	tractor	
10	ambulance	bulldozer	bus	car	firetruck	pickup_truck	police_car	school_bus	tractor	train

Using the exact sequence in Table 5.1 only implies that the experiment can be done in just 10 iterations. This is not true, since the order and combinations of the classes that are trained matter. To illustrate this, consider two models are trained for 10 iterations and decide which model has the unfair advantage:

- Model A starts with 'car' in iteration 1 and learns 'police_car' in iteration 2.
- Model B starts with 'car' in iteration 1 and learns 'truck' in iteration 2.

Even after only two iterations, model B has the advantage since it has already seen two vehicles that are considerably different, while model A has simply only been trained on 'car' + another type of car. This advantage helps model B generalise much better for new test samples as it has 'seen' a larger variety of relevant classes. We, therefore, need to treat more combinations and take their average to make this a fair test. In iteration 1 alone, there are 10 combinations to pick the first class when we select each class only once. In iteration 2 there are 90 combinations and this blows up for further iterations. The total amount of combinations without duplicates that can be made for the scheme in Table 5.1 is equal to the number of proper subsets. Given a set with k elements, the amount of proper subsets is given by $2^k - 2$. This contains the empty set and the power set as well, which will not be used. Removing these two sets will lead to a set of

$$2^k - 2 = 2046 \text{ combinations for } k = 11.$$

Running our relevance detection module 2046 times will be impractical and computationally infeasible. To overcome this issue, we sample 10 combinations per iteration, reducing the number of runs from 2046 to 100. Sampling 10 combinations for each iteration means we obtain 10×10 model outputs. Taking the average per iteration leaves us with 10 average performance values, where each value indicates the performance at $k = 1, \dots, 10$.

Fig. 5.3 displays the 10 average values (vertical axis) with the corresponding standard deviation, for iterations 1-10 (horizontal axis). This is an example plot of artificial data we manually generated. We expect that increasing the number of classes will yield higher performance and that the performance will saturate after a certain amount of prior domain knowledge.

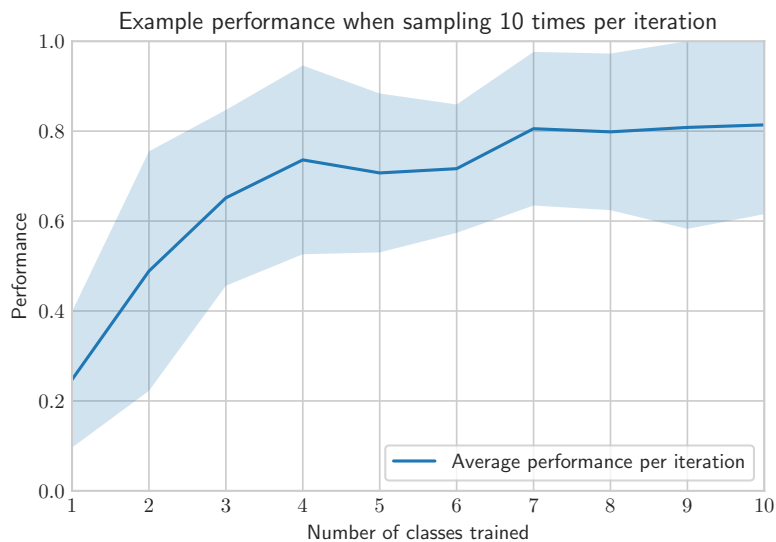


Figure 5.3

The experiments in later sections will show to what extent our expectations above align with reality.

5.4.2. Step 2: Calculating a relevance score

Recall from [Section 4.5.2](#) that we use three categories of distance measures to determine relevance: two are statistical distance metrics and one is ML anomaly detection models. The ML anomaly detection models use a built-in point-based approach to determine what is relevant or not; it is executed automatically for us. For statistical distances, we need to build the relevance detection ourselves, which we illustrate below using a 'ranking' approach.

Having established what 'relevance' means in step 1, we rank the 345 classes into three groups from most to least relevant:

1. **ID_R: Known relevant classes that we have trained on:** \mathbb{P}_i
 These classes are the prior domain knowledge specified by the user, road vehicles in our case. These seen during training and testing. \rightarrow varying from $k = 1, \dots, 10$ classes.
2. **OOD_R: Unknown relevant classes that we have not trained on**
 These are unseen road vehicles that have not been trained on and are only seen during testing. This group is the complement of the first group. $\rightarrow 11 - k$, e.g. our model trains on $k = 2$ classes, then the amount of **OOD_R** = $11 - 2 = 9$ classes.
3. **OOD_I: Unknown irrelevant classes that we have not trained on**
 These are anything but road vehicles (screwdrivers, airplanes, animals etc.) in our case and are only seen during testing. \rightarrow fixed at $345 - 11 = 334$ classes, meaning that in our experiments, we will always encounter all 334 irrelevant classes.

[Fig. 5.4](#) shows how we expect the clusters to be ordered if we were to rank the 345 clusters visually from most to least relevant based on their distances as well; the more relevant a test cluster \mathbb{Q}_j , the smaller its distance to the training clusters \mathbb{P}_i . Note that in feature space, test clusters \mathbb{Q}_j can be on any side of the training clusters ([Fig. 4.17](#)), but since we are ranking them, we sketch all clusters from left to right.

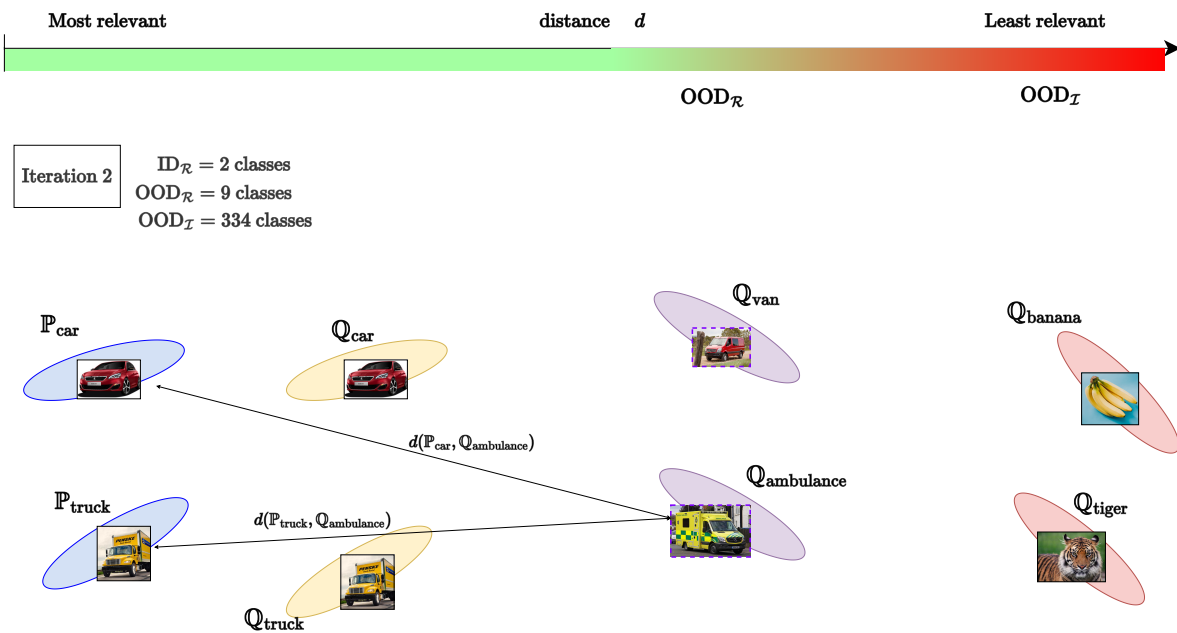


Figure 5.4: As an example we present the one possible second iteration where we train on cars and trucks. Note that we test on all 345 classes at every iteration, despite not being able to show all their clusters in this figure.

The first phase of relevance detection is to calculate for each cluster $\mathbb{Q}_1, \dots, \mathbb{Q}_{345}$ at test time, the average distances to all training clusters $\mathbb{P}_1, \dots, \mathbb{P}_k$ previously shown in [Fig. 4.18](#). This leads to 345 distances; one average distance \bar{d} per test cluster. To convert each \bar{d} into a relevance score r , we define $r = 1/\bar{d}$, so that a large distance leads to a small relevance score and a large distance to a high relevance score. We proceed by sorting these 345 scores from smallest to largest in a vector $\mathbf{r} = \{r_1, \dots, r_{345}\}$, where the largest r implies 'the most relevant class' and vice versa. Finally, we normalise all scores in \mathbf{r} to obtain scores in $[0, 1]$, where a

score of 0 indicates 'completely irrelevant' and 1 indicates 'completely relevant'. What is left, is to determine a threshold λ within $[0, 1]$ such that relevant clusters will be classified as relevant, and vice versa.

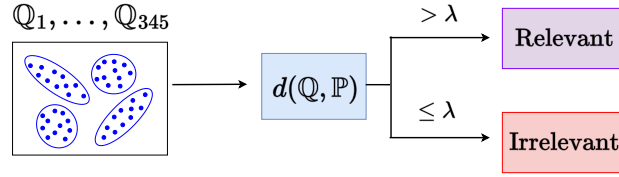


Figure 5.5: Based on the average distance \bar{d} per test cluster, the threshold λ determines relevance.

Fig. 5.5 portrays the workaround for removing the OOD detection module. Instead of having two separate batches shown in Fig. 4.18, we take a more general approach by mixing all clusters in one batch of 345 classes. This eliminates the OOD detection module's bottleneck and has no negative effects on relevance detection, on the contrary, it improves the validation and testing of relevance detection.

5.4.3. Step 3: Threshold determination and measuring performance

Step 2 leaves us with a list of scores that do not mean anything without a threshold λ that separates them into relevant and irrelevant, but how does one determine λ ? When is a score high enough that the corresponding class is relevant? Instead of manually attempting to pick a perfect λ , it is wiser to iteratively determine a λ that will classify most of the relevant classes as 'relevant', and most of the irrelevant classes as 'irrelevant'.

The most common evaluation metrics to determine such a threshold are known as the True Positive Rate (TPR) and False Positive Rate (FPR). Let all relevant classes have a perfect score of 1, and all irrelevant classes have a score of 0.0. The TPR then denotes how often a true relevant class is correctly classified as 'relevant' (true positive). Conversely, the FPR denotes how often an *irrelevant* class is classified as 'relevant' (false positive). These metrics are expressed as a ratio (Eq. (5.1)); TPR is the number of true positives over the number of positives P and the TPR is the number of false positives over the number of negatives N .

$$TPR = \frac{TP}{P}, \quad FPR = \frac{FP}{N} \quad (5.1)$$

Lower thresholds classify more classes as irrelevant while higher thresholds more classes as relevant. Our goal now is to find a λ such that the TPR is maximised while minimising the FPR. This is commonly done using the receiver operating characteristic (ROC) curve which plots the FPR against the TPR for various λ . In Fig. 3.14 two ROC curves are drawn, where the blue dot indicates the optimal threshold of the blue curve.

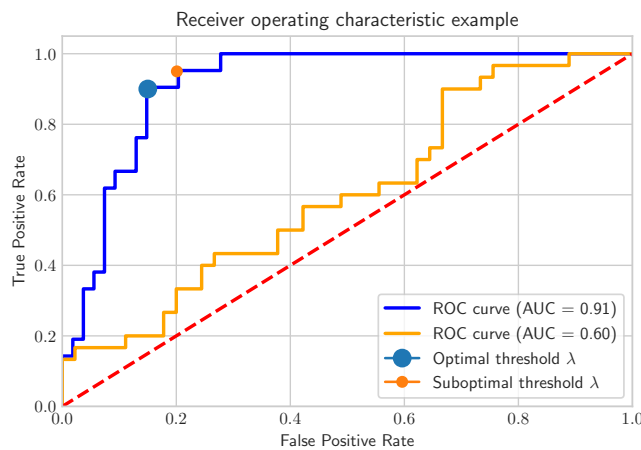


Figure 5.6: Example ROC curves, where the red dashed line indicates the ROC curve of a random classifier.

The optimal threshold is at a sweet spot where TPR is the highest relative to the FPR. The dot on the blue curve in Fig. 5.6 is an optimum since tweaking λ to increase the TPR (moving to the orange dot on the

right) leads to a significantly higher FPR (from 0.15 to 0.25), while only a moderate increase in FPR (from 0.9 to 1.0) and vice versa.

Now that we can determine an optimal threshold, how do we measure our model's performance? The metric that is linked to this is the **Area Under the ROC Curve (AUC)**. A high AUC implies that a model is better at predicting relevant classes as relevant, and irrelevant classes as irrelevant, than a model with a low AUC. This is because, for larger areas, an optimal λ can be found with high TPR + low FPR. This is illustrated in Fig. 5.6 where the blue curve scores significantly better with a higher AUC, than the orange curve. Ideally, the ROC curve should rest in a region where TPR is high and FPR is 0, yielding a perfect AUC of 1.0. An AUC of 0.5 means that the model cannot make a distinction between relevant and irrelevant, i.e. the classification is completely random, meaning that our models should score higher than 0.5 to be meaningful at all. We shall use the AUC to assess the performance of all detection models (statistical distance metrics, f -divergences, and ML anomaly detection models) as it contains the information of both the TPR and FPR.

In step 1 of this setup, we explained that the order of training the model on prior information can cause unfair advantages and that we should take the average performance of 10 combinations to obtain the plot in Table 5.1. Since the AUC will be the metric of our choice in the experiments that will follow, we will have to calculate an AUC score ten times per iteration and take its average. Fig. 5.7 depicts this process per iteration, where the AUC of 10 ROC curves is calculated and then averaged. Each average is one point in Fig. 5.3, repeating this for all 10 iterations results in the full graph.

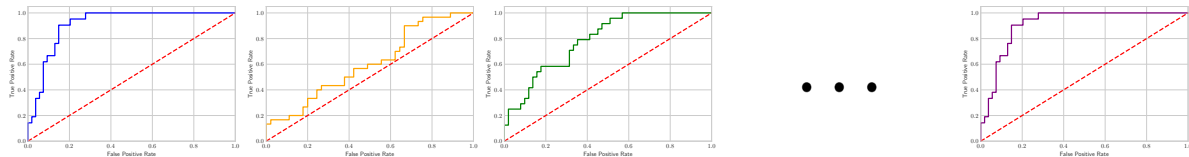


Figure 5.7: Ten example ROC curves of a single iteration shown in Table 5.1. Each ROC curve corresponds to one combination per iteration. Imagine that the blue curve belongs to a model that is trained on 'car' and 'truck', and the orange curve to a model that is trained on 'car' and 'police_car'. The 'blue model' generalises better, hence its higher AUC. Taking the average of the AUC values of each curve results in an average AUC per iteration.

The description of the experimental setup is now complete, and it has been demonstrated how the conceptual pipeline outlined in Chapter 4 can be put to the test in a real-world experiment using real data. The experiments and results in the following sections are based on this setup.

5.5. Experiment 1: Increasing Prior Domain Knowledge

SRQ5: How does prior information about the domain and related classes to the OOD classes of interest, help detection?

In our first experiment, we deploy our models in the exact process laid out in the main setup above. For each model, we incrementally train from 1 to 10 relevant vehicles for both 3D and 10D UMAP data. By doing so, we shall not only see the effect of increasing prior domain knowledge for a variety of models, but also experience the potential effects of (high) dimensionality (which answers **SRQ3**).

We split this experiment in two parts, testing separately on the following methods:

1. **Point-based methods** (ML models)
2. **Distribution-based methods** (statistical distance metrics & f -divergence)

Our objective is to find out to what extent, knowledge of even a single class generalises well to other classes, i.e. how well can our models perform relevance detection when only trained on 1 class? Does their performance significantly increase, or even increase at all when we incrementally train on more classes?

5.5.1. Point-based detection models

In [Section 5.5.1](#), the results of all models are presented and plotted relative to $AUC = 0.5$ (red), which is the score of a random classifier. They simply serve as a quick and high-level overview of all models. We shall zoom in on the best performers next.

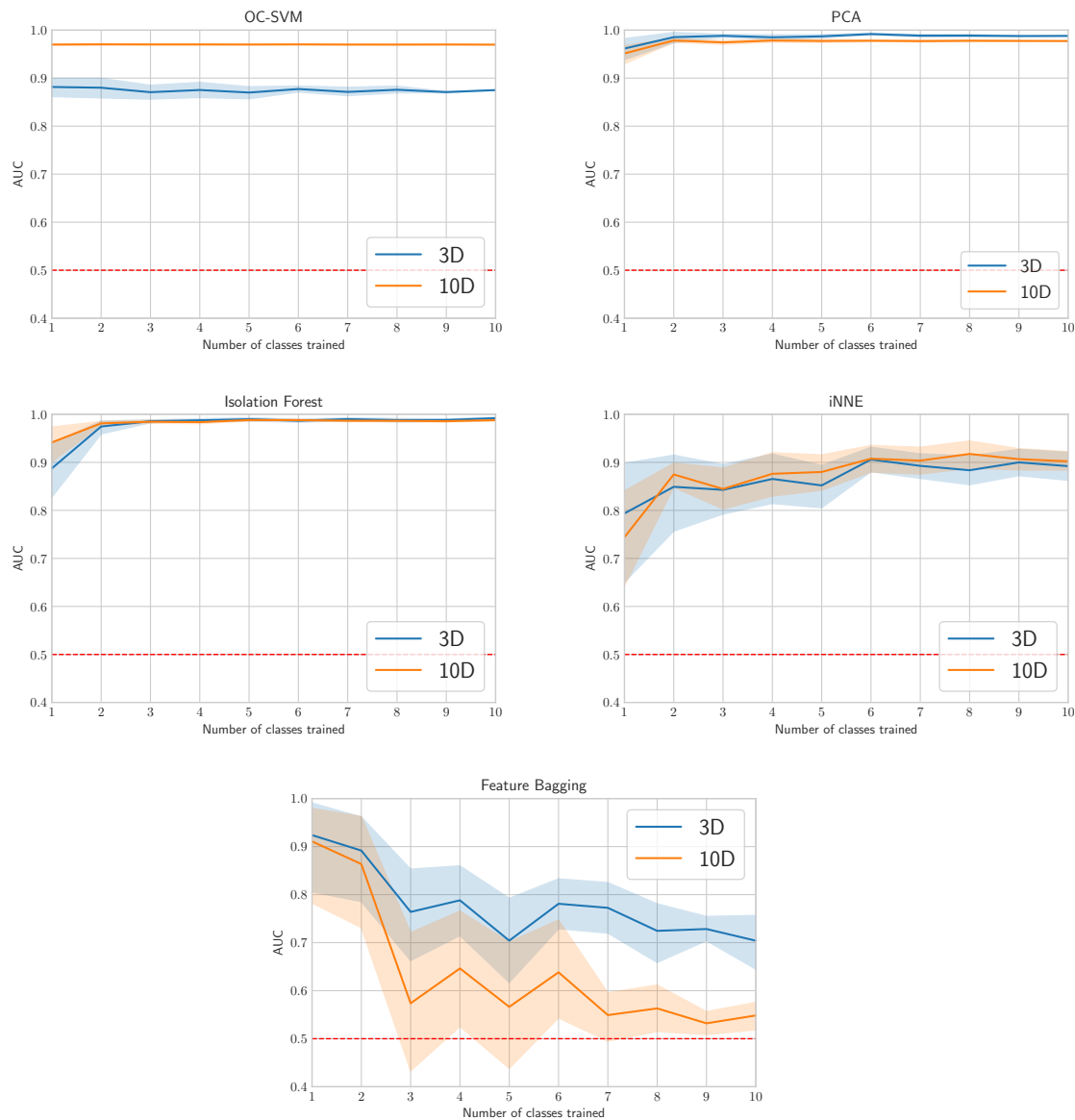


Figure 5.8: All five models from left to right, top to bottom: SVM, PCA, IF, iNNE, FB

Through visual inspection, it is already clear that PCA outperforms all others, since it reaches the highest AUC scores and exhibits the lowest standard deviation. While SVM and Isolation forest show high performance as well, especially for the 10D data (orange curves), we found that PCA still tops them when directly plotting them against PCA. The last two models (iNNE and Feature Bagging), however, do not do so well. Since the goal of using point-based methods was to obtain a baseline model (which we now have), we shall not investigate why some models perform less well and move on instead.

Zooming in on the best performer (PCA), for both 3D and 10D, leads to the following curves in [Fig. 5.9](#).

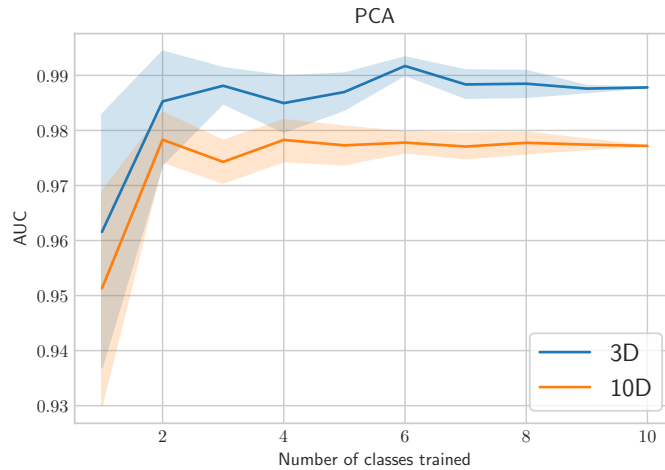


Figure 5.9: Zooming in on PCA.

The main insight obtained from Section 5.5.1, is that (most of) our point-based models generalise surprisingly well. Even when only training on 1 relevant class, PCA achieves AUC scores in the range of 93–99%, which are exceptionally good scores. This implies that irrelevant and relevant data points in feature space are far apart to such an extent, that the PCA hyperplane (Fig. 4.23) can make an accurate distinction between them. It is hard to draw any general conclusions regarding the dimensionality of the data as of now since the models in Section 5.5.1 are each fundamentally different and therefore show contrasting behaviour, but based on our tests so far, 3D data is the better choice for point-based methods.

Table 5.2: Best performing dimension per model, where "same" indicates that 3D and 10D have similar performance.

	Best dimension
OC-SVM	10D
PCA	3D
IF	same
iNNE	same
FB	3D

5.5.2. Distribution-based detection models

For the distribution-based models, we use a clustering algorithm as pointed out in Section 4.4, since the UMAP output is comprised of points, rather than clusters Q_1, \dots, Q_{345} . We selected K-means as the clustering algorithm in place of a density-based model like DBSCAN, despite some of its drawbacks. We discuss this in Section 5.7. Furthermore, we would like to point out that K in K-means was fixed at $K = 34$, since our models perform well for this value. Why this is the case, and how K was determined shall be treated in a separate study in Section 5.7 as well. For now note that all distribution-based models were tested on clustered data, where relevance detection worked best when data was divided into 34 clusters, rather than 345. Thus, numerous classes might be conveniently combined without negatively affecting relevance detection.

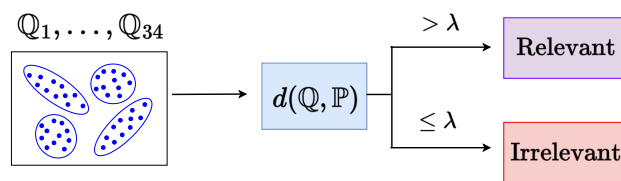


Figure 5.10: Test data of 345 classes, clustered into 34 clusters by K-means before fed to the distribution-based models.

In the following figures, we present and compare the results of the distribution-based methods using PCA as a baseline for 3D and 10D data.

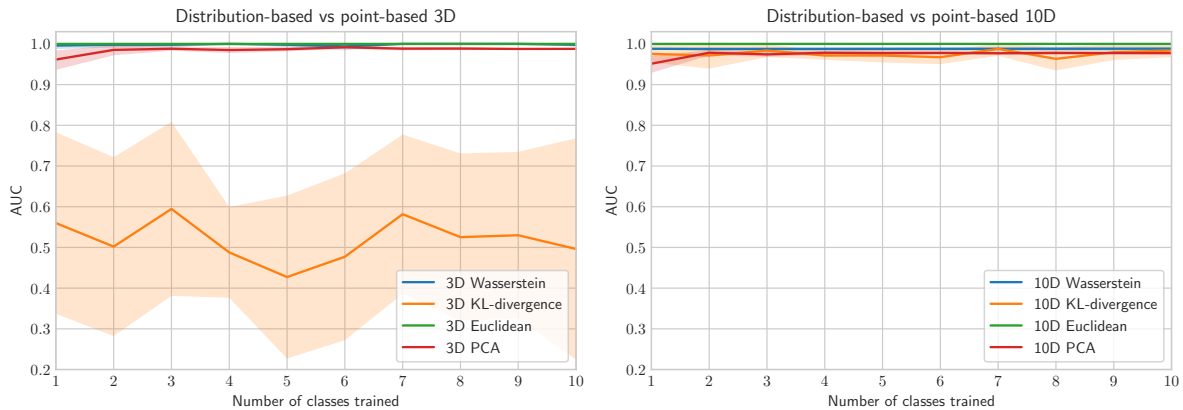


Figure 5.11: Distribution-based vs point-based. Left: 3D, right: 10D.

The plots of Fig. 5.11 feature a variety of events, so to fully grasp the displayed model behaviour we study the following:

1. **Understanding what went well:**
Euclidean & Wasserstein in 3D, Euclidean, Wasserstein & KL-divergence in 10D
2. **Understanding what went wrong:**
KL-divergence in 3D

To obtain a clearer picture, we zoom in on what went well in Fig. 5.12, by omitting the poorest performing distribution-based model.

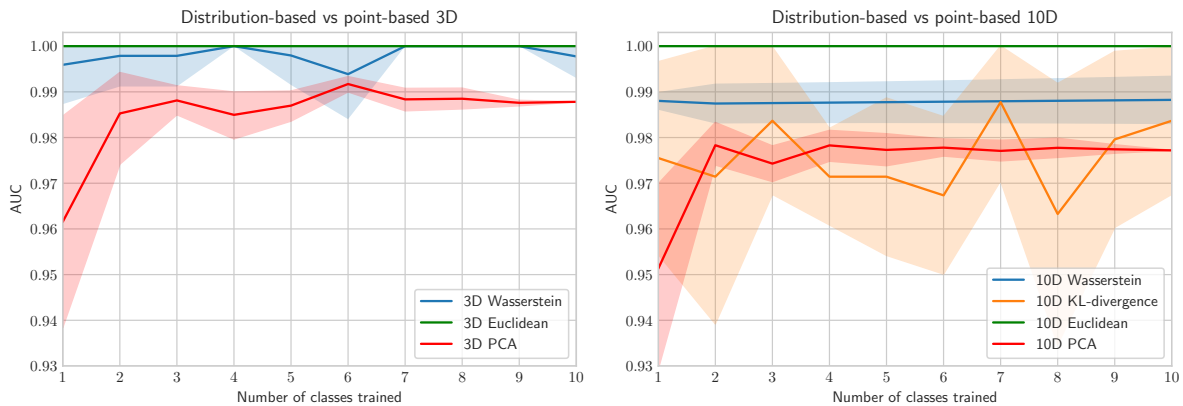


Figure 5.12: Distribution-based vs point-based (zoom). Left: 3D, right: 10D.

In 3D, both Euclidean and Wasserstein outperform point-based models, especially when one trains on 1 class only: PCA averages around 96% for 1 class, while Wasserstein and Euclidean average around 99.63% and 100% respectively. In 10D, the out-performance holds for Euclidean and Wasserstein as well. The main insight obtained here is that distribution-based models can beat the point-based baseline models and can especially add value when one does not have access to much prior information.

The perfect scores of the Euclidean distance raise concerns about its validity. Also, how can some AUC scores be so high, while others are so low? Conversely, why is Wasserstein so poor in 10D while it works so well in 3D? (the same holds for KL-divergence, where the opposite is true). Are there any potential flaws in our data, models, or methodology? After looking for these potential flaws and finding nothing, we have reason to suspect that the way the distances are computed is what is causing this disparity in performance.

Why else would the performance be so different for various measures, when the data and methodology are exactly the same? We believe that for each of the three distance measures, the ratio between the distances to relevant classes/ the distance to irrelevant classes might be different. In other words, for some measures, irrelevant classes might be (much) farther away from the training clusters, than relevant classes:

$$d(Q_{\text{irrelevant}}, \mathbb{P}) \gg d(Q_{\text{relevant}}, \mathbb{P}) \quad (5.2)$$

To prove and further investigate this for various measures, we calculate the ratio between these distances for each metric as follows.

$$\text{Ratio} = \frac{d(Q_{\text{irrelevant}}, \mathbb{P})}{d(Q_{\text{relevant}}, \mathbb{P})} \quad (5.3)$$

We repeat and plot this for each iteration to see if these ratios diverge when more prior information is added. Fig. 5.13 depicts the values of these ratio in 3D (left) and 10D (right).

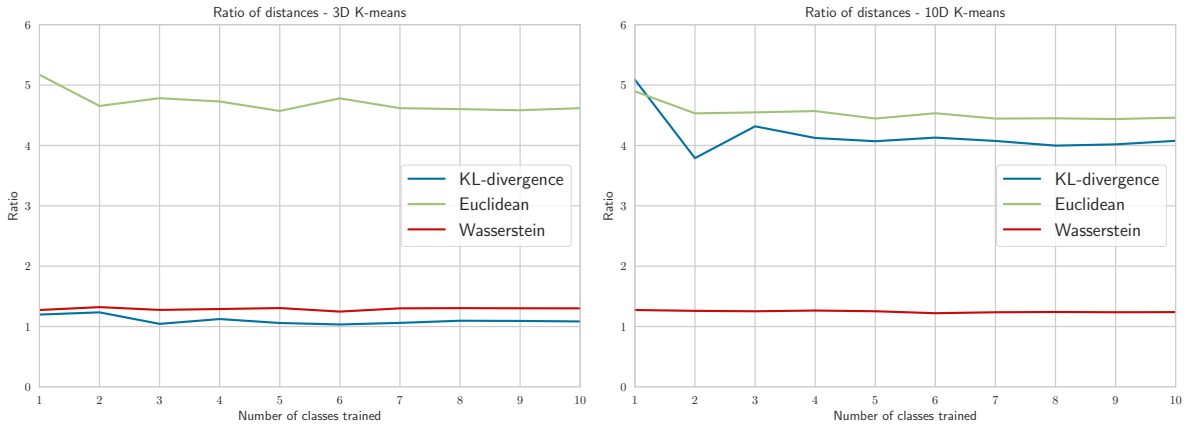


Figure 5.13: The ratio between irrelevant classes and relevant classes in feature space, per model. Left: 3D, right: 10D.

The first and foremost observation we make is that the ratio for the Euclidean distance at a ratio between 4 – 5, is considerably distinct from the others in 3D, i.e. irrelevant classes $Q_{\text{irrelevant}}$ are 4 – 5 \times as far from the training clusters $\mathbb{P}_1, \dots, \mathbb{P}_k$ than relevant classes Q_{relevant} when using the Euclidean distance measure. This holds for both 3D and 10D data, and explains why the performance of the Euclidean metric is so high and consistent in Fig. 5.12; it is relatively easier to distinguish relevant from irrelevant if irrelevant classes lie much farther than relevant classes.

The second notable pattern we see is that the ratios for all distance measures do not (significantly) diverge from their starting point at iteration 1 and remain almost constant over the iterations. Meaning that the ratio $\frac{d(Q_{\text{irrelevant}}, \mathbb{P})}{d(Q_{\text{relevant}}, \mathbb{P})}$ remains roughly constant even when we add more training clusters \mathbb{P} to calculate the average distance from. The main insight we obtained from this is that the training clusters of road vehicles (our prior information) are located close to each other, meaning that adding more classes to our prior information did not affect the average distance from one test cluster Q_j to all training clusters \mathbb{P}_i .

At last, we notice that the boost in performance of the KL-divergence in 10D can be explained by the right plot above which shows that the ratios for the KL-divergence have gone up considerably. We would have expected Wasserstein's ratio in 3D to be high as well since it performs on par with the Euclidean distance, meaning that there might be more to the performance disparity than the ratios alone.

Based on the results and corresponding intermediate evaluation so far, we believe that there can be other factors that can influence the performance. We hypothesise that the clustering algorithm we used, K-means, can affect the KL-divergence and Wasserstein distance since K-means is responsible for generating the points per cluster (or per distribution). This leads to a new research question on what would happen if we removed K-means and used the ground truth clusters instead. We treat this in the next experiment.

5.6. Experiment 2: Ablation Study for Removing K-means

SRQ6: Does performance improve if we remove K-means, and use ground truth clusters instead?

While K-means clustering has no effect on point-based models, it plays a critical role in distribution-based methods. This is because the clustering algorithm directly decides which points will be put into which cluster/distribution. For instance, K-means groups together points of three different classes that can lead to a new distribution with significantly different statistical properties than the three individual distributions. Since the Euclidean distance is a centroid-based distance (it calculates straight distances between the means of each cluster) it does not take cluster shape into account and therefore is not affected by K-means as much as the Wasserstein distance and KL-divergence (which look at the distribution as a whole since they are non-parametric techniques).

To see if K-means affects the distance measures we set up an ablation study where, instead of clustering, we sample distributions directly from the ground truth classes. This means we manually create clusters of each class without using K-means. This leads to 345 pure clusters/distributions which we will refer to as 'ground truth' clusters. In Fig. 5.14 we present the ground truth equivalent of Fig. 5.11, compared to PCA again.

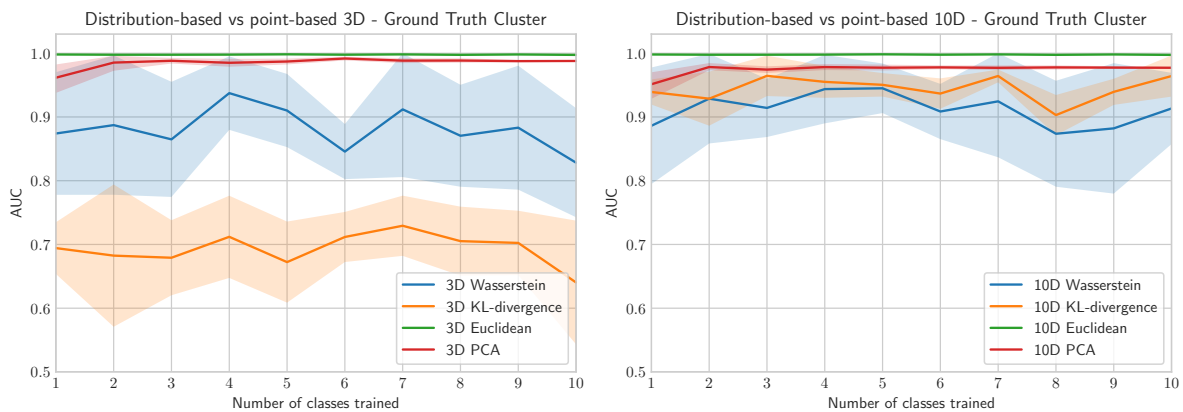


Figure 5.14: Distribution-based vs point-based, ground truth clusters. Left: 3D, right: 10D.

We shall compare the K-means version of each model side-by-side to its ground truth counterpart on the next page. Before doing so we also visualise the development of the ratios, previously shown in Fig. 5.13, for the ground truth clusters in Fig. 5.15; where we observe the same patterns as in the K-means case.

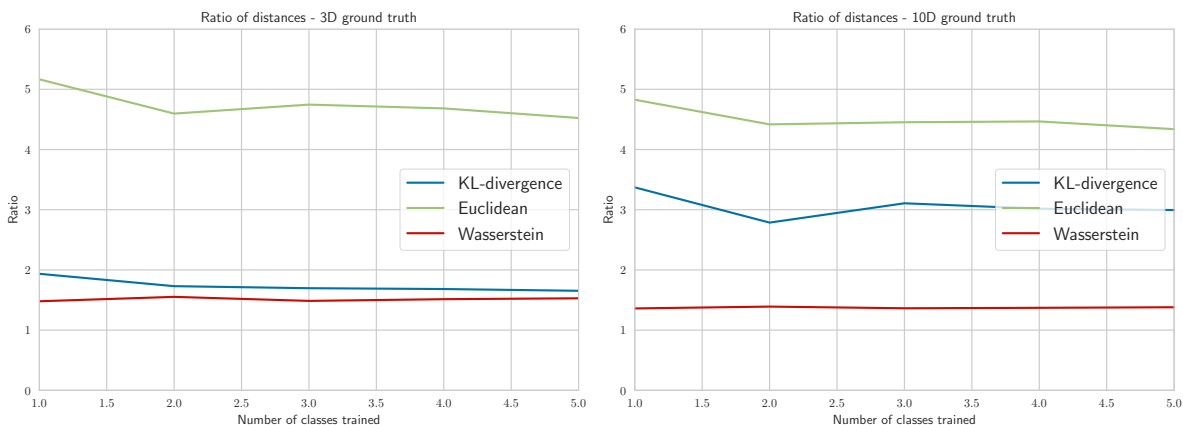


Figure 5.15: The ratio between irrelevant classes and relevant classes in feature space, per model. Left: 3D, right: 10D.

To assess the impact of switching from K-means to ground truth clusters, we now compare the three distance measures in Fig. 5.14 to their K-means counterpart displayed in Fig. 5.11. The plots on the left are in 3D,

and the plots on the right are in 10D, as seen in the images below. To ease visual analysis, all blue curves correspond to K-means while all orange curves to ground truth clusters.

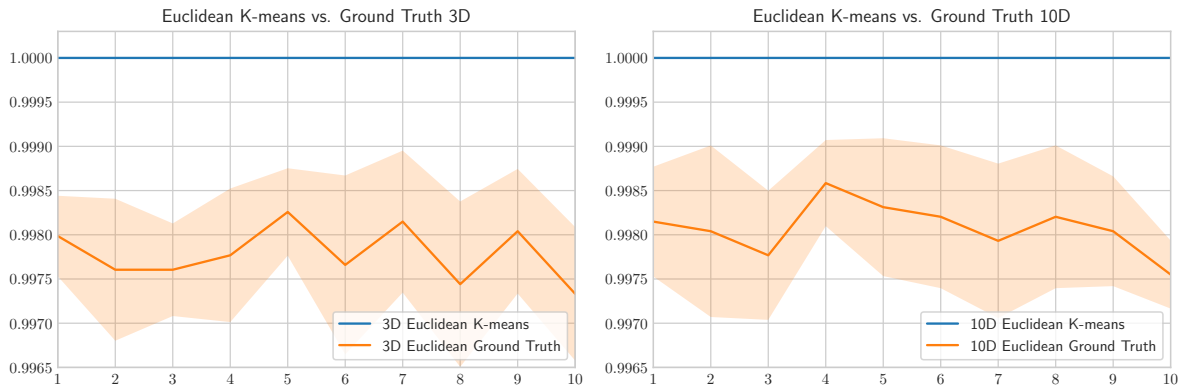


Figure 5.16: Ablation study with Euclidean. Left: 3D, right: 10D. The blue curves are equal, at a perfect score of 100%.

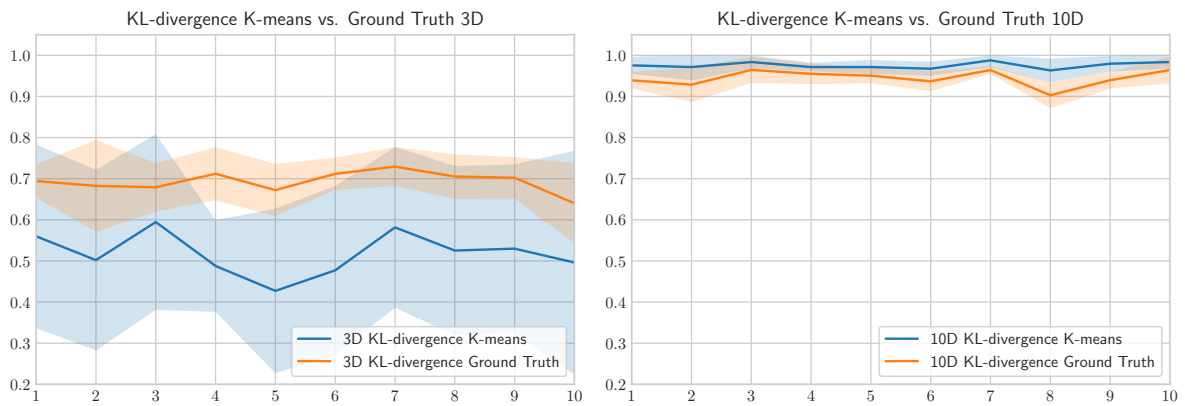


Figure 5.17: Ablation study with KL-divergence. Left: 3D, right: 10D.

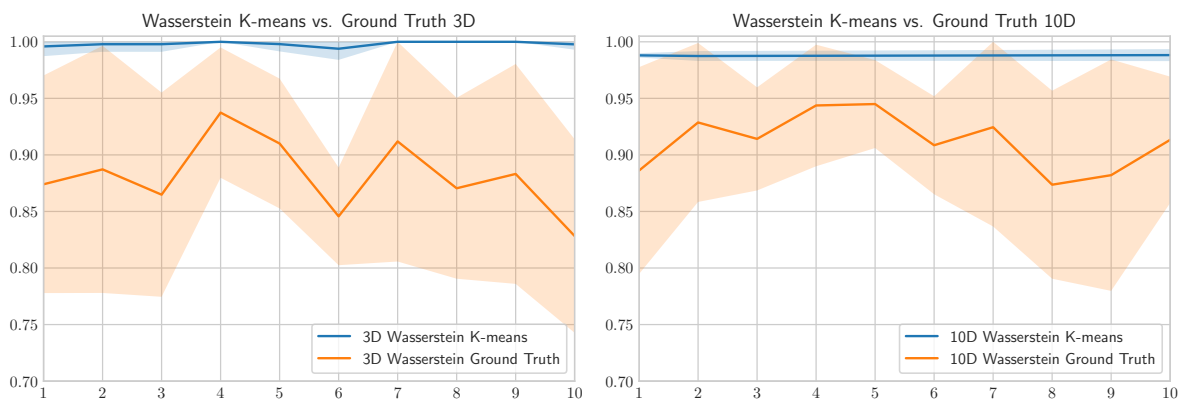


Figure 5.18: Ablation study with Wasserstein. Left: 3D, right: 10D.

It is striking that for almost all distance measures, in both 3D and 10D, relevance detection performs better on K-means clusters (mixed clusters with $K=34$) than on ground truth clusters (pure clusters). The performance discrepancy is not only reflected in terms of sheer AUC but also in terms of standard deviation since the blue curves exhibit significantly less variance than the orange curves. The only case for

which this does not hold is for KL-divergence, where both curves do not display great performance. We shall investigate the anomalous case of KL-divergence in [Section 5.8](#). For now, we would like to understand why relevance detection underperforms on ground truth clusters.

We expected performance to be better using ground truth clusters since these are pure clusters per class. Whenever we are using ground truth clusters, each of the 345 classes gets its own pure cluster. The result is 345 unique clusters in feature space, which is comparable to running K-means for $K = 345$, except for the fact that K-means can mix classes into clusters and ground truths are pure. This leads to the second additional research question on what would happen if we were to increase K in K-means all the way to 345; what value of K for our data-set optimises relevance detection? We hypothesise that if K is too great, it will result in over-fragmented clusters, which could lead to misleading cluster distances. We test this in the next section.

Finally, we provide the best-performing dimensions for both the K-means and ground truth cases. In contrast to point-based models in [Table 5.2](#) where 3D data was the better choice for relevance detection, for distribution-based models, we have evidence to suggest that 10D data is the best option.

Table 5.3: Best performing dimension per model, where "same" indicates that 3D and 10D have similar performance. This table is based on the results of the previous sections where both the values of the AUC and standard deviation are taken into account.

	K-means	Ground truth
Euclidean	same	same
Wasserstein	3D	10D
KL-divergence	10D	10D

5.7. Experiment 3: Varying K in K-means

SRQ7: What value of K for our data-set optimises relevance detection?

Recall that UMAP does not per se produce spherical clusters making centroid-based clustering algorithms a sub-optimal candidate for clustering. A density-based clustering method like DBSCAN would from a technical perspective be a wiser choice as it can capture non-linear patterns in the clusters. In spite of this, we still selected K-means (a centroid-based algorithm). The reason for doing so is that we wanted to see what happens in the case where clustering is not optimal. Do our methods fail in such a scenario? Are advanced clustering algorithms essential for our distribution-based models? Furthermore, K-means is straightforward to implement with great interoperability. We decided to move to more advanced models only when K-means would cause too much of a bottleneck.

5.7.1. Determining the optimal K for clustering

For K-means clustering, it is necessary to provide the number of clusters (K) beforehand. In our case, we knew prior to experiments that we would encounter 345 classes at test time, while in realistic applications one does not have access to the information on the number of classes. How does one determine the optimal K for clustering in that case? And why did we use $K = 34$ in our experiments when we knew that the total amount of classes is 345? The answer lies in using various heuristics.

In machine learning and statistics, the Elbow method is a common and powerful heuristic to determine the optimal K. Is it an iterative method which runs the K-means algorithm for a range of values for K, e.g. 5 to 40, and determines which K is optimal based on some loss. The elbow method is based on finding a value in the range of K, from where diminishing returns are no longer worth the additional cost. This value lies on the loss curve's inflection point or 'elbow'. A frequently used and suitable loss function is the distortion score, which is the mean sum of squared distances from a point x_i to its corresponding cluster centre, μ_X where X is a cluster.

$$\text{distortion} = \sum_i (\text{data point } x_i - \text{centroid } \mu_X)^2 + (\text{data point } y_i - \text{centroid } \mu_Y)^2 \dots \quad (5.4)$$

The idea behind this is as follows, if a point x_i actually belongs to a cluster X , it will be relatively close to μ_X . Therefore, correctly clustered points lead to smaller distances $(x_i - \mu_X)^2$, thus smaller distortion scores

and vice versa. Fig. 5.19 depicts the distortion scores for varying K from 2 to 100, leading to an elbow at $K = 13$. According to this method, increasing K will lead to diminishing improvements.

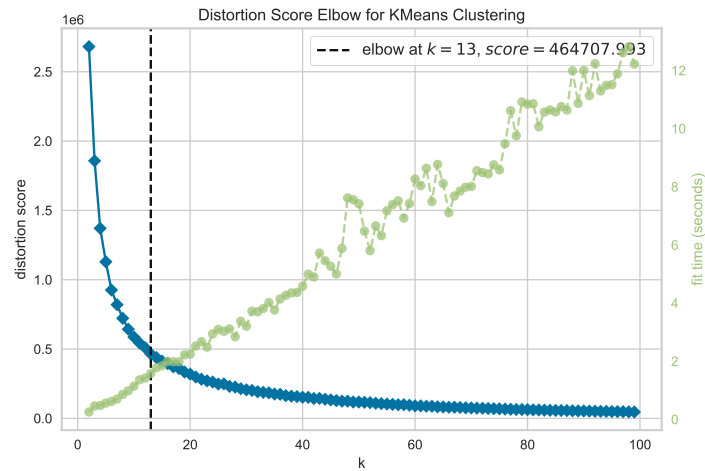


Figure 5.19: Distortion scores with elbow at $K=13$.

Fig. 5.19 gives the impression that when ignoring computational costs, values above $K = 13$ will still be better since the drop in distortion is not insignificant. Does this mean that when one has the resources, a higher K will always be better? We verify the elbow score of 13 for our own our model's performance by running it for values for K that are both lower and higher than the elbow score in the next test.

5.7.2. Determining the optimal K for relevance detection

The following test is conducted to understand why (most of) our distribution-based models performed better on K-means clustered data at $K = 34$, than on ground truth clustered data. We hypothesised a number of clusters (K) that is too great, will result in over-fragmented clusters which could lead to misleading cluster distances. To test this we ran Wasserstein, trained on only 1 class, on data clusters where K is varying from $K = 10$ to $K = 345$, with increments of 5. We expect that the AUC of the model will indeed start to deteriorate from a certain K onward. For reference, we have provided the AUC values for the K-means and ground truth performances at iteration 1 (from Fig. 5.18) as dashed lines.

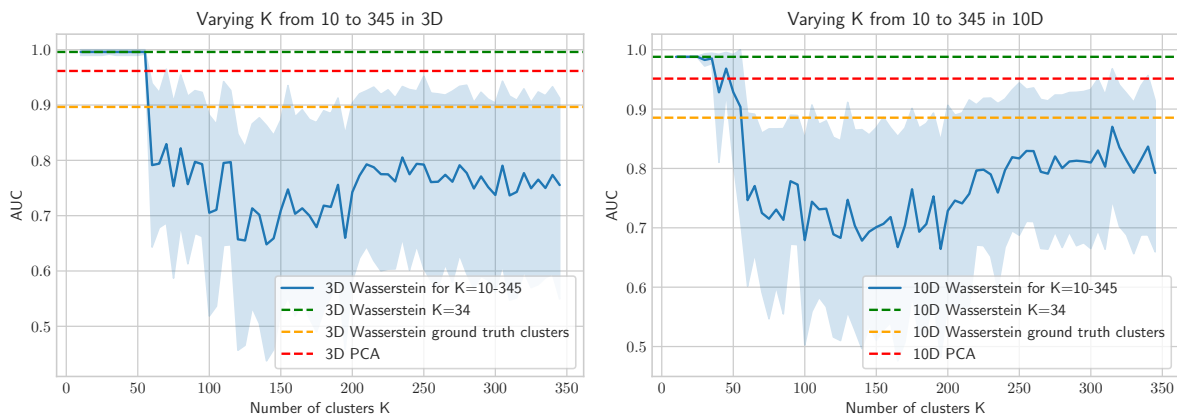


Figure 5.20: Running a distribution-based model, the Wasserstein distance 'trained' on 1 class in this case, for very high values of K ranging from 10 to 345. Left: 3D, right: 10D.

As expected, the AUC score has dropped significantly at $K = 345$, in both 3D and 10D. We make two striking observations from the graph; the first being that for low values of K ($K \in [10, 50]$ in 3D and $K \in [10, 35]$ in 10D) the model not only performs at its best with low variance and high AUC. The second observation is

Table 5.6: Accuracy per road vehicle class - KL-divergence 3D K-means

	ambulance	bulldozer	bus	car	firetruck	pickup_truck	police_car	school_bus	tractor	train	truck
1	0.8571	0.8571	0.8571	0.8571	0.8571	0.8571	0.8571	0.8571	0.8571	0.8571	0.8571
2	0.6000	0.6000	0.6000	0.6000	0.6000	0.6000	0.6000	0.6000	0.6000	0.6000	0.6000
3	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000
4	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000
5	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000
6	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000
7	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000	0.7000
8	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000
9	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000
10	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000

Table 5.7: Accuracy per road vehicle class - Wasserstein 3D K-means

	ambulance	bulldozer	bus	car	firetruck	pickup_truck	police_car	school_bus	tractor	train	truck
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
4	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
6	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
10	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

From the tables above it is clear that the models at least detect the classes of interest well. Table 5.6 proves why such a class-specific is useful; in 3D the KL-divergence was by far the lowest-performing model with an AUC oscillating around 0.7. From the KL-divergence table above, we learn that in spite of low(er) AUC, our relevant classes are still correctly detected with high accuracy. When we average each column in Table 5.6 per road vehicle over 10 iterations, we obtain an accuracy of 85.78%.

Moving on, we would like to find out how well the other 345-11=334 classes compare to the vehicle classes in terms of accuracy. Since it is impractical to create a table like the ones presented above for 345 classes, we shall make use of heat maps. First, we transpose the accuracy table so that the classes are now on the vertical axis, allowing for the long list of 345 classes. Next, rather than displaying a value in each cell, we display colours. Each cell is coloured where a black cell corresponds to 0.00% accuracy and a beige-coloured cell to 100%. Fig. 5.21 depicts a miniature example of what will follow; classes on the vertical axis and increasing amounts of prior knowledge on the horizontal axis.

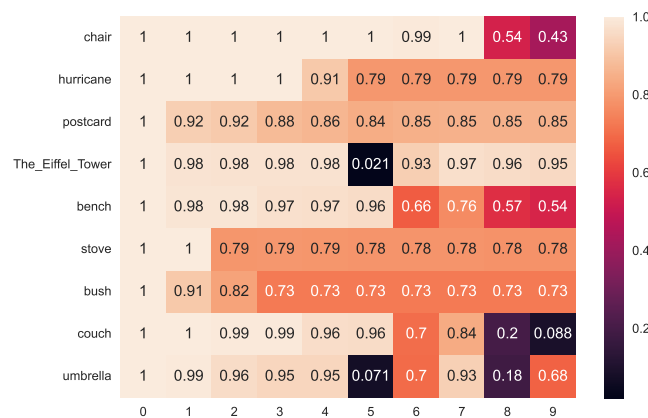


Figure 5.21: An example of what the heat maps will look like. For the 345-long heat map, we shall omit numerical values.

Heat maps shall allow us to visually detect where the weak points of each model lie, as well as how each model performs; the lighter the map, the better the performance. Fig. 5.22 depicts the four heat maps of

the four models discussed earlier on 3D data again, now for all 345 classes. We have sorted the rows of each heat map from highest to lowest average accuracy in order to observe visual patterns.

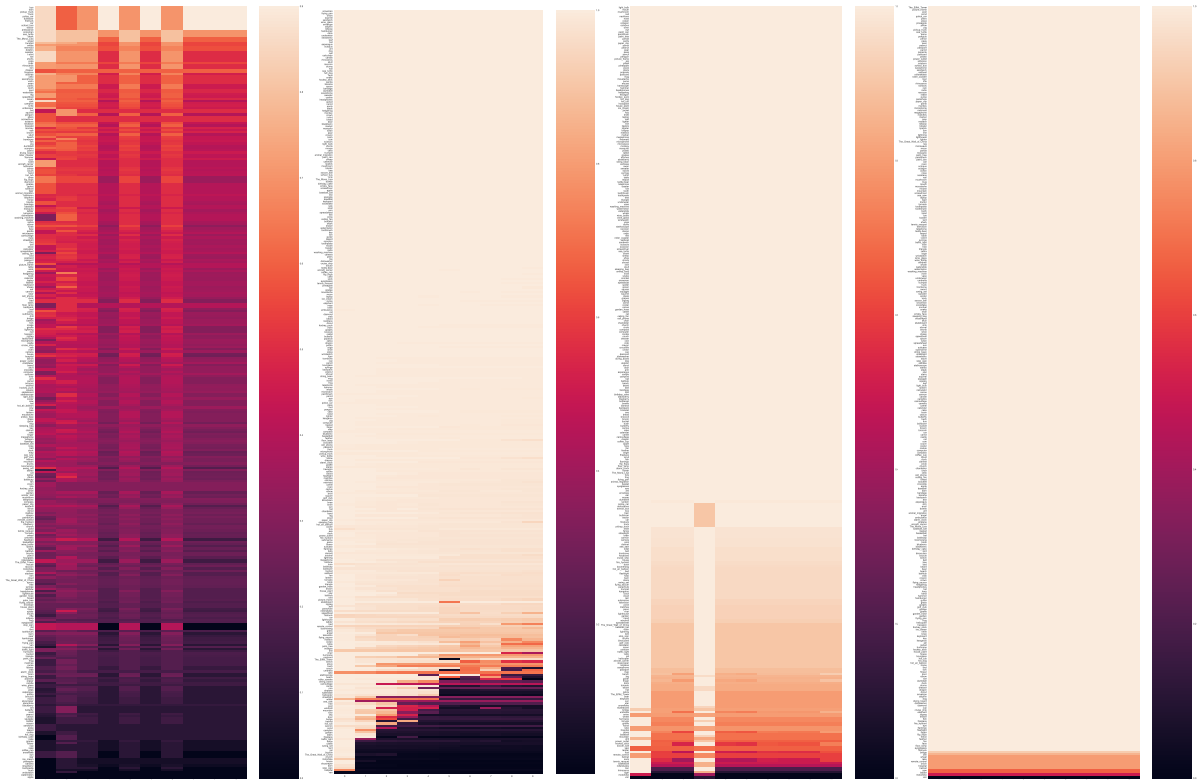


Figure 5.22: From left to right on 3D data: KL-divergence, PCA, Wasserstein, and Euclidean.

It is immediately clear from the heat maps that the Euclidean distance has proven itself to be the best performer, yet again. The KL-divergence had already shown less than optimal AUC scores in Fig. 5.17 for 3D, which is reflected in the heat map. We suspect that the drop in performance can be explained by the possibility that data features have drastically different 'probability' distributions for 3D and 10D data, the differences in ratios shown in Fig. 5.13 supports this. We would like to emphasise the term 'probability' here, since out of all models we used, the KL-divergence is the only statistical distance that models directly models point clouds as probabilities (by converting discrete data into probability mass functions). More research on higher moments beyond the mean and variance, such as kurtosis should be carried out to provide clarity on this. The other models perform well for the majority of classes, so we zoom in on their top 10 worst classes.

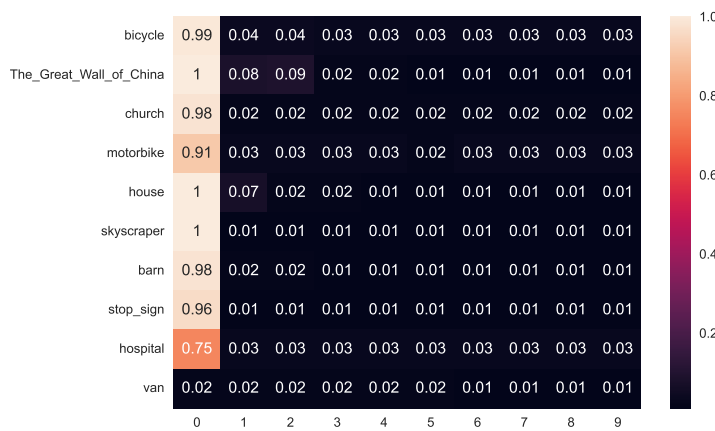


Figure 5.23: Top 10 worst detected classes for PCA.

Not only does PCA perform poorly for some classes, we see that the performance drops abruptly after we train on even one additional class. This means that for those classes, PCA cannot generalise well. The main implication that follows from this, is that the hyperplane used by PCA, cannot successfully separate relevant from irrelevant when more training data is added. We suspect that the irrelevant classes shown in Fig. 5.23, contain elements of road vehicles making them harder to classify as irrelevant. We now check for the other models whether these classes are also harder to label as irrelevant.

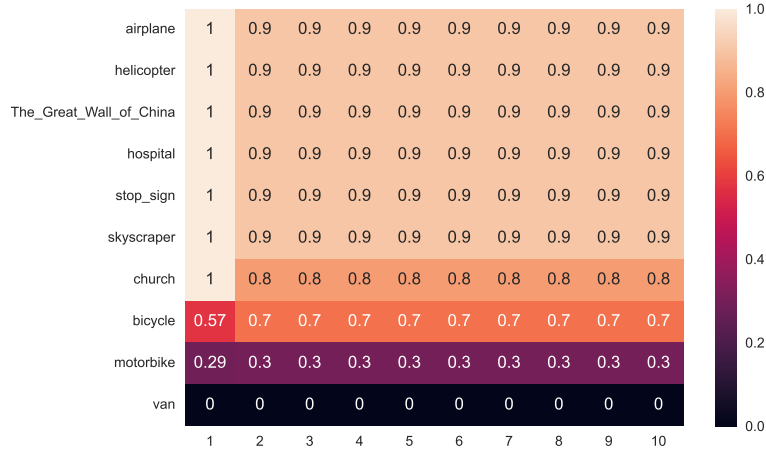


Figure 5.24: Top 10 worst detected classes for Euclidean.

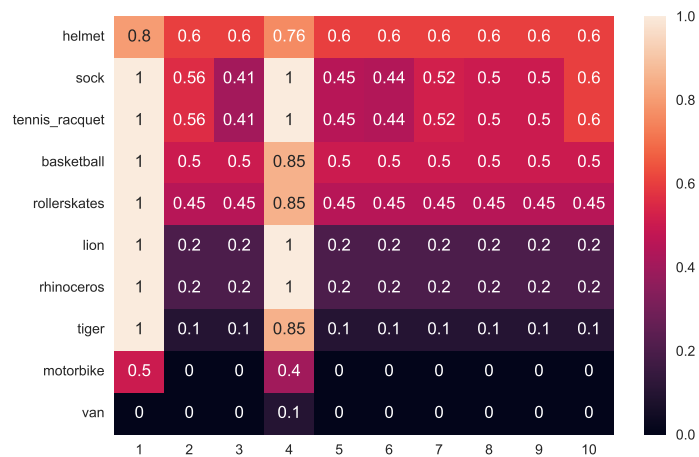


Figure 5.25: Top 10 worst detected classes for Wasserstein.

The one class that is at the bottom of all heat maps, with a 0% accuracy, is the class 'van'. Van is obviously a road vehicle and therefore relevant. In spite of that, we are pleased to see this, as this is good news. Recall from Fig. 5.2 that we manually searched and organised the classes of interest in the DomainNet data-set. During this process, we completely missed the 'van' class and did not include it in our list of 11 road vehicle classes. This means that we hard-coded the ground truth value for van as 'irrelevant'. We are glad that our model, even though we made a mistake on our end, has still classified (almost) all vans as 'relevant'. The 0% accuracy in the heat maps resulted from our hard-coded label that essentially told the model it was wrong every time it classified a van as relevant.

For all three distribution-based models, we see some overlap classes that seem to be hard to detect. We hypothesise that this performance drop is caused by elements of relevant classes, that are present in images of irrelevant classes. E.g. an image of a stop sign (irrelevant class) with cars in the background (element of a relevant class). To test this, we take a few of these classes and images of those classes that have proven to be especially hard to correctly classify. We have selected the classes bicycle, motorbike, and church. In

the figures below, we present the top 4 images of these irrelevant classes that were misclassified the most through all runs:



Figure 5.26: Motorbike images that were 'incorrectly' classified as relevant.

The second image in Fig. 5.26 does not contain any other vehicles but is most likely to be misclassified as 'relevant' since it is a police vehicle. Our training set of relevant classes contains police cars and ambulances that share the bright colours and patterns shown on the motorbike.

What is most interesting is that we observe that three out of four motorbike images contain elements of the road vehicle classes as well; the first, third, and fourth all contain cars or trucks, and all of these three images are related to accidents. The main insight gained from this is that we can use our model to find accidents in large data sets when setting vehicles of emergency services as relevant, and testing on irrelevant vehicle classes. This is a very valuable finding since accidents can be very relevant to a safety and security use case like ours.

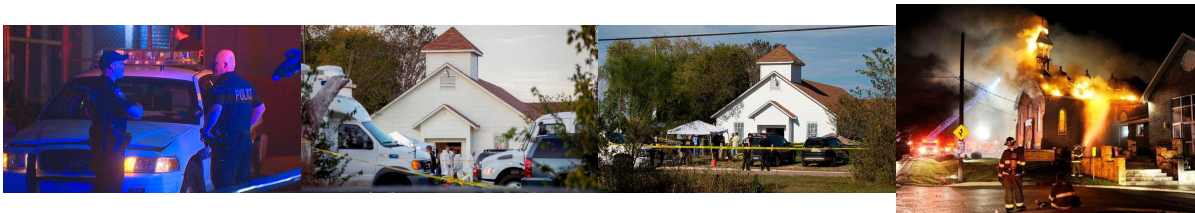


Figure 5.27: Church images that were 'incorrectly' classified as relevant.

We are pleased to discover the same phenomenon that we found for motorbikes, for the church class Fig. 5.27. The images of the church class are mostly clean and do not contain any unrelated elements, which is why we were surprised at first to see the church class at the bottom of the heat map. After loading the top 4 misclassified images, however, it is clear what went wrong. All images were related to accidents that happened at churches, containing therefore vehicles of emergency services. The first image of the church class is not even a photograph of a church, but a police car that is located in front of a church. This is again a very intriguing finding since we have shown that even the most irrelevant of classes (churches, which are less relevant than motorbikes), that are by themselves completely unrelated to vehicles and thus our safety and security use case, can contain incredibly *relevant* events.



Figure 5.28: Bicycle images that were 'incorrectly' classified as relevant.

For the bike class, the reasons for misclassification are similar to the previous cases. The first image of Fig. 5.28 is of particular interest; as it is not a regular bike, but a rather large and peculiar vehicle that bears

close resemblance to Batman’s Batcycle. We are glad that our model flagged this image as relevant even though it belongs to an irrelevant class. The insight gained from this is that vehicle classes that are not relevant, can become relevant when strange variations of adaptations of it arise (like a Batcycle, or strange military vehicle that looks like a bike, but actually is not).

The main conclusion we draw from this error analysis is that whenever our model classifies images of irrelevant classes as ‘relevant’, we should be extra cautious and check those instances as they can contain *relevant events / relevant classes*. Through this test we have shown that our model can even find relevance, within irrelevance, making it one of the most valuable insights gained throughout this thesis, for practical use cases.

5.8.2. Why the Euclidean distance performs so well

The Euclidean distance has consistently outperformed advanced and non-parametric SOTA methods like the Wasserstein distance and the KL-divergence. We have shown in Fig. 5.13 that its out-performance can be explained by the discrepancy in ratios, where irrelevant classes lie significantly farther than relevant classes. We extend this through a visual analysis by plotting a few training clusters of relevant vehicle classes (blue), testing clusters of relevant vehicle classes (green), and multiple irrelevant classes in Fig. 5.29.

Please focus on the red ellipse drawn around the blue training classes; it is clear that the green relevant vehicle classes at test time lie considerable closer to the blue training classes than the other irrelevant classes.

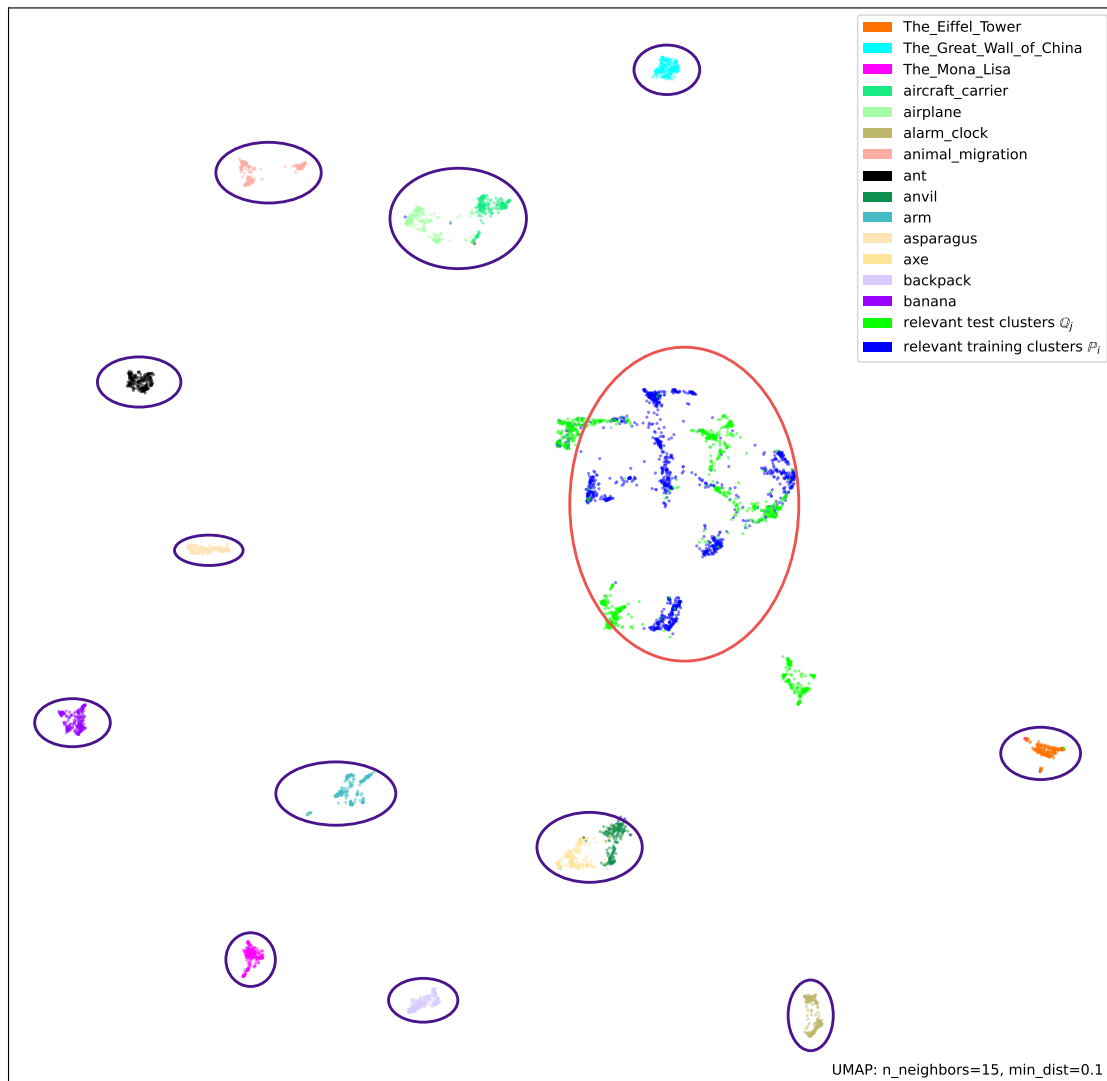


Figure 5.29: Purple ellipses: irrelevant test classes. Red ellipse: relevant training classes.

We used default values for the UMAP parameters (previously discussed in [Section 4.2.2](#)) as these preserve a good balance between local and global structure. [Fig. 5.29](#) shows data that is reduced to 2D since that is easier to interpret for this analysis.

From [Fig. 5.29](#) it is clear that irrelevant classes lie so much further apart from relevant classes in feature space, that computing straight distances between them should suffice. While this explains why Euclidean performs well, it does not entirely explain why it outperforms non-parametric techniques. The non-parametric distance measures (Wasserstein, and KL-divergence) also take statistical information like cluster shape, relative weights of the tails of a distribution (kurtosis), and many other statistical properties into account, apart from the mean and variance. From the plot in feature space, we see that it is more difficult to determine similarity based on even cluster shape alone since visually similar classes do not necessarily have the same cluster shape; both the Great Wall of China and the Mona Lisa have a similar shape but are not similar classes. Vehicle classes do appear to have a very irregular shape, unlike any other classes, which is one reason they still perform well. All in all, we conclude that a parametric straight distance, based on cluster centres is enough for relevance detection in UMAP feature space.

5.8.3. Aligning results with hypotheses

In the previous sections, we have already answered all the research questions and treated our results with respect to our hypotheses. Therefore, this sub-section serves as a brief overview and recap of how the results align with the hypotheses stated at the start of this chapter.

H1: Relevance detection can be done in a more robust way when using a cluster-based approach.

Through extensive testing we have validated our first hypothesis and found that indeed a cluster-based approach is more robust in terms of AUC and accuracy per class. Using statistical distances between clusters or 'distributions' better takes atypical points (strange buses and/or dogs) into account, and are also more adept at separating classes in features space than for instance the hyperplane used by PCA. Our results of relevance detection therefore fully align with this hypothesis.

H2: Statistical properties, such as cluster shape, play an important role in relevance detection.

The most unexpected result of this thesis was that apparently, one is better off using simple Euclidean distances than advanced non-parametric ones. We are aware of the fact that UMAP plays a part in this result, but that is also precisely why we expected cluster shape to matter; UMAP usually produces very irregular and non-spherical and we had therefore anticipated that by including non-parametric methods in our arsenal. Therefore, this hypothesis did not align with what we found in our research.

H3: Relevant OOD classes are better discovered when we have more prior domain information.

We have found that distributions-based methods could generalise very well to the unseen relevant classes after having only been trained on one class. Increasing the amount of prior information did not have a significant effect on both the AUC and accuracy scores. For point-based models, however, it was worthwhile to train on more classes. Even though point-based models generalised well to unseen relevant classes too, they could benefit from training on at least two to three classes, while distribution-based methods only needed one. This is yet another reason why cluster-based techniques are favourable, especially when prior information is not widely available. This result was unexpected and therefore also not in line with our hypothesis.

H4: Our pipeline will perform better on low-dimensional data.

The dimensionality of the data did not have any drastic effects on most models. Only the KL-divergence benefited greatly from higher dimensions while the rest of the models either increased or decreased to a much lesser extent. It is hard to say whether this hypothesis also did not align with the results since it depends on what one defines as high-dimensional. In our case, 10D is already high-dimensional and we have discovered that the tested models work well on 10D data, making the hypothesis invalid. But we do expect the performances of especially the Euclidean distance to decrease in very high dimensions like 100-2000D (which are common output dimensions of neural network architectures before dimensionality reduction) which would render this hypothesis valid.

6

Conclusions & Recommendations

In the open world, machine learning models can encounter a multitude of unknown or novel classes. In a surveillance or safety use case described in [Chapter 1](#), where potential threats of unknown vehicles have to be detected, it is crucial that the model can distinguish which unknown classes it encounters are relevant to our use case (unknown enemy vehicles) and which ones are not (harmless unknown birds). This motivated us to research this topic and develop a novel mathematical approach to answer the following (main) research question in this thesis on computer vision.

(How) Can relevant unknown classes be distinguished from irrelevant unknown classes?

Through this thesis, for the first time to the best of our knowledge, a method is developed that can assess the relevance of unknown classes, by modelling their image features as clusters (or distributions) and comparing them using statistical distance measures. Given images of classes that are defined as 'relevant' by a user, we have shown that this novel method can accurately determine the relevance of unknown classes at test time for both low and high-dimensional data, and is therefore the answer to our main research question.

6.1. Conclusions

Given the mathematical nature of our methodology, we wanted to research how the statistical properties of classes can be leveraged to distinguish the relevant from the irrelevant ones. We hypothesised that cluster shape would play a major role and that non-parametric statistical distances should be able to capture this best. Throughout our experiments in [Chapter 5](#), we have deployed both parametric and non-parametric approaches to compare statistical distributions of classes. To our surprise, the Euclidean distance, which is a parametric distance, outperformed our non-parametric candidates (Wasserstein distance, KL-divergence, and the point-based ML models) by a great deal in all tests. Through various tests that followed, we have shown that classes are far enough apart in feature space to such an extent, that cluster shape indeed does not have to matter for relevance detection. This has proven to be one of the most unexpected and major insights of this thesis since one is better off using the elementary Euclidean distance than a state-of-the-art non-parametric technique.

Through increasing the training set of prior knowledge we found that even after only training on a few classes, most models performed well at around 97-99% AUC. With Euclidean scoring a perfect 100% when training on 1 class only, and point-based models like PCA benefitting from training on two to three classes. This implies that one does not need much prior information to perform accurate relevance detection, making our method very suitable for cases where data on a specific group of interesting/relevant classes is scarce.

During the ablation study of our clustering algorithm of choice, K-means, we found that relevance detection works less well on ground truth clusters, which seems counter-intuitive. After investigating this further we discovered that for relevance detection, the user is better off by not choosing the number of clusters (K) equal to the number of classes but much lower. In our case, a K between 10 and 50 led to the best performance.

After running all tests on the general performance of the models, we conducted an error analysis for which we computed the accuracy per class. Through this analysis, we could understand which classes are hard to detect and why. The main conclusion we have drawn from this error analysis is that even when irrelevant samples are incorrectly classified as relevant, there is something relevant about them. For instance, we found that the Euclidean distance was often not entirely wrong whenever it was classifying images of churches as relevant; images of churches that were found relevant by the model contained road vehicles. As a matter of fact, all those images were related to accidents that happened at churches, containing therefore vehicles of emergency services. We found the same to be true for relevant images of motorcycles; most were involved in road accidents and there contained police cars and ambulances as well. This is a valuable finding as it has revealed that our model could possibly be used to find accidents in large data sets when setting vehicles of emergency services as relevant, and testing on irrelevant vehicle classes. This is a very valuable finding since accidents can be very relevant to a safety and security use case like ours.

6.2. Recommendations for Future Work

1. A separate, more extensive study on dimensionality

All tests were conducted using both 3D and 10D data. Although it was hard to draw very general conclusions since this is model-dependent, throughout our testing we found that point-based methods perform slightly better on 3D data, while distribution-based methods benefit from 10D data. Despite the fact that both 3D and 10D have proved themselves to be suitable dimensions for relevance detection, more research on the effect of dimensionality should be done. We recommend running a separate test where the dimensionality is varied from 2D to 512D (CLIP output) for various models, in a similar fashion we have varied K from 10 to 345. Such a study can hopefully lead to more general insights into what dimensionality practitioners should use for relevance detection.

2. Perform relevance detection on 'extremely' hard cases

We performed detection on high-resolution data of classes that were already visually close (thus challenging): we trained on road vehicles, and tested on aircrafts/ships which is already much harder than what we have found in literature. One could try relevance detection on classes that are even closer than that:

- Training on military road vehicles and testing on military aircrafts.

3. Choose relevant training classes that are not similar.

In our cases our training classes (road vehicles) were close to each other in feature space (see Fig. 5.29), meaning that every iteration an extra training class was added, the overall average distance $\frac{1}{k} \sum_{i=1}^k d(Q_j, P_i)$ did not change much. This will be different when our relevant training classes are not close to each other. As an example take the following case:

(a) Test on classes of the animal kingdom

One could select all animals with two feet as the relevant class, and all other animals (four feet) as irrelevant. What makes this problem much harder is that even within the group of relevant classes there is a considerable amount of variance; not all two-footed are alike and therefore probably far in feature space (e.g. birds and chimpanzees, and kangaroos).

4. Adaptations to determining relevance

We used average distances from one test cluster to all training clusters $\frac{1}{k} \sum_{i=1}^k d(Q_j, P_i)$ which has worked well so far. But for recommendation 3, where one uses training classes that are located far apart, it would be highly useful to also consider alternatives to the average. Let the relevance of a class now also be determined its nearest neighbouring cluster, or perhaps even the nearest neighbouring point of another cluster. Average distances to training clusters that are far apart will lead to misleading results.

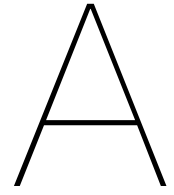
References

- [1] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: [1412.6572](https://arxiv.org/abs/1412.6572) [stat.ML].
- [2] Dubravko Miljković. “Review of novelty detection methods”. In: May 2010, pp. 593–598. ISBN: 978-1-4244-7763-0.
- [3] Saikiran Bulusu et al. “Out-of-Distribution Detection in Deep Learning: A Survey”. In: (Mar. 2020).
- [4] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. “Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images”. In: *CoRR* abs/1412.1897 (2014). arXiv: [1412.1897](https://arxiv.org/abs/1412.1897). URL: <http://arxiv.org/abs/1412.1897>.
- [5] Yann LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE*. Vol. 86. 11. 1998, pp. 2278–2324. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665>.
- [6] Adwin Jahn. “Keras Image Preprocessing: scaling image pixels for training”. In: (2017). URL: <https://www.linkedin.com/pulse/keras-image-preprocessing-scaling-pixels-training-adwin-jahn/>.
- [7] Brandon. “Course 137 Signal Processing Techniques - How to Convert an RGB Image to Grayscale”. In: (2019). URL: https://e2eml.school/convert_rgb_to_grayscale.html.
- [8] Google. “A.I. Experiments: Visualizing High-Dimensional Space”. In: (2016). URL: <https://experiments.withgoogle.com/collection/ai>.
- [9] 3Blue1Brown. “But what is a neural network? | Chapter 1, Deep learning”. In: (2017). URL: https://www.youtube.com/watch?v=aircAruvnKk&t=227s&ab_channel=3Blue1Brown.
- [10] Ross B. Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Computing Research Repository* abs/1311.2524 (2013). arXiv: [1311.2524](https://arxiv.org/abs/1311.2524). URL: <http://arxiv.org/abs/1311.2524>.
- [11] Ross B. Girshick. “Fast R-CNN”. In: *Computing Research Repository* abs/1504.08083 (2015). arXiv: [1504.08083](https://arxiv.org/abs/1504.08083). URL: <http://arxiv.org/abs/1504.08083>.
- [12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [13] Joris Bierkens. *Statistical Learning Lecture Slides Week 2.3*. <https://wi4630.zulipchat.com/>. TU Delft, 2021.
- [14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8). URL: <https://www.sciencedirect.com/science/article/pii/0893608089900208>.
- [15] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *ICML*. 2010.
- [16] Chigozie Nwankpa et al. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. In: *CoRR* abs/1811.03378 (2018). arXiv: [1811.03378](https://arxiv.org/abs/1811.03378). URL: <http://arxiv.org/abs/1811.03378>.
- [17] Sweta Shaw. “Activation Functions Compared with Experiments”. In: (2022). URL: <https://wandb.ai/shweta/ActivationFunctions/reports/Activation-Functions-Compared-with-Experiments--VmlldzoxMDQwOTQ>.
- [18] Ludwig Boltzmann. *Studien über das Gleichgewicht der lebendigen Kraft zwischen bewegten materiellen Punkten*. K.k. Hof- und Staatsdruckerei, 1868.
- [19] R. Duncan. Luce. *Individual choice behavior; a theoretical analysis*. Wiley, 1959.

- [20] Dario Radečić. “Softmax Activation Function Explained”. In: (2020). URL: <https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60>.
- [21] RInterested. “NOTES ON STATISTICS, PROBABILITY and MATHEMATICS”. In: (). URL: <https://rinterested.github.io/statistics/softmax.html>.
- [22] Katarzyna Janocha and Wojciech Marian Czarnecki. “On Loss Functions for Deep Neural Networks in Classification”. In: *CoRR* abs/1702.05659 (2017). arXiv: [1702.05659](https://arxiv.org/abs/1702.05659). URL: <http://arxiv.org/abs/1702.05659>.
- [23] Elliott Gordon-Rodriguez et al. *Uses and Abuses of the Cross-Entropy Loss: Case Studies in Modern Deep Learning*. 2020. DOI: [10.48550/ARXIV.2011.05231](https://doi.org/10.48550/ARXIV.2011.05231). URL: <https://arxiv.org/abs/2011.05231>.
- [24] Zhilu Zhang and Mert R. Sabuncu. “Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels”. In: *CoRR* abs/1805.07836 (2018). arXiv: [1805.07836](https://arxiv.org/abs/1805.07836). URL: <http://arxiv.org/abs/1805.07836>.
- [25] R. E. Wengert. “A simple automatic derivative evaluation program”. In: *Commun. ACM* 7 (1964), pp. 463–464.
- [26] epommweb. “Create 3d loss surface visualizations with optimizer path”. In: (2021). URL: <https://www.epommweb.org/post/create-3d-loss-surface-visualizations-with-optimizer-path/>.
- [27] Educative Answers Team. “Overfitting and underfitting”. In: (). URL: <https://www.educative.io/answers/overfitting-and-underfitting>.
- [28] Preetum Nakkiran et al. “Deep Double Descent: Where Bigger Models and More Data Hurt”. In: *CoRR* abs/1912.02292 (2019). arXiv: [1912.02292](https://arxiv.org/abs/1912.02292). URL: <http://arxiv.org/abs/1912.02292>.
- [29] Kishan G. Mehrotra, Chilukuri K. Mohan, and HuaMing Huang. *Anomaly Detection Principles and Algorithms*. 1st. Springer Publishing Company, Incorporated, 2017. ISBN: 3319675249.
- [30] Lukas Ruff et al. “A Unifying Review of Deep and Shallow Anomaly Detection”. In: *Proceedings of the IEEE* 109.5 (2021), pp. 756–795. DOI: [10.1109/JPROC.2021.3052449](https://doi.org/10.1109/JPROC.2021.3052449).
- [31] Mohammadreza Salehi et al. “A Unified Survey on Anomaly, Novelty, Open-Set, and Out-of-Distribution Detection: Solutions and Future Challenges”. In: *CoRR* abs/2110.14051 (2021). arXiv: [2110.14051](https://arxiv.org/abs/2110.14051). URL: <https://arxiv.org/abs/2110.14051>.
- [32] Jing kang Yang et al. “Generalized Out-of-Distribution Detection: A Survey”. In: *CoRR* abs/2110.11334 (2021). arXiv: [2110.11334](https://arxiv.org/abs/2110.11334). URL: <https://arxiv.org/abs/2110.11334>.
- [33] Chuanxing Geng, Sheng-Jun Huang, and Songcan Chen. “Recent Advances in Open Set Recognition: A Survey”. In: *CoRR* abs/1811.08581 (2018). arXiv: [1811.08581](https://arxiv.org/abs/1811.08581). URL: <http://arxiv.org/abs/1811.08581>.
- [34] Xiaoran Chen and Ender Konukoglu. “Unsupervised Detection of Lesions in Brain MRI using constrained adversarial auto-encoders”. In: *CoRR* abs/1806.04972 (2018). arXiv: [1806.04972](https://arxiv.org/abs/1806.04972). URL: <http://arxiv.org/abs/1806.04972>.
- [35] David Zimmerer et al. *A Case for the Score: Identifying Image Anomalies using Variational Autoencoder Gradients*. 2019. arXiv: [1912.00003](https://arxiv.org/abs/1912.00003) [eess.IV].
- [36] Thomas Schlegl et al. “f-AnoGAN: Fast Unsupervised Anomaly Detection with Generative Adversarial Networks”. In: *Medical Image Analysis* 54 (Jan. 2019). DOI: [10.1016/j.media.2019.01.010](https://doi.org/10.1016/j.media.2019.01.010).
- [37] Thomas Schlegl et al. “Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery”. In: *CoRR* abs/1703.05921 (2017). arXiv: [1703.05921](https://arxiv.org/abs/1703.05921). URL: <http://arxiv.org/abs/1703.05921>.
- [38] Markos Markou and Sameer Singh. “Novelty detection: a review—part 1: statistical approaches”. In: *Signal Processing* 83.12 (2003), pp. 2481–2497. ISSN: 0165-1684. DOI: <https://doi.org/10.1016/j.sigpro.2003.07.018>. URL: <https://www.sciencedirect.com/science/article/pii/S0165168403002020>.
- [39] Markos Markou and Sameer Singh. “Novelty detection: a review—part 2: neural network based approaches”. In: *Signal Processing* 83.12 (2003), pp. 2499–2521. ISSN: 0165-1684. DOI: <https://doi.org/10.1016/j.sigpro.2003.07.019>. URL: <https://www.sciencedirect.com/science/article/pii/S0165168403002032>.
- [40] Walter J. Scheirer et al. “Toward Open Set Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.7 (2013), pp. 1757–1772. DOI: [10.1109/TPAMI.2012.256](https://doi.org/10.1109/TPAMI.2012.256).

- [41] Dan Hendrycks and Kevin Gimpel. “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks”. In: *CoRR* abs/1610.02136 (2016). arXiv: [1610.02136](https://arxiv.org/abs/1610.02136). URL: <http://arxiv.org/abs/1610.02136>.
- [42] Peyman Morteza and Yixuan Li. “Provable Guarantees for Understanding Out-of-distribution Detection”. In: *CoRR* abs/2112.00787 (2021). arXiv: [2112.00787](https://arxiv.org/abs/2112.00787). URL: <https://arxiv.org/abs/2112.00787>.
- [43] Kimin Lee et al. *A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks*. 2018. DOI: [10.48550/ARXIV.1807.03888](https://doi.org/10.48550/ARXIV.1807.03888). URL: <https://arxiv.org/abs/1807.03888>.
- [44] Weitang Liu et al. “Energy-based Out-of-distribution Detection”. In: *CoRR* abs/2010.03759 (2020). arXiv: [2010.03759](https://arxiv.org/abs/2010.03759). URL: <https://arxiv.org/abs/2010.03759>.
- [45] Shiyu Liang, Yixuan Li, and R. Srikant. “Principled Detection of Out-of-Distribution Examples in Neural Networks”. In: *CoRR* abs/1706.02690 (2017). arXiv: [1706.02690](https://arxiv.org/abs/1706.02690). URL: <http://arxiv.org/abs/1706.02690>.
- [46] Chuan Guo et al. “On Calibration of Modern Neural Networks”. In: *CoRR* abs/1706.04599 (2017). arXiv: [1706.04599](https://arxiv.org/abs/1706.04599). URL: <http://arxiv.org/abs/1706.04599>.
- [47] Yann LeCun et al. “A tutorial on energy-based learning”. In: *PREDICTING STRUCTURED DATA*. MIT Press, 2006.
- [48] Rui Huang and Yixuan Li. “MOS: Towards Scaling Out-of-distribution Detection for Large Semantic Space”. In: *CoRR* abs/2105.01879 (2021). arXiv: [2105.01879](https://arxiv.org/abs/2105.01879). URL: <https://arxiv.org/abs/2105.01879>.
- [49] Henryk Maciejewski, Tomasz Walkowiak, and Kamil Szyk. “Out-of-Distribution Detection in High-Dimensional Data Using Mahalanobis Distance - Critical Analysis”. In: *Computational Science – ICCS 2022*. Ed. by Derek Groen et al. Cham: Springer International Publishing, 2022, pp. 262–275. ISBN: 978-3-031-08751-6.
- [50] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.
- [51] Dan Hendrycks, Mantas Mazeika, and Thomas G. Dietterich. “Deep Anomaly Detection with Outlier Exposure”. In: *CoRR* abs/1812.04606 (2018). arXiv: [1812.04606](https://arxiv.org/abs/1812.04606). URL: <http://arxiv.org/abs/1812.04606>.
- [52] Vivek Roy et al. “Few-Shot Learning with Intra-Class Knowledge Transfer”. In: *CoRR* abs/2008.09892 (2020). arXiv: [2008.09892](https://arxiv.org/abs/2008.09892). URL: <https://arxiv.org/abs/2008.09892>.
- [53] Xingchao Peng et al. “Moment matching for multi-source domain adaptation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1406–1415.
- [54] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *CoRR* abs/2103.00020 (2021). arXiv: [2103.00020](https://arxiv.org/abs/2103.00020). URL: <https://arxiv.org/abs/2103.00020>.
- [55] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: [1512.03385](https://arxiv.org/abs/1512.03385). URL: <http://arxiv.org/abs/1512.03385>.
- [56] Andrzej Maćkiewicz and Waldemar Ratajczak. “Principal components analysis (PCA)”. In: *Computers & Geosciences* 19.3 (1993), pp. 303–342. ISSN: 0098-3004. DOI: [https://doi.org/10.1016/0098-3004\(93\)90090-R](https://doi.org/10.1016/0098-3004(93)90090-R). URL: <https://www.sciencedirect.com/science/article/pii/009830049390090R>.
- [57] Alaa Tharwat et al. “Linear discriminant analysis: A detailed tutorial”. In: *Ai Communications* 30 (May 2017), pp. 169–190. DOI: [10.3233/AIC-170729](https://doi.org/10.3233/AIC-170729).
- [58] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: <http://jmlr.org/papers/v9/vandermaaten08a.html>.
- [59] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2018. DOI: [10.48550/ARXIV.1802.03426](https://doi.org/10.48550/ARXIV.1802.03426). URL: <https://arxiv.org/abs/1802.03426>.
- [60] Jiazhi Xia et al. “Revisiting Dimensionality Reduction Techniques for Visual Cluster Analysis: An Empirical Study”. In: *CoRR* abs/2110.02894 (2021). arXiv: [2110.02894](https://arxiv.org/abs/2110.02894). URL: <https://arxiv.org/abs/2110.02894>.

- [61] Tim Sainburg, Leland McInnes, and Timothy Q. Gentner. “Parametric UMAP: learning embeddings with deep neural networks for representation and semi-supervised learning”. In: *CoRR* abs/2009.12981 (2020). arXiv: [2009.12981](https://arxiv.org/abs/2009.12981). URL: <https://arxiv.org/abs/2009.12981>.
- [62] Parul Pandey. “Visualising-kannada-mnist-with-umap”. In: (2019). URL: <https://www.kaggle.com/code/parulpandey/part3-visualising-kannada-mnist-with-umap/notebook>.
- [63] Pratyush Pranav et al. “The Topology of the Cosmic Web in Terms of Persistent Betti Numbers”. In: *Monthly Notices of the Royal Astronomical Society* 465 (Aug. 2016). DOI: [10.1093/mnras/stw2862](https://doi.org/10.1093/mnras/stw2862).
- [64] UMAP-learn. “How UMAP Works”. In: (). URL: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html.
- [65] Wei Dong, Moses Charikar, and Kai Li. “Efficient K-nearest neighbor graph construction for generic similarity measures”. In: Jan. 2011, pp. 577–586. DOI: [10.1145/1963405.1963487](https://doi.org/10.1145/1963405.1963487).
- [66] UMAP-learn. “Using UMAP for Clustering”. In: (). URL: <https://umap-learn.readthedocs.io/en/latest/clustering.html>.
- [67] P.J.C. Spreij. *Measure Theoretic Probability*. UvA. URL: <https://staff.fnwi.uva.nl/p.j.c.spreij/onderwijs/master/mtp.pdf>.
- [68] Gabriel Peyré and Marco Cuturi. “Computational Optimal Transport”. In: (2018). DOI: [10.48550/ARXIV.1803.00567](https://doi.org/10.48550/ARXIV.1803.00567). URL: <https://arxiv.org/abs/1803.00567>.
- [69] Alfred Galichon. *Optimal Transport Methods in Economics*. Princeton University Press, 2016. URL: <http://www.jstor.org/stable/j.ctt1q1xs9h> (visited on 09/25/2022).
- [70] G. Monge. *Mémoire sur la théorie des déblais et des remblais*. Imprimerie royale, 1781. URL: <https://books.google.nl/books?id=IG7CGwAACAAJ>.
- [71] Liuba. “Introduction to Optimal Transport”. In: (2020). URL: <https://medium.com/analytics-vidhya/introduction-to-optimal-transport-fd1816d51086>.
- [72] Leonid Kantorovich. *On the Translocation of Masses*. 1942. URL: <https://doi.org/10.1287/mnsc.5.1.1>.
- [73] Ofir Pele and Michael Werman. “Fast and robust Earth Mover’s Distances”. In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 460–467. DOI: [10.1109/ICCV.2009.5459199](https://doi.org/10.1109/ICCV.2009.5459199).
- [74] Marco Cuturi. “Sinkhorn Distances: Lightspeed Computation of Optimal Transportation Distances”. In: (2013). DOI: [10.48550/ARXIV.1306.0895](https://doi.org/10.48550/ARXIV.1306.0895). URL: <https://arxiv.org/abs/1306.0895>.
- [75] Yury Polyanskiy. *f-divergences*. MIT. URL: https://people.lids.mit.edu/yp/homepage/data/LN_fdiv.pdf.
- [76] *PyOD package*. URL: <https://pyod.readthedocs.io/en/latest/>.
- [77] Cédric Villani. “Optimal transport – Old and new”. In: vol. 338. Jan. 2008, pp. xxii+973. DOI: [10.1007/978-3-540-71050-9](https://doi.org/10.1007/978-3-540-71050-9).
- [78] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. “On the Surprising Behavior of Distance Metric in High-Dimensional Space”. In: *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)* (Feb. 2002).
- [79] OpenAI. “CLIP: Connecting Text and Images”. In: (2021). URL: <https://openai.com/blog/clip/>.
- [80] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: [10.48550/ARXIV.1706.03762](https://doi.org/10.48550/ARXIV.1706.03762). URL: <https://arxiv.org/abs/1706.03762>.
- [81] Tianyang Lin et al. “A Survey of Transformers”. In: *CoRR* abs/2106.04554 (2021). arXiv: [2106.04554](https://arxiv.org/abs/2106.04554). URL: <https://arxiv.org/abs/2106.04554>.



Background on CLIP

A.1. CLIP Neural Network Architecture

CLIP (Contrastive Language-Image Pre-Training) is a neural network released by OpenAI on January 5th 2021, that is trained a vast amount of data; 400,000,000 digital images and their corresponding captions to be precise. What sets CLIP apart is that it is a multi-modal network for vision and text; unlike conventional computer vision models, CLIP is **not** limited to assigning *fixed labels* to each image as it has also been explicitly been trained on text data. This means that CLIP is able to "predict the most relevant custom text snippet, given an image." (Radford et al., 2021), rather than a predefined or fixed label. As a result, one can input any image into CLIP, and it will return the likeliest label, caption, or summary of that image. This is a considerable feat as it makes CLIP a suitable candidate for zero-shot learning (predicting labels of classes it has never seen before).

While CLIP has become a powerful bridge between computer vision and natural language processing, and an extraordinary zero-shot classifier, it was not necessarily intended for our use case. But as it has gone through extensive pre-training, we will be using it as a feature generator and therefore very briefly discuss its architecture below.

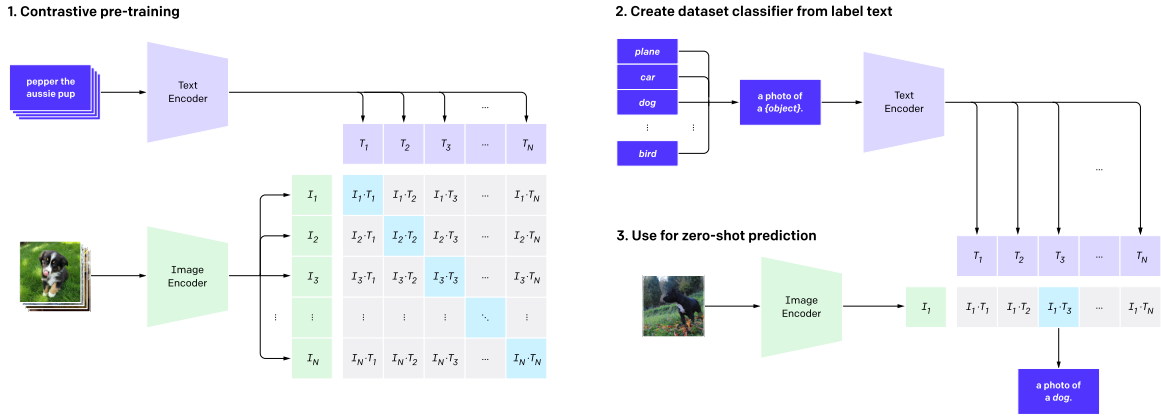


Figure A.1: Visual overview of the operational dynamics of CLIP [79]. Violet: text processing, mint green: image processing. Part 1 shows the training (left), while part 2 & 3 depicts the testing operations (right).

CLIP is comprised of two neural networks¹, denoted as f_T and f_I :

1. **Text encoder** f_T : transforms text data to a vector in a mathematical feature space.
2. **Image encoder** f_I : transforms image data to a vector in a mathematical feature space.

CLIP is pre-trained as follows:

Let (x_I, x_T) denote an (image, text) pair, with $I_i = f_I(x_{I_i})$ and $T_i = f_T(x_{T_i})$ the corresponding vector representations (called embeddings) obtained by networks of the i -th data pair. CLIP then learns how close a text embedding is to an image embedding by computing the cosine similarity with angle θ between the vectors.

It uses the following losses to train: the text-to-image loss $\mathcal{L}_i^{(I \rightarrow T)}$ and the image-to-text loss $\mathcal{L}_i^{(T \rightarrow I)}$, which are cross entropy losses of the cosine similarity:

$$\mathcal{L}_i^{(I \rightarrow T)} = -\log \frac{\exp(S_\theta(I_i, T_i))}{\sum_{k=1}^N \exp(S_\theta(I_i, T_k))}, \quad \mathcal{L}_i^{(T \rightarrow I)} = -\log \frac{\exp(S_\theta(T_i, I_i))}{\sum_{k=1}^N \exp(S_\theta(T_i, I_k))},$$

where $S_\theta(\cdot, \cdot)$ denotes the cosine similarity, and N the sample size. Through a variant of gradient descent, CLIP's final objective is to minimise the average loss (see [54] for details):

$$\frac{1}{N} \sum_{i=1}^N (\mathcal{L}_i^{(I \rightarrow T)} + \mathcal{L}_i^{(T \rightarrow I)}) / 2$$

At test time it will predict a fitting caption for a new image, based on the highest cosine similarity of the new image vector I_i and multiple text candidates: $(I_i, T_1), (I_i, T_2), \dots, (I_i, T_N)$.

¹The encoders used by CLIP to achieve the highest performance are *transformer* neural networks, which is beyond the scope of this thesis. It is not necessary to understand their architecture, but we refer curious readers to the following: [80, 81].