UNIVERSITY
OF OSLO

Master's thesis

Explainable Artificial Intelligence for Out-of-Distribution Detection

Using irregularities in machine learning explanations to detect when a model is faced with unusual data

Jonatan Hoffmann Hanssen

Robotics and Intelligent Systems
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences

Spring 2025

Jonatan Hoffmann Hanssen

Explainable Artificial Intelligence for Out-of-Distribution Detection

Using irregularities in machine learning explanations to detect when a model is faced with unusual data

Supervisors:
Hugo Lewi Hammer
Kyrre Harald Glette

Abstract

As Artificial Intelligence becomes a larger and larger part of society, the need for robust and understandable models becomes paramount. When neural networks are used in high-impact settings such as cancer detection or autonomous driving, we must require that they

Sammendrag

Here comes the abstract in a different language.

Contents

List of Figures

List of Tables

OOD Out-of-Distribution

DL Deep Learning

ID In-Distribution

CAM Class Activation Mapping

GradCAM Gradient Class Activation Mapping

GAP Global Average Pooling

MSP Maximum Softmax Probability

LIME Local Interpretable Model-Agnostic Explanations

CNN Convolutional Neural Network

FFNN Feed Forward Neural Network

ML Machine Learning

LRP Layer Relevance Propagation

XAI Explainable Artificial Intelligence

AI Artificial Intelligence

AUROC Area Under Receiver Operating Characteristic

AUPR Area Under Precision Recall Curve

ROC Receiver Operating Characteristic

FPR False Positive Rate

TPR True Positive Rate

FPR95 False Positive Rate at 95% Recall

Preface

I would like to thank my dog and Ed Bickley, who is still at large.

Chapter 1

Introduction

1.1 Motivation

Machine Learning (ML) generally, and Deep Learning (DL) specifically, have seen a tremendous increase in performance in recent years, performing comparable to humans in tasks such as image classification, speech and handwriting recognition, as well as many others [1]. Consequently, DL methods have been deployed in a multitude of fields and have become a part of our daily lives through their role in web search, text translation, computer vision, and in many other technologies which are taken for granted. In medicine, deep learning has the potential to provide faster and more accurate detection of diseases by being trained on cases from thousands of previous patients [2]. Despite this, "surprisingly little in health care is driven by machine learning" [3].

To explain this discrepancy, we should consider that despite their impressive performance, the application of deep learning methods is not without drawbacks. Firstly, deep neural networks are inherently unexplainable due to the large number of parameters that any non-trivial network has. State of the art models will perform millions of operations to evaluate a single data point, and it is therefore impossible for humans to comprehend and explain the entire process which lead the model to make a particular decision. In medicine, this is a major limitation of deep learning methods, as both doctors and patients expect to be able to understand why a decision was made [4].

Secondly, although neural networks may attain high accuracy on test data and appear to have learned great insights about the tasks they are employed in, they often lack robustness and can suffer large drops in performance on data points which are slightly different from the training data. As [5] has shown, it is possible to create data points which are imperceptibly different from normal data points, yet still fool otherwise high performing models. More problematically, unlike humans, who recognize when they are faced with a novel situation where their expertise might be lacking, DL methods will predict equally confidently on data points which are far outside the data they have been trained on [4].

These two problems lead to the fields of XAI, and OOD detection. XAI attempts to explain the reasons why a model came to a decision, which helps to remedy the black-box nature of complicated DL models. In a healthcare setting, such explanations can be inspected by medical practitioners to confirm the diagnosis, and can be used to give patients information about why decisions regarding their health were made. OOD detection attempts to uncover when a data point is too different from the training data to be classified reliably. These methods could alert medical practitioners when such data points occur, thus avoiding potentially fatal misclassifications.

Both of these fields have seen increased interest in recent years, and are vital parts of any integration of DL in medical settings. This thesis will focus on OOD detection, but will attempt to use methods inspired by XAI to improve detection performance. The overarching intuition is that by inspecting the explanation of a model on a specific data point, we may be able to uncover flaws or irregularities in the explanation which could help us determine whether the data point is OOD.

1.2 Problem Statement

As explained in the previous segment, OOD detection is a developing field, which has become more important in recent years as machine learning is being used for higher impact tasks, such as disease detection. Finding novel methods which improve a model's ability to detect when input is OOD is important to increase the robustness of machine learning models as they are used in these real-world scenarios. The field of XAI is concerned with understanding the inner workings of a model, and could thus offer insights which could help us detect unusual behaviour in the model as a result of OOD data points. The problem statement is thus as follows:

Can methods inspired by the field of Explainable Artificial Intelligence be used to improve Out-of-Distribution Detection?

To answer this question, I introduce 1 objective:

1. Develop theoretically sound proof of concept OOD detection methods that are inspired by insights gained from the field of XAI, and compare these methods on a variety of different benchmarks against existing OOD detection methods

1.3 Scope

As we will see in chapter 2, both the fields of XAI and OOD detection are very large, which make it impossible to explore all the possible ways one might combine XAI and OOD detection. Thus, it is necessary to restrict the scope of both the XAI and OOD detection methods used. In this section, I will describe the choices I've made and give a short explanation. In later chapters, the choices will be justified more thoroughly.

The field of OOD detection is primarily concerned with image classification tasks. Thus, my project will also deal exclusively with image classification datasets. Given this type of data, the choice of XAI algorithms naturally gravitates towards post-hoc, saliency based methods. The choice of OOD detection algorithms is not significantly restricted by the choice to deal exclusively with image data, but I will exclude methods which use outlier exposure to improve performance, or which require retraining of the model. These choices are informed by [6], which have found that post-hoc methods

1.4 Research Methods

Use ACM

1.5 Ethical Considerations

1.6 Main Contributions

Objective 1 is accomplished in chapter 2. Here, the fields of XAI and OOD detection are introduced, their respective taxonomies are explained and a selection of specific methods are explained in more detail. This chapter provides a good entry for machine learning researchers who wish a decent introduction to the two fields.

1.7 Thesis Outline

Chapter 2 gives a short introduction to machine learning, followed by a deeper look at the fields of XAI and OOD detection. Chapter **??** introduces my methodology; the methods I will use to compare and contrast XAI explanations on ID and OOD data, as well as a handful of OOD detection methods which integrate explanations into their functioning. Furthermore, I introduce the different datasets which will be used in chapter 4, among them the HyperKvasir gastrointestinal dataset [7]. 4 will first go into the results of the comparison between ID and OOD explanations, detailing the differences between different methods. Then, I will report the performance of the different OOD detection algorithms I have developed, on a handful of datasets, including HyperKvasir, CIFAR10 and ImageNet. After the experiments follow a discussion (chapter 5), where I reflect on the benefits and drawbacks of using XAI methods for OOD detection, and summarize the results from 4. Finally, the conclusion (chapter 6) concludes.

Chapter 2

Background

In this chapter I give a short introduction to important concepts in the field of machine learning generally, followed by a more in-depth look at the fields of OOD detection and XAI. Then, I give an overview of related works; papers which have attempted to use XAI for OOD detection. Finally, the datasets used in chapter 4 are covered.

2.1 Machine Learning

Machine Learning is the field of algorithms that are able to learn from data, as opposed to being explicitly programmed. Such algorithms use statistical methods to learn relationships in data, and use these relationships to generalize to unseen data. More formally, [8] gives the following definition of machine learning algorithms:

Definition. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Thus, machine learning is a different paradigm from traditional problem solving, where programs are made to solve problems by following explicit rules. For example, a traditional image classification system attempting to differentiate between malignant and benign tumors might use hand-crafted rules which consider the texture, color and size of a tumor. As one might imagine, such rules will quickly become very complicated when we consider all the possible factors which might influence the appearance of a tumor. Using a machine learning approach, we would instead feed an algorithm with thousands of images of both benign and malignant tumors, and the rules could then be automatically updated until the algorithm predicted the correct category with a high enough accuracy.

Machine Learning is commonly divided into the three subcategories of supervised, unsupervised and reinforcement learning

2.1.1 Supervised Learning

Supervised Learning is a subcategory of machine learning where we have a dataset containing both inputs and desired outputs. In the example above, we could use supervised learning by creating a dataset of images of tumors (the input) and corresponding labels which indicate whether each tumor is malignant or benign (the desired output). The learning goal of the algorithm is then to associate images with the correct label. Because we know the correct answer, we are able to fine-tune the

algorithm automatically whenever it makes a mistake. However, supervised learning requires labeled data, which can be very costly, especially in the medical domain, where deciding whether a tumor is malignant or benign requires expert knowledge.

2.1.2 Unsupervised Learning

In unsupervised learning, we do not have any labels. In these cases, we might not know whether data points belong to different classes or not. Instead, we can use machine learning to uncover patterns in the data, for example by attempting to cluster the data into different groups and seeing if these groups are sufficiently separated. An example use case could be for fraud detection in a bank. By feeding financial transaction from many different users into an unsupervised learning model and asking it to perform clustering of the data, it might be possible to find a group of users whose transactions differ substantially from the rest, which might indicate that their transactions are fraudulent.

2.1.3 Reinforcement Learning

Reinforcement Learning deals with problems where we do not know exactly what the correct solution is, but we are able to assess whether a given solution is good or not. For example, when controlling a robot arm, it is difficult to say exactly what angles each joint should be for every millisecond when picking up an object, but if the arm does not pick up the object, we know the algorithm has failed. In these problems, the algorithm is trained through reinforcement, where good attempts are rewarded and bad attempts punished.

2.2 Neural Networks

Neural Networks are a class of machine learning algorithms, which have become the clear state of the art in almost all fields where machine learning is applied. Notable examples are computer vision, image classification, speech recognition, text and image generation and machine translation. Neural networks are loosely inspired by our own brains, where neurons are connected together and send information between each other. By connecting thousands of neurons together, neural networks are able to learn complicated relationships between the input and output.

2.2.1 Feed Forward Neural Networks

The Feed Forward Neural Network (FFNN), also known as a Multilayer Perceptron, traces its roots to the very beginning of machine learning, through the work of Frank Rosenblatt [9]. It forms the basic structure for neural networks which has been adapted and modified over the years to form more complex architectures such as convolutional, recurrent or residual neural networks. The basic structure of an FFNN is that the input values are passed through an affine transformation (a matrix multiplication followed by the addition of a bias), and then passed through an activation function, which produces outputs. These outputs can then go through the same process again, which constitutes a single "layer". By stacking several of these layers, with non-linear activation functions, an FFNN is able to learn arbitrarily complex mappings between inputs and outputs[1].

[1] In fact, by the Universal Approximation Theorem [10], only a single hidden layer between the input and output is necessary, although this theorem does not give a way to construct such a network for any given function

Figure 2.1[2] shows a simple FFNN architecture with three hidden layers. In this case, the bias has been omitted for brevity. Here, we can see how all nodes of a layer are connected to the following layer. By using an activation function on the nodes of the hidden layer, before their values are sent to the next layer, we achieve the non-linearity required to learn complex patterns.

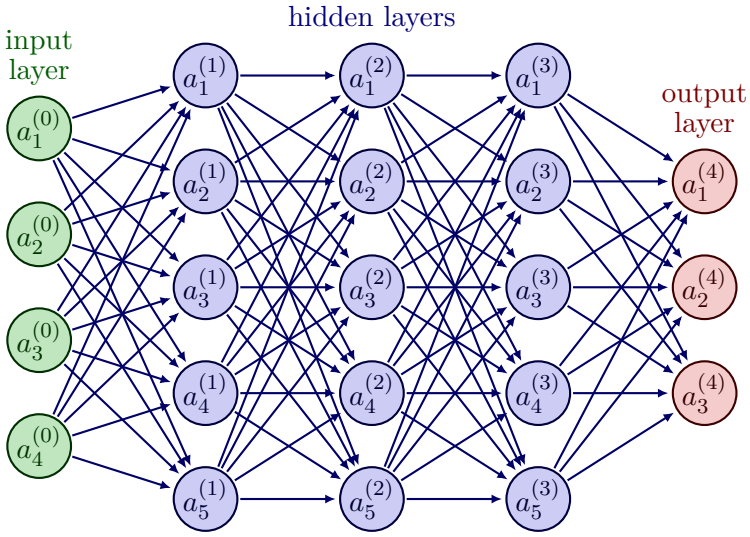

Figure 2.1: Figure showing a simple Feed Forward Neural Network, with nodes labeled. The number in parentheses indicates the layer number while the subscript indicates the node number within the layer.

Mathematically, a single layer can then be described as follows:

$$\boldsymbol{x_{i+1}} = \sigma_i(A_i\boldsymbol{x_i} + \boldsymbol{b_i}) \tag{2.1}$$

Here, the input $\boldsymbol{x_i}$ is linearly transformed by the weights of the matrix A_i from the input space to the output space, then each value of the new vector in the output space is adjusted by an addition of a bias term, and finally an activation function (σ_i) is applied to each value. The size of the input and output layers is determined by the number of input and output features, respectively. Furthermore, the activation function of the output layer is also determined by the application, most commonly *sigmoid* for binary classification, *softmax* for multi-class classification and simply identity for regression.

2.2.2 Convolutional Neural Networks

FFNNs have some inherent flaws which make them unsuitable for working with high dimensional, spatially connected data, such as the pixels which make up an image. Firstly, each input of a FFNN is connected to every output of the following layer. If we want to connect the input pixels of a 224 by 224 image to a layer of 100 nodes, our first layer will have over 5 million weights, which is already quite a lot for a relatively small image. Furthermore, these weights will have to encode redundant information, because each pixel is considered separately. Consider a network attempting to detect the presence of a cat in an image. We would want the network to detect the cat regardless of whether

[2]Figure by Izaak Neutelings, "Neural Network with coefficients, arrows", TikZ.net, licensed under CC BY-SA 4.0, [https://tikz.net/neural_networks/].

it is in the middle, the right corner, or any other position in the image. In an FFNN, the weights connected to any of these positions in the image would then have to encode a cat detector separately from all the others.

Convolutional Neural Networks (CNNs) solve both these issues by using small kernels of weights which are "slid" across the entire input. By using the same weights across all positions of the image, we do not need to train separate detectors for different positions, giving us translation invariance. Figure 2.2 shows the functioning of a convolutional kernel on a 3-channel image. Each value in the output is a weighted sum of a neighbourhood of values in the input image, where the weights are defined by the kernel. As we can see, the same weights are used on all positions, drastically reducing the number of parameters that need to be tuned. In a 2d-convolution, the kernel has the same number of channels as the input, and is only slid across the height and width dimension. The kernel in this figure is a *Sobel Operator*, and detects vertical edges. In a CNN, the weights of each kernel are not specified manually, but rather learned through backpropagation.

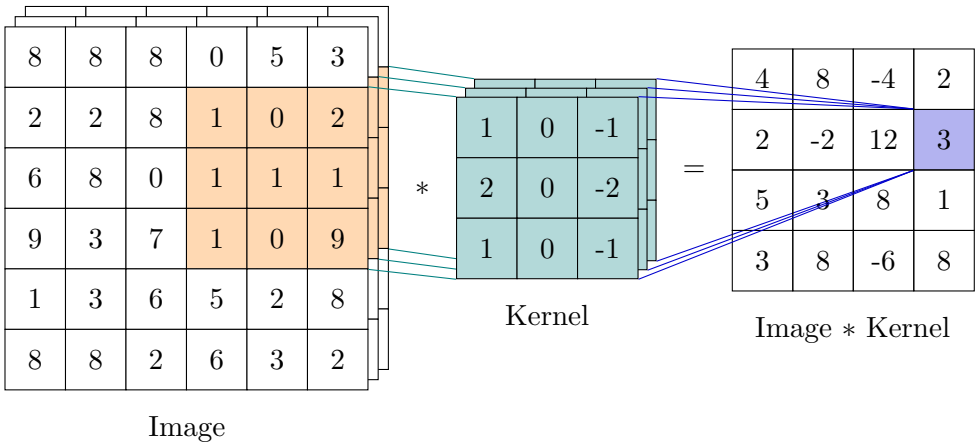

Figure 2.2: Figure showing a convolutional kernel applied to a 3-channel image.

By using several different kernels, we can detect many different patterns despite each kernel only detecting a single type. By using the outputs of all the kernels as inputs to a new set of kernels, we can use the same type layer structure as in an FFNN, allowing us to extract information in a hierarchical manner. It is common to see that trained CNNs have early layers that detect edges and texture, later layers that use these edge and pattern detections to detect larger shapes, while the final layers combine the shapes to detect entire objects [11].

By not evaluating every possible position in the input image, CNNs downsample the image, and are able to reduce the number of operations considerably. Simultaneously, this downsampling enables each subsequent layer to consider a larger area of the input image than the previous (a larger field-of-view), which allows larger patterns to be discovered. Simultaneously, it is common to use a larger and larger amounts of kernels on the new input, thus increasing the channel depth while the spatial dimensions are reduced. Between each layer, we use non-linear activation functions, similarly to how the are used in FFNNs.

After several such convolutions, we can flatten the output, either by aggregating each channel using Global Average Pooling (GAP) or a similar method, or we can simply

flatten all dimensions and consider the three dimensional feature map as a long vector of shape $C \times H \times W$. By doing this, we can pass the output to one or more linear layers, which can perform classification or regression on the extracted features and give us a final prediction. Figure 2.3 shows a high level overview of this process. Here, the input, which has only 3 channels, has its spatial dimensions reduced while its channel depth increases through consecutive convolutions. Finally, we have a certain number of channels in our final feature map, which are flattened (in this case with GAP) and processed through a linear layer to give a final prediction.

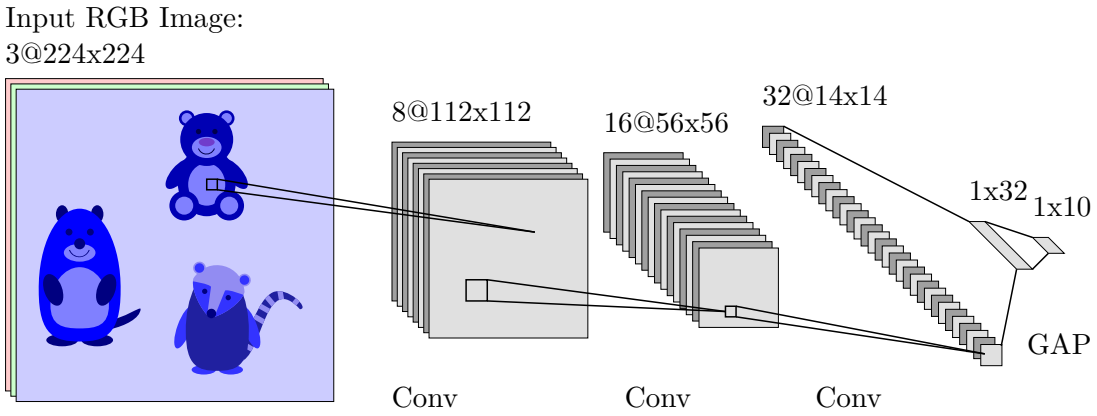

Figure 2.3: Figure showing a high level overview of how a CNN functions

2.3 Model evaluation

2.3.1 Metrics

OOD detection is essentially simply a binary classification problem. Thus, the metrics I will use in this thesis are those used for such problems. In the field of OOD detection, Area Under Receiver Operating Characteristic (AUROC) and False Positive Rate (FPR) are most commonly used. As such I shall focus on these metrics, as opposed to for example the Area Under Precision Recall Curve (AUPR).

Accuracy

Accuracy is the simplest metric used in binary classification. It is simply the ratio of correct predictions over all instances in the data set.

Accuracy has the advantage of being simple to understand and calculate, but it is very often insufficient. A simple (and quite common) scenario where accuracy fails to capture the performance of a model is any situation where there are large class imbalances. For example, imagine we have trained a CNN to predict whether a person has lung cancer or not, based on CT-scans of their lungs. As most people do not have lung cancer, we can imagine that such a dataset is highly imbalanced, for example that only 1 in 100 people actually have cancer. If a model simply predicts that no one ever has lung cancer, it will be correct in 99% of cases, and will thus have an accuracy of 99%, although it has missed every instance of cancer and is completely unusable in any real context.

For the purposes of OOD detection, accuracy is thus insufficient, as we have no guarantees that ID and OOD will be balanced. In fact, we expect OOD data to be relatively rare, given that the goal of developing an Artificial Intelligence (AI) model is

to ensure high performance during deployment, which necessitates having training data which covers as much as possible of the data seen during inference.

Metrics utilizing the binary classification confusion matrix

Instead of simply considering whether a prediction was correct or not, we should take into account the different combinations of prediction and ground truth, considering positive and negative classes separately. Figure 2.4 shows the possible four possible combinations given a ground truth class and a predicted class.

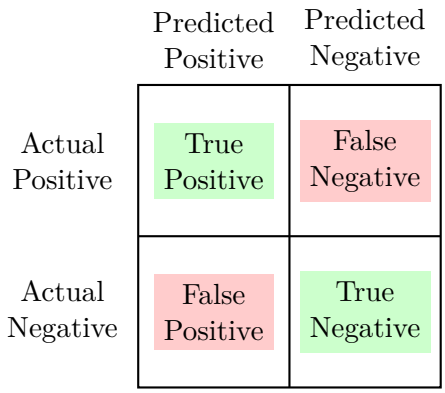

Figure 2.4: Figure showing the binary classification confusion matrix, denoting the four possible combinations created by the ground truth and predicted class. Green cells denote correct predictions, while red cells denote wrong predictions.

With these possibilities defined, we can begin to gain a clearer picture of the performance of a model.

Precision and Recall

Precision is the share of positive predictions that were actually positive. With a high false positive rate, we have a low precision, which means that the model is erroneously flagging many negative classes as positive. In an OOD detection setting, a model which flags many ID samples as OOD would have a low precision, if we treat OOD samples as the positive class.

Recall is the share of actual positive samples that were predicted positive. Recall tells us how many of positive samples we missed. In and OOD detection context, a model which lets many OOD samples slip by undetected will have a low recall score.

Precision and recall are often used together because evaluate the model in different ways that complement each other. If a model has both high precision and high recall, it does not erroneously flag many negative classes as positive, nor does it miss many positive classes.

Sensitivity and Specificity

Sensitivity and specificity is another pair of metrics that is commonly used for evaluating binary classification. Sensitivity is equivalent to recall; the share of positive samples that were correctly predicted as positive. Specificity is the share of the negative samples that were correctly predicted as negative. Sensitivity and specificity are also known as True Positive Rate (TPR) and *True Negative Rate*.

Threshold Independent Metrics

The previous metrics are a clear improvement over simply using accuracy. However, they still have the problem that they are all dependent on what threshold one sets when predicting something to be a negative or positive class. Thus, it becomes harder to compare different models by using these metrics. Indeed, by simply increasing the threshold of any classifier, we can increase the true negative rate. Similarly, by decreasing the threshold, we can increase the true positive rate.

Area under Receiver Operating Characteristic:
AUROC remedies this problem by looking at all possible thresholds, and calculating the TPR (equivalent to sensitivity, recall), and the FPR (equivalent to 1− specificity) for each possible threshold. With these values calculated, we can plot each point on a graph, giving us an Receiver Operating Characteristic (ROC) plot. Figure 2.5 shows this plot, for three different models.

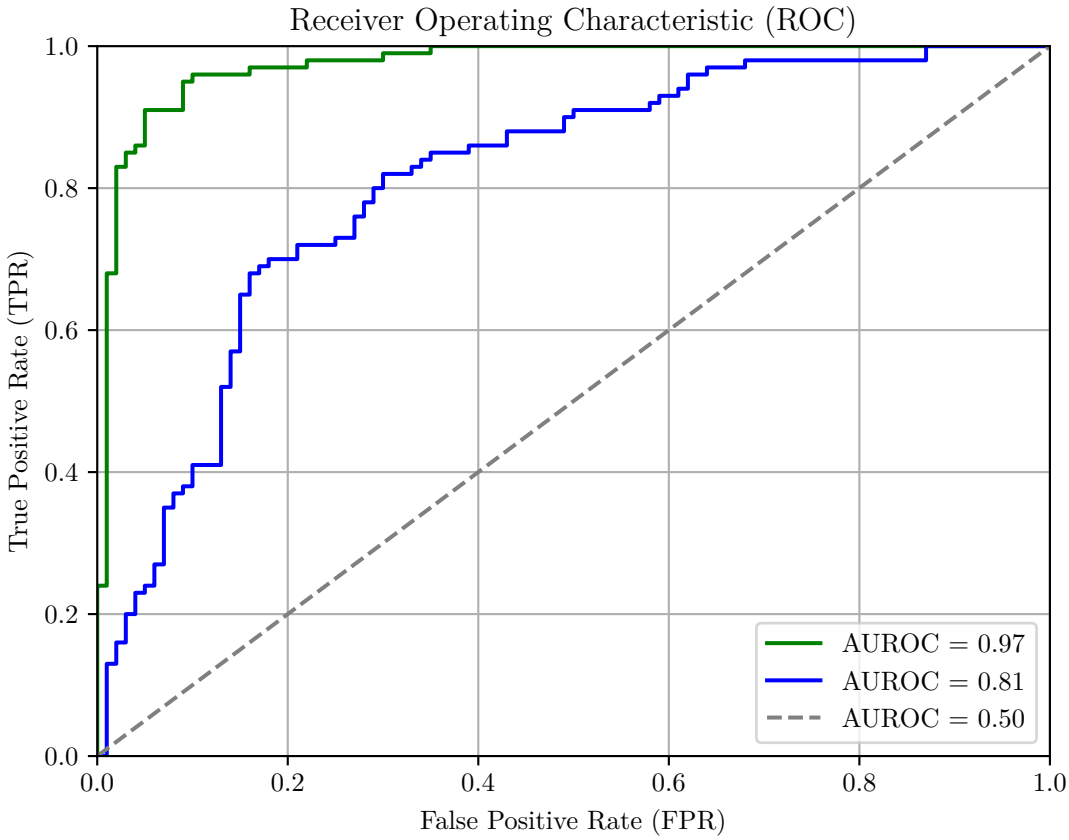

Figure 2.5: Graph showing the distribution of hypothetical ID, Near-OOD and Far-OOD data for an unspecified metric. The shaded region shows the overlap between the ID and OOD samples.

Once we have done this, we can calculate the integral under this curve, giving us the AUROC. If a binary classification model can perfectly separate the two classes, then all possible thresholds will either have 100% TPR or 0% FPR, giving an area under the curve of 1. If instead a model has no discriminative power, then the predicted values

of positive and negative classes are entirely random, and all changes to the threshold will increase one of the metrics at the expense of the other. Such a model would have TPRs and FPRs making a straight line of points, and an AUROC of 0.5. In between these extremes, we can evaluate different models, without having to consider different thresholds. 2.5 shows what the ROC-curve of a model with either 0.97 or 0.80 AUROC looks like, as well as that of a random classifier with AUROC = 0.50.

False Positive Rate at 95% Recall
False Positive Rate at 95% Recall (FPR95) is another way of comparing models without having to consider specific thresholds. Instead, we simply select the threshold which gives a recall (equivalent to TPR) of 0.95, and calculate the FPR at this threshold. The drawback to this metric as opposed to AUROC is that we do not get a general view of how the model performs. However, if we have a requirement that the model has a very high true positive rate, we may not care about how the model performs at any other threshold, and thus this metric is suitable. It is of course also possible to calculate this metric at any other recall value, depending on the application. However, in the field of OOD detection, FPR95 is the metric that is used in the vast majority of cases [6, 12–15].

2.3.2 Statistics: Bootstrapping and T-tests

Bootstrapping [16] is a way to get a better estimation of the true generalization error of a model, by performing the same experiments several times on resampled versions of the original dataset.

2.4 Explainable Artificial Intelligence

Below follows a thorough introduction to XAI, as well as detailed look at some important methods for explainability for neural networks applied to images. Specifically, saliency methods will be explained in detail, as they constitute a core part of my thesis.

2.4.1 The motivation for Explainable Artificial Intelligence

Given the impressive performance of DL methods, one might be convinced that these models do not need to be explainable or interpretable, and that we instead should just place our faith in the model without knowing exactly how it came to a decision. However, as [17] points out, "a single metric, such as classification accuracy, is an incomplete description of most real-world tasks". Small differences between the data distribution when the test data was collected and when the model is deployed may have a large impact on the model's performance, or the model may have learned artifacts or specificities in the training dataset which were also present in the test dataset, leading to a false belief that the model has gained generalizable knowledge when it has not. By using explainable methods, we may reveal these shortcomings.

XAI is also especially important whenever the model is used in settings where its decisions have a high impact. If a model is used by a hospital for disease detection, both the patient and doctor will probably want to be able to understand why the model has found that a disease is present. For them, high performance on a test set of different cases may not be enough. As [2] states, "for the regulated healthcare domain, it is utmost important to comprehend, justify, and explain the AI model predictions for a wider adoption of automated diagnosis". In other high impact areas, such as autonomous driving, the impact of wrong decisions by the network can have fatal consequences, and

customers and regulators will want to be absolutely sure that the models used are robust and base their decisions on relevant factors as opposed to quirks in the training data. Furthermore, the right to an explanation of an automated decision affecting a person is included in the EU's General Data Protection Regulation, which states that "In any case, such processing should be subject to suitable safeguards, which should include specific information to the data subject and the right [...] to obtain an explanation of the decision reached after such assessment and to challenge the decision." [18].

2.4.2 The Properties of an Explanation

2.4.3 Taxonomy of Explainable Artificial Intelligence

This section goes through three axes which define an XAI method:

- Intrinsically explainable models versus post hoc methods
- Model dependent versus model agnostic methods
- Global versus local explanations

Intrinsically explainable models versus post hoc methods

Intrinsically explainable models are models which have sufficiently low complexity, such that it is feasible for a human to understand them without further modifications. Examples of such methods are linear regression, logistic regression and decision trees [19].

Post hoc methods are methods which are applied to the model after training. These methods do not aim to constrain the model to be interpretable, but inspect the model after training. For example, after using a convolutional neural network to classify a CT-scan of a tumour (which gave a prediction of malignant), we could run post hoc algorithms on the network which are able to extract which part of the image contributed the most to the prediction. Thus, post hoc methods remove the need for the model to be simple enough for a human to understand by extracting the relevant information for us.

Model dependent versus model agnostic methods

Model dependence/agnosticity denotes whether an XAI method uses specifics of a particular type of model to generate the explanation, or whether the method can generate an explanation without using specifics of the model at all. Explanations based intrinsically explainable models are clearly model dependent, while methods that only use the input and output of the model instead of looking at the internal operations are model agnostic. An example of a model dependent method (which is not simply an intrinsically explainable model) is Class Activation Mapping, which requires a CNN with a specific architecture to function, while an example of a model agnostic method are Shapley values, which use the inputs and outputs to calculate the marginal effect of a single feature on the output value.

Global versus local explanations

Global explanations provide general relationships between the input features and outputs learned by the model over the entire dataset [20]. In this way, they can show how a specific feature affects the output in general, instead of just how it affects the output of a single point. These methods are ideal for finding trends in the data, but may not be suitable for a patient wanting an explanation for their specific case.

Local explanations do not describe general trends, but focus only on a single data point. These methods give insight into how the features influenced the prediction of a single data point, but these relationships may not hold for other data points, and as such these methods do not give the same insight into the general behaviour of the model.

2.4.4 Benchmarking

In general, it is difficult to evaluate an AI explanations, and there is no clear consensus in the field as to what metrics should be the standard [19, 21].

2.4.5 Specific methods

The following section goes through several specific XAI methods. This serves two purposes: Firstly, it aids the first objective mentioned in section 1.2, by explaining how commonly used XAI methods function in practice, and by elucidating exactly how one is able to go from gradients, counterfactuals and other model outputs to human interpretable explanations. Secondly, this section also covers the specific methods that will be analyzed and evaluated as part of the rest of the objectives. By thoroughly detailing the functions of these methods in this chapter, it becomes easier to analyse and explain the results in chapter 4

Local Interpretable Model-Agnostic Explanations (LIME)

LIME [22] is built on the idea that while the decision function of a large neural network (or any other large model) might be far too complex to easily interpret, it can most likely be approximated quite well by a simpler function, as long as we only look at the feature space around a single data point. For example, we could approximate a large feed forward neural network with a simple linear regression model, which can be intrinsically explained due to its low complexity.

To create a locally interpretable model, we need a neighbourhood of data points around our point of interest. To do this, we can sample a number of points from our dataset and weigh them by their distance to our original point. This sampling can be done in many ways, for example by calculating a mean and variance for each feature and sampling from a normal distribution. For image data, we can create new points similar to the image by masking out different regions of the image [19]. The distance measure depends on the type of data we are dealing with. Regardless, the distance values are passed through a smoothing kernel which can be tuned to adjust the size of the "neighbourhood".

With these new data points, we can generate new predictions using the original, complex model. Thus, we now have a series of points, each with a weighting based on their distance to our original point, and each with a predicted score from our original model. With such a dataset, we can train a simpler model, which will then approximate the complex model around the point of interest. By inspecting this simple model (for

example the learned weights of a linear regression model, or the structure of a decision tree), we can learn approximately how the complex model functions in a region around this single data point.

Class Activation Mapping (CAM)

Class Activation Mapping (CAM) [23] is a model dependent, post hoc XAI method, which is used on Convolutional Neural Nets (CNNs). For a specific output node of a model (for example, the one denoting the presence of a specific class, such as "cat"), CAM outputs a heat map showing which areas of the input image contributed to this node. In this way, CAM gives a visual explanation to which parts of an image the model focused on when making a decision to classify an image to a specific class. This method is model dependent, because it requires a specific architecture in the final layers of the network to work.

CAM is a relatively simple method to understand. It exploits the fact that various convolutional layers of CNNs actually behave as object detectors, even when the training objective is classification [23]. As [11] explains, the earlier layers "extract elementary visual features such as oriented edges, end-points [or] corners", which can be used by subsequent layers to detect higher-order features. In this manner, the final convolutional layer will detect very high level visual features, combining the extracted information from all the previous layers. This layer is composed of several feature maps, where each map can be thought of as denoting the presence of some specific feature across the original image. The authors perform global average pooling (GAP) on these feature maps, giving a single value for each map, which is followed by a single dense layer and the Softmax activation function. In this way, each output node in the final layer is a weighted sum of all the global average pooled feature maps from the final convolutional layer. This means that we can represent the areas of the image which were used to perform the classification by performing the same weighted sum on the actual feature maps instead, which gives us a heat map which we can overlay on the original image (after upsampling the feature maps).

Figure 2.6 shows the process visually. From this we can see that the resulting Class Activation Map (bottom right) gives an intuitive explanation for why the image in the top left gives a high score for the presence of the class "Australian Terrier".

Figure 2.6: Figure taken from [23], showing the steps required to create a Class Activation Map

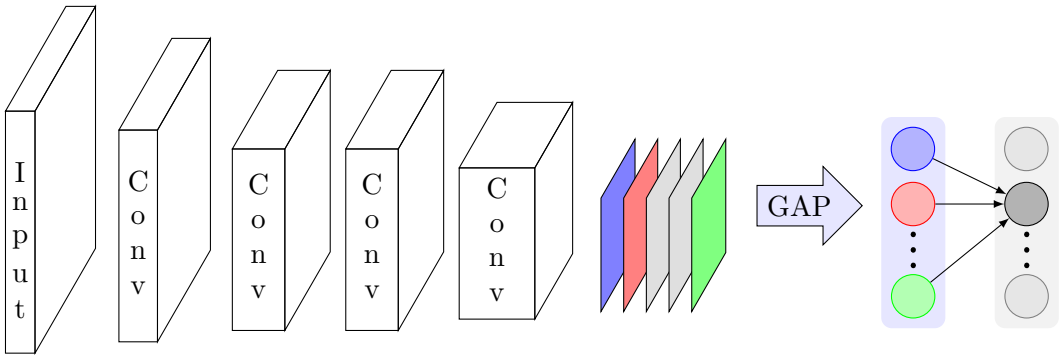

Figure 2.7: Figure showing a high level overview of how a CNN functions

Although CAM is an intuitive and effective method of visualizing the inner workings of a CNN, it has some downsides. Firstly, it is highly model dependent, requiring that the model only have a single dense layer after the convolutions. Although there are some state of the art models which only use a single dense layer, this still places a limit on what models can be used, or requires the simplification of models that use more than a single dense layer. [23, p. 4] notes a 1-2% drop in classification performance when performing this simplification. Secondly, the output of CAM is simply a weighted sum of all the feature maps after the final convolutional layer. As we move deeper in a CNN, we reduce the spatial resolution by downsampling, while increasing the number of channels (increasing the depth of the output while reducing the height and width). Because of this, the CAM will have a drastically lower resolution than the original image, often less than $10\,x\,10$, while the input image may be hundreds of pixels in both dimensions. Because of this, CAM can only show general areas, as opposed to pixel wise explanations.

Gradient Class Activation Mapping (GradCAM)

GradCAM [24] is an improvement on CAM, which generalizes the method to function with any CNN architecture, thus making the method much less model dependent and avoiding the performance drop incurred when simplifying the model with CAM. Instead of using the weights of a final layer to calculate a weighted sum of feature maps in the last convolutional layer, GradCAM uses gradients flowing from the relevant output node to the activation maps to calculate the weights for each feature map. Furthermore, the authors prove that this method is a strict generalization of CAM [24, p. 5], so that no information is lost by using gradients instead of weights.

Like the simplicity of the CAM method, the calculation of the weights using the gradients is also quite simple, as seen in Equation 2.2.

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\delta y^c}{\delta A_{ij}^k} \tag{2.2}$$

Here, c represents the index of the class we are interested in, k the index of the feature map, and i and j the width and height of the image. y^c is the element of the output vector y which corresponds to the class c, while A^k is the k'th feature map. Z is equal to $i * j$, and simply normalizes the sum. Thus, we are actually just performing global average pooling of the gradients of A^k with respect to y^c, which gives us a single value we can use as the weight for this feature map. Doing this for all feature maps for

a specific class gives us all the weights we need to calculate a weighted sum, which we can upsample and visualize to get an explanation for the decision of the CNN.

Thus, GradCAM improves upon CAM by making the method less model dependent. However, the explanations are still the same low resolution, which may not be ideal in all cases.

Integrated Gradients

Layer-Wise Relevance Propagation (LRP)

LRP is another XAI method which generates a visual explanation of the areas of an image which lead to a classification decision. Unlike CAM and GradCAM LRP outputs a map which describes the relevance of every single pixel in the input image, and is thus produces a much more fine grained explanation than these other methods. LRP also differs in that [25] does not define it as a specific method, but rather as a concept defined by a certain set of constraints which can be satisfied by different implementations depending on the type of model.

LRP assumes that we can model the relevance R_i for any node i in a neural network, and aims to find the relevance for all the input nodes (the pixels in an image). Relevance is the contribution of any node to the final prediction $f(x)$ of a network, and the idea is to take the relevance of the output layer (simply defined as the output $f(x)$), and iteratively propagate this backwards through the network. Relevance scores are subject to a conservation property, which means that the sum of relevances must be equal for all layers (Equation 2.3). Furthermore, nodes must also conserve relevance, such that the sum of relevances a node receives from the previous layer is equal to the amount it distributes to the next layer. Once the relevance scores have been propagated from the output to the input layer in accordance with these constraints, we have a measure for how each input pixel contributed to the final output.

$$f(x) = ... = \sum_{d \in l+1} R_d^{(l+1)} = \sum_{d \in l+1} R_d^{(l+1)} = ... = \sum_d R_d^{(1)} \tag{2.3}$$

To distribute relevance between the nodes in a way which obeys the constraints defined in Equation 2.3, a rule for propagation of relevances must be defined. As [25] shows, simply satisfying the constraints is not guaranteed to lead to meaningful explanations, nor is the decomposition of relevance unique. However, they show that by using a suitable propagation rule, we gain a visual explanation which shows which areas contribute to the final decision and which areas make the final decision less likely [25, p. 28].

Occlusion methods

Occlusion methods are a family of post-hoc model independent XAI methods. They function by masking different parts of the image and inspecting the change in output score. If an area leads to a large drop in softmax score for the predicted class when masked, this area must have been important for the network when making the prediction. The mask can be as simple as replacing all masked pixels with a single color, such as gray [26], or they could use more advanced inpainting methods using generative models, for example by replacing a masked tumor with generated healthy tissue.

Regardless of the mask, one can easily calculate the importance of any pixel for a prediction by calculating the average change in the output score for all masks which

contain the specific pixel [27]. Occlusion methods have the advantage of being completely model independent, since they do not consider the internals of the model. However, the computation can be expensive, because we need to run a forward pass for each position of the mask on the image. Figure ?? shows the process visually.

2.4.6 XAI methods adapted to images: Saliency maps

Before delving into specific XAI methods, I will take some time to explain how XAI methods are adapted to images. When explaining tabular data made up of categorical and numerical values, it is often common to explain each feature by associating it with some change in the output prediction. For example, one might say that increasing the number of rooms in a house by one increases the predicted sale price by 30 000 NOK, or that the absence of a balcony decreases it by 25 000 NOK. But, given that images are made up of tens of thousands of features (pixels), considering each feature individually poses a couple of problems.

Firstly, many methods which are designed for data tabular data (which have may have 10-20 features at most) are far too slow when the number of features may be over a hundred thousand. Methods such as Lime, Occlusion or Shapley fall into this category, as they permute the input at a rate which scales with the number of inputs.

The solution to this is to segment the image into larger regions which

Secondly, given that we have so many features, it may no longer be interesting to know exactly how much each feature contributes to a prediction, given that we don't expect any single pixel to have a very large impact. Instead we may be interested in how important pixels are relative to each other, or we may not be interested singular pixels at all.

With these limitations in mind,

2.5 Out-of-Distribution Detection

This section discusses OOD detection, the field which attempts to tackle the second problem discussed in the introduction; that ML models have significantly worse performance on OOD data points and will often "fail silently", making completely wrong predictions with apparent high confidence [28]. OOD detection is a developing field, and still in an initial stage [29]. In 2017, [12] proposed a baseline OOD detection method. This section will discuss this method and the methods which follow it.

2.5.1 Motivation for Out-of-Distribution Detection

When training a model using supervised learning, we implicitly use the "closed-world assumption", which means that we assume that test data will be drawn from the same distribution as the training data [14]. However, when a model is deployed, the data we see may not obey this assumption. Without OOD detection, the model will behave in the exact same way when encountering OOD samples or in distribution (ID) samples, and may even claim to be highly confident in its prediction although the sample is far away from the distribution of the training data [30, p. 1]. In any system where models make high impact decisions, this is a huge problem. We do not want a model to claim high confidence when predicting if a woman has lung cancer if the model has only been trained on men, nor do we want a model to attempt to classify a rare disease that was not part of the training data. Thus, OOD detection methods are necessary, so that OOD samples can be caught before the model makes a prediction and dealt with correctly.

Intuitively, one might assume that distinguishing ID and OOD samples from each other can be solved by simple binary classification using a dataset of ID samples and one of OOD samples. Indeed, if one has sufficient amount of high quality OOD samples, this can be done. However, this can be difficult to obtain in practice [14, p. 15], thus requiring more sophisticated methods of OOD detection.

2.5.2 Semantic versus covariate shift

The first distinction to make in OOD detection tasks is whether an OOD sample is OOD because of *semantic* or *covariate* shift. Semantic shift refers to samples with different classes than the ones the model is trained on. A picture of a giraffe would represent a semantic shift for a model trained to differentiate between cats and dogs, as a giraffe does not belong to either the "dog" or "cat" class. Covariate shift refers to samples which come from a different distribution while still belonging to one of the classes of the original data set. A picture of a chihuahua could represent a covariate shift for the same cat-versus-dog model if the training data contained only other races of dogs. Likewise, an image of a dog in a dark room could represent covariate shift, if all the ID images were of dogs outside, in well lit conditions. The detection of semantic shift, as opposed to covariate shift, is the main focus of most OOD detection tasks [14]. In many applications, it is expected that the model should be able to generalize its prediction to covariate-shifted data, and therefore the focus is on detecting semantic shift. However, the field of medical image classification is one where detecting covariate shift is also important, as the model should only make predictions on data points which are very similar to its training data [14].

Given that the detection of semantic shift has been the main focus of most OOD literature, my work will primarily deal with semantic shift as well. Thus, unless otherwise specified, when I refer to OOD data points, I mean data points which are semantically shifted, i.e that come from another class than those the model has been trained on.

2.5.3 Benchmarking

The performance of an XAI is hard to quantify, because the quality of an explanation is not easily reduced to a number. For OOD detection, performance is much easier to measure, as the problem can be described as a binary classification problem, with OOD and ID samples as the positive and negative class. Thus, we can calculate many different metrics and compare methods against each other. For OOD methods, the two most common metrics to report is the False Positive Rate at 95% recall (FPR95) and the Area Under Receiver Operating Curve (AUROC). FPR95 is, as the name implies, the number of false positives (i.e the number of ID data points that the method falsely believes to be OOD) when 95% of the OOD data points are correctly detected. AUROC can defined as the chance that a random ID data point has a higher ID-score than a random OOD data point [6]. An AUROC of 0.5 is equivalent to random guessing, while an AUROC of 1 is a perfect model that catches all OOD data points. It is common to use ImageNet or CIFAR as the ID dataset, and calculate FPR95 and AUROC on other datasets which contain no overlapping class labels. When selecting OOD datasets, it is common to differentiate between **near-OOD** and **far-OOD**. Far-OOD samples are samples which are drastically different from the ID samples, while near-OOD samples only differ slightly. For our cat-versus-dog classifier, a tiger and a wolf would represent near-OOD semantic shift, while a plane and a car would represent far-OOD semantic shift. As one might expect, detecting near-OOD samples is much harder.

In 2021, [14] defined a generalized OOD detection framework, and in 2023 [6] introduced a comprehensive benchmark of all relevant OOD methods under this framework, which allows for accurate comparisons of methods within the field. This benchmark is discussed in detail in section 3.2.1.

2.5.4 Methods

This section will follow the same outline as section 2.4; firstly, the overarching categories of methods will be discussed, followed by a more detailed look at a selection of specific methods within the field.

The field of OOD is separated into four categories of methods [14]:

- Classification-based methods
- Density-based methods
- Distance-based methods
- Reconstruction-based methods

All methods can also be categorized by whether they are post-hoc or training based. Post-hoc methods take an already trained network and attempt to extract information which separates ID and OOD samples out of the network during inference. These methods have the obvious advantage that they can work out of the box with large pre-trained network without requiring expensive training from scratch. Training based methods train the network in ways which maximize the difference between ID and OOD samples. These methods do not necessarily require OOD samples, but can train using auxiliary loss functions which amplify the differences in network behaviour when faced with OOD data as opposed to ID. Regardless, these methods come with a much higher computational requirement than post-hoc methods, as they require training from scratch or at least retraining using the new loss criterion.

Given the fact that post-hoc methods can be applied to trained networks out of the box, it is quite common to combine both post-hoc and training strategies to achieve the best performance.

Below follows a short explanation of each the four categories mentioned above.

Classification-based methods

Classification-based methods usually use the softmax score or logits of a model to attempt to distinguish OOD and ID samples. [12] made the observation that while the softmax score may be a poor indication of the actual confidence of the model on a single data point, it is still higher on average for ID samples as opposed to OOD samples. By using this simple distinction, they created a baseline model which separated OOD and ID samples. Using input perturbations and temperature scaling, [13] further improved on this method, by amplifying the difference in softmax score of ID and OOD data.

More generally, classification-based methods do not need to use the softmax score, but may attempt to find any metric which separates the distribution of ID samples from OOD samples. Figure **??** shows the probability density for an unspecified metric for both OOD and ID samples. The goal of classification-based OOD detection is to find

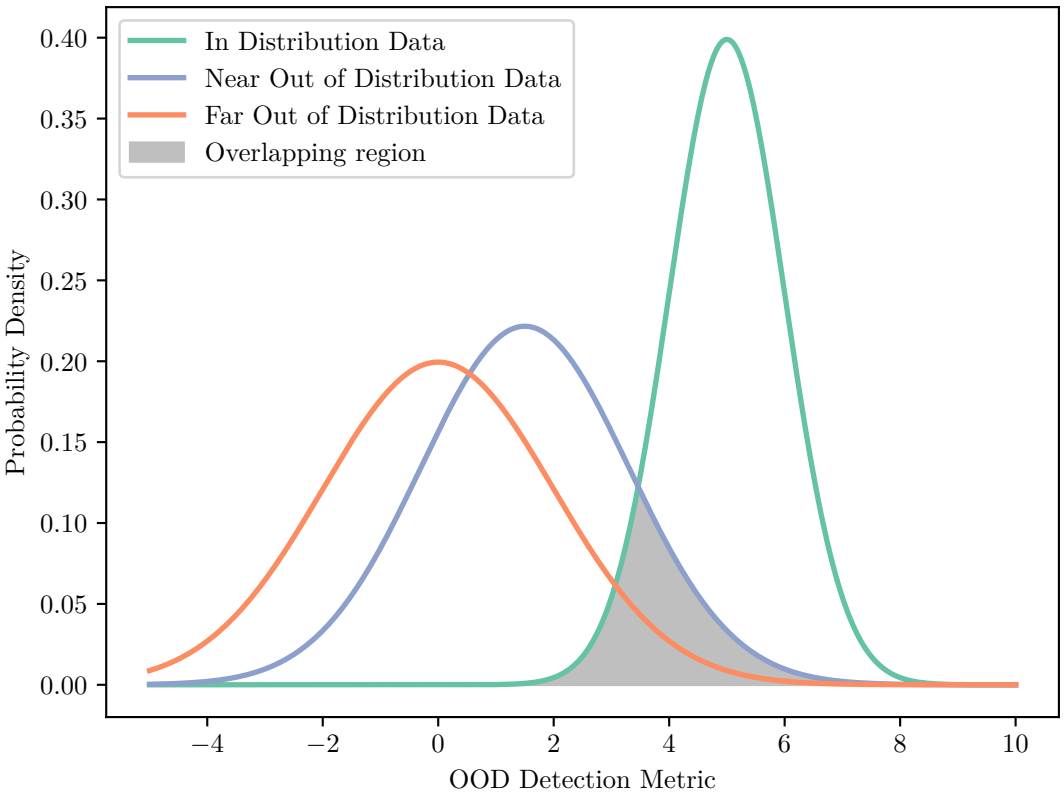

Figure 2.8: Graph showing the distribution of hypothetical ID, Near-OOD and Far-OOD data for an unspecified metric. The shaded region shows the overlap between the ID and OOD samples.

metrics or training methods which make these probability densities have as little overlap as possible, such that they are easily separated by a threshold.

There are several state of the art methods which utilize a classification-based approach, and these make up a large part of the representative methodologies for OOD detection today [14, p. 8]. As such, I shall devote the majority of section 2.5.5 to classification-based methods.

Density-based methods

Density-based methods explicitly try to model the in-distribution [14], which is then used to detect outliers in low likelihood regions. Although the idea is intuitive, learning the distribution of the data set can often be prohibitively expensive, and thus these methods often lag behind classification-based methods [14].

Distance-based methods

Distance-based methods attempt to detect OOD samples by calculating their distance to ID samples. Many different distance measures are used, such as Mahalanobis distance to estimated Gaussian distributions, cosine distance to the first singular vector of the

data set or Euclidean distance in an embedding space.

Reconstruction-based methods

Reconstruction-based methods are based on encoder-decoder frameworks, where the core idea is that the model will be much worse at reconstructing OOD data than ID. By measuring the reconstruction loss, we can detect OOD samples.

2.5.5 Specific methods

Below follows a more detailed look a selection of specific OOD detection methods.

Baseline model

The baseline model created by [12] is extremely simple, yet effective. It simply compares the softmax score the predicted class to a threshold, and labels it as OOD if it falls below this threshold. Let us assume we have a model $f : \boldsymbol{x} \to \mathbb{R}^C$ that takes an input $\boldsymbol{x}$ (which may be an image, a vector of values, or something else) and returns a vector of logits with length equal to the number of classes C. If we define a softmax score function

$$S_i(\boldsymbol{x}) = \frac{\exp\left(f_i(\boldsymbol{x})\right)}{\sum_{j=1}^{N} \exp\left(f_j(\boldsymbol{x})\right)}, \tag{2.4}$$

then the OOD detector has the following simple form, given a threshold δ:

$$g(\boldsymbol{x}; \delta) = \begin{cases} \text{in} & \max_i S(\boldsymbol{x}) \geq \delta \\ \text{out} & \max_i S(\boldsymbol{x}) < \delta \end{cases}, \tag{2.5}$$

This method reasonably well, because the softmax scores for ID data generally is higher than for OOD data. Two years later, [31] showed that this baseline could be improved in settings with larger datasets by forgoing the softmax normalization and instead only looking at the maximum logits, with the even simpler form

$$g(\boldsymbol{x}; \delta) = \begin{cases} \text{in} & \max_i f(\boldsymbol{x}) \geq \delta \\ \text{out} & \max_i f(\boldsymbol{x}) < \delta \end{cases}, \tag{2.6}$$

Perhaps surprisingly, both these methods are quite good at detecting OOD samples. However, there are still many ways to improve these methods, as will be shown in the following sections.

Out-of-Distribution Detector for Neural Networks (ODIN)

[13] improves on the work of [12] by introducing two simple modifications to the method which amplify the difference between the softmax score of ID and OOD samples. Firstly, they alter the input image slightly by adding small perturbations based on the gradients of the cross-entropy loss, as shown in equation 2.7

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \epsilon \operatorname{sign}(-\nabla_{\boldsymbol{x}} \log_{\hat{y}}(\boldsymbol{x}; T)) \tag{2.7}$$

Secondly, they add temperature scaling to the softmax calculation, as shown in equation 2.8:

$$S_i(\tilde{\boldsymbol{x}}; T) = \frac{\exp\left(f_i(\tilde{\boldsymbol{x}})/T\right)}{\sum_{j=1}^{N} \exp\left(f_j(\tilde{\boldsymbol{x}})/T\right)} \tag{2.8}$$

Thus, the OOD detector has the following form, given a threshold δ:

$$g(\boldsymbol{x}; T, \varepsilon, \delta) = \begin{cases} \text{in} & \max_i S(\tilde{\boldsymbol{x}}; T) \geq \delta \\ \text{out} & \max_i S(\tilde{\boldsymbol{x}}; T) < \delta \end{cases}, \tag{2.9}$$

With these modifications, they report large improvements over the baseline [13, p. 4]. To explain this increase, we should look at the mathematical justification for these modifications.

The idea behind perturbing the input image based on the gradient of the cross entropy loss is that ID data points have a higher gradient than OOD data in general. By moving our data point slightly in the direction of the negative gradient, we should expect to see a higher softmax score than if we did not move, regardless of whether the data point is ID or OOD. However, because the gradients are larger for ID data, we expect that the difference between the new softmax scores will be larger than they were before the perturbations, because the ID data point has moved further towards higher softmax values, as shown in figure 2.9, taken from [13, p. 8]. Here we see two data points, one ID (red) and one ODD (blue), which are both perturbed. As we can see, the resulting softmax scores after the perturbations differ more than before the perturbation.

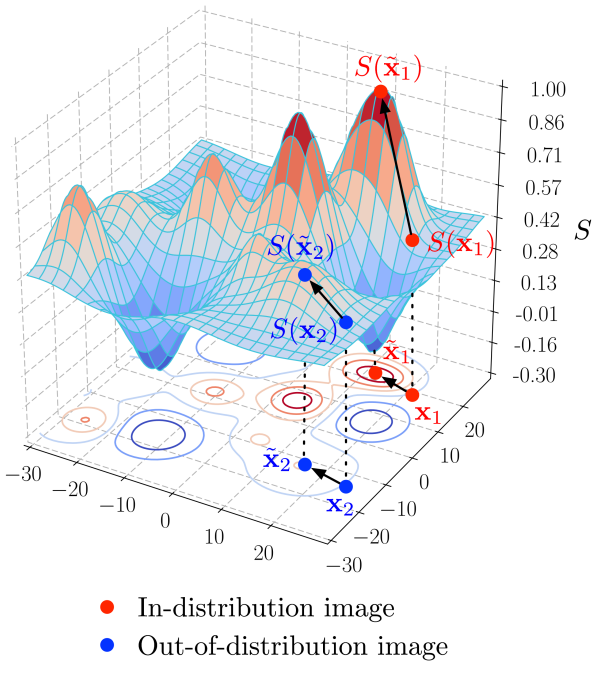

Figure 2.9: Figure taken from [13], showing the difference in gradients between ID and OOD data points

The interpretation of the temperature scaling is slightly more complex. By performing a Taylor expansion and omitting third and higher orders, we can rewrite the softmax score (i.e the value of the predicted class, the highest value) as $S \propto (U_1 - U_2/2T)/T$, with U_1 and U_2 defined as follows [13, p. 4]:

$$U_1(\boldsymbol{x}) = \frac{1}{N-1} \sum_{i \neq \hat{y}} [f_{\hat{y}}(\boldsymbol{x}) - f_i(\boldsymbol{x})] \tag{2.10}$$

$$U_2(\boldsymbol{x}) = \frac{1}{N-1} \sum_{i \neq \hat{y}} [f_{\hat{y}}(\boldsymbol{x}) - f_i(\boldsymbol{x})]^2 \tag{2.11}$$

$\hat{y}$ is the predicted class, and is thus also the index for the highest value in $f(\boldsymbol{x})$. Thus, U_1 represents "the extent to which the largest unnormalized output deviates from the remaining outputs", while U_2 measures how the remaining outputs deviate from each other [13, p. 6].

[13] makes the two following observations with regards to these two values: Firstly, they find that the largest unnormalized output tends to deviate more for ID samples, making U_1 larger than for OOD samples, because the model is more confident in its prediction. Secondly, they find that $E[U_2|U_1]$ is larger for ID data samples than for OOD samples, which shows that ID samples have more separation in the remaining unnormalized inputs than OOD samples.

Returning to the Taylor approximated softmax score $S \propto (U_1 - U_2/2T)/T$, we see that U_1 contributes to making the softmax score higher, while U_2 reduces the softmax score. Given that both these values are higher for ID data, we will want to reduce the impact of U_2 and increase the impact of U_1. As U_1 is divided by T, while U_2 is divided by $2T^2$, increasing the temperature achieves this, as U_2 will decrease much faster than U_1. Thus, we can see how an increased temperature increases the softmax scores for ID data, and thus increases the gap between softmax scores for ID and OOD samples, making them easier to differentiate.

With these two modifications to the simple baseline proposed by [12], [13] manages to increase the gap between the softmax scores of ID and OOD data and thus facilitates much more effective OOD detection.

Energy Based OOD Detection

[30] proposes using an *energy score* as opposed to the softmax score. They show mathematically that "the softmax confidence score is a biased scoring function that is not aligned with the density of the inputs" [30], and thus seek to use a different measurement which is better aligned with the probability density.

An energy function is a function $E(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$ which maps any data point into a non-probabilistic scalar called energy. Energy values can be converted to probabilities using the Gibbs distribution defined below (equation 2.12):

$$p(y \mid x) = \frac{e^{-E(x,y)/T}}{\int_{y'} e^{-E(x,y')/T}} = \frac{e^{-E(x,y)/T}}{e^{-E(x)/T}}, \tag{2.12}$$

This equation is quite similar to the softmax function, and we can see that by defining $E(\boldsymbol{x}, y) = -f_y(x)$. We can write the Gibbs distribution as the normal softmax output of a neural network:

$$p(y \mid x) = \frac{e^{f_y(x)/T}}{\sum_i^K e^{f_i(x)/T}}, \tag{2.13}$$

By using the *Helmholtz free energy* measurement, we can get an energy score for each data point given to the model, which can be used to detect OOD data points. Given that we define $E(\boldsymbol{x}, y) = -f_y(x)$, we can write the Helmholtz free energy $E(\boldsymbol{x})$ as:

$$E(x; f) = -T \cdot \log \sum_i^K e^{f_i(x)/T}. \tag{2.14}$$

The authors show that when training with negative log likelihood loss, the optimization will reduce the free energy of ID data points, and that the difference between ID and OOD energy scores is higher than the difference in softmax scores [30]. Thus, thresholding the free energy function is an effective way to separate ID and OOD data points. Furthermore, they also present a method for fine tuning a pre trained model using a loss function that is based on the energy score. By doing this, the gap between ID and OOD energy scores can be increased even further.

ReAct

ReAct [32] is a very simple method, which also aims to increase the difference in confidence scores between ID and OOD data. It does this by rectifying high activations in the penultimate layer, which surprisingly achieves this very effectively. Figure 2.10 gives an intuition for why this is the case:

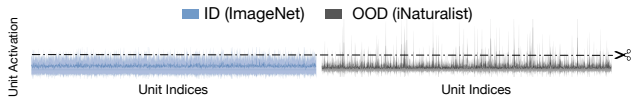

Figure 2.10: Figure taken from [32], showing the activations for the nodes in the penultimate layer for ID and OOD data

From this, we see that OOD samples have much more irregular activations, with a higher variance and many high value outliers. This gives an explanation for why OOD samples produce highly confident softmax scores: sharp positive outliers manifest in the model output, producing high logits in the output layer [32]. By using a positive upper limit to rectify these outliers, we can remove their impact and reduce the confidence for OOD data.

This gives rise to a very simple OOD detector: Let us denote the feature vector of the penultimate layer as $h(\boldsymbol{x})$, where $\boldsymbol{x}$ is the input feature vector. The logits of the network would be calculated by the function

$$f(\boldsymbol{x}) = W\, h(\boldsymbol{x}) + \boldsymbol{b}, \tag{2.15}$$

where W is a matrix which projects $h(\boldsymbol{x})$ down to the output space. $h(\boldsymbol{x})$ is the vector which contains the high activations for OOD data, so by rectifying this vector with $\text{ReAct}(\boldsymbol{x}; c) = min(\boldsymbol{x}, c)$ for a $c > 0$, we can remove these outlier activations. We then get

$$\bar{h}(\boldsymbol{x}) = \text{ReAct}(h(\boldsymbol{x}; c), \tag{2.16}$$

which gives us the new output logits

$$f^{\text{ReAct}}(x;\theta) = W^\top \bar{h}(x) + \mathbf{b}. \tag{2.17}$$

These logits can be used by any other OOD method which uses the output values to separate ID and OOD samples [32]:

$$g(\boldsymbol{x}; f^{\text{ReAct}}, \delta) = \begin{cases} \text{in} & S(\boldsymbol{x}; f^{\text{ReAct}}) \geq \delta \\ \text{out} & S(\boldsymbol{x}; f^{\text{ReAct}}) < \delta \end{cases}, \tag{2.18}$$

This simple methods performs well on many benchmarks, with the added benefit that it can be combined with many other methods. For example, we can use ODIN or Energy with output scores calculated using ReAct instead of unrectified outputs, which leads to improvements over the methods used by themselves.

Virtual Outlier Synthesis (VOS)

Generating outliers to expose to the model during training is another way to reduce the model's confidence on OOD data. However, creating realistic OOD data points can be difficult, especially if the input space is of a high dimension, such as in image classification. [33] presents a more tractable method, which synthesizes outliers not in the input space, but in the feature space, which can be of a much lower dimensionality.

In this lower dimension space, previously intractable methods are now less computationally expensive. To synthesize outliers, [33] simply estimates class conditional Gaussian distributions by computing empirical class means and covariances, and sample outliers from the class boundaries between these Gaussians.

Using these outliers, they present a "unknown-aware training objective", which can be used during training to maximize the separability between ID and OOD data during inference.

GradNorm

As opposed to using the feature or output space, GradNorm [34] attempts to use the gradient space of a network to calculate OOD-ness. They find that the gradients of the weights actually contain valuable information that allows for effective separation of ID and OOD samples, and perform ablation studies which show that this methods outperforms many other methods, including the previously mentioned ODIN and Energy methods.

The gradients are calculated with regards to the Kullback-Leibler divergence between the softmax values and a uniform distribution. An important distinction from other methods is that all the softmax values are used, as opposed to the *softmax score* which would be only the score of the predicted class. Thus, this method captures information about the uncertainty across all categories, as opposed to just the most likely class [34, p. 3]. Once the gradients have been calculated, the threshold is simply done on the L_p-norm of these gradients, giving us the following thresholding function [34]:

$$S(x) = \| \frac{\partial D_{\text{KL}}(u \parallel \text{softmax}(f(x))}{\partial w} \|_p \tag{2.19}$$

As shown in figure 2.11, we see that the gradient norms are consistently lower for OOD data (gray) than ID data (blue).

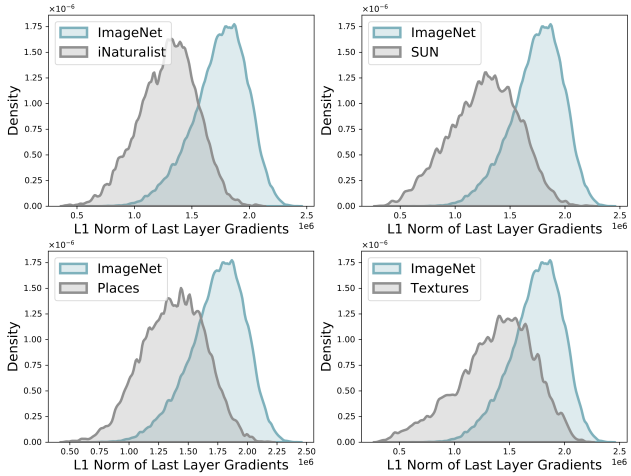

Figure 2.11: Figure taken from [34], showing the difference in gradient norms between ID and OOD data

[34] find that it is sufficient to only calculate the gradients for the last layer of the network, and that the L_1-norm performs the best, as it weights all gradients equally, as opposed to higher norms which place more importance on larger values.

In their mathematical analysis, they show that GradNorm captures joint information from both the feature and output space. By decomposing the L_1-norm of gradients of weights of the last layer with regards to the Kullback-Leibler divergence, they reach the following equality:

$$S(\boldsymbol{x}) = \frac{1}{CT}\left(\sum_{i=1}^{m}|x_i|\right)\left(\sum_{j=1}^{C}\left|1 - C \cdot \frac{e^{f_j/T}}{\sum_{j=1}^{C} e^{f_j/T}}\right|\right) \tag{2.20}$$

From this, we see that $S(\boldsymbol{x})$ is a product of a factor which is simply the L_1-norm of the feature vector $\boldsymbol{x}$, and another term which captures information about the softmax values in the output space.

Virtual Logit Matching (ViM)

[15] attempts to improve OOD detection by calculating a score based on the feature, the logit and the softmax probability at once, as opposed to just one of them. By looking at all three elements in conjunction, they see an increase in performance over models which only rely on a single input source (such as the previously mentioned ODIN).

The reasoning behind not just looking at the logits or softmax probability is that there is a lot of information that is lost when going from features to logits [15]. Once we project the features down to logits, we have only class dependent information, and have lost the class agnostic information which is contained within the features. To show how this information is lost, the authors give an example based on null space analysis [35]:

Let us assume that we have a simplified network with only a single layer. Then, we have $\hat{\boldsymbol{y}} = W\boldsymbol{x}$, where $\hat{\boldsymbol{y}}$ is the vector containing the logits, $\boldsymbol{x}$ is the feature vector of the input (with an additional 1 for the bias term) and W is the matrix containing the weights and biases transforming the feature vector into logits. A null

space Null(W) of a matrix W is the set of all vectors that map to the zero vector, such that $W\boldsymbol{a} = \mathbf{0} \iff \boldsymbol{a} \in \text{Null}(W)$. The null space of a matrix may be trivial (empty), but a matrix which projects vectors to a lower dimension have non-trivial null spaces. Given that the final layer of a neural network projects down to logits, which are the same dimension as the number of classes, this will almost always be the case. Because of the distributivity of matrix multiplication, we have the following:

$$W(\boldsymbol{x} + \boldsymbol{a}) = W\boldsymbol{x} + W\boldsymbol{a} = W\boldsymbol{x} + \mathbf{0} = W\boldsymbol{x} \tag{2.21}$$

The vector $\boldsymbol{x}$ can be decomposed into $\boldsymbol{x}^W + \boldsymbol{x}^{\text{Null}(W)}$, where $\boldsymbol{x}^W$ is the projection of $\boldsymbol{x}$ onto the column space of W and $\boldsymbol{x}^{\text{Null}(W)}$ is the projection of $\boldsymbol{x}$ onto the null space of W. It follows from this and equation 2.21 that when going from features to logits using the projection $W\boldsymbol{x}$, we lose all information contained in $\boldsymbol{x}^{\text{Null}(W)}$. [35] shows how this can be exploited by adversarial methods, by creating images with added noise derived from the null space of a matrix within the network, which are classified as if the noise was not present, despite having no resemblance to the original image. See figure 2.12.

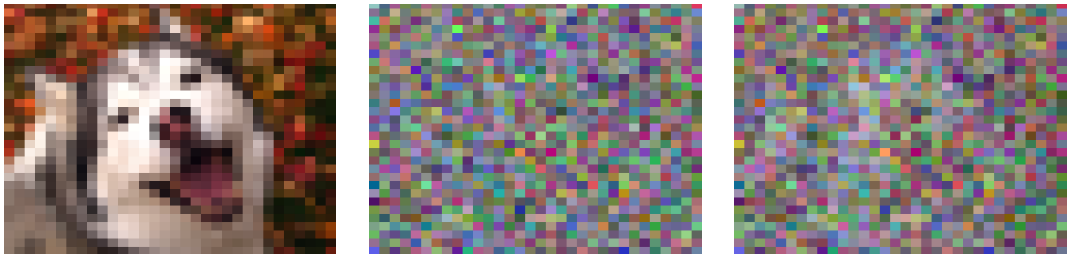

Figure 2.12: Image taken from [35]. Left: Original image. Center: Additive null space noise. Right: Final image, indistinguishable from original image according to the network the noise in the center column is sampled from.

From this, we can see that potentially large amounts of information can be lost when going from features to logits. Using this information, it is also possible to perform OOD detection, as shown by [35]. Another method which uses the features performs Principal Component Analysis (PCA) and looks at the residual information lost when using the first N principal components [36]. However, the information in the features is still class agnostic, and [15] aims to go beyond using just one input source and combine several elements of the network.

To do this, they propose using a *Virtual Logit*. The Virtual Logit is calculated as follows: First, they center the feature space, so that "it is bias free in the computation of logits" [15]. They then perform PCA as in [36], and calculate the residual of $\boldsymbol{x}$ with regards to the principal components, which is the projection $\boldsymbol{x}$ onto the null space of the principal subspace P. The residual represents the information lost when using the projection P.

$$\text{Residual}(x) = ||\boldsymbol{x}^{\text{Null}(P)}|| \tag{2.22}$$

This value is scaled based on the average values of the maximum logit across the dataset, and is appended to the rest of the logits as a Virtual Logit:

$$l_0 := \alpha||\boldsymbol{x}^{\text{Null}(P)}|| \tag{2.23}$$

This now takes part in the computation of the softmax values, and thus is affected by the size of the rest of the logits. They call the softmax value of the Virtual Logit the

ViM score. In this way, the ViM score represents the size of the residual in comparison with the predictions of the model. If the model is very confident, then the norm of the residual will be small in comparison, and the ViM score will be low. If the residual is very large, the ViM score will be higher, and more indicative of an OOD sample. In this way, [15] have combined information from the feature, the logit and the softmax probability level to perform OOD detection.

2.6 Related work

While the combination of XAI and OOD detection has been explored in many previous works, the majority of them focus on explaining why a data point was marked as OOD, as opposed to using XAI to aid the detection itself. Works like [37], [38] and [39] are papers which combine XAI and OOD for this purpose. Within network security, XAI has been as part of anomaly detection systems to detect malicious or faulty network traffic. Here, it has been used to explain detections ([40], [41]), but also to aid in detection itself by inspecting the explanations of the detection system ([42], [43]). These methods thus use XAI to aid OOD detection in a similar manner to my work, however they are strictly focused on sequential network traffic data as opposed to images, and are mostly concerned with detection "unnatural" data samples such as intentionally malicious traffic or that generated by faulty equipment, as opposed to natural OOD data caused by semantic or covariate shift occurring when a model is deployed.

[44] is the most relevant previous work. Here, the authors explicitly aim to use XAI to improve OOD detection on images. They do this by looking at saliency maps produced by a GradCAM-based XAI method (section 2.4.5) during inference, i.e the heatmaps that explain which parts of the image was most influential to classify the image as a specific class. Using these heatmaps, they perform distance-based OOD detection (section 2.5.4): By collecting all explanations for each image in the ID dataset, they are able to construct archetypical explanations, and can make clusters of explanations. To perform OOD detection, they simply compare the explanation of a new data point to the clusters of archetypical explanations, and mark it as OOD if it has a distance which is over a certain threshold.

This method performs decently on toy datasets, achieving scores similar to State-of-the-Art methods when using *Fashin MNIST* as ID and *MNIST* as OOD. However, this method fails catastrophically in more complicated scenarios, achieving an AUROC score of only 52% on *CIFAR10* vs *SVHN*, which is only marginally better than pure guessing and far below any other method. The paper thus ends with the authors concluding that "OoD detection approaches that are specifically designed for the purpose achieve in general better detection scores at the cost of an additional computational burden in the model's construction" [44].

For more potential related work, we can look to OpenOOD ([6]), which aims to provide a comprehensive benchmark of all relevant methods in the field of OOD detection. Out of all 41 OOD detection methods included in this benchmark, there are no methods which use XAI.

From the absence of any relevant method utilizing XAI in OpenOOD and from the poor results of [44], we can see that the potential for a truly effective OOD detection system using XAI has not been fully realized in any previous work.

2.7 Summary

In this chapter, I have given a thorough introduction.

Chapter 3

Methodology

The core of this work is the

As shown in the preceding chapter, there exists a plethora of XAI methods, which exploit gradient information, differences in output scores when altering model inputs, marginal contributions of input features, as well as many other intricacies of deep learning models. The core idea of this master thesis is that these methods, in their attempt to explain a model decision, may also inadvertently extract information which is valuable for OOD detection. Thus, there may be an unexplored potential in these methods to function not just as explanations, but also as classifiers which allow us to separate ID and OOD data. Intuitively, we might expect the explanations to be more spread out on OOD images, given that there are (by definition) no objects of interest in the image that the model can definitely be said to focus on. In contrast, we might expect an explanation on an ID image to be more focused on a specific area, which contains an object of interest. Furthermore, given that saliency methods give a numerical value to each region of the image, we might be able to extract information about the "OOD-ness" of an image by inspecting the magnitudes of these values. Intuitively, it makes sense that such values should be lower for OOD than ID, reflecting the higher uncertainty present in OOD data.

As an example, let us consider GradCAM. As explained in section 2.4.5, GradCAM uses the gradients of the weights within the final convolutional layer with regard to the output prediction, along with the final feature maps, to generate an explanation.

3.1 Proposed XAI methods for OOD detection

In this section, I introduce my proposed methods for OOD detection, which utilize XAI methods as part of their detection pipeline.

3.1.1 Stand-alone saliency methods

As we have seen from section 2.6, there has been little research into using explanations for OOD detection, aside from the work of [44]. Thus, I begin by presenting a simple baseline method which uses saliency values generated by XAI methods to calculate an OOD score.

Aggregate of Saliency

As mentioned previously, the field of OOD detection started with the simple baseline introduced by [12], which simply uses the Maximum Softmax Probability (MSP) (i.e the

confidence score of the predicted class) as a way to measure OOD. As such, I propose a similarly simple baseline when using explanations for OOD detection. The analogue of the maximum logit in an XAI context can reasonably be said to be the explanation generated of the predicted class (the class corresponding to the maximum logit). As explained previously, this explanation will take the form of an $n \times m$ saliency map. This saliency map is not a single scalar value, and does thus not make a suitable OOD score by itself. Instead, we may perform some form of aggregation on the saliency map (such as taking the mean, the max, the variance or some other form), and use this as the OOD score.

The intuition for this method is informed by the fact that there are many forms of aggregation over saliencies which one might reasonably expect to be different for ID and OOD data. As an example, let us consider the implications of aggregating in some way which captures the magnitude of the saliencies, such as the *mean*, the *norm*, or the *max value*. When we generate a saliency map using the predicted class, the XAI saliency method attempts to calculate a measure of importance for each region of the input image, with regard to this class. For ID data, as long as the model predicted correctly, we know that there really are regions of the input image which contain the predicted class. If we instead are looking at semantically shifted OOD data, we know that no input image contains any of the ID classes. Thus, when a neural network makes a prediction on such a data point, it will always be wrong, because it will always predict one of the ID classes. By generating a saliency map of this prediction, we are asking a method to decide how each region contributed to a false decision. Given that there are no objects of the predicted class, in any region, we should expect the saliency values to be lower than in an ID case, where such objects actually are present.

To further illustrate this theoretical justification, I present a simple example scenario (figure 3.1). Here, I imagine a model which has been trained to differentiate between different breeds of dogs. In the first case, it is given an image of a dog, and a prediction of "English Foxhound" is made, which happens to be correct. Generating an explanation for this prediction, each region of the image is given a measure of importance, calculated using gradient information, differences in prediction score on counterfactual examples or by other means. As there actually is an English Foxhound in the image, we expect that these methods generate saliencies which have a magnitude which is higher than if there was no dog present.

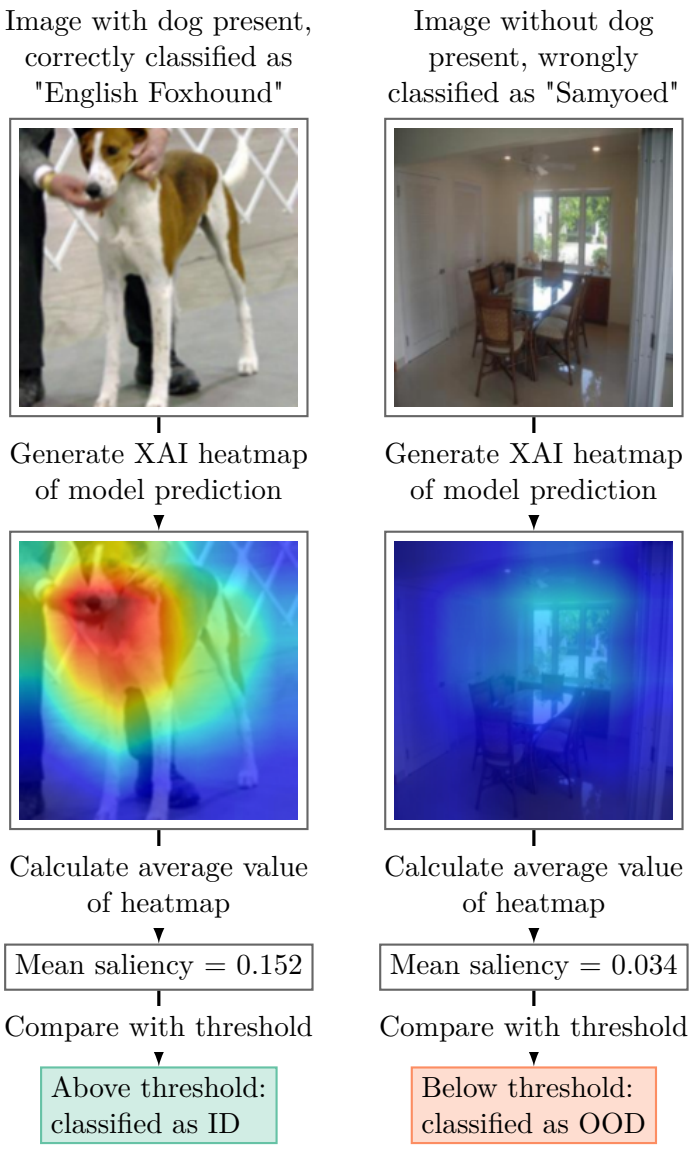

Figure 3.1: Figure showing the functioning of the Aggregate of Saliency OOD detector, using mean as the aggregation, in a hypothetical scenario where a model trained on images of dogs is shown an image with no dogs present

In the second case, the model is given an image without any dogs present. Given that there is no class for images without dogs in them, the model will classify the image as one of the possible dog breeds. In this case, the model predicted the class of "Samoyed", a decision which can be considered essentially arbitrary. When a explanation is generated for this decision, the methods for calculating importance scores will most likely assign saliencies to most regions, given that no region contains a dog. As such, if we calculate mean saliencies for both the image with the dog and the one without, we expect the image with the dog to have a higher mean saliency. As long as we work with semantically shifted OOD data, it will always be the case that the prediction on OOD data is wrong, and thus we may also expect that the generated explanations in general output lower saliencies.

Apart from aggregations which convey information about the magnitude of the

saliencies, we may also expect the variation or spread of the saliencies to be different between ID and OOD data. The intuition is that one might expect the heatmap on OOD data to be less concentrated and more spread out, given that there are no actual objects of interest present.

Figure 3.2 shows a hypothetical scenario in which calculating the spread could be beneficial in OOD detection. Like in the previous case, we imagine a model trained on dogs, which is fed two images, one which contains a dog and one which does not. In this case, we do not care about the magnitudes, but instead only the spread of the values in relation to each other, and thus the heatmap is normalized. As we can see, the heatmap is more spread out in the explanation where there is no dog present than for the image with a dog. By calculating a suitable metric, such as the Gini coefficient, entropy or another measure of spread, we can get a single number which represents how spread out the heatmap is, regardless of its magnitude. Using these values, we can define a threshold which is below most ID data and above most OOD data, allowing us to separate these distributions.

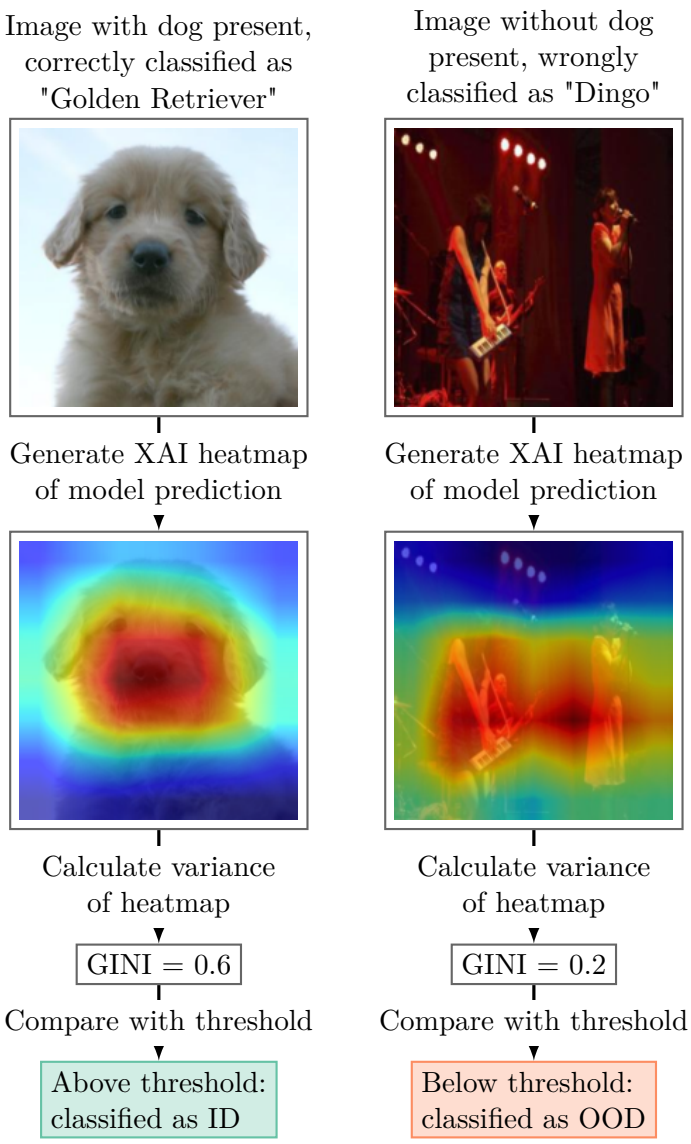

Figure 3.2: Figure showing the functioning of the Spread of Saliency OOD detector in a hypothetical scenario where a model trained on images of dogs is shown an image with no dogs present

With the intuition explained, we may now formalize the method. To define this detector mathematically, let us first define the necessary components. As in chapter 2, we assume we have a model $f : \boldsymbol{x} \to \mathbb{R}^C$. In this case, $\boldsymbol{x} \in \mathbb{R}^{D \times H \times W}$, i.e a D channel image of height H and width W. In addition, we define a XAI saliency mapping method $s : (f, \boldsymbol{x}) \to \mathbb{R}^{H_s \times W_s}$. This function takes the model f and an input $\boldsymbol{x}$ and returns a H_s by W_s saliency map for the predicted class; the class corresponding to the highest logit. Finally, we define an aggregation function $A : \boldsymbol{x} \to \mathbb{R}$, where $\boldsymbol{x}$ can be of any shape.

Thus, the OOD detector has the following form, given a threshold δ:

$$g(\boldsymbol{x}; s, A, \delta) = \begin{cases} \text{in} & A(s(\boldsymbol{x}, f)) \geq \delta \\ \text{out} & A(s(\boldsymbol{x}, f)) < \delta \end{cases} \tag{3.1}$$

Explanation Clustering

3.1.2 Saliency integrated into existing OOD detection algorithms

Given the poor results of [44], one might expect that saliency maps on their own are insufficient to differentiate ID and OOD data.

Virtual Logit Matching with Saliencies

3.2 Datasets

With a thorough introduction to both XAI and OOD detection as well as methods which combine the two, I shall present the datasets that will be used to test the new method that I introduce in chapter 7. The two datasets used are OpenOOD ([6]) and HyperKvasir ([7]). OpenOOD is a comprehensive benchmark which can test the new method's performance in a wide range of OOD scenarios, while HyperKvasir is a medicinal dataset which can be used to test the method in the specific use case of gastrointestinal OOD detection.

3.2.1 OpenOOD

As mentioned previously, OpenOOD was introduced in 2023 by [6] as an attempt to unify the performance metrics of the field, such that accurate comparisons of different methods could be made. Prior to this work, different methods were tested on different datasets, with different image preprocessing procedures, and with other externalities which inhibited effective comparison between methods [6].

OpenOOD includes 11 different benchmarks across Anomaly Detection, Open Set Recognition and OOD detection, three fields which are very closely related. Of these, 6 benchmarks are used to test methods for OOD detection. Each benchmark is defined by an ID dataset, with 6 or more corresponding OOD datasets, separated into Near-OOD and Far-OOD.

CIFAR10

CIFAR10 [45] is a 10-class dataset for general object recognition [6]. This dataset is commonly used in AI research [46]. Each image is 32×32, and there are 50 000 training images and 10 000 test images. The ten classes are as follows: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. OpenOOD presents a series of Near-OOD and Far-OOD datasets which are used in conjunction with this dataset for evaluation. Table 3.1 presents a short description of each dataset and the number of samples.

In addition, a sample of images from CIFAR10 and the corresponding OOD datasets are shown in figure 3.3.

Dataset	Size	Description
CIFAR10	10 000	General objects
CIFAR100	10 000	General objects from same collection as CIFAR10
TinyImageNet	10 000	General objects
MNIST	10 000	Handwritten digits from 0 to 9
SVHN	10 000	Street view house numbers
Texture	10 000	Street view house numbers
Places365	10 000	Places, scenes, locations

Table 3.1: Hello

ImageNet200

Figure 3.4: Figure showing three example images from ImageNet200 and from the corresponding OOD datasets

3.2.2 HyperKvasir

HyperKvasir is the largest gastrointestinal (GI) machine learning datasets, containing 110 079 images and 374 videos collected during gastro- and colonoscopy examinations [7].

Dataset	Size	Description
ImageNet200	10 000	General objects
SSB-Hard	10 000	General objects
NINCO	10 000	General objects
iNaturalist	10 000	Handwritten digits from 0 to 9
Texture	10 000	Street view house numbers
OpenImage-O	10 000	General objects

Table 3.2: Hello

It is of utmost importance to detect and correctly classify disease found in the GI-tract, as the different diseases can range from minor annoyances to highly deadly diseases such as GI-cancer, which has a mortality rate of 63% [7]. Furthermore, the effectiveness of endoscopy in locating these diseases is highly dependent on the skill and knowledge of the human operator, with polyps in the colon having a 20% miss rate [47]. Thus, this is a field which could benefit greatly from computer aided diagnosis, but such an integration requires robust and trustworthy AI-systems which do not make erroneous decisions with high confidence. This makes HyperKvasir an ideal dataset to test a new OOD detection method in a more practical real-world scenario.

Of the 100 079 images, 10 662 have been labeled, with a total of 23 classes. HyperKvasir also contains segmentations and labeled video, however, I shall limit myself to the labeled images and to the task of classification, as per the scope of this thesis as outlined in section 1.3.

3.3 Networks

While it may be interesting to see how these new methods function on different network architectures, the combination of several different novel OOD detection algorithms and three different datasets already presents a considerable amount of evaluation. Thus, to focus the thesis on comparing different OOD detection methods against each other, I believe it is best to fix other parameters such as network architecture. With this in mind, I choose to limit myself to the ResNet [48] family of neural networks. In particular, I use ResNet18 for all tests, either one trained on 32×32 images (for Cifar10), or one trained on 224×224 images (for ImageNet200, HyperKvasir).

Aside from CNNs, Vision Transformers also perform exceptionally on computer vision tasks, and achieve state-of-the-art results in many settings [49]. On ImageNet, they are even dominant, and the top 10 models when considering (top-1) accuracy are all based on vision transformers, as opposed to CNNs.[1] As such one might question the choice of a CNN model, when newer and better models have been developed.

However, given that a large part of the XAI methods have been developed in the CNN paradigm, they are not easily adapted to vision transformers. Methods such as GradCAM and Layer Relevance Propagation (LRP) exploit specific parts of CNN architectures when generating explanations [50], and are thus difficult to use with different architectures. To be able to use a broad section of the representative XAI methods in use today, it is thus preferable to use CNNs as opposed to vision transformers.

[1] https://paperswithcode.com/sota/image-classification-on-imagenet

3.4 XAI Saliency Methods

In the following section, I will describe the saliency methods I have chosen to use.

3.4.1 GradCAM

GradCAM is a natural choice of XAI method, given that is a seminal work which has inspired a large number of other methods [51–53].

3.4.2 LIME

3.4.3 Occlusion

3.5 Evaluation

3.5.1 Metrics

AUROC and FPR95 are the most common metrics used for OOD detection [6, 12–15]. In OpenOOD ([6]), AUROC is chosen as the primary metric used to rank methods against each other. As [6] by far presents the most comprehensive complete benchmark of all OOD detection methods to date, I have followed their methodology and used AUROC when evaluating my methods, while also calculating FPR95 as a secondary metric.

AUROC is agnostic to which class is the positive and which is the negative. Whether we consider OOD samples as 0 or 1, the result will be the same. However, the same is not true for FPR95, necessitating a choice of positive and negative classes. There is no correct answer, but [6] chooses to consider OOD samples as the positive class, to "align with ML convention". It is common to consider abnormalities, anomalies, or the unexpected as the positive class (for example, a cancer detection system would consider the presence of cancer to be part of the positive class), which aligns with the goal of OOD detection. Thus, I have also followed this convention and considered OOD samples as positive, and In-Distribution (ID) samples as negative. In this case, FPR95 has the following interpretation: When 95% of OOD samples are correctly classified as OOD, what percentage of ID samples are incorrectly classified as OOD. In other words, FPR95 measures the rate of "false alarms" when 95% of actual anomalies are caught.

For both of these metrics, the calculations are done on each OOD dataset individually, as opposed to comparing all OOD samples to the ID dataset. Afterwards, the metrics of all Near-OOD and Far-OOD are averaged, giving us two general performance metrics which tell us how a method functions on either Near-OOD or Far-OOD.

3.5.2 Development and Test Sets

Given the scope of my thesis, which focuses on post-hoc OOD methods which do not use outlier exposure or require any form of training, there is no need for a training dataset. Despite this, there is still a chance of overfitting, due to the exploratory nature of developing an entirely new class of methods. By trying out different methods and by tuning different parameters to find out how one might best separate ID and OOD data using XAI, I am also biasing the results. Therefore, it may be possible that potential improvements in performance are not due to the increased quality of the method, but instead because of peculiarities in the specific datasets that the method is used on.

Thus, I have split all the datasets mentioned in 3.2 into development and test sets, and have performed all development of the methods exclusively on the development

set. Only when calculating the final metrics have I used the test set, to remove any bias associated with performing multiple experiments on the same data. Due to the large amount of data available in the CIFAR10 and ImageNet200 datasets, and their associated OOD datasets, I have simply split all datasets into two equally large parts.

3.5.3 Bootstrapping

When evaluating a new method, it is not enough to simply report the results from a single experiment. Instead one should run the same experiment multiple times and perform statistical analysis to ensure that the results are robust. Therefore, I have.

3.6 Implementation

This section goes over the implementation details of my thesis.

3.6.1 Basic hardware and software

Python is the most popular programming language for data science and ML research [54], and as such it is my language of choice as well. Key libraries that I have used are *PyTorch*, *SciKitLearn*, *Matplotlib*, *NumPy* and others. Below is a table of the key libraries and their version numbers.

Library	Version number
PyTorch	1.1
Scikit-Learn	1.2
NumPy	1.3
NumPy	1.3
OpenOOD	1.5

All development and computation was done on a single computer with an `Intel i7-8700K` CPU and an `Nvidia GeForce RTX 3090` GPU.

3.6.2 Method Evaluation: OpenOOD

As explained previously, OpenOOD [6] represents the most comprehensive benchmark for OOD detection methods. It is also a framework which easily allows for development and benchmarking of new methods, and is thus the ideal framework for my purposes.

In particular, OpenOOD includes a "unified, easy to use evaluator" [55] that makes evaluating new methods very simple. All that is required is that new methods inherit from a base class (`BasePostprocessor`[2]), and override the calculation of OOD scores. Code listing 3.1 shows all the code required to create the Aggregate of Saliency OOD detector.

```
class SaliencyAggregatorPostprocessor(BasePostprocessor):
    def __init__(self, config, saliency_generator, aggregator):
        super().__init__(config)

        self.saliency_generator = saliency_generator
        self.aggregator = aggregator
```

[2]In OpenOOD, `Postprocessors` are the OOD detection algorithms that generate an OOD score during inference

```
    def postprocess(self, net: nn.Module, data: Any):
        predictions = torch.argmax(net(data), dim=-1)

        saliencies = self.saliency_generator(net, data)
        score_ood = self.aggregator(saliencies, dim=-1)

        return predictions, score_ood
```

Listing 3.1: Source code listing for the Aggregate of Saliency OOD detector

With this `postprocessor` defined, evaluating it on a specific dataset is similarly simple (listing 3.2):

```
resnet18_pretrained = get_network('cifar10')

ood_detector = SaliencyAggregatorPostprocessor(None, GradCAM, torch.mean)

evaluator = Evaluator(
    net=resnet18_pretrained,
    id_name='cifar10',
    postprocessor=ood_detector,
)

metrics = evaluator.eval_ood()

print(metrics)
```

Listing 3.2: Source code listing for evaluating methods within the OpenOOD framework

This code will calculate the OOD scores for all data samples in both the ID and OOD datasets, and subsequently calculate the AUROC and FPR95 for all the OOD datasets when comparing their OOD values to the values of the ID datasets. Code listing 3.3 shows this output when using the baseline MSP method.

```
            FPR@95   AUROC   AUPR_IN   AUPR_OUT    ACC
cifar100     61.36   86.51     84.20      85.05  95.56
tin          42.02   88.88     88.81      85.57  95.56
nearood      51.69   87.69     86.51      85.31  95.56 # average of two above
mnist        19.38   93.86     79.72      98.89  95.56
svhn         24.78   91.38     84.26      95.49  95.56
texture      43.31   88.68     91.01      80.97  95.56
places365    41.62   89.21     68.49      96.28  95.56
farood       32.27   90.78     80.87      92.91  95.56 # average of four above
```

Listing 3.3: Output of calling `evaluator.eval_ood` with CIFAR10 as the dataset and MSP as the detector

As we can see, OpenOOD allows for very easy evaluation of new methods. Furthermore, it allows for easy comparisons between methods, one of the stated goals of the framework [6]. This makes it an ideal framework for this thesis.

Modifications to OpenOOD

As explained previously, continually testing new methods on the same datasets will bias the final results. As of the writing of this thesis, there are no functionalities in OpenOOD which allow for creating development or test sets, all evaluations are done with entire datasets. For my purposes, which involve continuous exploration of different methods and careful inspection of the datasets, this is inadequate. Thus, I have made modifications to the `Evaluator` class such that it takes a `data_split` parameter during

initialization, and have also modified the function `get_id_ood_dataloader` to accept this parameter and return the correct split accordingly.

Furthermore, there is no functionality for sampling from the datasets as opposed to using them as they are, which is necessary to perform bootstrapping and calculate the statistical significance of the results. I have done this by passing a seed to the `Evaluator` class, which, if defined, will be used to seed a random sampling operation on the datasets. By instantiating several `Evaluator` classes with different seeds, we can bootstrap the evaluation and perform statistical analysis on the results.

Listing 3.4 shows the relevant snippet of code from `get_id_ood_dataloader` which enables both dataset splitting and bootstrapping:

```
# This code is run once for the ID dataset and once for all OOD datasets
sampler = None
if data_split is not None:
    # Always seed with the same value to ensure
    # development and test sets always are the same
    generator = Generator().manual_seed(0)
    val_set, test_set = random_split(
        dataset, [0.5, 0.5], generator=generator
    )

    if data_split == 'val':
        dataset = val_set
    elif data_split == 'test':
        dataset = test_set
    else:
        raise ValueError("data_split only accepts 'val' or 'test'")

    # Create a sampler if bootstrap seed is set, otherwise it is None
    if bootstrap_seed is not None:
        # Variable seed allows for new randomly sampled
        # datasets each time
        generator = Generator().manual_seed(bootstrap_seed)
        sampler = RandomSampler(
            dataset, replacement=True, generator=generator)
# If sampler=None, this just makes a dataloader, if sampler
# is a RandomSampler the dataloader samples with replacement
# from the original dataset
dataloader = DataLoader(dataset, sampler=sampler, **loader_kwargs)
```

Listing 3.4: Code snippet from `get_id_ood_dataloader` which contains the modifications necessary for splitting and bootstrapping the datasets

3.6.3 Implementation of Saliency Methods

There are many libraries which implement XAI methods [22, 56, 57]. When these implementations were suitable for my purposes, I used them. However, given I am not using these explanations for their original purpose (elucidating why a model came to a specific decision), there are many cases where the current implementations are inadequate. The two main problems are lack of access to the raw saliency values and slow speeds.

GradCAM

GradCAM has been implemented from scratch in `PyTorch`. There are several libraries which implement GradCAM [56, 57]. However, given that these libraries are concerned

with simply producing heatmaps that users can inspect, they do not output the raw saliency values, but upscale the saliencies to match the input image dimensions. As explained in 2.4.5, GradCAM uses the final feature map to generate an explanation, which usually has a spatial dimension which is far smaller than the input image (e.g. 7×7 vs 224×224). When overlaying these values on the input image, it is common to use bilinear interpolation [57], which interpolates all 224×224 positions based on the original 7×7 saliency map. For visualizations, this is reasonable. When attempting to use this data to separate ID and OOD data however, this is undesirable. Bilinear interpolation introduces new values, which changes many statistical qualities of the saliency values. This may reduce the separability of different samples. Furthermore, given that these new values do not add any new information, it is inefficient to involve these upscaled saliency maps in any computational operation.

Although it is relatively simple to modify the source code of these libraries to remove the interpolation, GradCAM is not very difficult to implement, and as such I have simply used my own implementation. Listing 3.5 shows the Python code.

```
class GradCAM(torch.nn.Module):
    def __init__(self, model, target_layer):
        super().__init__()
        self.model = model
        self.target_layer = target_layer
        self.grads = None
        self.acts = None

        self.handles = list()
        self.handles.append(
            self.target_layer.register_full_backward_hook(self.grad_hook)
        )
        self.handles.append(
            self.target_layer.register_forward_hook(self.act_hook)
        )

    def __del__(self):
        for handle in self.handles:
            handle.remove()

    def grad_hook(self, module, grad_input, grad_output):
        self.grads = grad_output[0]

    def act_hook(self, module, input, output):
        self.acts = output

    def forward(self, x):
        batch_size = x.shape[0]

        # self.act_hook is called here
        preds = self.model(x)

        self.model.zero_grad(set_to_none=True)

        idxs = torch.argmax(preds, dim=1)

        # backward pass, this gets gradients for each prediction
        # self.grad_hook is called here
        torch.sum(preds[torch.arange(batch_size), idxs]).backward()

        # average last two dimensions so that we have Batch x Channel
        average_gradients = self.grads.mean(-1).mean(-1)
```

```
        # make shape Batch x Channel x 1 x 1 to allow for broadcasting
        average_gradients = average_gradients.unsqueeze(-1).unsqueeze(-1)

        saliency = self.acts * average_gradients
        saliency = torch.sum(saliency, dim=1)
        # remove negative saliencies in accordance with original paper
        saliency = torch.nn.functional.relu(saliency)

        return saliency
```

Listing 3.5: Source code listing for my GradCAM implementation

LIME

LIME has an implementation for Python, written by the original authors [22]. However, this implementation is not suitable for my purposes. The main issue is that it is far too slow, being implemented with `NumPy` which restricts the computation to the CPU. Furthermore, [22] also returns full size heatmaps, although the actual number of saliency values used to create this heatmap is far smaller than the number of pixels in the image.

Thus I have implemented LIME myself, using PyTorch whenever possible.

Occlusion

Captum [56] is a library of XAI methods implemented in PyTorch, and this library contains a suitable implementation of occlusion. As explained previously, occlusion occludes parts of the image and compares the prediction scores before and after the occlusion. By occluding all parts of the image, we can get a saliency value for all positions. Occlusion is usually done using a sliding window, similar to a convolutional kernel, which is slid over all parts of the image. Such a window is rarely a single pixel, because it is often not interesting to see how a single pixel contributes to a prediction, but rather a larger region. What this means is that the final heatmap, although it is the same size as the input image, actually contains far fewer unique values. To avoid performing computations on thousands of repeated values, I reduce the size of the heatmap by sampling one pixel from each of the positions the sliding window has been applied, which can be efficiently done using a 1×1 MaxPooling kernel in PyTorch. Listing 3.6 shows the code listing for this implementation.

```
def occlusion(data, model, block_size):
    targets = torch.argmax(model(data), dim=-1)
    lrp = captum.attr.Occlusion(model)

    attributions = lrp.attribute(
        data,
        target=targets,
        sliding_window_shapes=(3, block_size, block_size),
        strides=(3, block_size, block_size),
    )

    attributions = attributions.sum(dim=1)
    attributions = torch.nn.MaxPool2d(1, stride=block_size)(attributions)

    return attributions
```

Listing 3.6: Source code listing for the Captum occlusion implementation

Chapter 4

Experiments and Results

This chapter contains my findings. In the first section, I show how different forms of aggregation over saliency maps separate ID and OOD data points. This step is important, because it informs the choice of aggregation used in the Aggregation-of-Saliency OOD detection method. I report AUROC for different forms of aggregation over the three datasets. This is done on the validation sets, to avoid biasing the final results.

In the second section, perform OOD detection on the test sets, and report the AUROC and FPR95 for each method on each dataset. In addition, I perform statistical analyses to investigate whether my XAI OOD detection methods outperform baseline methods.

4.1 Data Analysis of Explanations

This section will detail how different XAI methods generate explanations which differ

4.1.1 GradCAM

CIFAR10

ImageNet200

HyperKvasir

4.1.2 LIME

CIFAR10

ImageNet200

HyperKvasir

4.1.3 Occlusion

CIFAR10

ImageNet200

HyperKvasir

Figure 4.7 shows an example of a case which corresponds to the intuition mentioned in chapter 7.2. Here, we see how the spatial extent of explanations generated for a

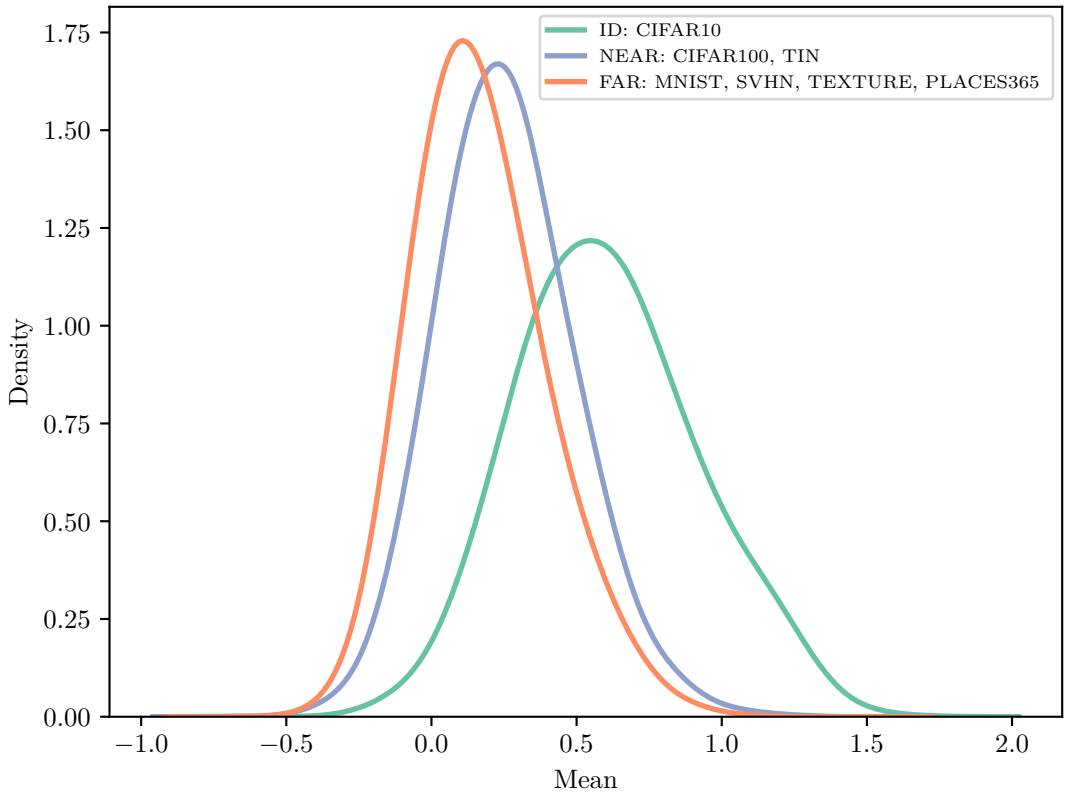

Figure 4.1: Graph showing the distribution of hypothetical ID, Near-OOD and Far-OOD data for an unspecified metric. The shaded region shows the overlap between the ID and OOD samples.

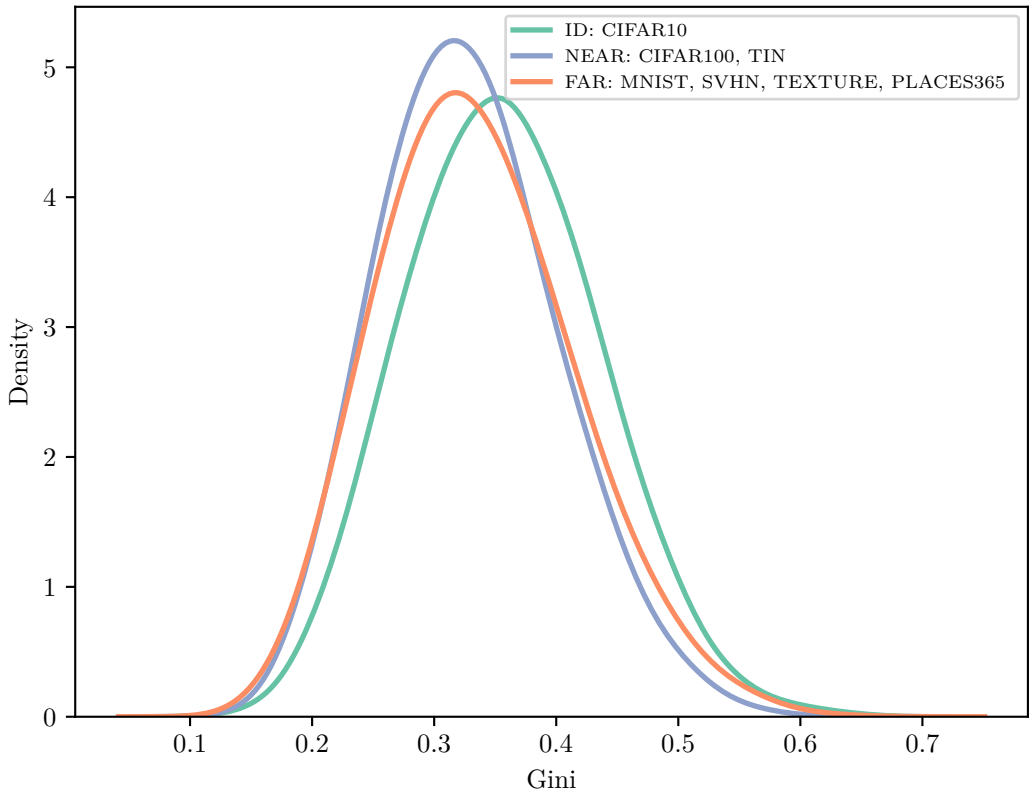

Figure 4.2: Graph showing the distribution of hypothetical ID, Near-OOD and Far-OOD data for an unspecified metric. The shaded region shows the overlap between the ID and OOD samples.

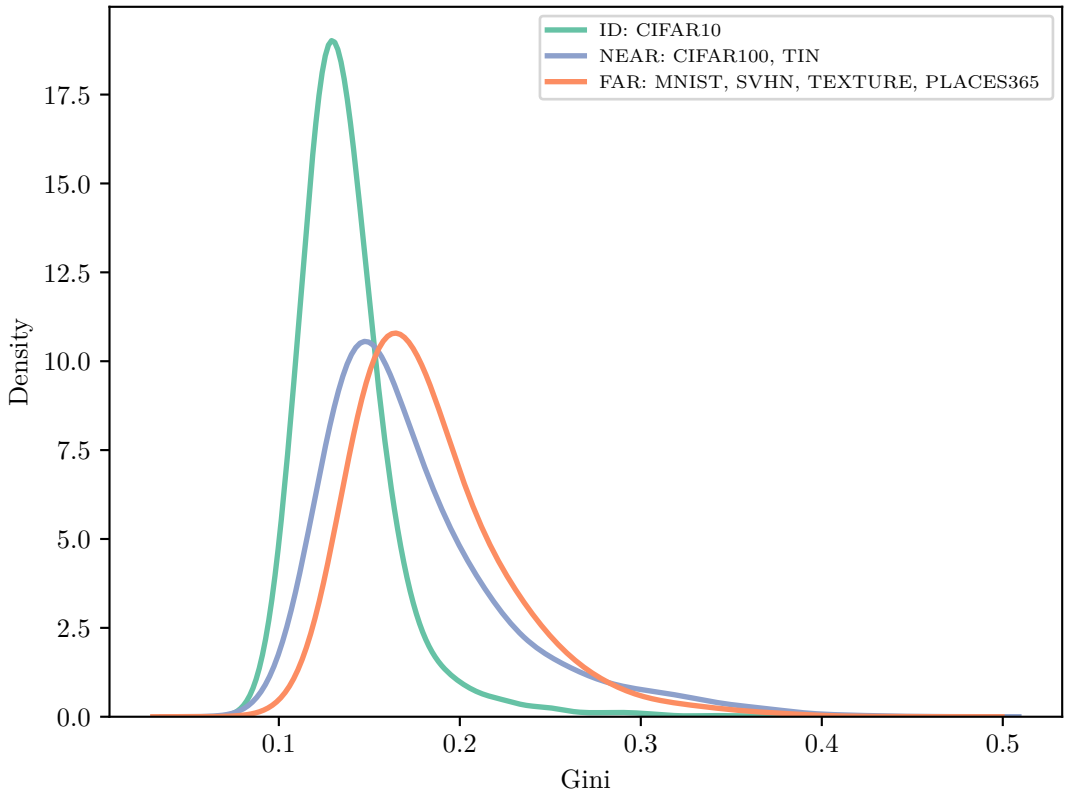

Figure 4.3: Graph showing the distribution of hypothetical ID, Near-OOD and Far-OOD data for an unspecified metric. The shaded region shows the overlap between the ID and OOD samples.

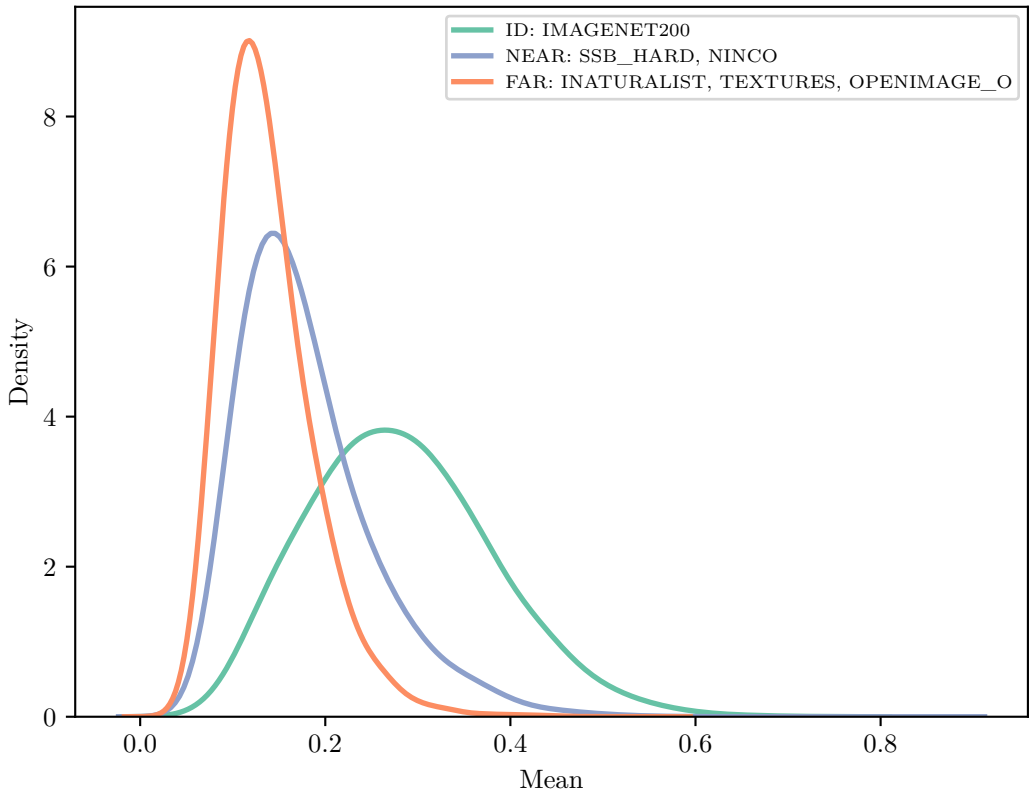

Figure 4.4: Graph showing the distribution of hypothetical ID, Near-OOD and Far-OOD data for an unspecified metric. The shaded region shows the overlap between the ID and OOD samples.

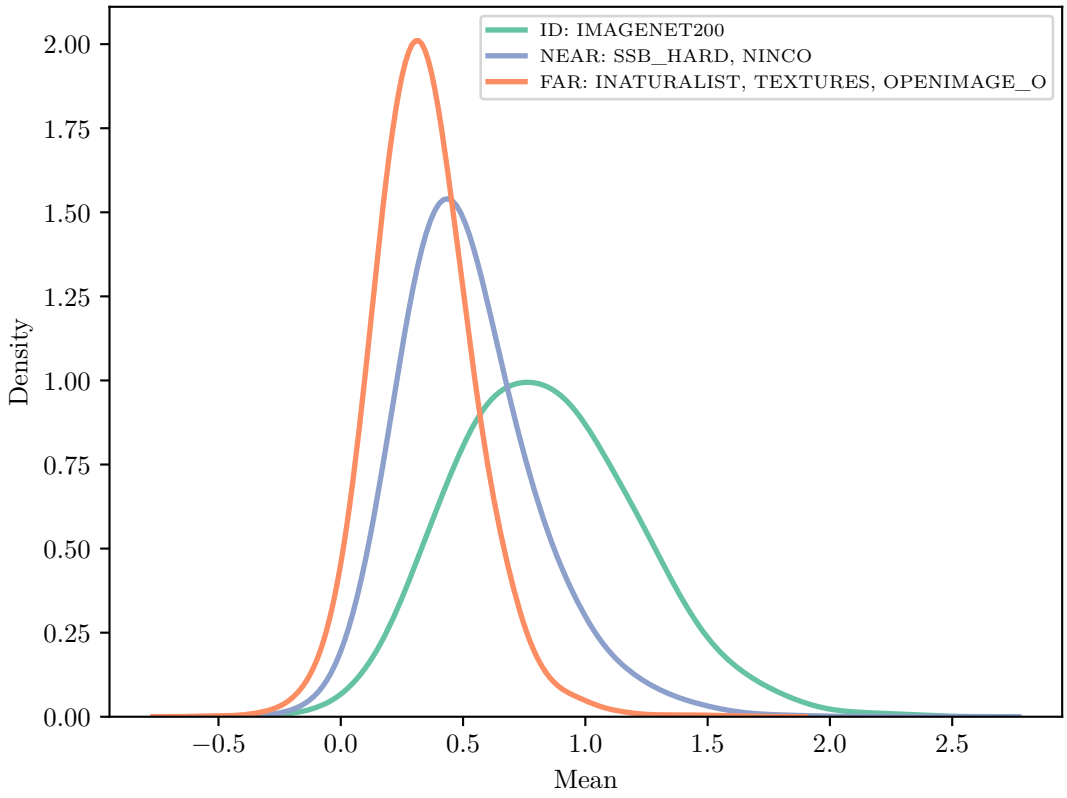

Figure 4.5: Graph showing the distribution of hypothetical ID, Near-OOD and Far-OOD data for an unspecified metric. The shaded region shows the overlap between the ID and OOD samples.

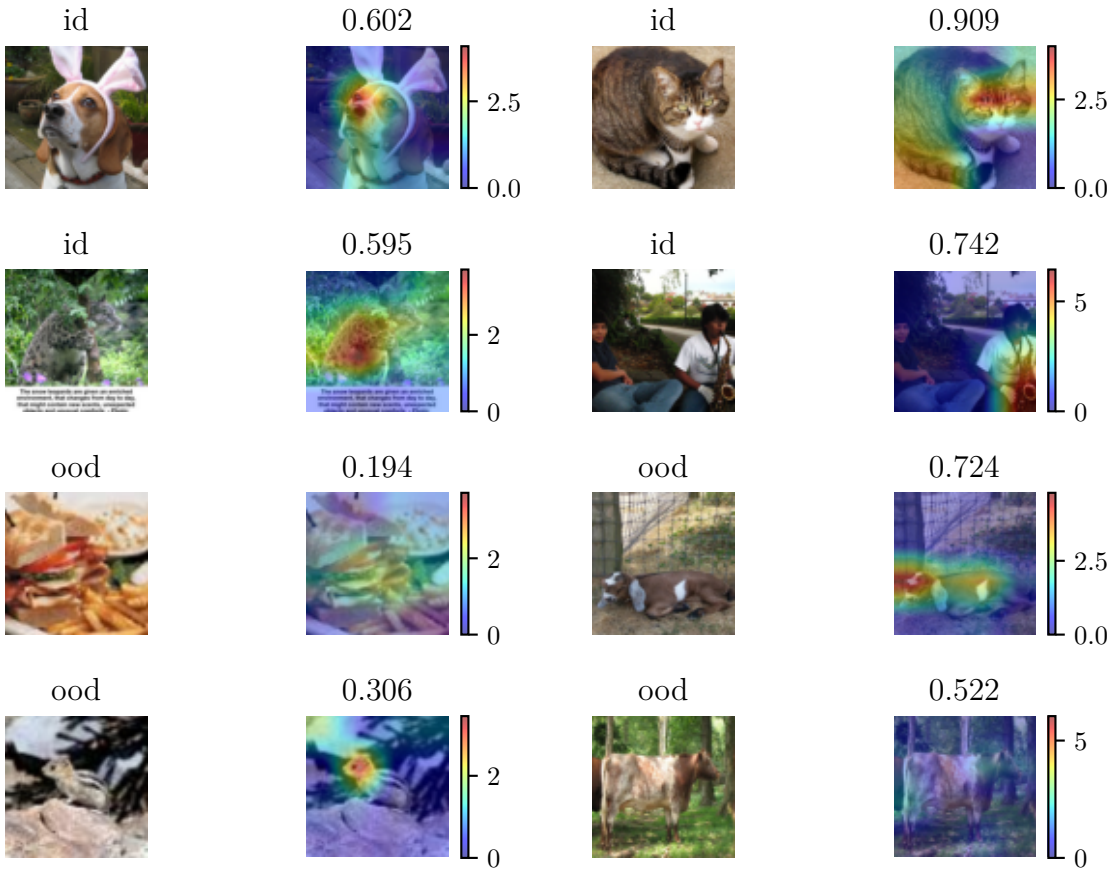

Figure 4.6: Graph showing the distribution of hypothetical ID, Near-OOD and Far-OOD data for an unspecified metric. The shaded region shows the overlap between the ID and OOD samples.

data point in the ImageWoof dataset (ID) differ from the explanations generated for a data point in the Places365 dataset (Far-OOD). In the first instance, the correct class is predicted, and we see that both GradCAM, LIME and Occlusion XAI methods highlight only a small, localized area of the image, corresponding roughly to the face of the dog. In the second instance, the prediction is much less localized, due to the fact that there is no single location corresponding to the predicted class (which is this case was Samoyed).

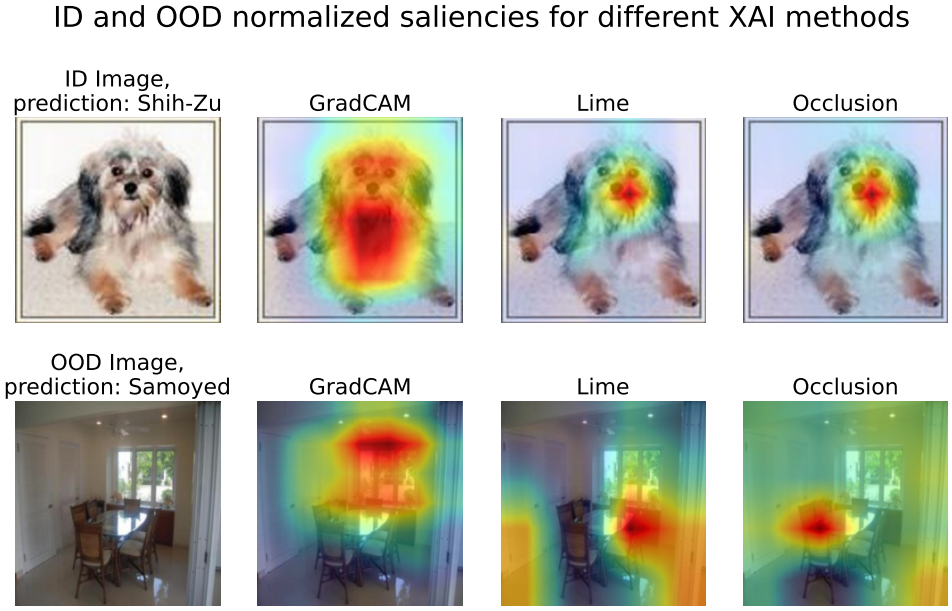

Figure 4.7: Figure

If we look at the same examples as in figure 4.7, but without normalizing the saliencies, we can see from figure 4.8 that there are not just spatial differences in the saliencies generated for ID and OOD data, but also great differences in magnitude. Intuitively, this also makes sense: For ID data, we expect there to be regions which are very important for the predicted class, which get corresponding high saliency values. For OOD data, the predicted class is always wrong (given that the image is, by definition, not of any of the known classes), and thus it less likely that any one region in the image was particularly important for the prediction. It is more likely that the responses of all parts of the image for all the different classes were quite low, and that one of these low responses simply happened to be higher than the others.

Figure 4.9 shows that this can even be the case for near-OOD. Here we see that all though both images contain a dog, only the dog which is actually part of the ID classes generates a significant saliency response.

Looking at the mean saliency values for each image over all datasets (figure **??**, we can see that these examples were not outliers, but consistent with a clear trend in all explanations across the datasets. Here, we can see that the mean saliency per image is on average higher for ID than Near- or Far-OOD, although there is a certain amount of overlap, especially between ID and Near-OOD.

Furthermore, when looking at the average standard deviation for each saliency map (figure **??**), we see a similar result.

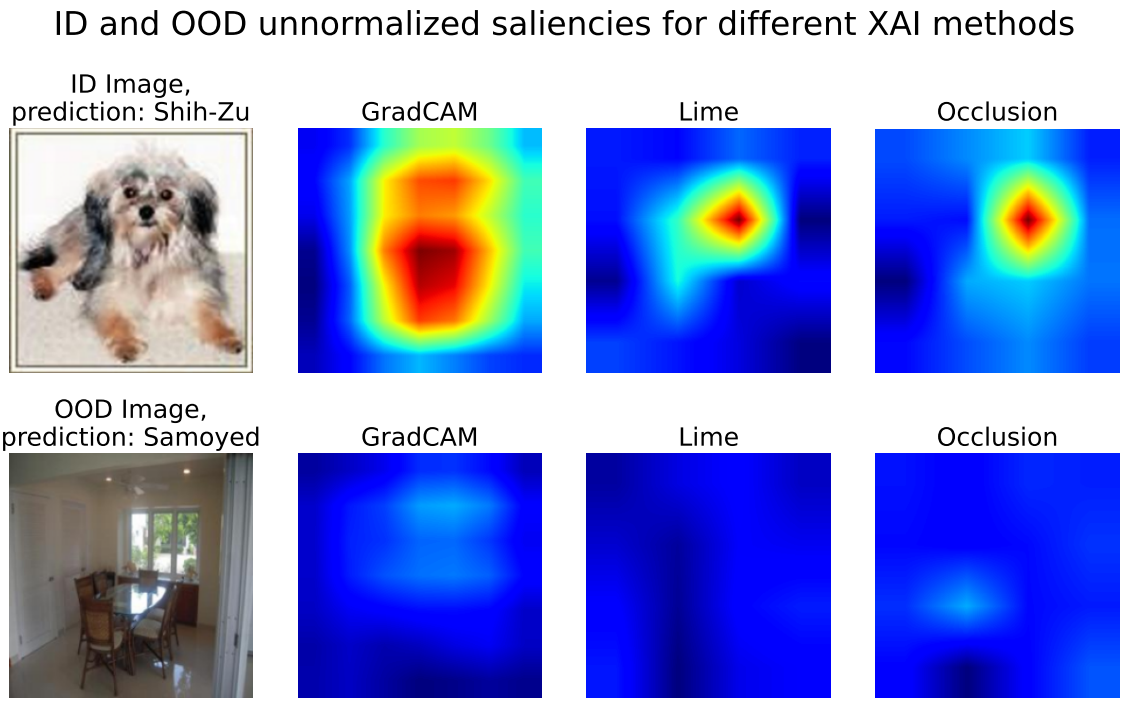

Figure 4.8: Figure

4.2 XAI methods for OOD detection

4.2.1 CIFAR10

4.2.2 ImageNet200

4.2.3 HyperKvasir

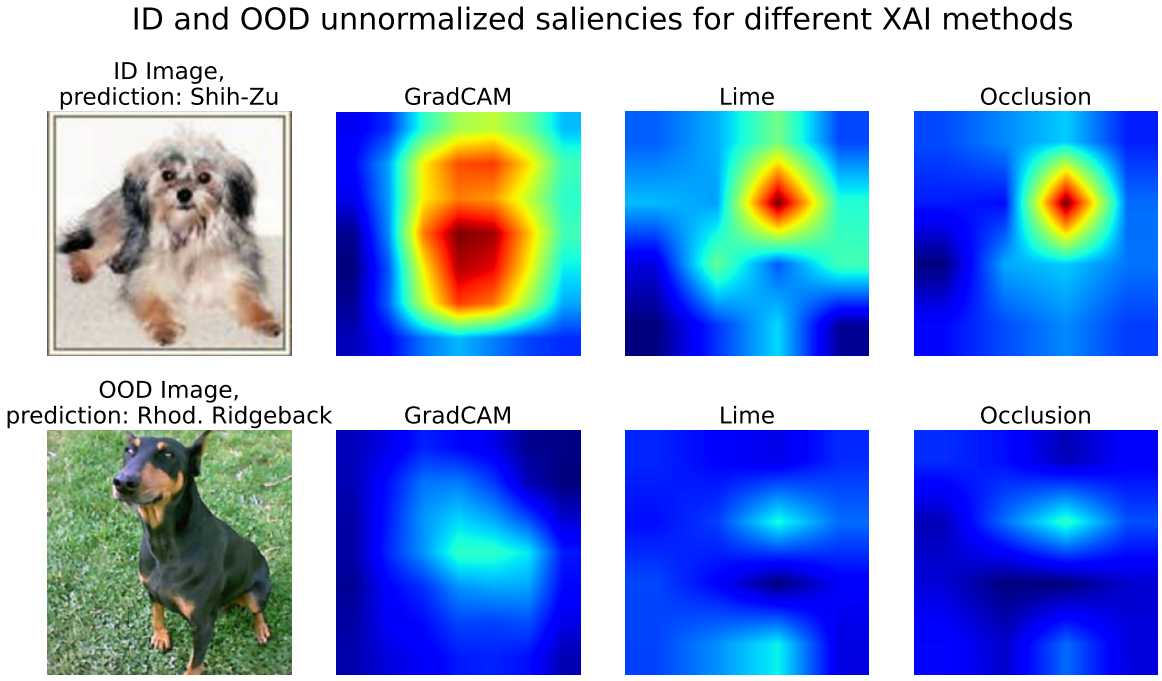

Figure 4.9: Figure

Chapter 5

Discussion

Chapter 6

Conclusion

6.1 Future work

Chapter 7

Temporary chapter

7.1 Overall plan

I will integrate XAI methods into a specific method from each of the four methodologies discussed in chapter 2; classification-, density-, distance-, and reconstruction-based methods. For each method, I shall compare the AUROC and FPR95-scores before and after integrating XAI values.

Furthermore, I will perform data analysis on

7.2 Motivation

Many OOD-detection methods use the final features of the network, i.e the values right before we generate logits for each class. In convolutional neural networks values typically contain a lot of semantic information, but little spatial information. For example, the ResNet family of convolutional networks use global average pooling on the feature map generated by the last convolutional layer to generate the penultimate features. When doing this, all positional information is lost, as we average over the pixel values in each channel, so that a 512 channel 7 by 7 feature map becomes a vector of 512 values. Thus, we have only the average activation of each channel, which gives us information about what is in the image, but not where.

Explainability methods, especially local saliency methods, give information about where a model is focused when making a prediction. Such information could be valuable to deciding whether a data point is OOD. For example, by comparing explanations for OOD and ID data, we may find that the model is focused on several different areas of the image in an OOD data point, while it is focused on a single area in ID samples. The reasoning for this is that if a data point is ID, it must contain an object of the class the model has been trained on. It is then likely that this object will generate a higher response from the network than other parts of the image, which will be picked up by XAI algorithms. For an OOD data point, there is, by definition, no object of any of the classes that the model has been trained on. In this case, the predicted class will be essentially random, and it is less likely that there is a single object which elicits a very high response from the network. Thus, we expect to see an explanation which is more spread out.

Furthermore, the explanations also contain magnitude information about the saliencies of different regions of the image, which one might expect to be higher for ID data, where there should be clear regions which contribute greatly to the final prediction. In OOD data points, we would expect there to be no region which is particularly

important to the predicted class the model happened to output, given that (in the case of semantic shift) this class is not even present in the image.

7.3 Proof of Concept, preliminary investigations: ImageWoof

7.3.1 Results

7.4 Combine XAI and OOD

Given the poor results of [44] in their attempt to perform OOD detection using only explanations, it is likely that explanations on their own will not be sufficient to discriminate between ID and OOD samples. Instead, I believe that by integrating explanations into OOD methods which use other features of the network, we could increase the gap between OOD and ID and increase the discriminatory power of these methods.

7.4.1 Virtual Logit Matching (ViM)

As a first choice of OOD method, I choose ViM. The reasoning is that ViM already is a method which attempts to use multiple sources of information from a network at once to determine an OOD-score. The baseline ViM uses the logits, features and softmax probabilities to determine OOD-ness. I propose to adapt this method to also include saliency values, which can come from different XAI methods.

The methods I choose are LIME, Occlusion and GradCAM. The values are appended to the logit values for each sample. In this way, we gain some positional information along with the strictly semantic information of the logits of ResNet model.

7.5 Implemented method

Although there exists many libraries for XAI methods such as LIME, Occlusion and GradCAM, I have implemented them all from scratch. Given that I am using XAI-methods for an entirely different purpose than the libraries are designed for (OOD detection versus explainability), this is not so surprising.

Lime and Occlusion have been implemented from scratch to work best in the context of OOD detection. Specifically, they have been implemented in such a way that the calculations can be batched, to take advantage of large

GradCAM has been implemented from scratch to allow for certain

7.6 Analysis of XAI methods

In this chapter, I will conduct a thorough analysis of the explanations generated by different XAI methods. The focus will be on the possible differences between explanations made on ID and OOD data points. With this in mind, I will compare and contrast the different methods, attempting to identify possible candidate methods for use in OOD detection.

7.7 Choice of XAI methods

While there exists a large multitude of XAI methods, the problem statement (OOD detection on image data) imposes a number of restrictions which restrict the choice of methods.

Firstly, given the wish to use XAI methods for OOD detection, it is natural to favour local methods, i.e those that give explanations to single samples. Global methods, which describe the relation between the input features over a whole dataset, give a greater understanding of how the model functions in general but are of little use when attempting to decide whether a single data point is OOD, which is the goal during deployment of OOD methods.

Secondly, since I am working with image classification problems, it is most reasonable to use post-hoc methods, as opposed to intrinsically explainable models. Image classification is complex problem which can not be satisfactorily solved by simple methods such as logistic regression or random forests, and as such intrinsically explainable models will have unacceptably poor classification performance. Instead, the field of image classification is dominated by complex models such as Vision Transformers and Convolutional Neural Networks, which are not intrinsically explainable. Indeed, as shown by [20], the vast majority of XAI methods applied to images in the medical field are post-hoc.

The majority of XAI methods applied to image models are so-called *saliency methods*, also known as *pixel attribution methods* [19]. These methods produce a heatmap denoting which regions of the image the model was "focused on" when making the prediction, attributing a numerical value of importance to each pixel or region in the image. For example, when producing an explanation for why a classification model chose to label an image as "cat", we would expect the explanation to highlight the regions where the cat is, and not parts of the background.For my purposes, these methods are well-suited, as they output numerical values which are easy to compare between examples. In contrast, image captioning methods generate textual explanations, which are much harder to compare.

In conclusion, given the requirements set by the type of data (images) and the task (OOD detection), the XAI methods I choose will:

- be local rather than global
- be post-hoc rather than intrinsically explainable
- output numerical values (such as saliency values) rather than text

Based on these criteria, I choose N number of methods, all of which have been introduced and explained in chapter 2.4.

- Gradient Class Activation Mapping (GradCAM)
- Occlusion
- Local Interpretable Model-agnostic Explanations (LIME)
- Layer Relevance Propagation (LRP)

7.8 Analysis of methods on a proof of concept dataset

To investigate the potential for XAI methods to aid in separating ID from OOD data, I introduce a simple collection of datasets, which form an OOD detection problem. This collection consists of an ID, Near-OOD and Far-OOD dataset. The ID dataset is ImageWoof [58], a subset of ImageNet which contains ten different breeds of dogs which are to be classified. For Near-OOD data points, the obvious choice would be pictures of dogs which are of different breeds than the 10 breeds used in ImageWoof. Thus, I use the Stanford Dog Dataset [59], and simply remove any classes which overlap with the 10 from ImageWoof. Given that dogs from two different breeds can have many similarities in terms of size, colour, texture and body and face structure, the Stanford Dog Dataset represents a very small semantic shift and consequently, a difficult OOD detection task.

For a far-OOD dataset, I use Places365 [60], a dataset used for scene recognition. This dataset has categories such as *Classroom*, *Forest road* or *Conference Room*, and thus does not contain anything resembling dogs or even any animal at all.

Below (figure 7.1) is a set of examples for each of these datasets. As we can see, ImageWoof and Stanford Dogs are quite similar, however, no breed is present in both datasets.

Figure 7.1: Figure

With this collection, I train a ResNet-50 model on the ID dataset and subsequently run inference on all three datasets, generating explanations for all data points.

7.9 Results

When inspecting the generated saliency heatmaps, we can either normalize the values (making the maximum value equal to one and the minimum equal to 0) before displaying them, or keep the saliency values unnormalized. Because the intended purpose of saliency methods is simply to show which regions are more important than others when making a prediction, it is common to normalize the values before visualization [19, 24]. However, as I will show, there is much understanding to be gained by looking at the unnormalized saliencies directly, especially for OOD detection. In fact, I posit that one of the reasons

for the poor performance attained by [44] is the presence of normalization in their implementation.

For each method, I will

7.9.1 GradCAM

Chapter 8

XAI for OOD Detection: Integration of Select XAI Methods

8.1 Saliency methods

8.2 Methods

Bibliography

[1] Shaveta Dargan et al. 'A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning'. In: *Archives of Computational Methods in Engineering* 27.4 (Sept. 2020), pp. 1071–1092. ISSN: 1886-1784. DOI: 10.1007/s11831-019-09344-w. URL: https://doi.org/10.1007/s11831-019-09344-w.

[2] Sajid Nazir, Diane M. Dickson and Muhammad Usman Akram. 'Survey of explainable artificial intelligence techniques for biomedical imaging with deep neural networks'. In: *Computers in biology and medicine* 156 (2023), p. 106668. URL: https://api.semanticscholar.org/CorpusID:257067347.

[3] Alvin Rajkomar, Jeffrey Dean and Isaac Kohane. 'Machine Learning in Medicine'. In: *New England Journal of Medicine* 380.14 (2019), pp. 1347–1358. DOI: 10.1056/NEJMra1814259. eprint: https://www.nejm.org/doi/pdf/10.1056/NEJMra1814259. URL: https://www.nejm.org/doi/full/10.1056/NEJMra1814259.

[4] Jordan Zheng Ting Sim et al. 'Machine learning in medicine: what clinicians should know'. en. In: *Singapore Med J* 64.2 (May 2021), pp. 91–97.

[5] Christian Szegedy et al. *Intriguing properties of neural networks*. 2014. arXiv: 1312.6199 [cs.CV].

[6] Jingyang Zhang et al. 'OpenOOD v1.5: Enhanced Benchmark for Out-of-Distribution Detection'. In: *arXiv preprint arXiv:2306.09301* (2023).

[7] Hanna Borgli et al. 'HyperKvasir, a comprehensive multi-class image and video dataset for gastrointestinal endoscopy'. In: *Scientific Data* 7.1 (2020), p. 283. ISSN: 2052-4463. DOI: 10.1038/s41597-020-00622-y. URL: https://doi.org/10.1038/s41597-020-00622-y.

[8] Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.

[9] F. Rosenblatt. 'The perceptron: A probabilistic model for information storage and organization in the brain.' In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519. URL: http://dx.doi.org/10.1037/h0042519.

[10] George V. Cybenko. 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314. URL: https://api.semanticscholar.org/CorpusID:3958369.

[11] Yann Lecun et al. 'Gradient-Based Learning Applied to Document Recognition'. In: *Proceedings of the IEEE* 86 (Dec. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[12] Dan Hendrycks and Kevin Gimpel. *A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks*. 2018. arXiv: 1610.02136 [cs.NE].

[13] Shiyu Liang, Yixuan Li and R. Srikant. *Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks*. 2020. arXiv: 1706.02690 [cs.LG].

[14] Jingkang Yang et al. *Generalized Out-of-Distribution Detection: A Survey*. 2024. arXiv: 2110.11334 [cs.CV].

[15] Haoqi Wang et al. *ViM: Out-Of-Distribution with Virtual-logit Matching*. 2022. arXiv: 2203.10807 [cs.CV].

[16] B. Efron. 'Bootstrap Methods: Another Look at the Jackknife'. In: *The Annals of Statistics* 7.1 (1979), pp. 1–26. DOI: 10.1214/aos/1176344552. URL: https://doi.org/10.1214/aos/1176344552.

[17] Finale Doshi-Velez and Been Kim. *Towards A Rigorous Science of Interpretable Machine Learning*. 2017. arXiv: 1702.08608 [stat.ML].

[18] European Union. *Article 71: European Data Protection Board*. Accessed: February 13, 2024. 2016. URL: https://www.privacy-regulation.eu/en/r71.htm.

[19] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. Independently published, 2022. URL: https://christophm.github.io/interpretable-ml-book.

[20] Bas H.M. van der Velden et al. 'Explainable artificial intelligence (XAI) in deep learning-based medical image analysis'. In: *Medical Image Analysis* 79 (2022), p. 102470. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2022.102470. URL: https://www.sciencedirect.com/science/article/pii/S1361841522001177.

[21] Meike Nauta et al. 'From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI'. In: *ACM Comput. Surv.* 55.13s (July 2023). ISSN: 0360-0300. DOI: 10.1145/3583558. URL: https://doi.org/10.1145/3583558.

[22] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016. arXiv: 1602.04938 [cs.LG]. URL: https://arxiv.org/abs/1602.04938.

[23] Bolei Zhou et al. *Learning Deep Features for Discriminative Localization*. 2015. arXiv: 1512.04150 [cs.CV].

[24] Ramprasaath R. Selvaraju et al. 'Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization'. In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://dx.doi.org/10.1007/s11263-019-01228-7.

[25] Sebastian Bach et al. 'On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation'. In: *PLOS ONE* 10.7 (July 2015), pp. 1–46. DOI: 10.1371/journal.pone.0130140. URL: https://doi.org/10.1371/journal.pone.0130140.

[26] Matthew D. Zeiler and Rob Fergus. 'Visualizing and Understanding Convolutional Networks'. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 818–833. ISBN: 978-3-319-10590-1.

[27] Håvard Horgen Thunold et al. 'A Deep Diagnostic Framework Using Explainable Artificial Intelligence and Clustering'. In: *Diagnostics* 13.22 (2023). ISSN: 2075-4418. DOI: 10.3390/diagnostics13223413. URL: https://www.mdpi.com/2075-4418/13/22/3413.

[28] Ian J. Goodfellow, Jonathon Shlens and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML].

[29] Peng Cui and Jinjia Wang. 'Out-of-Distribution (OOD) Detection Based on Deep Learning: A Review'. In: *Electronics* 11.21 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11213500. URL: https://www.mdpi.com/2079-9292/11/21/3500.

[30] Weitang Liu et al. *Energy-based Out-of-distribution Detection*. 2021. arXiv: 2010.03759 [cs.LG].

[31] Dan Hendrycks et al. *Scaling Out-of-Distribution Detection for Real-World Settings*. 2022. arXiv: 1911.11132 [cs.CV]. URL: https://arxiv.org/abs/1911.11132.

[32] Yiyou Sun, Chuan Guo and Yixuan Li. *ReAct: Out-of-distribution Detection With Rectified Activations*. 2021. arXiv: 2111.12797 [cs.LG].

[33] Xuefeng Du et al. *VOS: Learning What You Don't Know by Virtual Outlier Synthesis*. 2022. arXiv: 2202.01197 [cs.LG].

[34] Rui Huang, Andrew Geng and Yixuan Li. *On the Importance of Gradients for Detecting Distributional Shifts in the Wild*. 2021. arXiv: 2110.00218 [cs.LG].

[35] Matthew Cook, Alina Zare and Paul Gader. *Outlier Detection through Null Space Analysis of Neural Networks*. 2020. arXiv: 2007.01263 [cs.LG].

[36] Ibrahima Ndiour, Nilesh Ahuja and Omesh Tickoo. *Out-Of-Distribution Detection With Subspace Techniques And Probabilistic Modeling Of Features*. 2020. arXiv: 2012.04250 [cs.LG].

[37] Eoin Delaney, Derek Greene and Mark T. Keane. *Uncertainty Estimation and Out-of-Distribution Detection for Counterfactual Explanations: Pitfalls and Solutions*. 2021. arXiv: 2107.09734 [cs.LG]. URL: https://arxiv.org/abs/2107.09734.

[38] John Sipple and Abdou Youssef. 'A General-Purpose Method for Applying Explainable AI for Anomaly Detection'. In: *Foundations of Intelligent Systems*. Ed. by Michelangelo Ceci et al. Cham: Springer International Publishing, 2022, pp. 162–174. ISBN: 978-3-031-16564-1.

[39] AJ Tallón-Ballesteros and C Chen. 'Explainable AI: Using Shapley Value to Explain Complex Anomaly Detection ML-Based Systems'. In: *Machine Learning and Artificial Intelligence: Proceedings of MLIS 2020* 332 (2020), p. 152.

[40] Osvaldo Arreche et al. 'E-XAI: Evaluating Black-Box Explainable AI Frameworks for Network Intrusion Detection'. In: *IEEE Access* 12 (2024), pp. 23954–23988. DOI: 10.1109/ACCESS.2024.3365140.

[41] Basim Mahbooba et al. 'Explainable Artificial Intelligence (XAI) to Enhance Trust Management in Intrusion Detection Systems Using Decision Tree Model'. In: *Complexity* 2021.1 (2021), p. 6634811. DOI: https://doi.org/10.1155/2021/6634811. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/6634811. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/6634811.

[42] Erzhena Tcydenova et al. 'Detection of adversarial attacks in AI-based intrusion detection systems using explainable AI'. In: *Human-Centric Comput Inform Sci* 11 (2021).

[43] Tahmina Zebin, Shahadate Rezvy and Yuan Luo. 'An Explainable AI-Based Intrusion Detection System for DNS Over HTTPS (DoH) Attacks'. In: *IEEE Transactions on Information Forensics and Security* 17 (2022), pp. 2339–2349. DOI: 10.1109/TIFS.2022.3183390.

[44] Aitor Martinez-Seras, Javier Del Ser and Pablo Garcia-Bringas. 'Can Post-hoc Explanations Effectively Detect Out-of-Distribution Samples?' In: *2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2022, pp. 1–9. DOI: 10.1109/FUZZ-IEEE55066.2022.9882726.

[45] Alex Krizhevsky. 'Learning Multiple Layers of Features from Tiny Images'. In: (2009), pp. 32–33. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[46] Samira Pouyanfar et al. 'A survey on deep learning: Algorithms, techniques, and applications'. In: *ACM computing surveys (CSUR)* 51.5 (2018), pp. 1–36.

[47] Michal F Kaminski et al. 'Quality indicators for colonoscopy and the risk of interval cancer'. In: *New England journal of medicine* 362.19 (2010), pp. 1795–1803.

[48] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[49] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[50] Walid Bousselham et al. *LeGrad: An Explainability Method for Vision Transformers via Feature Formation Sensitivity*. 2025. arXiv: 2404.03214 [cs.CV]. URL: https://arxiv.org/abs/2404.03214.

[51] Aditya Chattopadhay et al. 'Grad-CAM++: Generalized Gradient-Based Visual Explanations for Deep Convolutional Networks'. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Mar. 2018. DOI: 10.1109/wacv.2018.00097. URL: http://dx.doi.org/10.1109/WACV.2018.00097.

[52] Ruigang Fu et al. *Axiom-based Grad-CAM: Towards Accurate Visualization and Explanation of CNNs*. 2020. arXiv: 2008.02312 [cs.CV]. URL: https://arxiv.org/abs/2008.02312.

[53] Rachel Lea Draelos and Lawrence Carin. *Use HiResCAM instead of Grad-CAM for faithful explanations of convolutional neural networks*. 2021. arXiv: 2011.08891 [eess.IV]. URL: https://arxiv.org/abs/2011.08891.

[54] Giang Nguyen et al. 'Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey'. In: *Artificial Intelligence Review* 52 (2019), pp. 77–124.

[55] Jingyang Zhang et al. 'OpenOOD v1.5: Enhanced Benchmark for Out-of-Distribution Detection'. In: *arXiv preprint arXiv:2306.09301* (2023).

[56] Narine Kokhlikyan et al. *Captum: A unified and generic model interpretability library for PyTorch*. 2020. arXiv: 2009.07896 [cs.LG].

[57] Jacob Gildenblat and contributors. *PyTorch library for CAM methods*. https://github.com/jacobgil/pytorch-grad-cam. 2021.

[58] Jeremy Howard. *Imagewoof*. URL: https://github.com/fastai/imagenette/.

[59] Aditya Khosla et al. 'Novel Dataset for Fine-Grained Image Categorization'. In: *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*. Colorado Springs, CO, June 2011.

[60] Bolei Zhou et al. 'Places: A 10 Million Image Database for Scene Recognition'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.6 (2018), pp. 1452–1464. DOI: 10.1109/TPAMI.2017.2723009.