REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

Département d'Electronique

المدرسة الوطنية متعددة التقنيات

Ecole Nationale Polytechnique

Thesis of Final Graduation Project

for the obtention of the State Engineer Diploma in Electronics

Enhancing deep learning based classifiers using out of distribution data detection

Presented by :

Abdelghani HALIMI

Ahmed HADJADJ

Under the direction of Mr. Sid-Ahmed BERRANI, Associate Professor

Presented and publicly defended on (30/06/2022)

Composition of the jury :

President	Mr. Mourad ADNANE	Professor	ENP
Supervisor	Mr. Sid-Ahmed BERRANI	Associate Professor	ENP
Examiner	Mrs. Nesrine BOUADJENEK	Assistant Professor	ENP

ENP 2022

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

Département d'Electronique

المدرسة الوطنية متعددة التقنيات

Ecole Nationale Polytechnique

Thesis of Final Graduation Project

for the obtention of the State Engineer Diploma in Electronics

Enhancing deep learning based classifiers using out of distribution data detection

Presented by :

Abdelghani HALIMI

Ahmed HADJADJ

Under the direction of Mr. Sid-Ahmed BERRANI, Associate Professor

Presented and publicly defended on (30/06/2022)

Composition of the jury :

President	Mr. Mourad ADNANE	Professor	ENP
Supervisor	Mr. Sid-Ahmed BERRANI	Associate Professor	ENP
Examiner	Mrs. Nesrine BOUADJENEK	Assistant Professor	ENP

ENP 2022

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

Département d'Electronique

المدرسة الوطنية متعددة التقنيات

Ecole Nationale Polytechnique

Mémoire de projet de fin d'études

pour l'obtention du diplôme d'Ingénieur d'Etat en Electronique

Amélioration des classificateurs d'apprentissage profond à l'aide de la détection des données hors distribution

Réalisé par :

Abdelghani HALIMI

Ahmed HADJADJ

Sous la direction de M. Sid-Ahmed BERRANI, Docteur

Présenté et soutenu publiquement le (30/06/2022)

Composition du Jury :

Président	M. Mourad ADNANE	Professeur	ENP
Promoteur	M. Sid-Ahmed BERRANI	Docteur	ENP
Examinatrice	Mme Nesrine BOUADJENEK	Docteur	ENP

ENP 2022

ملخص

في هذا العمل، تم تجرِبة ثلاث طرق للكشف عن بيانات خارج التوزيع، تقييمها ومقارنتها على العديد من المعايير الشائعة (مجموعات بيانات الصور الطبيعية المختلفة)، وكذلك على مجموعة بيانات ImageNet-O وهي مجموعة بيانات جديدة تم إنشاؤها للمساعدة في البحث في اكتشاف OOD لنماذج ImageNet. في هذه الأطروحة، نقوم أيضًا بالتحقيق في تأثير حجم البيانات المُعلمّة على أداء اكتشاف OOD، لأننا استخدمنا ثلاث مجموعات بيانات مختلفة داخل التوزيع (CIFAR-10 و CIFAR-100 و ImageNet-1K), وأظهرنا أن الأداء ينخفض بسرعة مع زيادة عدد البيانات المُعلمّة. اختتمنا باقتراح طريقة تتجاوز الأساليب الثلاث السابقة في أداء الكشف وعن طريق إنشاء واجهة مستخدم ويب لاختبار طريقة اكتشاف OOD الخاصة بنا.

كلمات مفتاحية : شبكة عصبية, كشف عن بيانات خارج التوزيع, بيانات الصور, تقييم بالمقارنة, تصنيف, طريقة مقترحة, واجهة مستخدم ويب.

Résumé

Dans ce travail, trois méthodes de détection des hors distribution sont mises en œuvre, évaluées et comparées sur plusieurs repères communs (différents ensembles de données d'images naturelles) ainsi que sur le jeu de données ImageNet-O, un nouveau jeu de données qui a été créé pour aider la recherche dans la détection des hors distribution pour les modèles ImageNet. Dans ce projet, nous étudions également l'effet du nombre de classes sur les performances de détection des hors distribution, pour cela nous avons utilisé trois jeux de données différents (CIFAR-10, CIFAR-100 et ImageNet-1K), et nous avons constaté que les performances se dégradent rapidement lorsque le nombre de classes augmente. Nous avons également proposé une méthode qui surpasse les trois méthodes précédentes en termes de performances de détection et nous avons créé une interface utilisateur web permettant de tester cette méthode de détection.

Mots clés : Réseau neuronal, détection de la hors-distribution, image, évaluation comparative, classification, méthode proposée, interface web

Abstract

In this work, three out-of distribution detection methods are implemented, evaluated and compared on several common benchmarks (different natural image datasets), as well as on the ImageNet-O dataset, a novel dataset that has been created to aid research in OOD detection for ImageNet models. In this thesis, we also investigate the effect of label space size on the OOD detection performance, for that we used three different in-distribution datasets (CIFAR-10, CIFAR-100 and ImageNet-1K), and we showed that the performance degrades rapidly as the number of in-distribution classes increases. We concluded by proposing a method that surpasses the three previous methods in detection performances and by creating a web user interface to test out our OOD detection method.

Keywords : Neural network, out-of-distribution detection, image data, comparative evaluation, classification, proposed method, web interface

Dedication

“

To my dearest parents

To my grandparents

To my brothers: Abderrahim, Abdelghafour and Abdessalem

To my partner: Ahmed

To my supervisor: Mr. Sid-Ahmed Berrani

To my friends at ENP

To all those who believed in me

To all those who are dear to me

I dedicate this modest work,

Thank you.

”

- *Abdelghani*

Dedication

“

I’d like to dedicate my work to my family and my friends. I want to express my gratitude to my caring parents, they were number one supporter during my journey, my siblings Massouada, Abdelhalim, Faiza, Omar and Khadidja have never left my side and I’m more than grateful, and not to forget my nephews Amine, Rayan, Soumia, Assia, Bilel, Maria, Hadjer and Mohamed, I dedicate this work for all my lovely family.

I also dedicate this work and give special thanks to my friends Abderrahmane, Amayas, Imad and Zinou for their support during my journey, to my colleagues and the friendships I had during my time in this school thank you, another dedication I want to make is for the “VIC” family whom I had such an unforgettable experience with.

Last but not least, I want to dedicate this work to my partner Abdelghani HALIMI without him this thesis won’t be complete, thank you for all the lab work we did together, I wish you best of luck for the rest of your journey, I want to thank my supervisor Mr. Sid-Ahmed Berrani who I had a real pleasure working with him.

”

- *Ahmed*

Acknowledgements

This work was carried out at the National Polytechnic School.

First of all, we would like to thank particularly and very warmly our project's supervisor, Dr. Sid-Ahmed BERRANI, with whom we had so much pleasure to work with and who made us benefit from his experience in the field of artificial intelligence. We are very sensitive to the honor that Prof. Mourad ADNANE gives us by chairing this jury. We would like to express our sincere thanks to him.

We express our deepest gratitude to Dr. Nesrine BOUADJENEK for having accepted to be agreeing to be member of the jury and to judge this modest work.

May all our professors who have contributed to our training find here our deepest gratitude.

We address our most sincere thanks to all those who contributed, from near or far, to the achievement of this work.

Finally, we wish to dedicate this thesis to our parents. Nothing would have been possible without their support, trust and generosity.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ANN	*Artificial Neural Network*
AUPR	*Area Under the Precision-Recall curve*
AUROC	*Area Under the Receiving-Operating Characteristic curve*
BG	*Background*
CIFAR	*Canadian Institute For Advanced Research*
CNN	*Convolutional Neural Networks*
EBM	*Energy Based Models*
FC	*Fully Connected*
FN	*False Negative*
FP	*False Positive*
FPR	*False Positive Rate*
GPU	*Graphics Processing Unit*
ID	*In-Distribution*
LSUN	*Large-Scale Scene UNderstanding*
ML	*Machine Learning*
MSE	*Mean Squared Error*
MSP	*Maximum Softmax Probability*
NN	*Neural Network*

ODIN	*Out-Of-Distribution detector for neural networks*
OOD	*Out-Of-Distribution*
PR	*Precision-Recall*
ReLU	*Rectified Linear Unit*
ResNet	*Residual Network*
SUN	*Scene UNderstanding*
TIC	*Tiny ImageNet (Crop)*
TIR	*Tiny ImageNet (Resize)*
TN	*True Negative*
TP	*True Positive*
TPR	*True Positive Rate*
UI	*User Interface*

Introduction

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

During development, NNs models are typically tested and assured as being reasonably accurate on a set of input samples which are analogous to the data used to train them. There is, however, serious risk when the model encounters data that is different from the limited domain of expected inputs: A type of data often referred to as being Out-Of-Distribution (OOD). Research has shown that when machine learning (ML) models like NNs encounter OOD samples as input, they often make an output prediction with high-confidence, despite being poorly equipped to handle such data [1, 2]. This is problematic because it may cause a complex system that relies on these predictions to perform an inappropriate and dangerous action.

Critically, a model may thus return overconfident predictions on OOD data, which may give serious consequences in high stake applications, such as cybersecurity, autonomous vehicles, infrastructure monitoring, disease outbreak detection or medical diagnosis, just to name a few.

Recent ML literature aims to rectify this problem by detecting when model input data falls far outside of the data-generating distribution of previously encountered samples where performance has been empirically measured as being good. This body of literature is collectively known as OOD detection. The main idea behind these methods is that when an OOD input is given to the ML model, the OOD detection system can detect it as being a sample in which the model has a high risk of making a poor prediction. It would then, for example, be possible to let a human take over from an automated system that is not trained to handle that input.

Problem Definition and Purpose

OOD detection is applicable to many types of data, including text, images, videos, time series, speech and more. However, in this work, we will focus on image data. In order to identify the most relevant and promising methods, this thesis critically evaluates and analyzes the literature to find positive and negative aspects of each method, such as scalability, complexity, performance, interpretability and required architectural changes. Of special interest are methods called Post-Hoc methods that do not require architectural changes to an existing classifier, avoiding retraining of the model, which make them have the advantage of being easy to use without modifying the training procedure and objective.

The purpose of this work is to study existing OOD methods, choose and focus on three of them. These are then implemented, evaluated and compared in order to assess how well they handle anomalous data. We also propose an enhanced new method that mainly consists in fusing the three previously considered methods. The new resulting technique is evaluated as well. In the specific context of image classification (the application considered in this work), OOD detection methods are typically tested using samples from a class the model has not learned. For this, we have performed tests using several common benchmarks (different natural image datasets) as well as a novel dataset that has been created to aid research in OOD detection for ImageNet models. As for the in-distribution data we consider three in-distribution datasets: CIFAR-10, CIFAR-100 and ImageNet-1K with a number of classes of 10, 100 and 1000 classes respectively, this lets us investigate the effect of label space size on the OOD detection performance.

Structure of the thesis

This thesis is organized into four chapters:

Chapter 1 “**Background**” covers the necessary background for understanding the base classifiers and OOD detection methods studied in this thesis.

Chapter 2 “**Principles and concepts of out-of-distribution detection**” provides a theoretical description of the OOD detection task, some related terminology and concepts, an overview of related works, some potential improvements as well as the used evaluation metrics.

Chapter 3 “**Experimental Setup**” describes the used datasets and implementation details of the base classifiers and OOD methods,

Chapter 4 “**Results and Analysis**” presents and analyzes the experimental findings, discusses possible causes for these results.

Finally, the thesis is concluded by summarizing the main findings and presenting possible avenues for future research.

Chapter 1

Background

1.1 Introduction

This chapter summarizes the concepts required for understanding the following chapters. It begins with a summary of deep neural network concepts, followed by an overview of the popular convolutional networks architectures, and in particular the DenseNet and EfficientNet architectures that have been used in our experiments.

1.2 Machine Learning

Machine learning is a subset of artificial intelligence that includes mathematical techniques for estimating functions that will map some input data to a desired output. These mathematical techniques are typically used when the function $f(x')$ is difficult to explicitly specify. They can be classified into two broad categories, based on the kind of data used to estimate the function parameters. Unsupervised methods are only provided with a dataset of possible inputs x' and are often used to find structure or patterns that are intrinsic to the dataset [3]. In contrast, in supervised models, parameters are estimated using a dataset with examples of both the inputs x' and corresponding desired outputs y' of f [3]. Supervised learning, which is the focus of this thesis, can be broken down by task into two sub-types: classification or regression.

1.2.1 Classification

Classification is the task of mapping inputs x' to outputs $y' \in 1, ..., C$ where C is the number of categorical classes. Classification tasks require that y is a categorical or nominal value in a finite set of possible classes [3]. If $C = 2$, the task is called a binary classification problem, while for any $C > 2$, the task is called multi-class classification. If it is the case that labels y' are not mutually exclusive and the model should potentially output more than one category for each input, the task is called a multi-label classification problem. It is most common for a classification model to provide a single discrete categorical output for each input and it should be assumed this is the case unless otherwise specified. Examples of classification problems include predicting the name of an object most prominently featured in an image or predicting if the content of an email is spam or a fishing attempt.

1.2.2 Regression

Regression is the task of predicting a function that can map an input x' to a continuous output $y' \in \mathbb{R}$ [3]. Examples of regression tasks include predicting the selling price of a home given details about its size, etc., and predicting the location of a robot in a room given the information from various on-board sensors.

1.3 Feedforward Neural Networks

Feedforward neural networks are a kind of machine learning model that approximate a function f through a series of intermediate computational layers. These layers are parameterized by values θ such that desired outputs y' are approximated by the function $\hat{y} = f(x'; \theta)$. Feedforward models are some of the simplest architectures of deep neural networks in that data x' passes through one or more intermediate layers of the model to the output layer with no cyclical connections between the output and any of the intermediate layers [2].

These kinds of models are referred to as networks because of how they are composed of a series of relatively simple functions. The network f is typically composed of simpler layer functions f^1, f^2, and f^3 as follows: $f(x') = f^3(f^2(f^1(x')))$. The depth of the network refers to how many of these layer functions are composed in this way. The outermost layer (f^3 in the example) is referred to as the output layer while functions f^1 and f^2 are referred to as hidden layers [2]. The desired output of the hidden layers is not explicitly specified by the training dataset. The structure of a simple feedforward neural network can be seen in Figure 1.1.

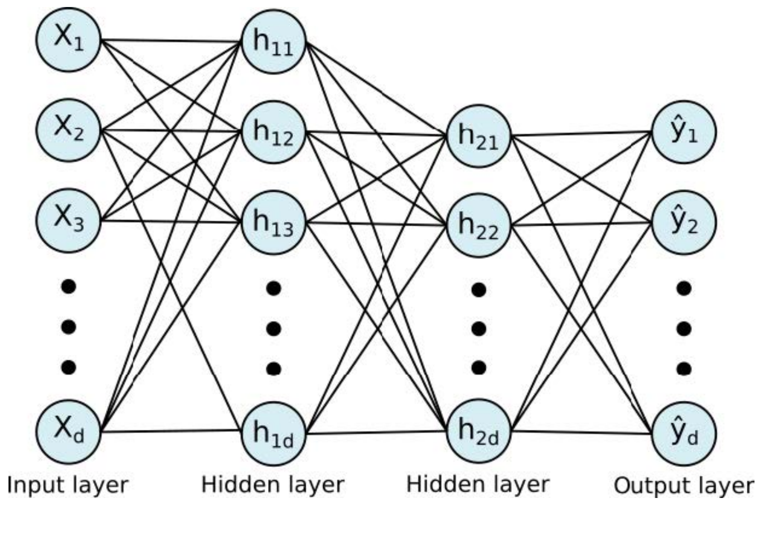


Figure 1.1: Example of a feedforward neural network.

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

1.3.1 Activation Functions

Activation functions refer to a set of functions used for computing the hidden layer output values [2]. These functions take a linear combination of the input data via layer weights W, add a bias term b, and then output values of a non-linear transformation of that computation. That is, a typical activation neuron has the form $h = f_{act}(W^T x' + b)$. There are several commonly used activation functions f_{act} with different properties based on the range of outputs, the magnitude of gradients at different points, and other characteristics. It is possible to use linear activation functions, however, these only allow function approximation by simple linear regression. Non-linear activation functions allow for learning more complex functions and allow for neural networks to behave as universal function approximators [2]. Some commonly used activation functions are the rectified linear unit (ReLU) and Softmax functions.

ReLU

The ReLU function is a piece-wise linear activation function, which retains many of the properties of linear functions that make them easy to optimize with gradient-based methods [2]. ReLU is defined by the equation 1.1 and a visualization of the output is shown in the Figure 1.2. ReLU activation functions are the default recommended non-linear activation function for use within neural networks, although others also exist [2]. While the gradient of the ReLU function is undefined at 0, software implementations typically define it heuristically as either the left or right derivative which allows the function to work well in practice [3].

$$ReLU(z) = max(0; z) \tag{1.1}$$

where max(a,b) denotes the maximum operator between a and b.

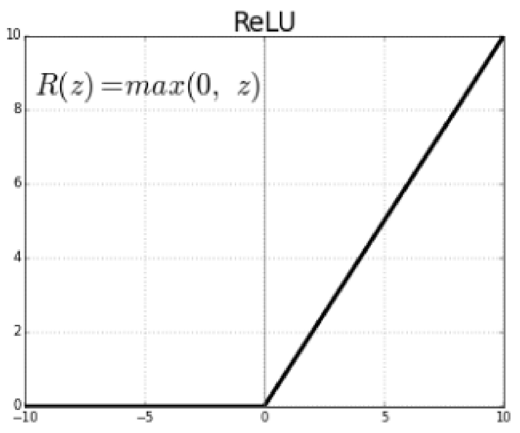


Figure 1.2: Output of the ReLU activation function.

Softmax

The softmax activation function is often used as the final activation function in multi-class classification neural networks. It is used to normalize the outputs of a neural network (logits) to the interval (0; 1) and change them such that the sum of all outputs is equal to 1. This is useful in practice when it is desirable to have outputs that can be interpreted as probability scores for each of the possible output classes as illustrated in Figure 1.3.

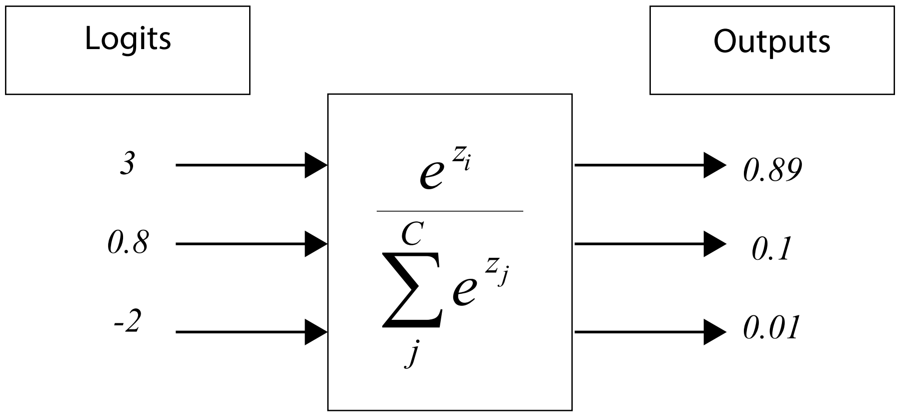


Figure 1.3: Illustration of softmax in a multi-class classification problem.

The formula for the softmax function is given by:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{1.2}$$

where e denotes the exponential function.

1.3.2 Loss Functions

The loss function of a neural network is a function that indicates how well the model is performing at the task it is supposed to accomplish. In supervised learning problems, this is often some kind of distance measure between the training data labels y' and the value predicted by the model $\hat{y} = f(x')$. These loss functions, which are interchangeably called objective functions, are meant to be used such that the value they output can be maximized or minimized using gradient-based optimization methods [2]. The choice of loss function depends on the objective of the model. The loss functions used within this thesis are covered in more detail below.

Cross-Entropy Loss

Cross-entropy loss is used to measure the performance of a model with categorical output probabilities in the range (0; 1) [4]. In the context of classification neural networks, the cross-entropy loss is applied to the output of a final softmax activation layer $\hat{y}$ and a one-hot representation of the ground truth label y'. That is, a one-hot vector of class label y' will be a vector of zeros of size C, the number of classes, with a value of 1 at the index corresponding to the ground truth label of data x'. Given these representations, cross-entropy is then defined as:

$$L(\hat{y}, y') = -\sum_{k=1}^{C} y'_k log(\hat{y}_k) \tag{1.3}$$

Cross-entropy loss is minimized when the predicted distribution $\hat{y}$ matches the distribution of the ground truth, and increases as the predicted class diverges from the true label [4]. When there are only two possible ground truth classes (a binary classification), it is more common to use the simpler binary cross-entropy formula, which encodes the label y' as a single 0 or 1, rather than a one-hot vector, as follows:

$$L(\hat{y}, y') = -y' log(\hat{y}) + (1 - y') log(1 - \hat{y}) \tag{1.4}$$

In this last case, the output layer is composed of only one neuron with a Sigmoid activation function.

L2 Loss

L2 or mean squared error (MSE) loss is used to minimize the difference between the ground truth and model-predicted value. The L2 loss function is defined by the Equation 1.5. L2 loss is used in regression models and increases with the square of the distance between predicted output ground truth value. As a result, the L2 loss is sensitive to outliers in the data that may skew the model due to their effect on drastically increasing the loss value [5].

$$L2(\hat{y}, y') = \sum_{i=0}^{n} (y'_i - \hat{y}_i)^2 \tag{1.5}$$

L1 Loss

L1 loss is used in similar situations as mean squared error loss. L1 loss has an advantage over L2 in that the effect of outliers is reduced, which may result in a better model when there is noise present in the data [5]. It is defined as:

$$L1(\hat{y}, y') = \sum_{i=0}^{n} |y'_i - \hat{y}_i| \tag{1.6}$$

1.3.3 Gradient Descent

The most common method to train a neural network is by using gradient descent. The way this works is by defining a loss function $l(\theta)$ that expresses how well the weights biases θ allow the network to fit the training data. A higher loss means that the network is bad and makes a lot of errors while a low loss generally means that the network performs well. We can then train the network by adjusting the network parameters in a way that reduces the loss.

Formally the update rule in the gradient descent algorithm is defined like this:

$$\theta_{t+1} = \theta_t - \epsilon \nabla l(\theta) \tag{1.7}$$

Here you take the gradient ∇ of the loss function l which tells you in which direction you have to move through parameter space to increase the loss. Then you go in the opposite direction $-\nabla l$ (in which the loss decreases) and move by a distance dependent on the learning rate ϵ . This works very well in most cases and is the foundation of much of modern deep learning.

The learning rate is a typically small fraction that optimizes the model in smaller steps such that loss minima are not over-shot by the update step. This prevents the learning algorithm from oscillating around a local minimum rather than decreasing the error more steadily [2].

1.3.4 Regularization

Regularization is a set of methods for improving the generalization capabilities of a machine learning model without drastically increasing the training error [2]. Without regularization, gradient descent optimizations will usually perform very well for the training data but poorly on any new data that was unseen during training, despite containing the same classes of image content, text, etc. When this undesirable scenario occurs, the model is said to have overfit the training data. The kind of regularization used within this thesis is briefly described below:

Dropout

Dropout is a method that effectively allows training an ensemble of models simultaneously without the expensive computation or memory overhead of doing so the traditional way (i.e. by optimizing multiple models initialized with different random parameters) [2]. Dropout does this by randomly turning off some fraction of specified model layer outputs for a batch of data used in a single training pass of the model. The computation of the loss function and the gradient descent update operation are performed using the network outputs that were computed using the subset of model parameters. This random deactivation of layer neurons (i.e. a single weighted parameter value in a layer) encourages the model to learn an internal representation with a level of redundancy. This is useful for learning more robust features from the dataset and often improves the generalization capability of the model.

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

1.3.5 Optimization

Optimization is the process of adjusting hyper-parameters in order to minimize the cost function by using one of the optimization techniques. It is important to minimize the cost function because it describes the discrepancy between the true value of the estimated parameter and what the model has predicted.The kinds of regularization used within this thesis are briefly described below:

Stochastic Gradient Descent (SGD)

SGD stands for Stochastic Gradient Descent. In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy or less random manner, but the problem arises when our datasets get really huge.

This problem is solved by Stochastic Gradient Descent. In *SGD*, it uses only a single sample to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

Since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm. But that does not matter all that much because the path taken by the algorithm does not matter, as long as we reach the minima and with significantly shorter training time.

Momentum

Momentum is essentially a small change to the *SGD* parameter update so that movement through the parameter space is averaged over multiple time steps. This is done by introducing a velocity component v. Momentum speeds up movement along directions of strong improvement (loss decrease) and also helps the network avoid local minima. It is intuitively related to the concept of momentum in physics. With momentum, the *SGD* update rule is changed to:

$$v_{t+1} = \mu v_t - \epsilon \nabla l(\theta) \tag{1.8}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{1.9}$$

Here v is the velocity and μ is the momentum parameter which controls how fast the velocity can change and how much the local gradient influences long term movement. At every time step the velocity is updated according to the local gradient and is then applied to the parameters.

Nesterov Momentum

Nesterov momentum is a simple change to normal momentum. Here the gradient term is not computed from the current position θ_t in parameter space but instead from a position $\theta_{intermediate} = \theta_t + \mu v_t$. This helps because while the gradient term always points in the right direction, the momentum term may not. If the momentum term points in the wrong direction or overshoots, the gradient can still go back and correct it in the same update step.

The revised parameter update rule is:

$$v_{t+1} = \mu v_t - \epsilon \nabla l(\theta + \mu v_t) \tag{1.10}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{1.11}$$

Learning rate decay

Learning rate decay is a technique for training modern neural networks. It starts training the network with a large learning rate and then slowly reducing (decaying) it until local minima is obtained. It is empirically observed to help both optimization and generalization.

When we have a constant learning rate, the steps taken by our algorithm while iterating towards the minima are so noisy that after certain iterations it seems wandering around the minima and do not actually converges. But when the learning rate is reducing over time, since the learning rate is large initially we still have relatively fast learning but as tending towards minima learning rate gets smaller and smaller, end up oscillating in a tighter region around minima rather than wandering far away from it.

1.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specialized kind of model for processing data with grid-like spatial correlations between the input features [2]. An example of data with this property is image data. CNNs form the basis for many state-of-art image classification neural networks such as DenseNet [6], Wide Residual Networks [7] and EfficientNet [8]. The difference between a simple feedforward neural network and a CNN is that a CNN makes use of at least one convolutional layer. These layers are a modification of the more simple linear vector multiplication layers (called fully-connected or dense layers) and account for relationships between adjacent data features.

1.4.1 Convolution operation

The convolution operation is most easily understood as a mathematical operation between some ordered input data matrix and a kernel matrix which slides over the input. The kernel matrix is used to compute a matrix dot product with the corresponding input data covered by the kernel function. The output of this operation is a single value in a convolved matrix output, commonly called the feature map. This definition is visualized and clarified by Figure 1.4. More formally, for multi-dimensional arrays of parameters as in machine learning, convolution is defined:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n) \tag{1.12}$$

Where S is the output convolved matrix, i and j are indices in the matrix, I is an input multidimensional array (such as an image), and K is the kernel matrix [2]. Here, K is assumed to be zero everywhere except for a finite set of points for which we store the values. In the context of machine learning, these values are shared weight parameters learned via gradient descent.

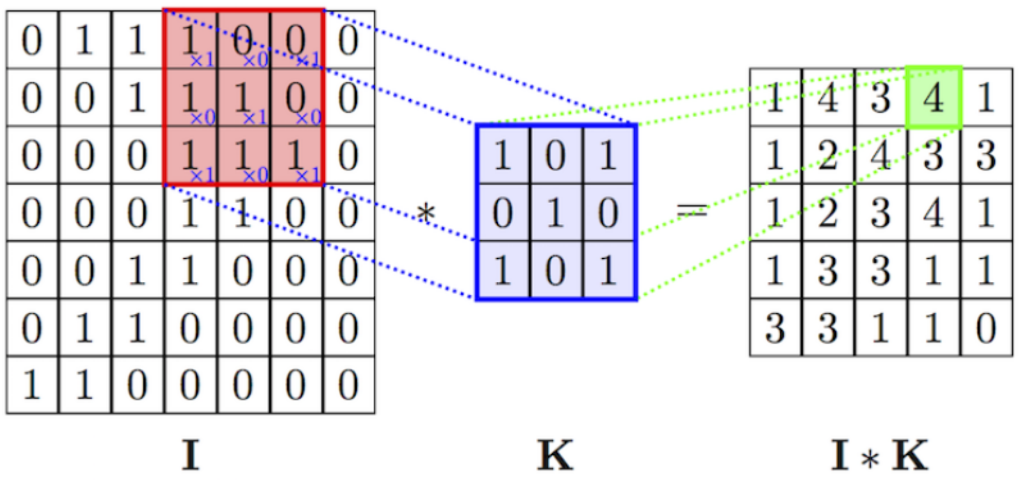


Figure 1.4: The convolution operation.

The dot product of kernel K is computed with image region I to produce a single value in the feature map output.

The shape of the non-zero elements of K in Figure 1.4 is described as the size of the kernel, with square 3x3 or 5x5 matrices being common in machine learning literature. The step size the kernel makes as it iterates over the input data is referred to as the stride. Finally, the input matrix I is sometimes modified before the convolution by padding the perimeter with zeros or some other value such that the output of the convolution operation is a matrix with some desired shape. Manipulating the kernel size, stride, and padding all affect the shape of the output feature map and also define how the spatially correlated input features are mapped to the hidden representation.

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

1.4.2 Pooling Layer

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

1.4.3 Normalization Layer

Normalization layers, as the name suggests, normalize the output of the previous layers. It is added in between the convolution and pooling layers, allowing every layer of the network to learn more independently and avoid overfitting the model.

However, normalization layers are not used in advanced architectures because they do not contribute much towards effective training.

1.4.4 Fully-Connected Layer

The Convolutional Layer, along with the Pooling Layer, forms a block in the Convolutional Neural Network. The number of such layers may be increased for capturing finer details depending upon the complexity of the task at the cost of more computational power.

Having been able to furnish important feature extraction, we are going to flatten the final feature representation and feed it to a regular fully-connected neural network for image classification purposes as illustrated in Figure 1.5.

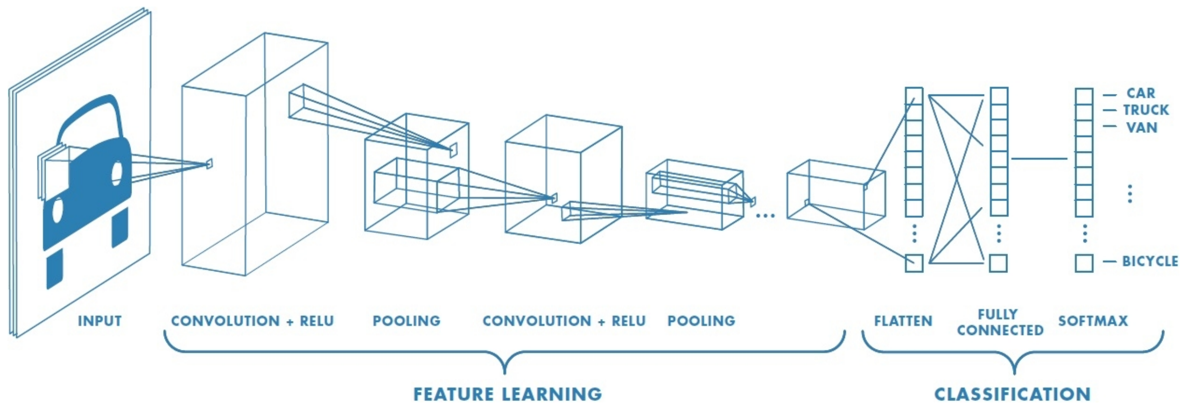


Figure 1.5: A simple CNN for classifying images.

1.5 Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision, given the vast compute and time resources required to develop neural network models on these problems. Transfer learning is so common that it is rare to train a model for an image or natural language processing-related tasks from scratch. Instead, researchers and data scientists prefer to start from a pre-trained model that already knows how to classify objects and has learned general features like edges, shapes in images. ImageNet and AlexNet are typical examples of models that have the basis of Transfer learning.

To achieve transfer learning, first we need to choose a pre-trained model a high correlation between source and knowledge dataset is required. The final output layer needs to be changed accordingly to the source. Freezing the starting layers from the pre-trained model is essential to make the model learn the basic features. One method of improving the performance is fine-tuning, fine-tuning involves unfreezing some part of the base model and training the entire model again on the whole dataset at a very low learning rate. The low learning rate will increase the performance of the model on the new dataset while preventing overfitting.

1.6 Popular Convolutional Networks Architectures

The following section present the details of some of the popular Convolutional Networks Architectures. In particular, the architectures of the neural network classifiers used in the experiments: DenseNet and EfficienNet.

1.6.1 LeNet

This was the first introduced convolutional neural network [9]. LeNet was trained on 2D images, grayscale images with a size of 32x32x1. The goal was to identify hand-written digits in bank cheques. It had two convolutional-pooling layer blocks followed by two fully connected layers for classification.

1.6.2 AlexNet

AlexNet [10], named after its creator Alex Krizhevsky, was trained on the ImageNet dataset with 15 million high-resolution images with 256x256x3.

ReLU activation function was used between convolution layers and pooling layers for the first time as well as the overlapping pooling with stride smaller than the window size. It had five convolutional-pooling layer blocks followed by three fully connected dense layers for classification.

1.6.3 VGGNet

VGGNet [11] came with a solution to improve performance rather than keep adding more dense layers in the model.

The key innovation came down to grouping layers into blocks that were repetitively used in the architecture because more layers of narrow convolutions were deemed more powerful than a smaller number of wider convolutions.

A VGG-block had a bunch of 3x3 convolutions padded by 1 to keep the size of output the same as that of input, followed by max pooling to half the resolution. The architecture had n number of VGG blocks followed by three fully connected dense layers.

1.6.4 GoogLeNet

This architecture [12] has Inception blocks that comprise 1x1, 3x3, 5x5 convolution layers followed by 3x3 max pooling with padding (to make the output of the same shape as the input) on the previous layer and concatenates their output.

It has 22 layers, none of which are fully connected layers. It requires a total of 4 million parameters which is still 12 times fewer parameters than previous architectures like AlexNet [10].

1.6.5 Residual Neural network (ResNet)

It was observed that with the network depth increasing, the accuracy gets saturated and eventually degrades. Therefore, data scientists proposed the ResNet [13] to rescue us from that scenario, and helps to resolve this problem.

Residual Network (ResNet) is one of the famous deep learning models ever introduced [13] and is one of the popular and most successful deep learning models so far.

Residual Blocks

The problem of training very deep networks has been relieved with the introduction of these Residual blocks and the ResNet model is made up of these blocks.

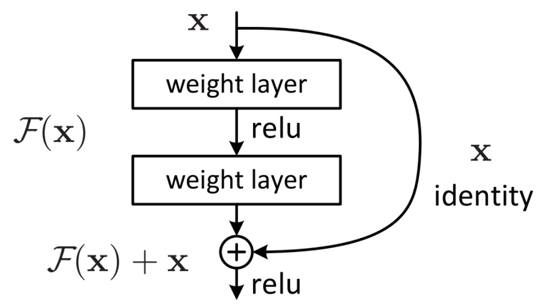


Figure 1.6: Residual learning: a building block.

In Figure 1.6, the very first thing we can notice is that there is a direct connection that skips some layers of the model. This connection is called 'skip connection' and is the heart of residual blocks. The output is not the same due to this skip connection. Without the skip connection, input x gets multiplied by the weights of the layer followed by adding a bias term.

Then comes the activation function, $f()$ and we get the output as $H(x)$.

$$H(x) = f(wx + b) or H(x) = f(x) \tag{1.13}$$

Now with the introduction of a new skip connection technique, the output is $H(x)$ is changed to

$$H(x) = f(x) + x \tag{1.14}$$

But the dimension of the input may be varying from that of the output which might happen with a convolutional layer or pooling layers. Hence, this problem can be handled with these two approaches:

• Zero is padded with the skip connection to increase its dimensions.

• 1×1 convolutional layers are added to the input to match the dimensions. In such a case, the output is:

$$H(x) = f(x) + w1.x \tag{1.15}$$

Here an additional parameter w1 is added whereas no additional parameter is added when using the first approach.

These skip connections technique in ResNet solves the problem of vanishing gradient in deep CNNs by allowing alternate shortcut path for the gradient to flow through (identity mapping). Also, the skip connection helps if any layer hurts the performance of architecture, then it will be skipped by regularization.

Architecture of ResNet

There is a 34-layer plain network in the architecture that is inspired by VGG-19 in which the shortcut connection or the skip connections are added. These skip connections or the residual blocks then convert the architecture into the residual network as shown in Figure 1.7:

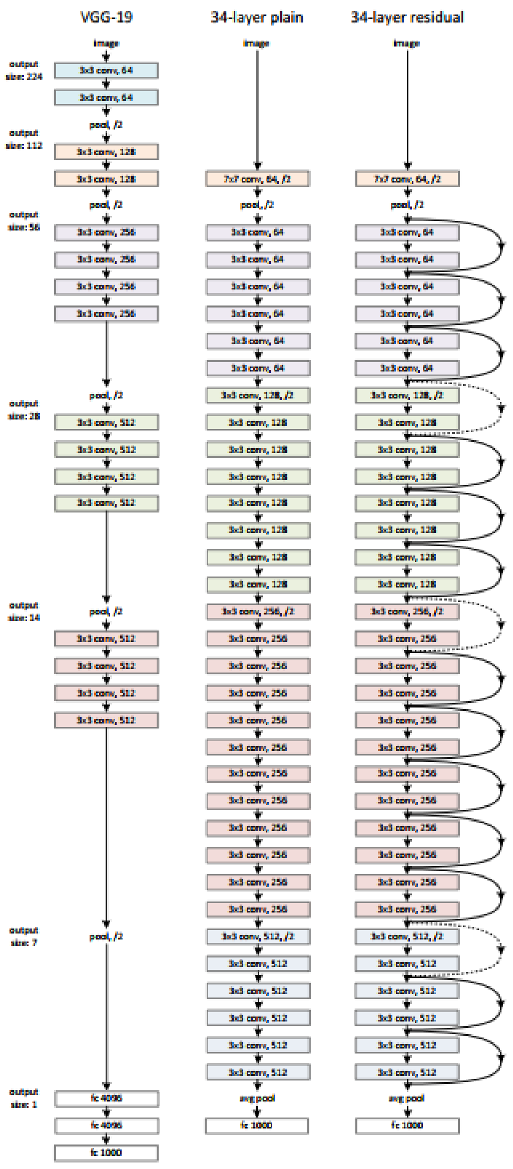


Figure 1.7: Example network architectures for ImageNet.

Left: the VGG-19 model (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions.

1.6.6 Dense Convolutional Network (DenseNet)

A limitation that ResNet has is that it require weeks for training, so in order to make a model more efficient and take advantage of shortcut connections that ResNet has, a novel architecture was proposed called DenseNet [6]. In this architecture, all the layers are connected directly with each other, like shown in Figure 1.8, to insure maximum information flow and feature reuse, allowing the network to achieve high-parameter efficiency.

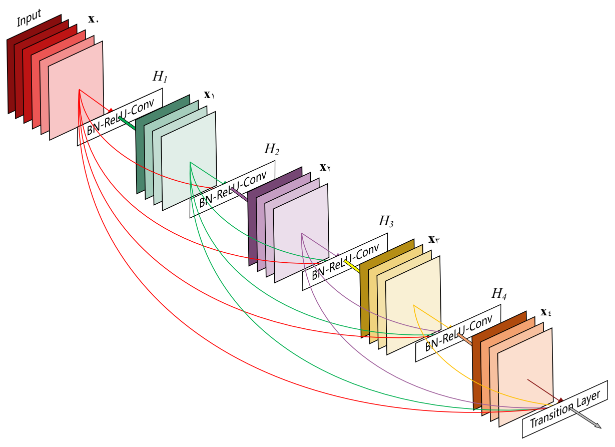


Figure 1.8: A 5-layer dense block with a growth rate of k = 4.

The growth rate k is the additional number of channels for each layer. Each layer takes all preceding feature-maps as input.

Architecture of DenseNets

DenseNets-B

In the basic DenseNets, For each composition layer, Pre-Activation Batch Norm (BN) and ReLU, then 3×3 Conv are done with output feature maps of k channels. As depicted in Figure 1.9, DenseNets-B are just regular DenseNets that take advantage of 1×1 convolution to reduce the feature maps size before the 3×3 convolution and improve computing efficiency. The B comes after the name Bottleneck layer.

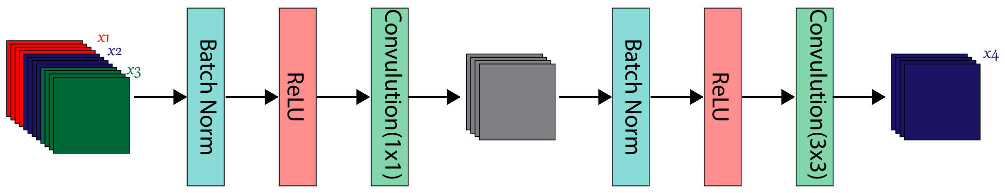


Figure 1.9: Architecture of DenseNet-B with 3×k input channel concatenated into k output channel.

DenseNets-BC

DenseNets-BC are another little incremental step to DenseNets-B, for the cases where we would like to reduce the number of output feature maps. The compression factor θ in the range $0 \leq \theta \leq 1$ determines this reduction. Instead of having m feature maps at a certain layer, we will have $\theta \times m$ output layer. If $\theta = 1$ DenseNets will remain the same as a DenseNets-B and the number of feature-maps across transition layers remains unchanged. Figure 1.10 shows a comparison of the parameter efficiency on CIFAR_10 between DenseNet variations and ResNet.

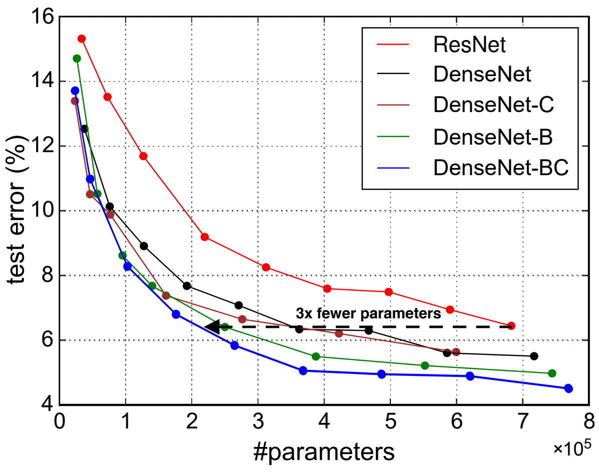


Figure 1.10: Comparison of the parameter efficiency on Cifar10 between DenseNet variations and ResNet.

1.6.7 EfficientNet

EfficientNet [8] model was proposed by Mingxing Tan and Quoc V. Le of Google Research, Brain team. These researchers studied the model scaling and identified that carefully balancing the depth, width, and resolution of the network can lead to better performance.

Based on this observation, they proposed a new scaling method that uniformly scales all dimensions of depth, width and resolution of the network. They used the neural architecture search to design a new baseline network and scaled it up to obtain a family of deep learning models, called EfficientNets, which achieve much better accuracy and efficiency as compared to the previous Convolutional Neural Networks.

Scaling

The researchers used the compound scaling method to scale the dimensions of the network. The applied grid search strategy to find the relationship between the different scaling dimensions of the baseline network under a fixed resource constraint. Using this strategy, the could find the appropriate scaling coefficients for each of the dimensions to be scaled-up. Using these coefficients, the baseline network was scaled by the desired size.

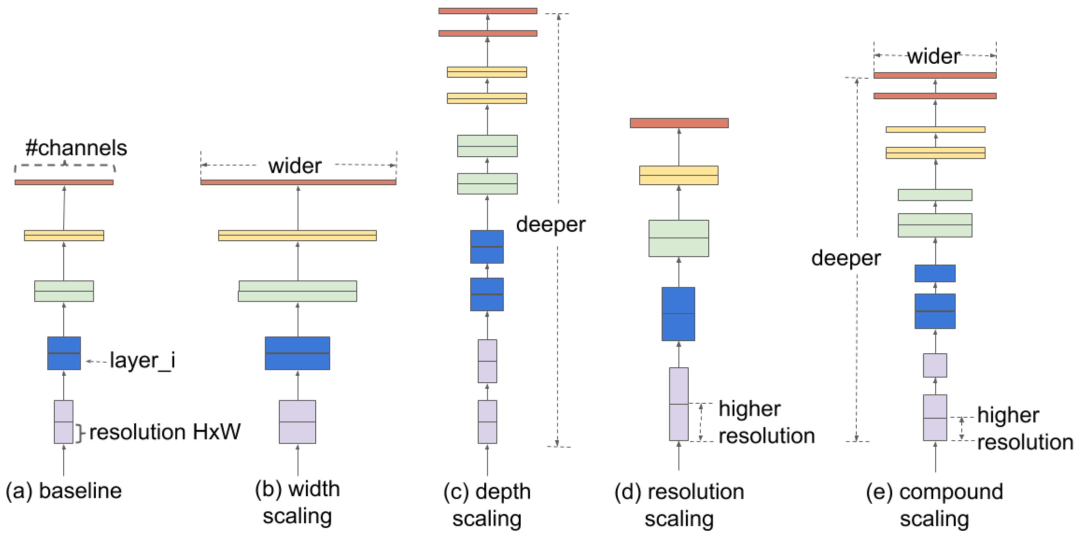


Figure 1.11: Comparison of different scaling methods.

Unlike conventional scaling methods (b)-(d) that arbitrary scale a single dimension of the network, EfficientNet compound scaling method uniformly scales up all dimensions in a principled way, as depicted in Figure 1.11. The researchers claimed in their work that this compound scaling method improved the model's accuracy and efficiency.

EfficientNet Architecture

The researchers first designed a baseline network by performing the neural architecture search, a technique for automating the design of neural networks. It optimizes both the accuracy and efficiency as measured on the floating-point operations per second (FLOPS) basis. This developed architecture uses the mobile inverted bottleneck convolution (MB-Conv). The researchers then scaled up this baseline network to obtain a family of deep learning models, called EfficientNets. Its architecture is given in Figure 1.12.

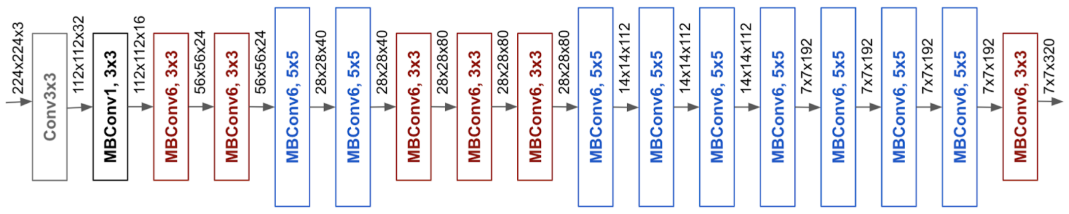


Figure 1.12: EfficientNet architecture.

They also presented a comparison of EfficientNet's performance with other powerful transfer learning models when worked on ImageNet dataset. As illustrated in Figure 1.13, it has been shown that the latest version of EfficientNet that is EfficientNet-B7 has the highest accuracy among all with less number of parameters. EfficientNet-B0 is the baseline network developed by AutoML MNAS, while Efficient-B1 to B7 are obtained by scaling up the baseline network. In particular, EfficientNet-B7 achieves 84.4% top-1 and 97.1% top-5 accuracy, while being 8.4x smaller than the best existing CNN.

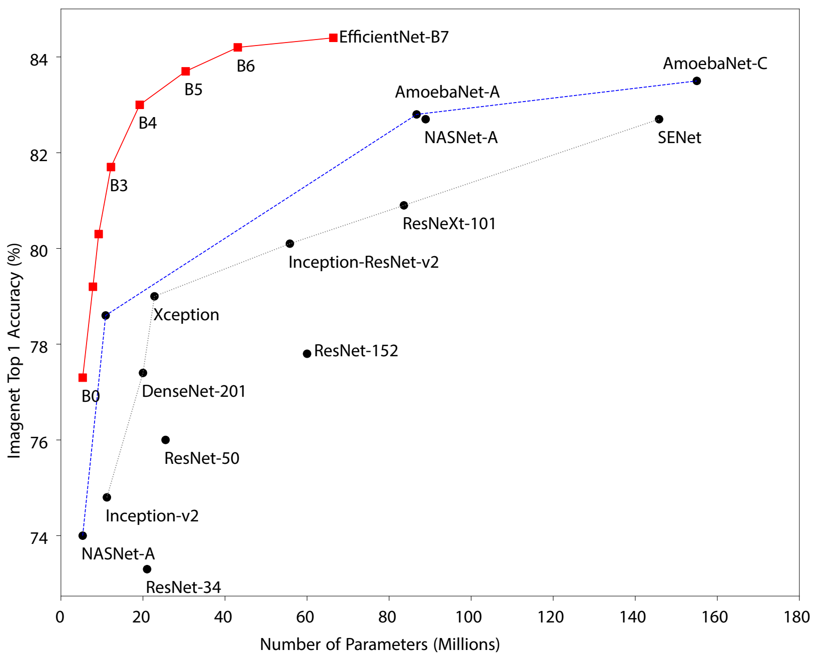


Figure 1.13: Model Size vs. Accuracy Comparison.

1.7 Conclusion

In this chapter, we have presented the basic concepts of deep learning that allow the reader to understand the upcoming chapters. In Chapter 2 and 3 for instance, the Softmax function is extensively used in an OOD detection method and both EfficientNet and DenseNet architectures have been used in our experiments, they have been trained using specific hyper-parameters that are described in the present chapter.

Chapter 2

Principles and concepts of out-of-distribution detection

2.1 Introduction

In this chapter, the out-of-distribution detection task is described. Following this, the evaluation metrics used to assess the performance of the OOD detection methods are presented. Finally, three detection techniques are explained and a new proposed method is introduced. This chapter presents the main contributions of our work.

2.2 Theoretical Definition: Out-Of-Distribution data

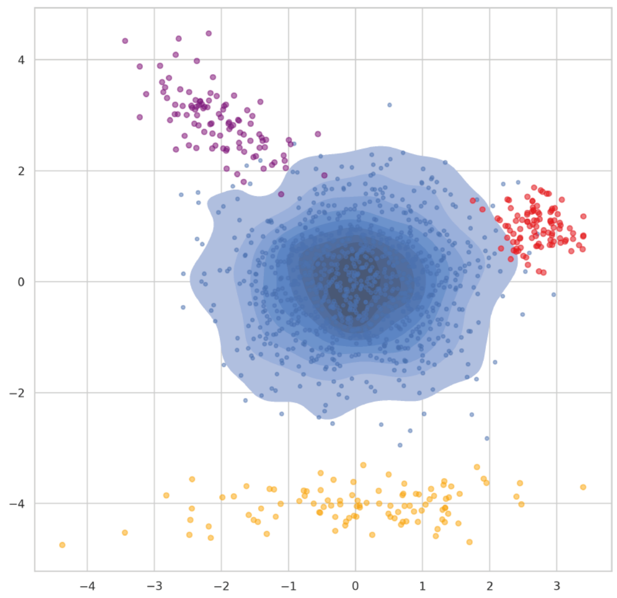


Figure 2.1: An illustration of an estimated standard normal distribution (blue) density, with samples from three other example distributions (red, purple, orange).

Let X $\in R^D$ denote the input space for some task. A concept of normality may then be defined as the distribution P^+ on X that captures a ground-truth law of normal behavior for the given task [14]. An anomaly may then be defined as a sample x $\in$ X that lies in a low probability region of P^+, as illustrated in Figure 2.1.

Given that there exists a corresponding probability density P^+(x), a set of anomalies may then also be defined as A = {x $\in$ X | P^+(x) $\leq \delta$ } for some threshold $\delta > 0$, chosen such that it ensures that the probability of each x $\in$ A is sufficiently small under P^+.

In Figure 2.1, the blue distribution represents an estimate of P^+, while the data from the other distributions are of low probability under P^+. This example illustrates one challenge with OOD detection, as some samples from OOD distributions may overlap with low-probability samples from the In-Distribution (ID) distribution.

2.3 Terminology

The problem of OOD detection is described using different terms in the literature, caused by the lack of a universally accepted definition [15]. Many of the works around OOD detection point to a similar problem, but with slight variations in problem description and underlying assumptions. However, the underlying problem is the same: To find a model that distinguishes samples originating from a different underlying distribution than that of the training data. The most common terms used in the literature are:

• **Anomalies** are samples originating from a data-generating distribution that is different from P^+. In the context of an airport security Xray application, weapons [16, 17] and illegal drugs are examples of anomalies that would be of interest. Given such an anomaly, the application would inform the operator and allow for manual inspection of the image.

• **OOD** samples are considered as samples that are rejected due to irrelevance to the application [14]. An example could be a dog breed image classifier to which an image of a cat is presented. The sample lies outside of the training distribution, and is thus anomalous, but the consequent action (rejection) differs from anomaly detection, where the anomalies are instead of special interest.

• **Outliers** are samples that lie within a low probability density region of P^+, but that are in fact generated by P^+. Examples of this could be instances that are highly infrequent, but also those that may be anomalous due to measurement errors or noise. An example of a rare instance could be a rare neurological disease observed by an assisting medical diagnosis model. The difficulty with outliers lies in determining whether an instance is in fact a true rare event or the result of some measurement noise due to some mechanical error. In Figure 2.1, the blue data points that lie outside of the shaded area can be considered outliers. As seen there, some purple and red data points have a higher probability than the blue (ID) data points, highlighting this challenge.

• **Novelties** are samples that originate from a new, previously unseen, mode or region of P^+. Imagine a developer creating a model for classifying car brands from images. When given an image of a previously unseen car brand, the model would ideally detect that this is a new class, and consequently update the model to be able to classify it correctly in the future. For example, if it was known that the distribution in Figure 2.1 was multimodal, then the samples from the red, purple and orange distributions would be considered novelties from three new modes of P^+. This scenario may be hard, as instances from different classes may be semantically close.

The slight differences in these terms are mainly important in specific applications and the actions taken at their respective detection. While an anomaly is often regarded as interesting and may provide valuable information for the user and the model, an outlier is often disregarded as a result of noise or measurement errors. Novelties may indicate that a model needs to be updated in order to facilitate modeling of the newly discovered class, which thereforth ought to be considered normal. Theoretically, the main objective of these approaches are the same: to determine whether a sample x lies within A or not.

A related field to anomaly and OOD detection is uncertainty estimation, which aims at quantifying how confident a model is in its prediction, i.e., if the model “knows what it doesn’t know” [18]. In uncertainty estimation, the goal is to quantify this uncertainty, which is usually approached from different theoretical angles. This uncertainty can be interpreted as an anomaly score and may be used as a means to discriminate between ID and OOD data. Although some research has suggested that the latter method may be misleading in specific cases [19], it has shown strong empirical results on image data [20, 21]. Another related field is active learning, in which a model receives feedback from a user on samples that deviate from the current notion of normality, and updates its notion of normality accordingly [22, 23].

This thesis focuses on methods that detect when a sample likely originates from outside the training distribution, and uses this information to reject samples that are deemed anomalous. Under the terminology presented, the problem statement is thus cast as **out-of-distribution detection**.

2.4 Data Properties

An important aspect to consider in OOD detection is how the OOD data differs from the In-Distribution (ID) data. As illustrated in Figure 2.2, semantic shift is the problem where the OOD data is drawn from a distribution that has no class overlap with the ID data, such as the example with dogs and cats, while non-semantic shift (covariate shift) considers distributions with the same classes in ID and OOD data. An example of non-semantic shift would be where ID data are camera pictures of dogs, while the OOD data are sketches or paintings of dogs.

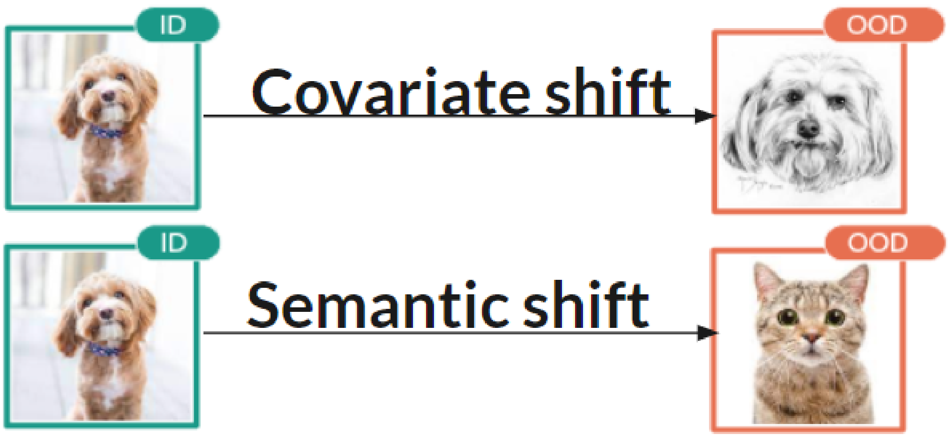


Figure 2.2: An illustration of covariate shift and semantic shift.

While most works in the current community interpret the keyword ”out-of-distribution” as ”out-of label/ semantic-distribution”, some OOD detection works also consider detecting covariate shifts, which claim that covariate shift usually leads to a significant drop in model performance and therefore needs to be identified and rejected. However, although detecting covariate shift is reasonable on some specific (usually high-risk) tasks, such as a medical diagnosis model that trained by one hospital should detect scans under distributional shift, research on this topic remains a controversial task [24]. Therefore, detecting semantic shift has been the mainstream of OOD detection tasks.

Furthermore, the amount of information that is available when training and testing a model for OOD detection must be considered. The unsupervised setting is arguably the most realistic scenario, in which no OOD samples are available during training or validation. In these scenarios, the models are trained in various ways to learn features of the normal data, and then use the learned features to discriminate incoming data that deviates significantly from the learned notion of normality. This discrimination can be done in many different ways and has been widely explored in the literature.

The unsupervised setting may require assumptions to be made on data, such as what type of anomalies to expect, and if any prior information is known about this data. It has previously been shown that unsupervised OOD detection models often fail to reach their required detection rates in applications, and are poor at detecting new unseen OOD samples, especially adversarial ones [25].

In the semi-supervised setting, there are labels available for ID data, and a smaller set of labels available for the OOD data. These samples may have been chosen and labelled by an expert and provide valuable prior information on the underlying distribution generating the anomalous samples, which can be incorporated during training. Work in the semi-supervised setting has shown that a small sample of anomalous data can be sufficiently informative and significantly improve OOD detection [25, 26].

In the supervised setting, labeled samples are available during training and testing, for both ID and OOD samples. However, because labeling samples is an expensive task, this is a very uncommon setting in OOD detection, and the problem can then instead be solved by regular classification techniques, although with special care taken to consider any potential class imbalance between anomalous and normal data[14].

2.5 Out-of-distribution detection

2.5.1 Background

With the observation that deep learning models can over-confidently classify samples from different semantic distributions, the field of out-of-distribution detection emerges, requiring models to reject label-shifted samples to guarantee reliability and safety. Formally, test samples in the OOD detection setting come from the distribution with semantic shift from ID.

2.5.2 OOD detection framework

Out-of-distribution detection is a super-category that includes multi-class novelty detection, and open-set recognition, as depicted in Figure 2.3:

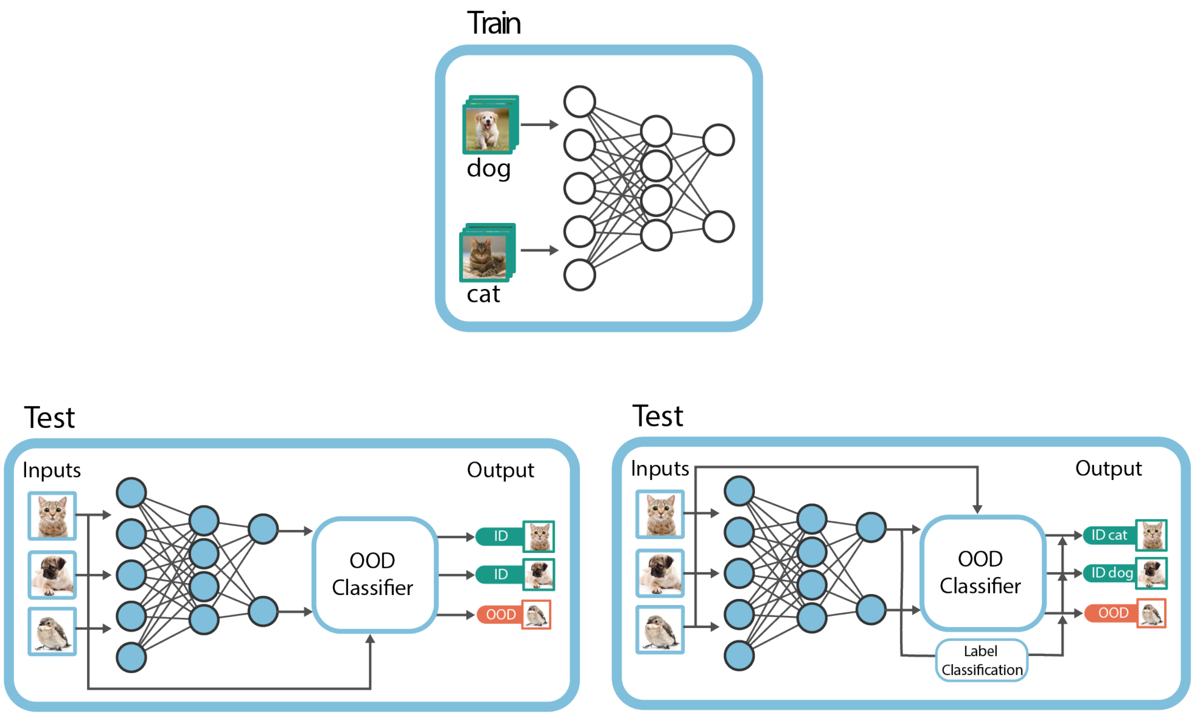


Figure 2.3: OOD detection framework with a model trained on cats and dogs. the left multi-class novelty detection and Open set recognition in the right.

• **In multi-class novelty detection**: ID images belong to multiple classes. Test images with semantic shift will be considered as OOD. So it's like a binary classification between ID and OOD only.

• **Open set recognition** is identical to multi-class novelty detection in the task of detection, with the only difference that open set recognition further requires In-Distribution (ID) classification label, so it's considered as an enhancement to the main classification.

Since our thesis is about enhancing deep learning based classifiers using OOD detection, we focus in this work on the "Open set recognition" setting.

2.5.3 Application and Benchmark

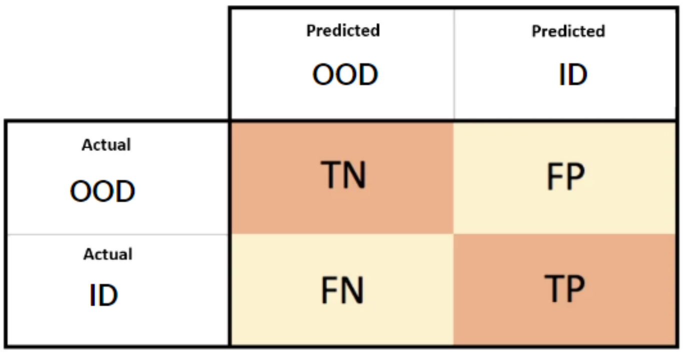
The application of OOD detection usually falls into safety-critical situations, such as autonomous driving. An example academic benchmark is to use CIFAR-10 as ID during training, and to distinguish CIFAR images from other datasets such as Places365, Textures, etc. it is worth pointing out that OOD datasets should NOT have any label overlapping with ID datasets when building the benchmark.

2.6 Evaluation Metrics

This section details several evaluation metrics used in the OOD detection. Unless specified, the range for each metric is between 0 and 1, and the arrows in each subsection title indicate whether a higher (↑) or lower (↓) value is desirable.

2.6.1 Confusion matrix

Confusion matrix is a very popular measure used while solving classification problems. It can be applied to binary classification as well as for multiclass classification problems. In our case of OOD detection (binary classification), the confusion matrix is shown in Figure 2.4:



	Predicted OOD	Predicted ID
Actual OOD	TN	FP
Actual ID	FN	TP

Figure 2.4: Confusion matrix in OOD detection task.

Definition of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) metrics. Columns indicate the predicted label and the rows the true labels. Confusion matrices represent counts from predicted and actual values. The output "TN" stands for True Negative which shows the number of negative examples classified accurately (OOD classified as OOD). Similarly, “TP” stands for True Positive which indicates the number of positive examples classified accurately (ID classified as ID). The term “FP” shows False Positive value, i.e., the number of actual negative examples classified as positive (OOD classified as ID); and “FN” means a False Negative value which is the number of actual positive examples classified as negative (ID classified as OOD).

2.6.2 Accuracy ↑ and Error rate ↓

The accuracy is defined as the fraction of correctly classified samples among all classified samples. The best accuracy is 1, whereas the worst is 0:

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{2.1}$$

Conversely, the error rate is defined as the fraction of misclassified samples among all classified samples. The best error rate is 0, whereas the worst is 1:

$$Error\ rate = \frac{FP + FN}{TP + FN + TN + FP} \tag{2.2}$$

The accuracy and error rate are complements of each other, meaning that we can always calculate one from the other. For example:

$$Accuracy + Error\ rate = 1 \tag{2.3}$$

2.6.3 Precision ↑

The precision of a model is defined as the fraction of samples classified as positive that were truly positive:

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

The higher the precision, the less the model is wrong on the positives.

In an OOD detection setting, this translates to what fraction of samples classified as ID were truly ID.

2.6.4 True Positive Rate ↑

The TPR, also known as the recall, is defined as the fraction of all truly positive samples correctly classified as positive:

$$TPR = Recall = \frac{TP}{TP + FN} \tag{2.5}$$

The higher the recall, the more positives the model finds.

In an OOD detection setting, this translates to the fraction of ID data that was correctly classified as ID.

2.6.5 False Positive Rate ↓

The FPR, also known as the fall-out, is defined as the fraction of negative samples that were incorrectly classified as positive samples:

$$FPR = \frac{FP}{FP + TN} \tag{2.6}$$

In an OOD detection setting, this translates to the fraction of missed anomalies.

2.6.6 True Negative Rate ↑

The TNR, also known as the specificity, is defined as the fraction of negative samples correctly classified as such:

$$TNR = \frac{TN}{FP + TN} = 1 - FPR \tag{2.7}$$

In an OOD detection setting, this translates to the fraction of samples classified as OOD that was in fact OOD.

2.6.7 False Negative Rate ↓

The FNR is defined as the fraction of positive samples that were incorrectly classified as negative:

$$FNR = \frac{FN}{FN + TP} = 1 - TPR \tag{2.8}$$

In an OOD setting, this translates to the fraction of ID samples incorrectly classified as OOD.

2.6.8 AUROC ↑

The area under the receiver-operating characteristic curve (AUROC) metric is a threshold independent performance measure for evaluating detection networks that output some real valued confidence or distance measure. AUROC is essentially a statistic that describes how well the positive and negative classes can be separated by the output confidence measure values without requiring that a threshold be determined beforehand. AUROC is calculated by sorting the output predictions by the confidence score and calculating the true positive rate (TPR) and false positive rate (FPR) for each possible threshold on that sorted data. The TPR is plotted against the FPR and the area under the curve that is formed by interpolation is taken to be the AUROC like shown in Figure 2.5.

With an equal number of positive and negative samples, the baseline AUROC of a random classifier will be 0.50, while a classifier that always gives positive samples a higher confidence value than negative samples will have an AUROC value of 1.0.

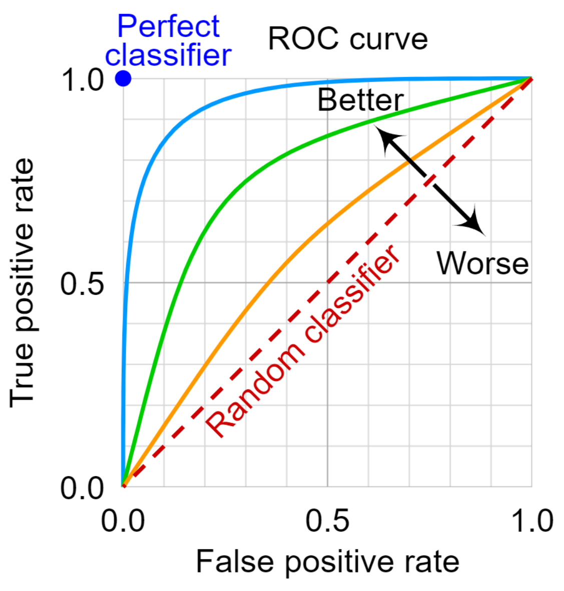


Figure 2.5: Example of a Receiver Operating Characteristic curve.

2.6.9 AUPR ↑

The Area under the Precision-Recall (AUPR) curve is the second most common threshold independent performance measure for evaluation detection networks with positive and negative samples. The AUPR, like AUROC, also sidesteps the issue of having to determine an appropriate threshold for a classifier but is more appropriate for datasets with imbalanced numbers of positive and negative samples. The metric is calculated similarly as AUROC, in that the data is sorted by the output confidence score and the precision and recall at each possible threshold $\delta \in [\delta_{min}, \delta_{max}]$ are calculated using the previous equations. These values are plotted and AUPR is calculated as the area under this curve, as depicted in Figure 2.6.

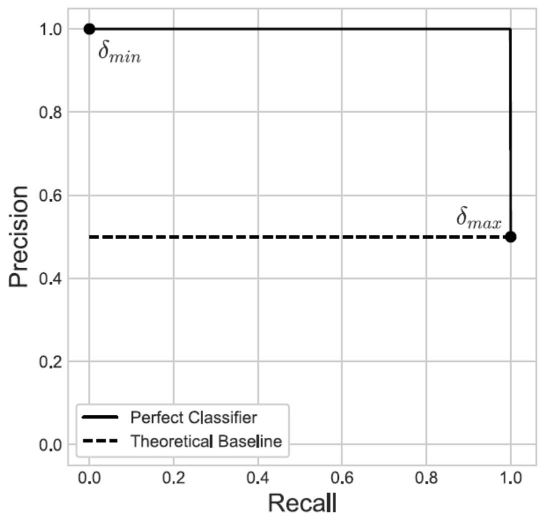


Figure 2.6: Example of a Precision-Recall curve.

A random classifier has an AUPR approximately equal to the precision, which tends to the fraction of ID samples in the dataset as the number of samples increases, while a perfect detector has an AUPR of 1.

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

2.6.10 FPR at 95% TPR ↓

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

2.6.11 Detection error ↓

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

$$P_e = \frac{n_+}{n}(1 - TPR) + \frac{n_-}{n}FPR \tag{2.9}$$

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

2.7 OOD detection: Existing techniques

Numerous methods have been proposed to avoid explicitly modelling the OOD class. The majority of these methods assume that the ID dataset has an unknown and unknowable underlying data generating distribution from which the ID samples are drawn. OOD detection methods model this ID data-generating distribution in some way and compare new samples to it. When a sample is determined to be unlikely as being generated by the ID data-generating distribution, it is labelled OOD. In this way, we can think of the task of OOD detection as a binary classification problem where the space of ID samples is modelled by using the training set. Samples that poorly fit the model of ID data are then classified as OOD. The way we model the data generating distribution and how this one-class bounding is performed on the ID set can be divided into three categories [24], as depicted in Figure 2.7:

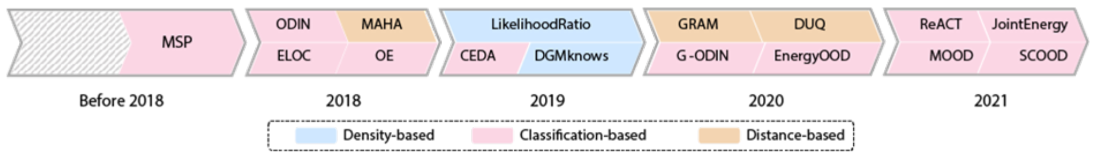


Figure 2.7: Timeline for representative methodologies of OOD detection. Different colors indicate different categories of methodologies.

2.7.1 Density-based Methods

Density-based methods in OOD detection explicitly model the in-distribution with some probabilistic models, and flag test data in low-density regions as OOD. When the ID contains multiple classes, class-conditional Gaussian distribution can explicitly model the in-distribution so that the OOD samples can be identified based on their likelihoods [27]. Flow-based methods [28, 29, 30, 31] can also be used for probabilistic modeling. While directly estimating the likelihood seems like a natural approach, some works [18, 32, 33] find that probabilistic models sometimes assign a higher likelihood for the OOD sample. Several works attempt to solve the problems using likelihood ratio [34]. [35] finds that the likelihood exhibits a strong bias towards the input complexity and proposes a likelihood ratio-based method to compensate the influence of input complexity. Recent methods turn to new scores such as likelihood regret [36] or an ensemble of multiple density models [32]. Overall, generative models can be prohibitively challenging to train and optimize, and the performance can often lag behind the classification-based approaches.

2.7.2 Distance-based Methods

The basic idea of distance-based methods is that the testing OOD samples should be relatively far away from the centroids or prototypes of in-distribution classes. ”MAHA” [27] uses the minimum Mahalanobis distance to all class centroids for detection. A subsequent work splits the images into foreground and background, and then calculates the Mahalanobis distance ratio between the two spaces [37]. Some works use cosine similarity between test sample features and class features to determine OOD samples [38, 39]. The one-dimensional subspace spanned by the first singular vector of the training features is shown to be more suitable for cosine similarity-based detection [40]. Moreover, other works leverage distances with radial basis function kernel ”Deterministic Uncertainty Quantification” (DUQ) [41] and Euclidean distance [42] between the input’s embedding and the class centroids.

2.7.3 Classification-based Methods

Research on OOD detection originated from a simple baseline, that is, using the maximum softmax probability (MSP) as the indicator score of whether in or out of distribution [1]. Early OOD detection methods focus on deriving improved OOD scores based on the output of neural networks.

Here are some classification-based methods as shown in Figure 2.7:

Maximum Softmax Probability (MSP) [1], Out-of-Distribution Detector for Neural Networks (ODIN) [43], Ensemble of self supervised Leave-Out Classifiers (ELOC) [44], Outlier Exposure (OE) [45], Confidence Enhancing Data Augmentation (CEDA) [46], Generalized ODIN (G-ODIN) [47], Energy-based Out-Of-Distribution Detection (EnergyOOD) [48], OOD detection with Rectified ACTivations (ReACT) [49], Multi-level Out-Of-Distribution detection (MOOD) [50], Semantically Coherent Out-Of-Distribution detection (SCOOD) [51] and OOD detection using joint energy (JointEnergy) [52].

2.8 Classification-based post-hoc methods

The post-hoc Detection is a classification-based Method that depends on the output of the neural network. These methods are easy to use since they rely on the output value only, without modifying the actual model or have a training procedure that could take hours.

This is the main advantage of these methods. This property can be used in OOD detection in real life problems where the cost of retraining a model is costly.
In this thesis, we will study some of Post-hoc methods, implement and evaluate them.

Post-hoc methods aim to amplify the ID/OOD separability, making them more separable by a threshold δ. An image x_i is classified as in-distribution if the method output score $S_y(x_i; f_i)$ is greater than the threshold and vice versa. The out-of-distribution detector can be described as shown in the Figure 2.8:

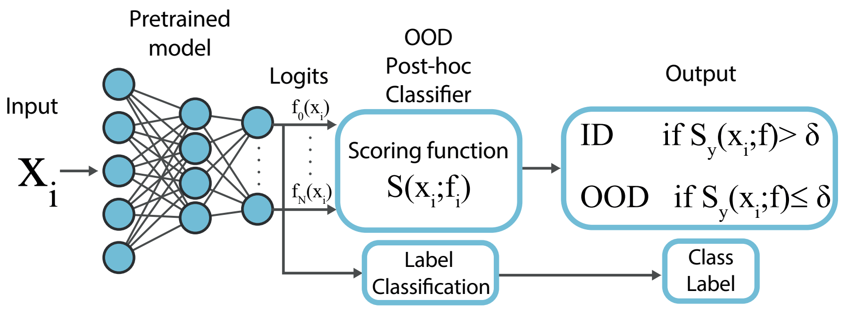


Figure 2.8: Post-hoc methods workflow.

The parameter δ is chosen so that the true positive rate (i.e., the fraction of in-distribution images correctly classified as in-distribution images) is 95%.

Studied post-hoc methods are presented in the following subsections.

2.8.1 Max-Softmax (MSP)

Maximum softmax probability (MSP) method [1] is called the baseline technique. It has been proposed in 2017 and is considered as the first OOD detection technique, yet it's very effective and commonly used. This technique can be applied to a neural network trained on N ID class. Softmax function turn logits $f_n(x)$ (which are the last output of classification neural networks that its values in range of $]-\infty;+\infty[$) into probabilities that could sum up to 1 using the Softmax function (described in Section 1.3.1).

We note the MSP function for a x_i input $S_{\hat{y}}(x_i)$:

$$S_{\hat{y}}(x_i) = \max(Softmax(f_i(x))) \tag{2.10}$$

The Softmax function has a behaviour where correctly classified examples tend to have higher confidence score, greater than erroneously classified and out-of-distribution examples, so as a result the maximum Softmax will be higher for the ID data and lower for the OOD as illustrated in Figure 2.9, and the distribution can be separable by a threshold δ, as shown in Figure 2.10.

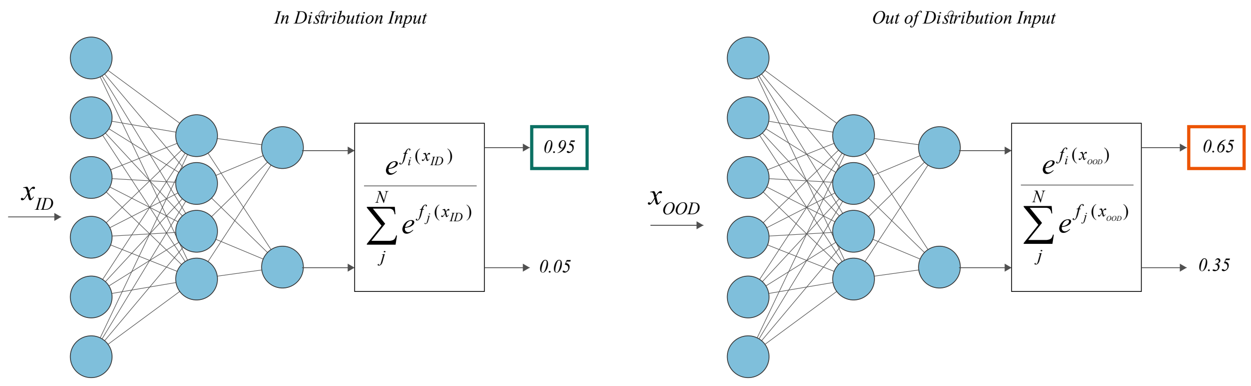

In Distribution Input
x_{ID}
$\frac{e^{f_i(x_{ID})}}{\sum_j^N e^{f_j(x_{ID})}}$
0.95
0.05
Out of Distribution Input
x_{OOD}
$\frac{e^{f_i(x_{OOD})}}{\sum_j^N e^{f_j(x_{OOD})}}$
0.65
0.35

Figure 2.9: Illustration of the Softmax function behaviour, where the ID data x_{ID} gives higher maximum probability than the OOD Input x_{OOD}.

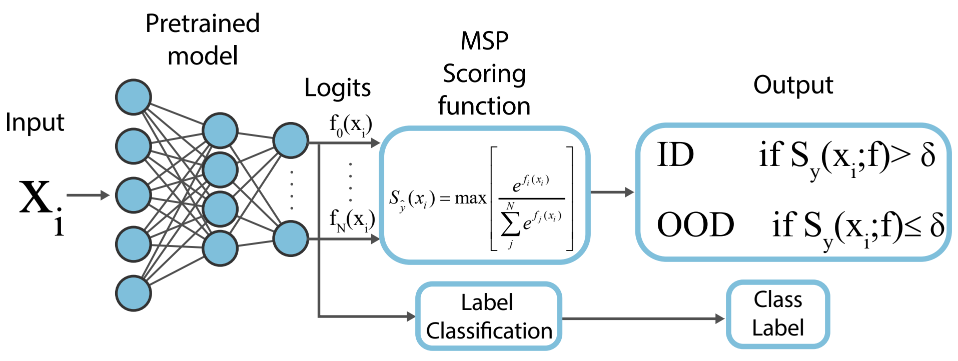

Pretrained model
Input
X_i
Logits
$f_0(x_i)$
$f_N(x_i)$
MSP Scoring function
$S_{\hat{y}}(x_i) = \max\left[\frac{e^{f_i(x_i)}}{\sum_j^N e^{f_j(x_i)}}\right]$
Output
ID if $S_y(x_i;f) > \delta$
OOD if $S_y(x_i;f) \leq \delta$
Label Classification
Class Label

Figure 2.10: MSP workflow.

As depicted in Figure 2.11, the parameter δ is chosen so that the true positive rate (i.e., the fraction of ID images correctly classified as ID images) is 95%.

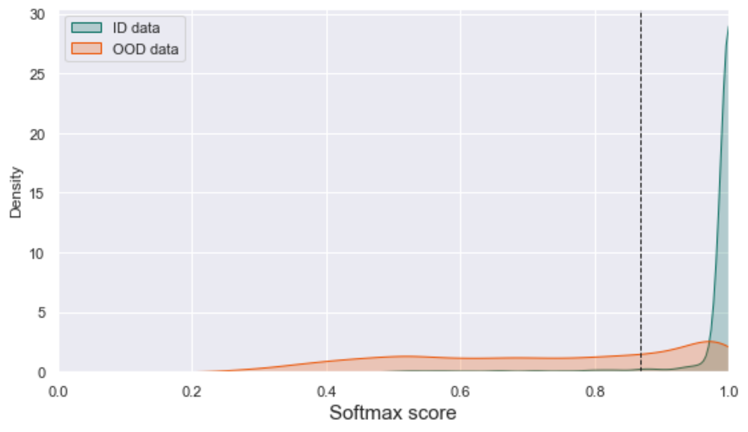


Figure 2.11: Illustration of the softmax score distribution for ID and OOD data and the classifier threshold.

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

ODIN [43] is a method used for improving the calibration of neural networks, for detecting out-of-distribution samples. The detector is built on two components: Temperature scaling and input preprocessing. These components are methods are both used in other purposes in data science and machine learning.

2.8.2.1 Temperature Scaling

For the temperature scaling, the previous MSP method is used and the logits are divided by a hyper-parameter called the temperature scaling factor T, where $S_{\hat{y}}(x_i; T)$ is the output of the MSP with temperature scaling:

$$T \in \mathbb{R}^+ : \qquad S_i(x_i; T) = \frac{e^{\frac{f_i(x)}{T}}}{\sum_j^N e^{\frac{f_j(x)}{T}}} \tag{2.11}$$

$$S_{\hat{y}}(x_i; T) = \max(S_i(x_i; T)) \tag{2.12}$$

The value for the temperature T is a hyper-parameter which the authors in the original paper optimize using a fraction of the test images [43].

Using temperature scaling, the Softmax scores will be more separable between in- and out-of-distribution images, making OOD detection more effective than the baseline method, as shown in Figure 2.12.
T is tuned from OOD data. We get higher output distributions with higher T values then we get spiky distribution where the probabilities start to converge. The ID data tend to

have higher distribution values. Above the value of T=1000 the gain is saturated and no improvement is noticed. In general T is fixed to T=1000. When we put T=1 we get the MSP method mentioned in the previous subsection.

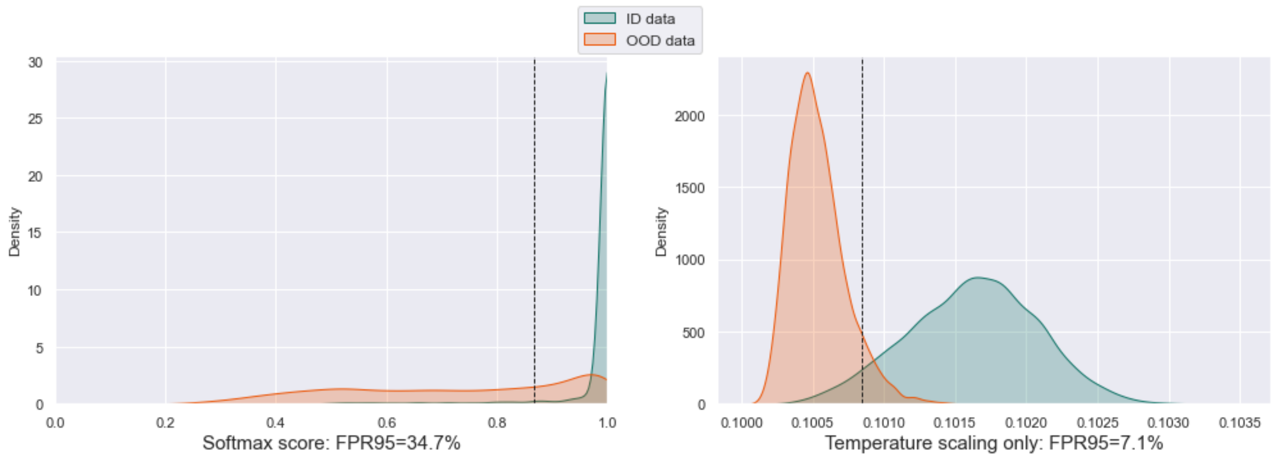


Figure 2.12: Comparison between the softmax score and temperature scaling for In and OOD datasets.

2.8.2.2 Input Preprocessing

Input Preprocessing consists of adding small controlled noise to the input data. Input is preprocessed by adding small perturbations:

$$\hat{x} = x - \varepsilon sign(-\nabla_x \log S_{\hat{y}}(x;T)) \tag{2.13}$$

$S_{\hat{y}}$ is the output of MSP of the clean image, and the parameter ε is called the "perturbation magnitude". This method relies on the idea of adversarial training [53], where small controlled noise is added to the input image to decrease the confidence score for the true label to force the neural network to make a wrong prediction, as illustrated in Figure 2.13. Adversarial attacks refer to finding adversarial examples for well-trained models.

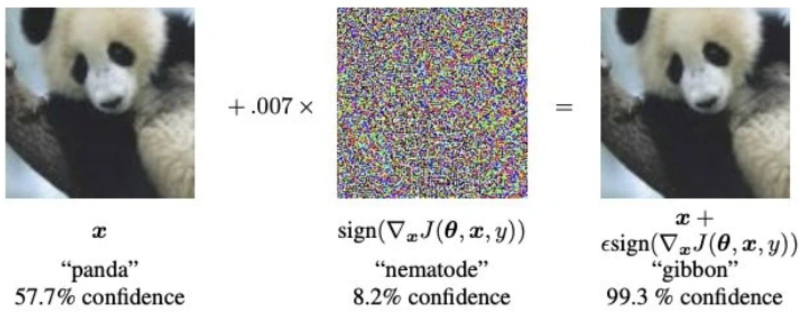

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 2.13: Illustration of an adversarial attack.

Where the noise magnitude is multiplied by the negative of the "fast gradient sign method" [53], this method generates adversarial examples. Small perturbations are designed to decrease the probability of the true class. Perturbations can be easily computed by back-propagating the gradient of the cross-entropy loss with reference to the input. The final output score is calculated using Equation 2.14, where $\widehat{x}_i$ is the preprocessed input and T is the previous temperature scaling factor and $S_{\hat{y}}(x_i; T)$ is the output of the MSP with temperature scaling and input preprocessing:

$$S_{\hat{y}}(x_i, T) = \max(S_i(\widehat{x}_i; T)) \tag{2.14}$$

Using the negative sign can have a stronger effect on the ID images than that on OOD images, making them more separable by a threshold δ like depicted in Figure 2.14.

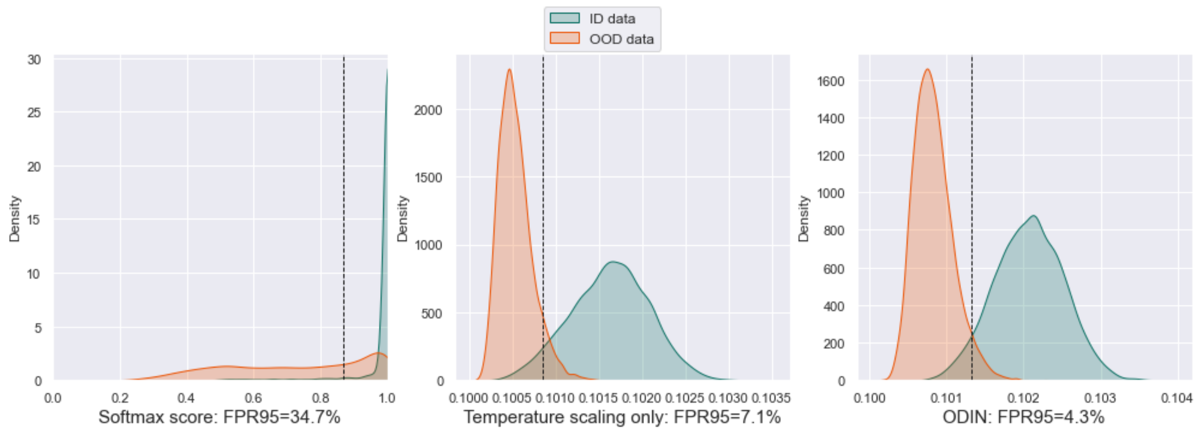


Figure 2.14: Comparison of MSP, temperature scaling and ODIN score distributions respectively for ID and OOD datasets.

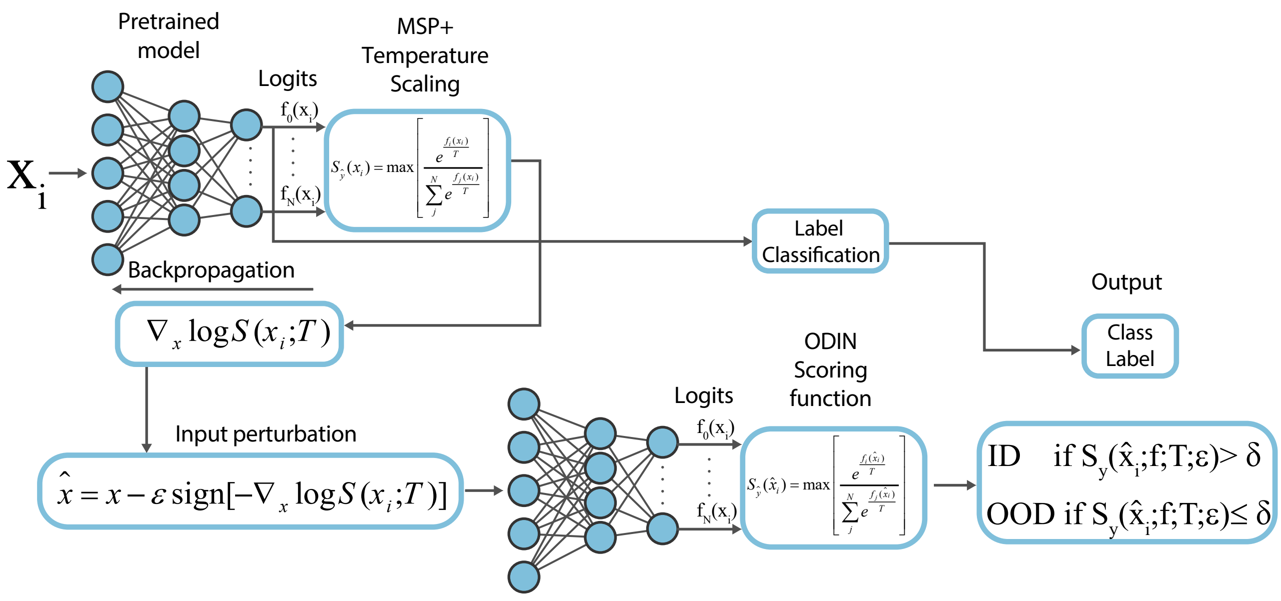

Pretrained model
Logits
MSP+ Temperature Scaling
X_i
$f_0(x_i)$
$f_N(x_i)$
$S_{\hat{y}}(x_i) = \max\left[\frac{e^{\frac{f_i(x_i)}{T}}}{\sum_j^N e^{\frac{f_j(x_i)}{T}}}\right]$
Label Classification
Backpropagation
$\nabla_x \log S(x_i;T)$
Output
Class Label
ODIN Scoring function
Logits
Input perturbation
$\hat{x} = x - \varepsilon\,\mathrm{sign}[-\nabla_x \log S(x_i;T)]$
$S_{\hat{y}}(\hat{x}_i) = \max\left[\frac{e^{\frac{f_i(\hat{x}_i)}{T}}}{\sum_j^N e^{\frac{f_j(\hat{x}_i)}{T}}}\right]$
ID if $S_y(\hat{x}_i;f;T;\varepsilon) > \delta$
OOD if $S_y(\hat{x}_i;f;T;\varepsilon) \leq \delta$

Figure 2.15: ODIN method workflow.

As illustrated in Figure 2.15, two passes of the network are required to do the input perturbation.

For the first pass, a pretrained model is fed with input data X_i, then the logits pass through the model's label classifier for later classification. Then the logits are scaled by T and passed through the MSP scoring function $S(x_i)$ that gives the temperature scaling confidence score. Then the gradient of the cross-entropy loss is calculated with respect to the input sample, which is done using the back-propagation algorithm. Then the preprocessed input $\widehat{x_i}$ is calculated. Next, the preprocessed image $\widehat{x_i}$ is passed into the neural network, Then through the scaled Softmax score $S(\widehat{x_i};T)$ where the ODIN confidence score is obtained. Lastly, the score $S(\widehat{x_i};T)$ is compared with a threshold δ and obtain the class label.

2.8.3 Energy score

The framework of energy-based models [48] is a unifying framework encompassing many different probabilistic and non-probabilistic learning models, including discriminative neural classifiers. The key point of energy-based models is capturing dependencies - described by the weights and biases in neural networks - between inputs x and the labels y, which minimize an energy function E(x, y), mapping x to a non-probabilistic scalar value, called the energy,

This framework enables highly flexible models, since it allows any function that maps data to a real value to be interpreted as an energy model. It was recently shown that every discriminative DNN classifier using a cross-entropy loss of $p(y/x)$ can be reinterpreted as an energy-based model for $p(x,y)$, for some sample x with labels y.
This induces that the logits produced by the DNN are energies. The energy-based OOD detection method considered in this work, called EBM, has been recently proposed [48] and uses the following energy function for OOD detection:

$$E(x;F) = -T.log\sum_{i=1}^{K} e^{\frac{S_i}{T}} \tag{2.15}$$

where $S_i = F(x)_i$ denotes the logit of class i produced by the classifier F, as depicted in Figure 2.16.

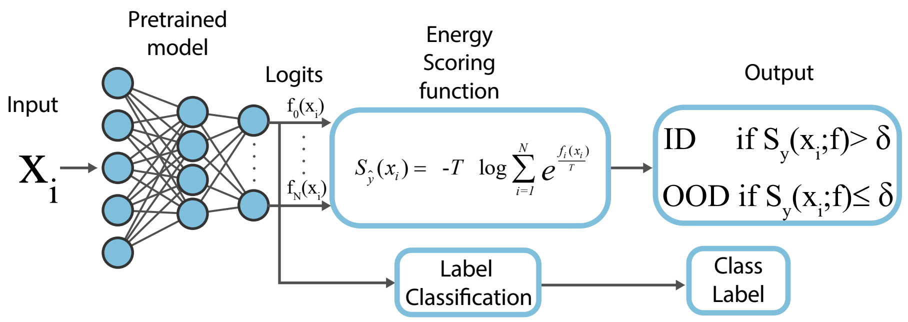

Pretrained model
Input
X_i
Logits
$f_0(x_i)$
$f_N(x_i)$
Energy Scoring function
$S_{\hat{y}}(x_i) = -T \log \sum_{i=1}^{N} e^{\frac{f_i(x_i)}{T}}$
Output
ID if $S_y(x_i;f) > \delta$
OOD if $S_y(x_i;f) \leq \delta$
Label Classification
Class Label

Figure 2.16: Energy score method workflow.

It can be shown that this energy score and the softmax score for some class k given data x, is related by:

$$ln \max_k (y = \frac{k}{x}) = E(x; F) + max_k F(x) \tag{2.16}$$

which shows that an energy-based detection score is, unlike the classic softmax score, theoretically aligned with the probability density of the inputs. It is therefore less susceptible to the overconfidence issue. Furthermore, in opposition to other energy-based models, this approach does not rely on full density estimation, which avoids the use of unstable generative models.

The energy function that is used as a scoring function assigns low energies to the ID data, and higher energies to the incorrect values OOD data, so we use negative energy scores to align with the convention that positive (in-distribution) samples have higher scores like depicted in Figure 2.17.

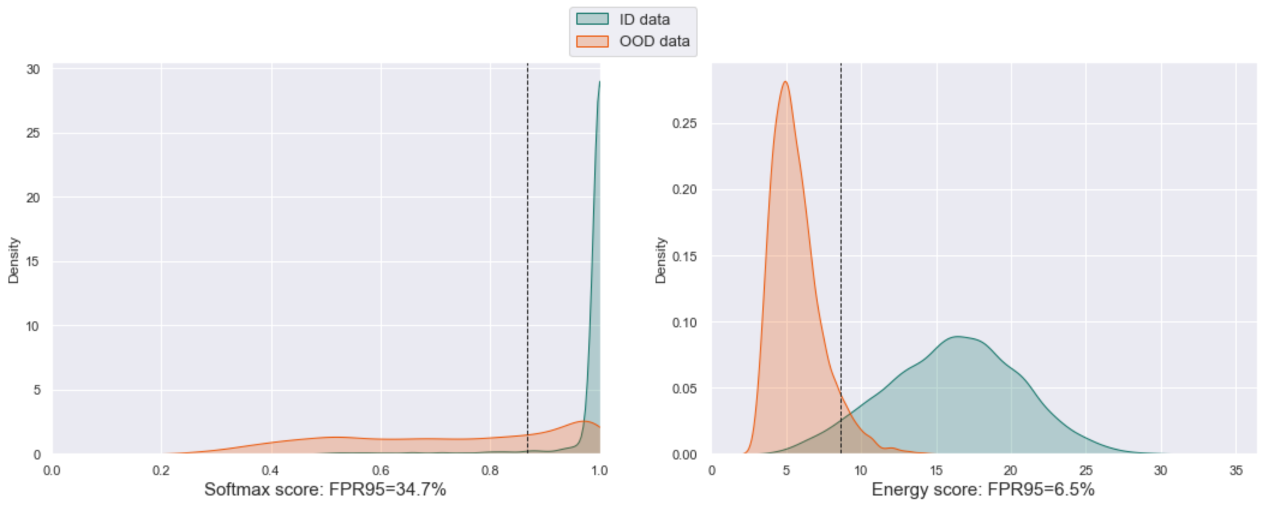


Figure 2.17: Illustration of the energy score for In and OOD datasets.

In figure 2.17 we used negative energy scores for to align with the convention that positive (in-distribution) samples have higher scores.

2.8.4 Proposed Method

In order to make use of the advantages of the previously presented methods, we propose a new method that consists of fusing the results of the three methods. The fusion relies on the *OR* operator, that requires a single method to detect a sample as OOD image to say it's OOD image, as shown in Figure 2.18. This can only enhance the detection performance or keep it the same as the best of the three performances. After a lot of experiments with different datasets, we have noticed that the previous methods are not as consistent as they should be. Every technique came as an improvement of the one that came before, so since our goal is to minimize the FPR error, the *OR* condition will classify an image as ID if only all the methods outputs were greater than their corresponding δ. In other words, it will classify an image as OOD if only one of the methods outputs was greater than their corresponding δ. As a result, more of the false positive results will be eliminated.

$$\begin{cases} S_{MSP}(x) \leq \delta || S_{ODIN}(x,T,\varepsilon) \leq \delta || S_{Energy}(x,T) \leq \delta : \;\; OOD \\ \qquad\qquad\qquad Else : \;\; ID \end{cases} \tag{2.17}$$

where || is the *OR* operator.

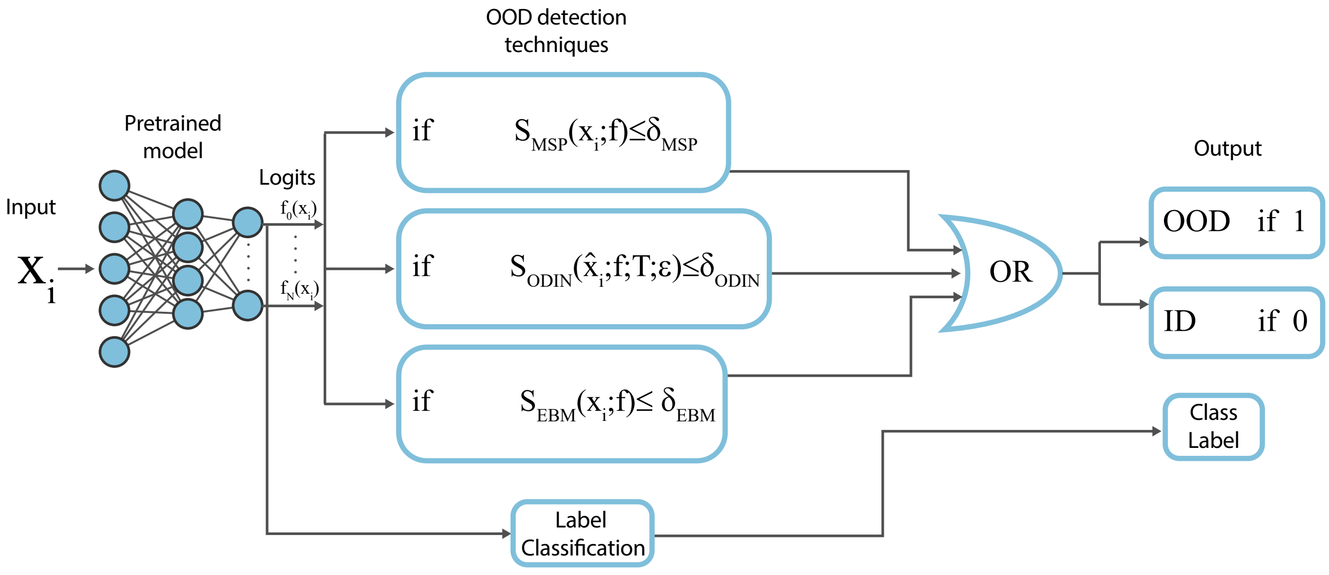

OOD detection techniques
Pretrained model
Input
Logits
X_i
$f_0(x_i)$
$f_N(x_i)$
if $S_{MSP}(x_i;f) \leq \delta_{MSP}$
if $S_{ODIN}(\hat{x}_i;f;T;\varepsilon) \leq \delta_{ODIN}$
if $S_{EBM}(x_i;f) \leq \delta_{EBM}$
OR
Output
OOD if 1
ID if 0
Class Label
Label Classification

Figure 2.18: Proposed method workflow.

2.9 Conclusion

After presenting OOD detection methods theory and the evaluation metrics used to assess their performance, we have discussed post-hoc techniques and how efficient they can be. We have also combined them in a single technique that we will evaluate later.

Chapter 3

Experimental Setup

3.1 Introduction

This chapter details the datasets that have been used in this thesis as well as the image classifiers and their training configurations. The implementation details of the three OOD detection methods studied are also described.

3.2 Datasets

Thirteen different datasets have been used in this work. Three datasets have been used as in-distribution datasets (CIFAR-10, CIFAR-100 and ImageNet) and the other 10 as OOD datasets for testing the OOD detection methods. The 13 datasets are described in more detail in the next subsections.

3.2.1 CIFAR-10

The CIFAR-10 [54] dataset consists of 60,000 32×32 color images. They represent 10 classes as shown in Table 3.1, with 6,000 images per class. There are 50,000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 1,000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5,000 images from each class.

Table 3.1: The list of the 10 classes in the CIFAR-10 dataset.

airplane	automobile	bird	cat	deer
dog	frog	horse	ship	truck

Here are 25 random images from this dataset presented in the Figure 3.1:

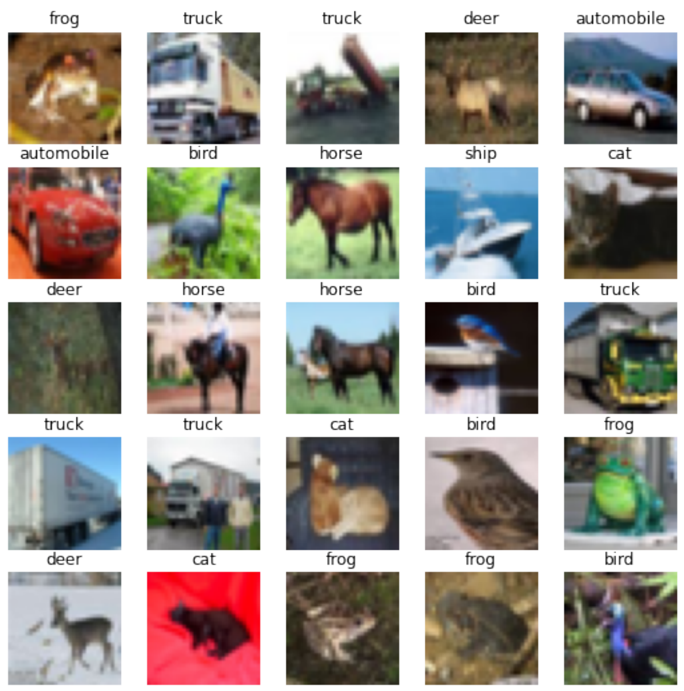


Figure 3.1: Random CIFAR-10 examples.

3.2.2 CIFAR-100

CIFAR-100 [54] similar to CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a ”fine” label (the class to which it belongs) and a ”coarse” label (the superclass to which it belongs). The list of classes is represented in Table 3.2.

Table 3.2: The list of classes in the CIFAR-100 dataset

apple’	Aquarium_fish’	Baby’	Bear’	Beaver’	Bed’	Bee’	Beetle’
mouse’	Mushroom’	Oak_tree’	Orange’	Orchid’	Otter’	Palm_tree’	Pear’
castle’	Caterpillar’	Cattle’	Chair’	Chimpanzee’	Clock’	Cloud’	Cockroach’
ray’	Road’	Rocket’	Rose’	Sea’	Seal’	Shark’	Shrew’
fox’	Girl’	Hamster’	House’	Kangaroo’	Keyboard’	Lamp’	Lawn_mower’
table’	Tank’	Telephone’	Television’	Tiger’	Tractor’	Train’	Trout’
Bicycle’	Bottle’	Bowl’	Boy’	Bridge’	Bus’	Butterfly’	Camel’
Pickup_truck’	Pine_tree’	Plain’	Plate’	Poppy’	Porcupine’	Possum’	Rabbit’
Couch’	Crab’	Crocodile’	Cup’	Dinosaur’	Dolphin’	Elephant’	Flatfish’
Skunk’	Skyscraper’	Snail’	Snake’	Spider’	Squirrel’	Streetcar’	Sunflower’
Leopard’	Lion’	Lizard’	Lobster’	Man’	Maple_tree’	Motorcycle’	Mountain’
Tulip’	Turtle’	Wardrobe’	Whale’	Willow_tree’	Wolf’	Woman’	Worm’
Can’	Raccoon’	Forest’	Sweet_pepper’				

A set of 25 random images from this dataset presented in Figure 3.2:

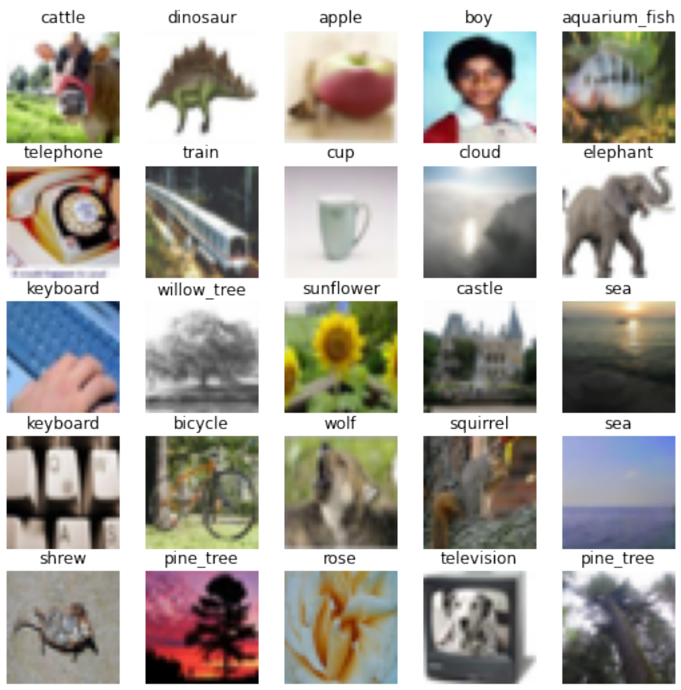


Figure 3.2: Random CIFAR-100 examples.

3.2.3 ImageNet

The ImageNet dataset [55] consists of three parts: training data, validation data, and image labels. The training data contains 1000 categories and 1.2 million images, packaged for easy downloading. The validation and test data are not contained in the ImageNet training data. Here are some random images from this dataset presented in Figure 3.3:

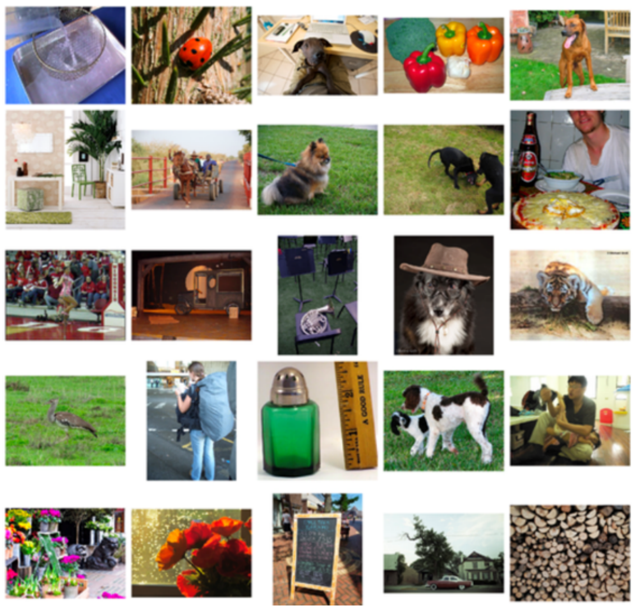


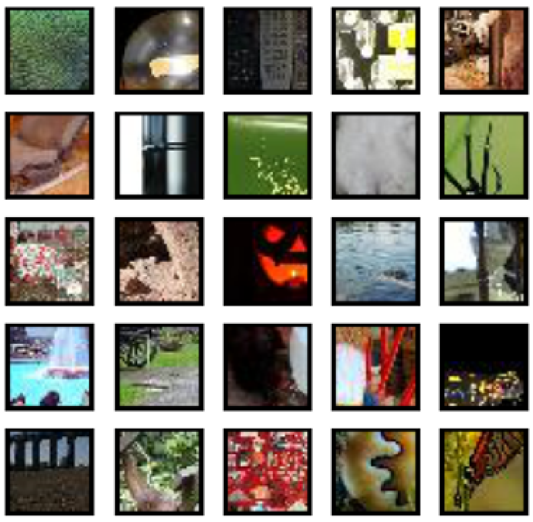
Figure 3.3: Random ImageNet examples.

The validation and test data consists of 150,000 photographs, hand labeled with the presence or absence of 1000 object categories. A random subset of 50,000 of the images with labels has been released as validation data along with a list of the 1000 categories. The remaining images are used for evaluation and have been released without labels.

3.2.4 TinyImageNet (crop) and TinyImageNet (resize)

The Tiny ImageNet dataset [56] consists of a subset of ImageNet images (Deng et al., 2009). It contains 10,000 test images from 200 different classes. We have used two datasets, TinyImageNet (crop) and TinyImageNet (resize), that gave been created by either randomly cropping image patches of size 32×32 or downsampling each image to size 32×32.

Here are some random images from these datasets presented in Figure 3.4 :


(a) Random TinyImageNet (crop) examples.

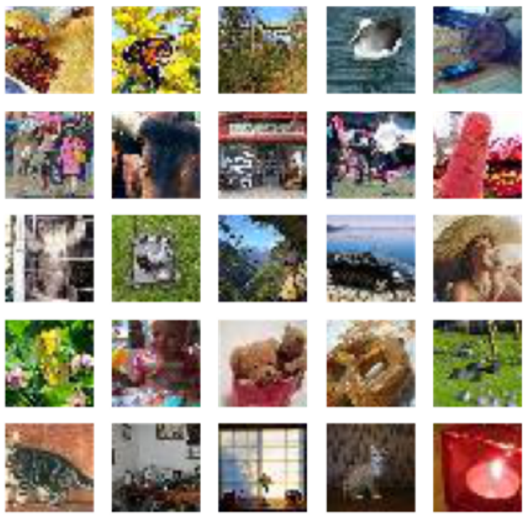

(b) Random TinyImageNet (resize) examples.

Figure 3.4: TinyImageNet (crop) and TinyImageNet (resize) examples.

3.2.5 Places365

The Places365 dataset [57] is a scene recognition dataset. It is composed of 10 million images comprising 434 scene classes. There are two versions of the dataset: Places365-Standard with 1.8 million train and 36,000 validation images from K=365 scene classes, and Places365-Challenge-2016, in which the size of the training set is increased up to 6.2 million extra images, including 69 new scene classes (leading to a total of 8 million train images from 434 scene classes). In this thesis, the former version have been used.

Here are some random images from this dataset presented in Figure 3.5:

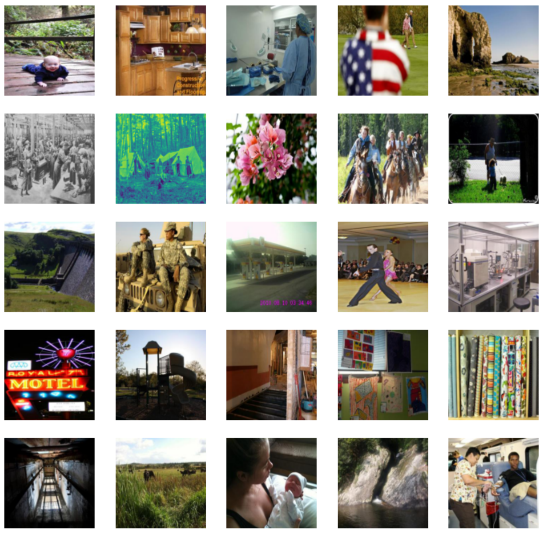


Figure 3.5: Random Places365 examples.

3.2.6 Textures

The Describable Textures Dataset (DTD) [58] is an evolving collection of textural images in the wild, annotated with a series of human-centric attributes, inspired by the perceptual properties of textures. This data is made available to the computer vision community for research purposes. It is consisting of 5640 images, organized according to a list of 47 terms (categories) inspired from human perception. There are 120 images for each category. Image sizes range between 300x300 and 640x640, and the images contain at least 90% of the surface representing the category attribute. The data is split in three equal parts, in train, validation and test, 40 images per class, for each split.

Here are some random images from this dataset presented in Figure 3.6:

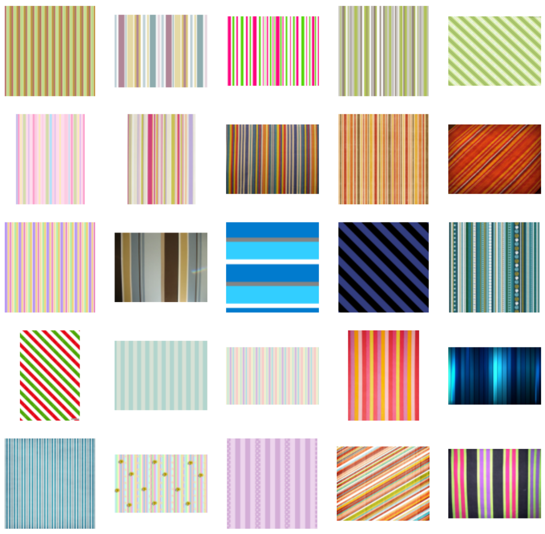


Figure 3.6: Random Textures examples.

3.2.7 INaturalist

The iNaturalist dataset [59] contains 675,170 training and validation images from 5,089 natural fine-grained categories. Those categories belong to 13 super-categories including Plantae (Plant), Insecta (Insect), Aves (Bird), Mammalia (Mammal), and so on. The iNat dataset is highly imbalanced with dramatically different number of images per category. For example, the largest super-category "Plantae (Plant)" has 196,613 images from 2,101 categories; whereas the smallest super-category "Protozoa" only has 381 images from 4 categories.

Here are some random images from this dataset presented in Figure 3.7:

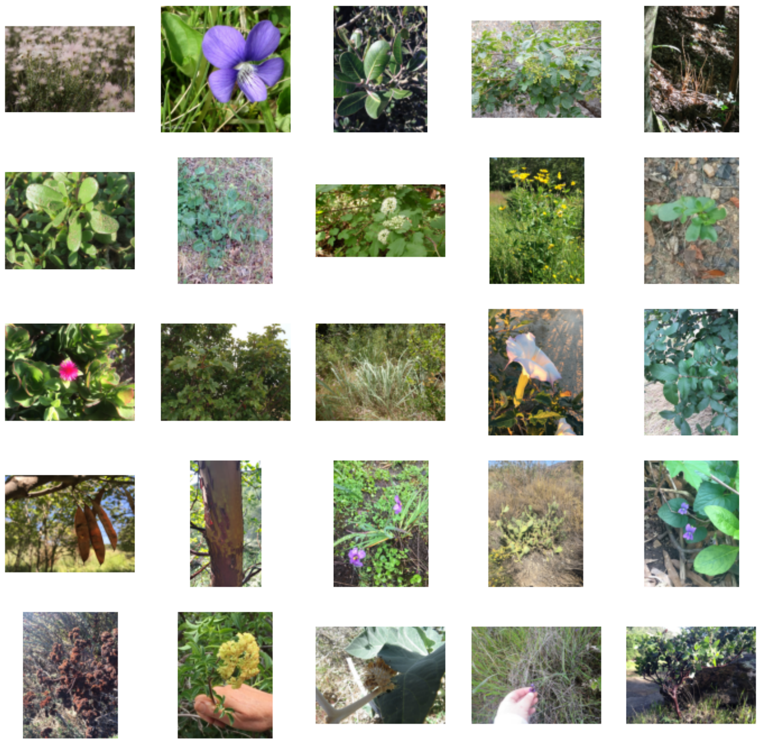


Figure 3.7: Random INaturalist examples.

3.2.8 SUN

The Scene UNderstanding (SUN) [60] database contains 899 categories and 130,519 images. There are 397 well-sampled categories to evaluate numerous state-of-the-art algorithms for scene recognition.

Here are some random images from this dataset presented in Figure 3.8:

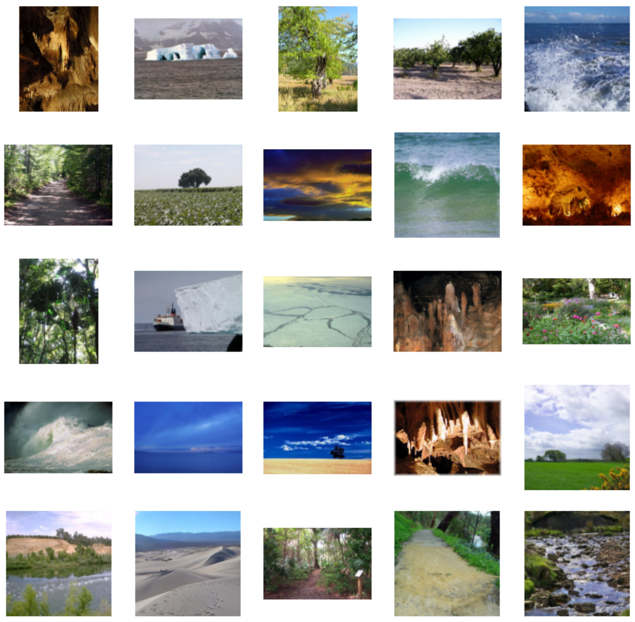


Figure 3.8: Random SUN examples.

3.2.9 ISUN

The iSUN [61] consists of a subset of SUN images. We include the entire collection of 8925 images in iSUN and downsample each image to size 32 by 32.

Here are some random images from this dataset presented in Figure 3.9:

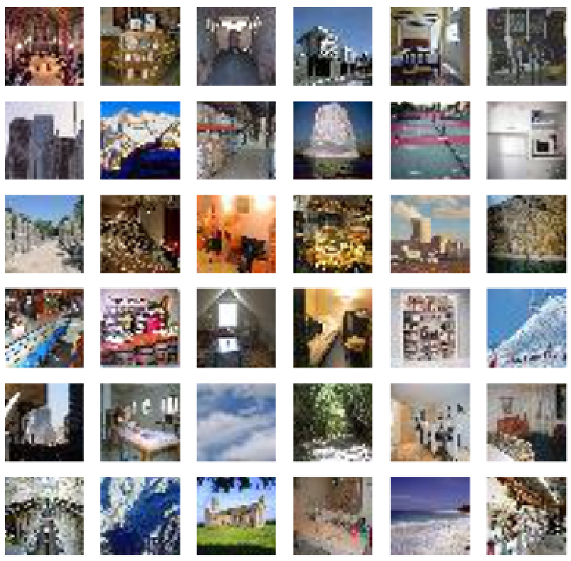
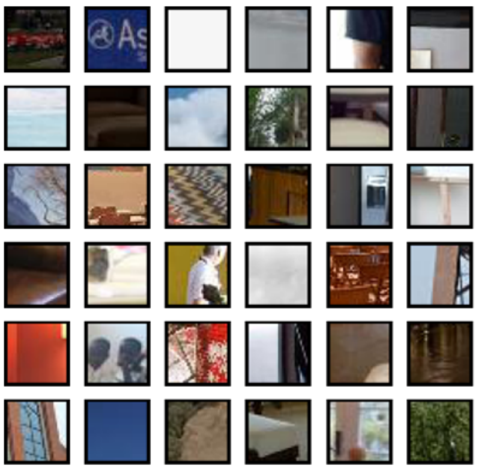


Figure 3.9: Random ISUN examples.

3.2.10 LSUN (crop) and LSUN (resize)

The Large-scale Scene UNderstanding dataset (LSUN) has a testing set of 10,000 images of 10 different scenes [62]. Similar to TinyImageNet, we used two datasets, LSUN (crop) and LSUN (resize), that were created by randomly cropping and downsampling the LSUN testing set, respectively.

Here are some random images from these datasets in Figure 3.10:


(a) Random LSUN (crop) examples.

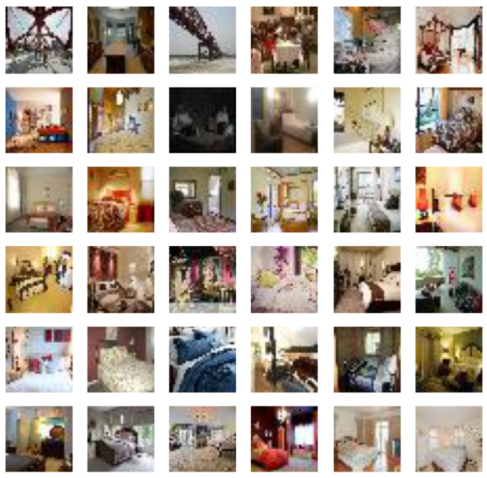

(b) Random LSUN (resize) examples.

Figure 3.10: LSUN (crop) and LSUN (resize).

3.2.11 ImageNet_O

ImageNet_O [63] contains 2000 images from 200 classes that contain semantic information differing from the ImageNet-1K classes. As depicted in Figure 3.11, there is a label distribution shift between ImageNet-1K classes and the ImageNet_O classes, albeit being visually similar.

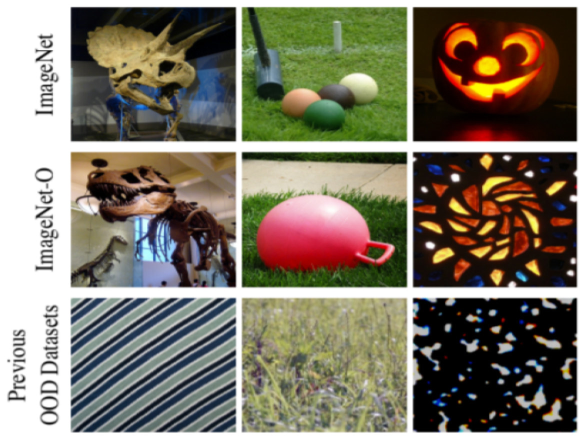


Figure 3.11: Comparison between ImageNet_O and the previous OOD benchmarks.

This is a more real world OOD scenario and hence makes it an important dataset to benchmark on.

Here are some random images from this dataset presented in Figure 3.12:

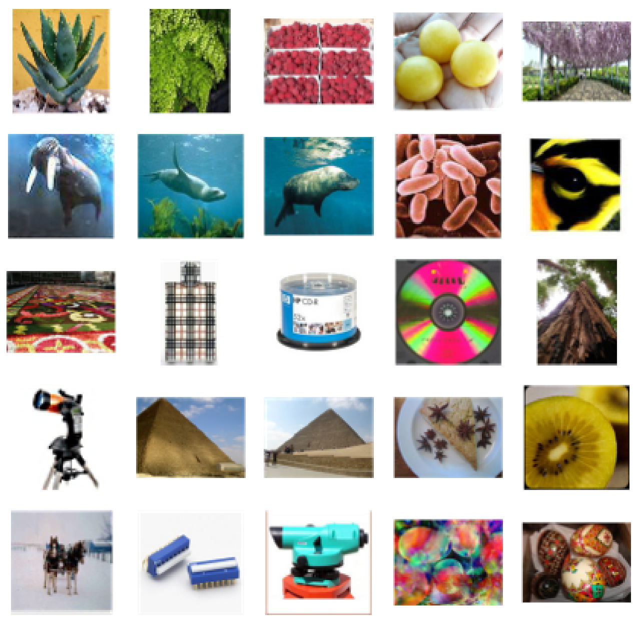


Figure 3.12: ImageNet_O examples.

3.3 Preprocessing

All the images should be normalized by the mean and standard deviation computed from each training dataset but since these values are very similar to each other in the case of CIFAR-10, CIFAR-100 and ImageNet, the normalization have been preformed using the same mean and standard deviation of the CIFAR datasets.

Finally, we have resized them to the right shape of each ID dataset as they are fed into their respective model: 32×32 for the DenseNet CNN models (CIFAR-10 and CIFAR-100) and 224×224 for the ImageNet model.

3.4 Architectures and training configurations.

We have used two neural network architectures: DenseNet [6] for the CIFAR-10 and CIFAR-100 and EfficientNet-B7 [8] for ImageNet dataset.

On the one hand, the DenseNet model follow the same setup as in [6], with depth L = 100, growth rate k = 12 (Dense-BC) and dropout rate=0. The hyper-parameters have been set to values that are identical to the original DenseNet implementation. They have been trained with stochastic gradient descent with Nesterov momentum for 300 epochs with batch size=64 and momentum 0.9. The learning rate starts at 0.1, and is dropped by a factor of 10 at 50% and 75% of the training progress, respectively.

On the other hand, EfficientNet-B7 have been trained using similar settings as [8]: RMSProp optimizer with decay 0.9 and momentum 0.9; batch norm momentum 0.99 weight decay 1e-5; initial learning rate 0.256 that decays by 0.97 every 2.4 epochs. They have also used SiLU (Swish-1) activation, AutoAugment, and stochastic depth with survival probability 0.8 and a 0.5 dropout.

Table 3.3 presents the test accuracies of the pretrained models used in our experiments.

Architecture (dataset)	**Accuracy**
DenseNet-BC-100 (CIFAR-10)	95.20%
DenseNet-BC-100 (CIFAR-100)	77.63%
EfficientNet_b7 (ImageNet)	84.12%

Table 3.3: Test accuracies on CIFAR-10, CIFAR-100 and ImageNet datasets considering only ID data.

3.5 Implementation details

For each OOD method, we have a score of 20,000 images which are equally splitted between ID dataset and the OOD dataset, in order to evaluate these methods.

3.5.1 Maximum softmax probability (MSP)

MSP requires no hyper-parameter tuning. The OOD score $OOD_{MSP} \in [0, 1]$ was defined as the maximum softmax score of the model output, and is considered as a baseline in our experiments.

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

On the other hand, ODIN requires the tuning of the noise magnitude ε. For this we have used 1,000 of ID images and 1,000 of OOD images for its tuning by doing a grid search and we chose the optimal value that gives the best separability between ID and OOD images. As for the temperature scaling T, it has been fixed to 1000 in all the experiments. At last, we have evaluated using these values of T and ε on the remaining images.

3.5.3 Energy score

The energy-based OOD score $OOD_{EBM} \in R^-$ simply applies the energy function $E(x; F)$ described previously to the logits produced by the classifier. In order to align with the convention that positive (in-distribution) samples have higher scores, we have used the negative energy instead: $OOD_{EBM} \in R^+$. In their original paper, Liu et al. show that tuning the temperature does not improve the results, and the method may therefore be used with T = 1, which reduces the anomaly score to the logarithm of the denominator of the softmax and makes it a hyper-parameter free method, as depicted in Figure 3.13.

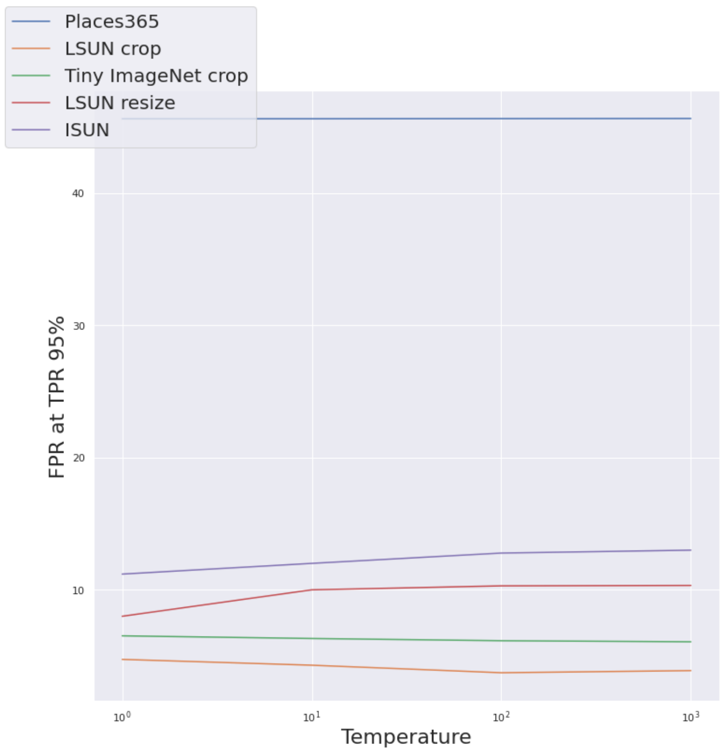


Figure 3.13: Effect of the temperature on the energy score.

3.6 Conclusion

After presenting the datasets used in this thesis, we introduced the image classification models architecture and their training configurations. Finally, we set the implementation details of the three OOD detection methods studied in this work.

Chapter 4

Results and Analysis

4.1 Introduction

This chapter presents, compares and analyzes the results of the experiments detailed in Chapter 2 and 3. First, the evaluation results of the three studied methods as well as the proposed one are presented while considering three different ID datasets: CIFAR-10, CIFAR-100 and ImageNet. The next section provides implementation results samples and few examples that show the limitation of the OOD detection techniques. The chapter is concluded by a presentation of a web user interface that allows to use and test our proposed method.

4.2 OOD detection evaluation

4.2.1 CIFAR-10 and CIFAR-100 datasets

Table 4.1 presents the obtained results after evaluating 3 different techniques and baseline. All methods are fine-tuned from the same pre-trained Densenet (BC_100) with CIFAR-10 and CIFAR-100 as in-distribution datasets. $\uparrow$ indicates larger values are better, while $\downarrow$ indicates smaller values are better. All values are percentages.

Table 4.1: OOD detection performance comparison between 3 different techniques and baseline. All methods are fine-tuned from the same pre-trained Densenet (BC_100) with CIFAR-10 and CIFAR-100 as in-distribution datasets. ↑ indicates larger values are better, while ↓ indicates smaller values are better. All values are percentages. **Bold** numbers are superior results.

		Evaluation metrics				
Model In-distribution dataset	Out-of-distribution dataset	FPR@95%TPR ↓	Detection error ↓	AUROC ↑	AUPR-In ↑	AUPR-Out↑
		MSP(Baseline)/ODIN/EnergyScore/ProposedMethod				
	ISUN	37.4/6.2/11.2/5.4	10.8/5.5/7.0/6.9	94.7/98.8/97.9/99.9	95.9/98.9/98.2/99.9	93.0/98.8/97.5/98.1
	LSUN (crop)	39.0/8.8/4.7/2.5	10.2/6.0/4.8/4.8	94.8/98.2/98.8/99.9	96.1/98.5/99.0/99.9	93.2/97.9/98.6/98.5
	LSUN (resize)	33.55/3.8/8.8/3.4	9.8/4.3/6.1/5.7	95.4/99.2/98.3/99.9	96.4/99.3/98.6/99.9	94.1/99.2/98.1/98.6
	SUN	51.5/48.5/34.1/23.6	14.2/19.4/12.2/13.8	92.1/88.1/94.2/99.4	92.9/87.4/95.1/99.2	89.4/87.8/92.6/91.2
Dense-BC-100 CIFAR-10	TinyImageNet (crop)	34.8/4.3/6.5/3.1	10.0/4.7/5.5/5.6	95.3/99.1/98.6/99.9	96.4/99.2/98.8/99.9	93.8/99.1/98.4/98.6
	TinyImageNet (resize)	41.2/7.6/14.2/6.5	11.5/6.0/7.9/7.5	94.1/98.5/97.4/99.9	95.0/98.6/97.7/99.8	92.3/98.5/97.0/97.7
	PLACES365	60.2/58.4/45.6/37.7	17.7/21.0/17.3/19.3	89.4/86.5/90.0/97.5	87.6/86.7/89.9/97.0	86.4/84.6/89.0/86.3
	TEXTURES	53.1/50.9/56.3/28.0	15.1/21.9/20.8/16.2	91.5/85.0/86.5/99.2	91.9/82.6/85.6/99.0	88.9/86.1/86.0/90.1
	INATURALIST	58.4/68.1/53.4/41.3	15.7/22.7/16.4/18.2	90.7/84.7/90.5/98.6	91.2/86.3/92.0/98.3	87.3/81.1/87.6/84.8
	ImageNet_O	55.2/47.0/45.6/30.3	16.5/18.3/15.9/16.5	90.3/89.2/90.8/98.5	90.1/89.1/91.1/98.1	87.7/88.7/89.7/89.6
	Average	46.4/30.4/31.2/**18.2**	13.2/13/**11.4**/11.5	92.8/92.8/94.3/**99.3**	93.4/92.7/94.6/**99.1**	90.6/92.2/**93.5**/93.4
	ISUN	85.1/65.6/80.4/62.4	36.2/24.3/29.4/30.0	69.5/83.5/77.3/90.5	72.4/84.6/79.5/91.3	66.4/81.2/73.5/74.3
	LSUN (crop)	59.6/17.2/20.9/11.9	23.2/9.4/10.6/10.6	85.5/96.7/96.2/99.0	87.0/96.9/96.3/99.0	84.4/96.5/96.2/96.0
	LSUN (resize)	83.2/61.5/76.8/59.0	35.1/23.4/27.5/29.0	70.8/84.8/79.7/91.2	73.9/85.9/81.7/92.0	67.9/83.0/76.3/76.5
	SUN	77.9/59.8/62.8/43.9	28.7/21.0/20.6/21.1	78.1/86.9/85.9/97.8	80.5/88.3/89.2/98.0	74.7/84.9/87.0/84.8
Dense-BC-100 CIFAR-100	TinyImageNet (crop)	66.7/27.7/38.9/26.2	27.0/13.1/16.6/15.9	81.1/94.4/91.5/96.8	82.9/94.6/91.7/96.9	80.1/94.4/91.7/92.2
	TinyImageNet (resize)	82.4/59.0/76.1/56.2	34.3/23.3/28.7/28.3	71.7/84.7/78.3/91.0	74.0/85.2/79.3/91.5	69.0/83.5/76.1/77.6
	PLACES365	79.9/77.6/78.7/54.1	30.0/27.2/28.6/22.3	76.4/79.0/77.8/95.0	78.2/79.7/78.5/95.9	73.0/75.3/74.7/80.6
	TEXTURES	76.1/56.7/60.3/44.3	30.6/21.1/21.6/22.8	76.1/86.6/85.7/94.3	77.0/86.6/84.9/94.2	74.1/85.4/85.0/84.1
	INATURALIST	78.6/76.8/76.4/62.7	26.1/24.9/25.7/26.0	80.3/81.7/81.4/93.0	83.5/83.9/83.3/94.1	75.5/76.4/77.2/75.8
	ImageNet_O	81.8/65.3/80.1/57.5	29.3/23.9/27.3/26.4	76.4/83.3/78.5/92.4	78.5/83.1/79.1/92.9	72.0/81.3/74.4/78.1
	Average	77.1/56.7/64.6/**47.8**	30/**21.2**/23.8/23.2	76.6/86.1/82.9/**94.1**	78.8/86.9/83.8/**94.6**	73.7/**84.2**/81.2/82

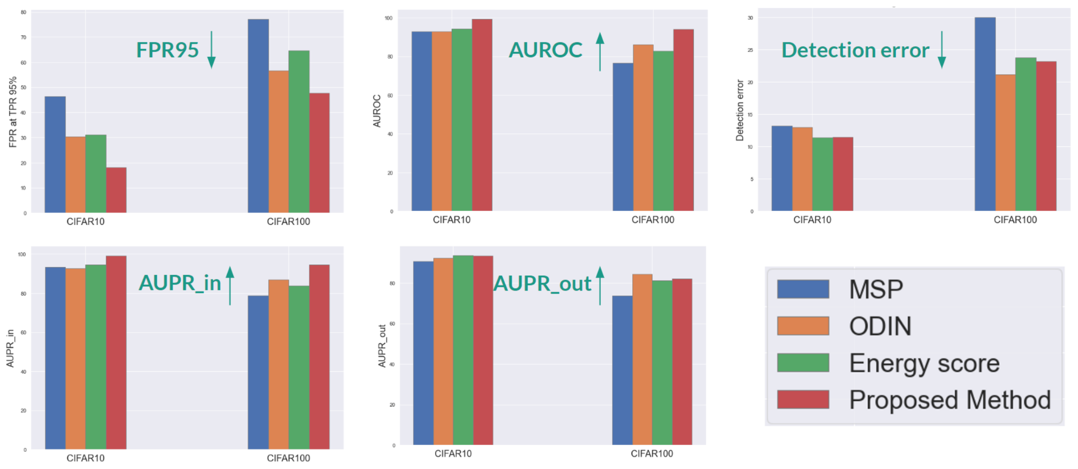


Figure 4.1: Performance comparison of 3 OOD detection techniques and baseline (MSP) considering two ID datasets: CIFAR-10 and CIFAR-100.

As depicted in Figure 4.1, all three methods ODIN, energy score and the proposed one give better performance results than the baseline MSP regarding all the 5 evaluation metrics. We notice that the AUPR_in evaluation metric is in general better than the AUPR_out, which makes the OOD detection techniques better in detecting ID images than OOD ones. We observe a drastic drop in performance when we increase the label class size from 10 to 100, for example when considering the proposed method, the FPR increased from 18.2 to 47.8 percent which means that the FPR has increased by 163% from its initial value in CIFAR-10.
Overall, the proposed method surpasses the other three studied techniques in OOD de-

tection performance, but we observe a slight difference considering the evaluation metric AUPR_Out making the proposed method always second in detecting truly OOD samples, which is foreseeable since our method work with an OR operator making the FP increase a little bit (OOD is considered as positive) regarding the other three methods.
But, in an OOD detection context it is better to be wrong in classifying ID as OOD than classifying OOD as ID given that they are applied in safety-critical environments, such as autonomous cars and medical diagnosis.

In the literature, the OOD detection techniques are often evaluated while considering CIFAR datasets as ID datasets, but in real world application CIFAR datasets have lots of limitations such as:

- Low resolutions (32x32)
- Small class space (10-100 labels)
- Not reflective of the visual complexity observed in the real world

One solution would be to test OOD detection on the ImageNet dataset.

4.2.2 ImageNet dataset

Some datasets were not included in this experiment due to their class overlapping with ImageNet dataset.

Table 4.2 presents the obtained results after evaluating 3 different techniques and baseline using ImageNet as ID dataset.

Table 4.2: OOD detection performance comparison between 3 different techniques and baseline. All methods are fine-tuned from the same pre-trained EfficientNet_B07 with ImageNet as in-distribution dataset. ↑ indicates larger values are better, while ↓ indicates smaller values are better. All values are percentages. **Bold** numbers are superior results.

		Evaluation metrics				
Model and In distribution dataset	Out-of-distribution dataset	FPR@95%TPR ↓	Detection error ↓	AUROC ↑	AUPR-In ↑	AUPR-Out↑
		MSP(Baseline)/ODIN/EnergyScore/ProposedMethod				
Efficientnet-B7 ImageNet-1K	LSUN (crop)	16.6/23.3/21.2/9.2	10.6/13.9/12.3/9.3	96.1/92.4/94.3/98.8	95.5/90.5/92.8/98.7	96.6/94.1/95.3/96.8
	SUN	56.9/79.6/87.3/48.5	23.5/32.3/40.3/27.0	84.7/71.4/50.3/99.6	84.2/75.7/59.5/99.6	84.7/72.4/70.9/81.4
	PLACES365	64.2/87.2/74.5/55.6	26.6/36.2/32.3/30.3	81.2/68.6/74.3/92.8	80.7/70.6/73.4/93.4	80.8/64.9/74.2/76.8
	TEXTURES	55.5/75.0/59.1/43.4	25.4/26.9/26.4/25.0	82.8/80.1/81.1/94.7	81.5/81.8/79.5/94.9	84.0/77.7/82.4/83.6
	ImageNet_O	86.5/80.6/82.6/69.8	43.9/30.1/32.5/34.0	59.2/75.6/73.2/89.2	61.4/76.3/73.6/89.9	60.2/73.3/71.0/70.9
	Average	55.9/69.1/64.9/**45.3**	26/27.9/28.8/**25.1**	80.8/77.6/74.6/**95.0**	80.7/79/75.8/**95.3**	81.3/76.5/78.8/**81.9**

Since ImageNet_O is a more real world OOD scenario than the other OOD datasets and it is a dataset that has been created specifically for ImageNet models to aid research in OOD detection, we have studied it in depth in the following.

As depicted in Figure 4.2, the three methods give better performance results than the baseline. The OOD detection techniques are better in detecting ID images than OOD ones since AUPR_in evaluation metric is in general better than the AUPR_out. The detection performance tend to fall when we increase the label class size from 10 to 100 to 1000. In general, the proposed method surpasses the other three studied techniques in OOD detection performance but we observe a slightly difference considering the evaluation metric AUPR_Out making the proposed method always second in detecting truly OOD samples.

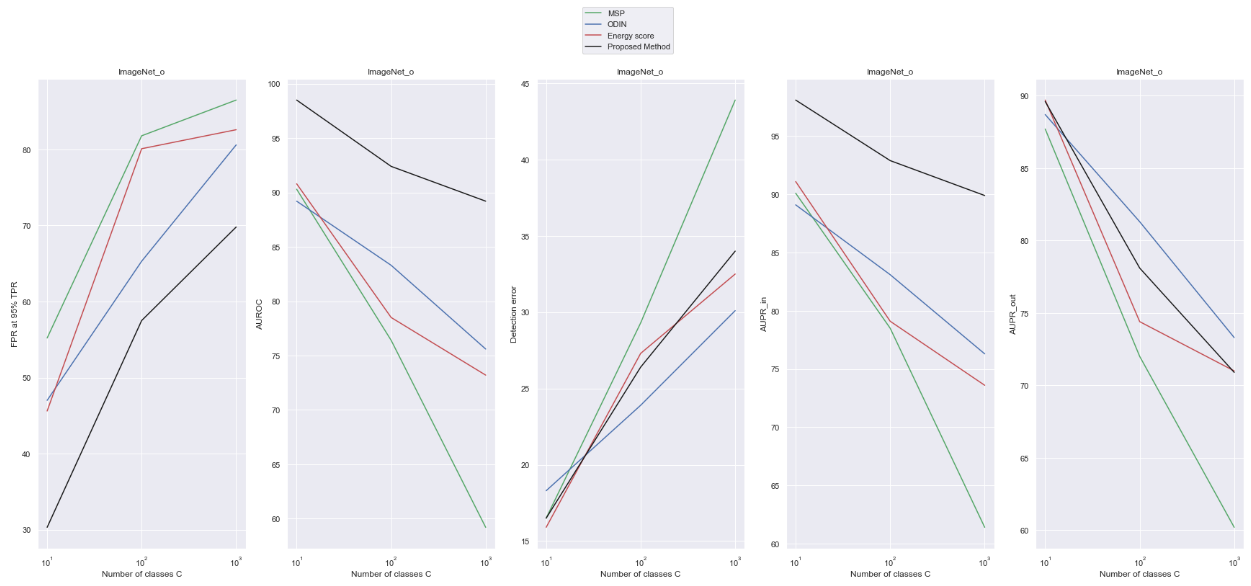


Figure 4.2: Effect of label space size on the OOD detection performance with ImageNet_O as OOD dataset.

The graphs in Figure 4.3 illustrate the ROC-curves for all methods in three representative scenarios (CIFAR10, CIFAR100 and ImageNet). This shows the effect of label space size on the AUROC metric with ImageNet_O as OOD dataset. The results suggest that the performance of all the techniques regarding the AUROC metric rapidly decreases until the MSP become performing like a random classifier in the case ImageNet.

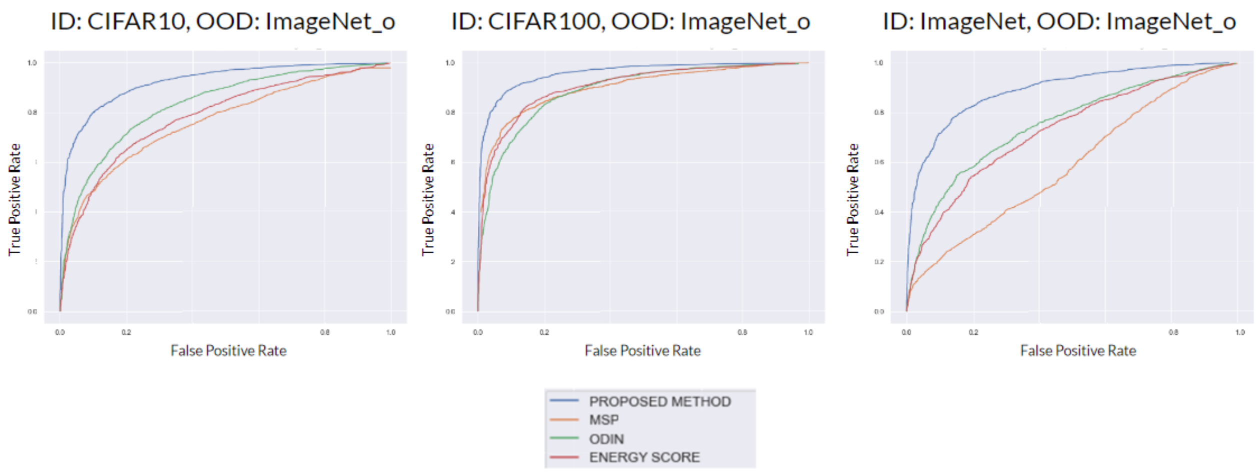


Figure 4.3: Effect of label space size on the AUROC metric with ImageNet_O as OOD dataset.

4.2.3 Runtime comparison

Table 4.3 summarizes the runtime required for each method to output a novelty score for all considered images. Runtime was determined by averaging the total runtime to obtain novelty scores by the total number of testing images: 20000 images splitted equally between ID images and OOD images.

Method	**Runtime (sec)**
Max Softmax (MSP)	0.05
ODIN	0.13
Energy score	0.05
Proposed Method	0.13

Table 4.3: Inference time for each of the OOD methods.

We note that ODIN and the proposed method take more time to give novelty scores for the considered images and are 2.5 slower that the baseline and the energy score techniques.

4.3 Accuracy comparison

In the Table 4.4, we compare the classification accuracies of our models in a multi-class novelty detection setting when the MSP and the proposed method are implemented. We note an enhancement of 14.1%, 14.65% and 5.3% when the ID dataset is CIFAR-10, CIFAR-100 and ImageNet respectively.

Architecture	ID Dataset	MSP (baseline)	Proposed method
DenseNet(BC-100)	CIFAR-10	74.30%	88.40%
DenseNet(BC-100)	CIFAR-100	58.95%	73.60%
EfficientNet-B7	ImageNet	69.55%	74.85%

Table 4.4: Comparison of the classification accuracy between MSP and the proposed method in multi-class novelty detection setting.

In Table 4.5, we have compared the classification accuracies of our pretrained models when OOD detection is applied or not. We have noted that in the case of classification without OOD detection the accuracies are pretty low despite being originally high (divided by two), that's due to considering only ID data in the former computation. As a result, when we tested with OOD data they were all misclassified as ID. In contrast, when the OOD detection techniques were being used the accuracies were pretty high and a clear improvement were observed.

Architecture	ID Dataset	Without OOD detection	With OOD detection
DenseNet(BC-100)	CIFAR-10	47.60%	91.80%
DenseNet(BC-100)	CIFAR-100	38.80%	75.60%
EfficientNet-B7	ImageNet	42.10%	79.50%

Table 4.5: Comparison of the classification accuracy when no OOD detection were used and when we are in an Open set recognition setting using our proposed method.

As it might be seen in Figure 4.4, when the OOD data increase at the test phase the accuracy will drop drastically in the case of a classification when no OOD detection were applied. In contrast, when we apply our proposed method to the pretrained models the accuracy performance will not be very affected by the OOD data and a smaller decrease is noticed.

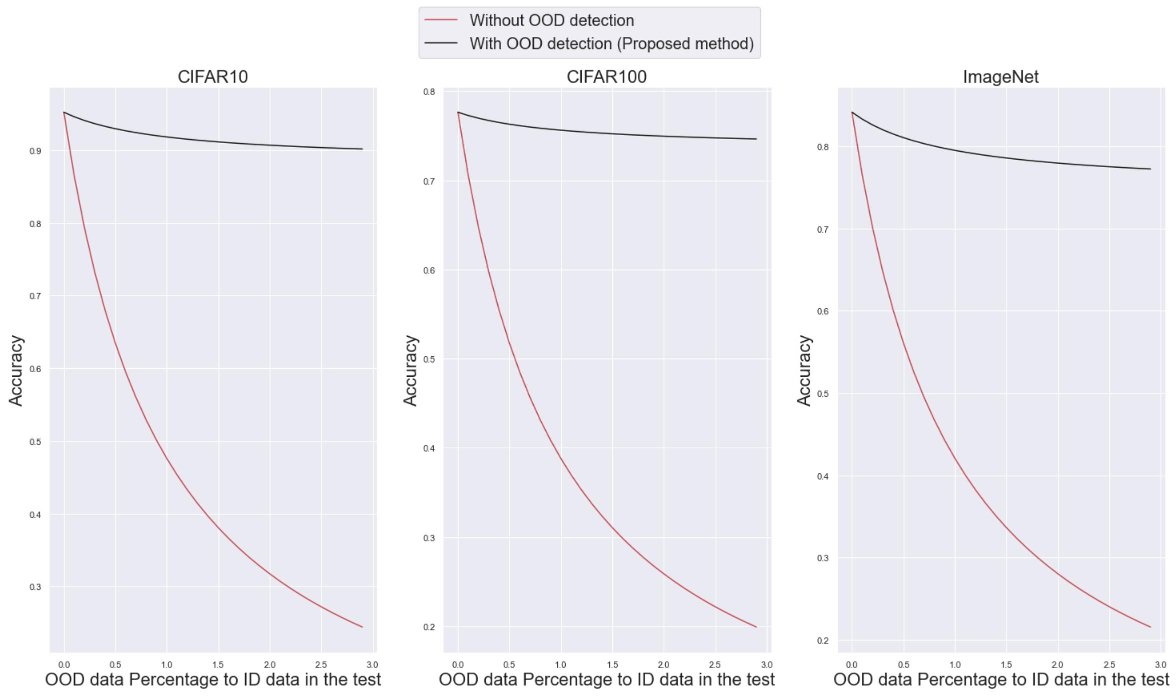


Figure 4.4: Effect of the size of OOD data in the test set on the classification accuracy of our models.

4.4 Implementation results samples

Here are some examples of the OOD detection after implementing the previous OOD detection techniques on 3 ID datasets: CIFAR-10,CIFAR-100 and ImageNet.

4.4.1 CIFAR10:

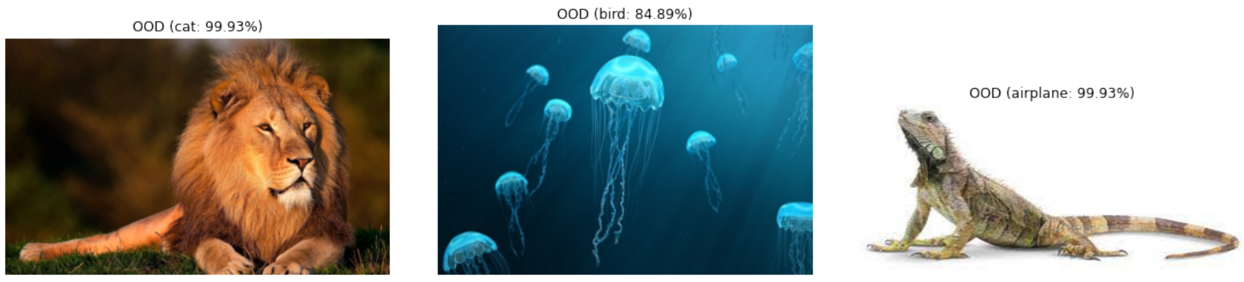


Figure 4.5: OOD examples: CIFAR-10

4.4.2 CIFAR100:

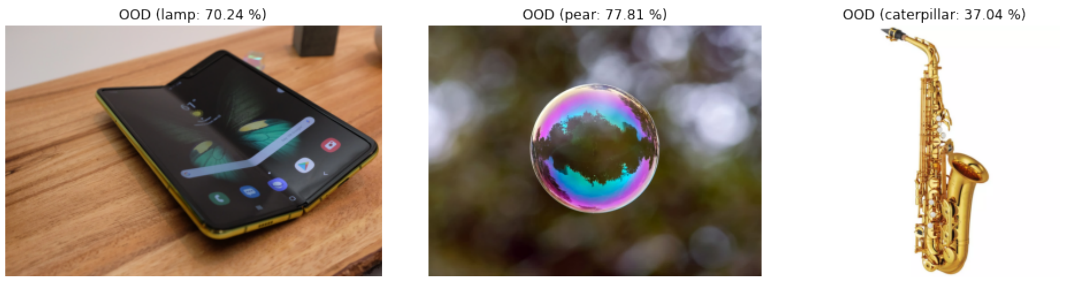


Figure 4.6: OOD examples: CIFAR-100

4.4.3 ImageNet:

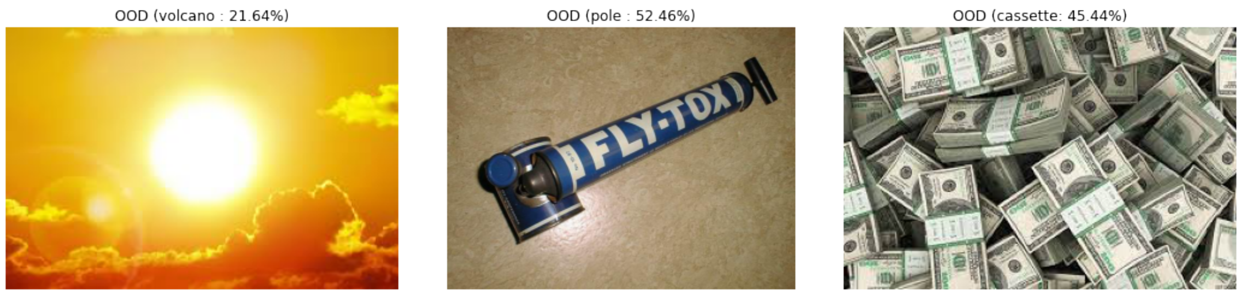


Figure 4.7: OOD examples: ImageNet

We can see in the Figures 4.5, 4.6 and 4.7 that if the OOD detection techniques were not applied, the model would have produced arbitrarily confident predictions, such as the example with the lion in the CIFAR-10 that was misclassified as a cat with a 99.93% probability.

4.5 Limitations

In addition to the large number of ID classes, OOD images with high semantic similarity to ID can pose a real challenge for the OOD detection techniques.

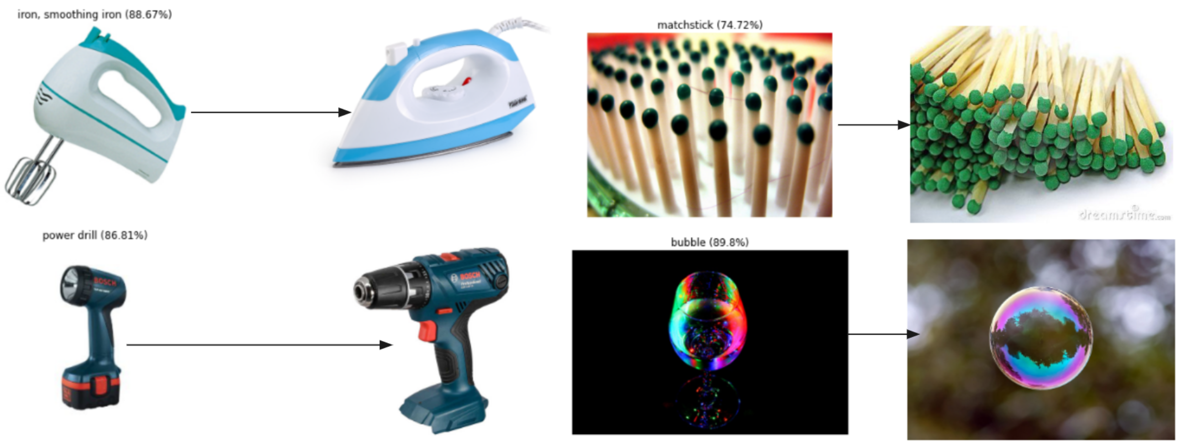


Figure 4.8: In the left we have the OOD examples from the ImageNet_O dataset, in the right we have the predicted class.

Due to the high semantic similarity between ImageNet and ImageNet_O and the previously introduced problem of producing arbitrarily confident predictions, an overlapping between the IN distribution an the Out distribution occurred making their separation more difficult which result as an important increase in the FPR like shown in Figure 4.9.

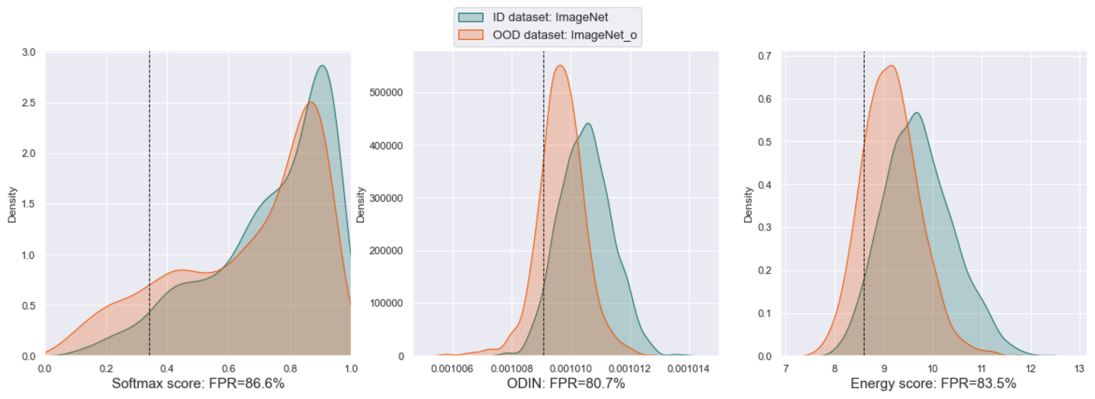


Figure 4.9: Distribution of softmax scores vs. ODIN detector vs. energy scores from pretrained EfficientNet_B07. We use negative energy scores to align with the convention that positive (in-distribution) samples have higher scores.

4.6 Web Interface

4.6.1 Introduction

We have created a web user interface to test out our OOD detection method, with this interface testing our method will be accessible by normal users. It can also be deployed on the web. The main goal of this UI is to identify OOD and ID data of a dataset.

4.6.2 Development tools

Python:

Python is a high-level, interpreted, general-purpose programming language, it is used widely in data science and deploying deep learning algorithms, is the main language used in the previous experiments, from creating and loading models to plotting classification results.

Google Colaboratory:

Google Colaboratory, or Google Colab, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser on the cloud, and is especially well suited to machine learning, data analysis. More technically, Colab provides a free development environment with resources like CPU and GPU, and it doesn't require any kind of setup to use.

Flask:

Flask a micro web framework written in Python, it can create APIs for python and allow the user to create simple or complex applications, it supports extensions that can add application features as if they were implemented in Flask itself, it is used to pass the classification function from the python function, to the Vue.JS interface and show it to the user.

Vue.JS:

Vue.js is an open-source front end JavaScript framework for building user interfaces and single-page applications, it has an HTML syntax by default which makes it a simple framework to develop with, also it has virtual DOM feature which renders only the elements that have changed in the page (the header elements in our interface), also Vue came in handy when exchanging API with the Flask app.

Ngrok:

Ngrok is a cross-platform application that facilitates putting local applications on the web for testing purposes, it can be considered as a temporary deployment solution.

4.6.3 Structure

Figure 4.10 presents the structure of our web interface.

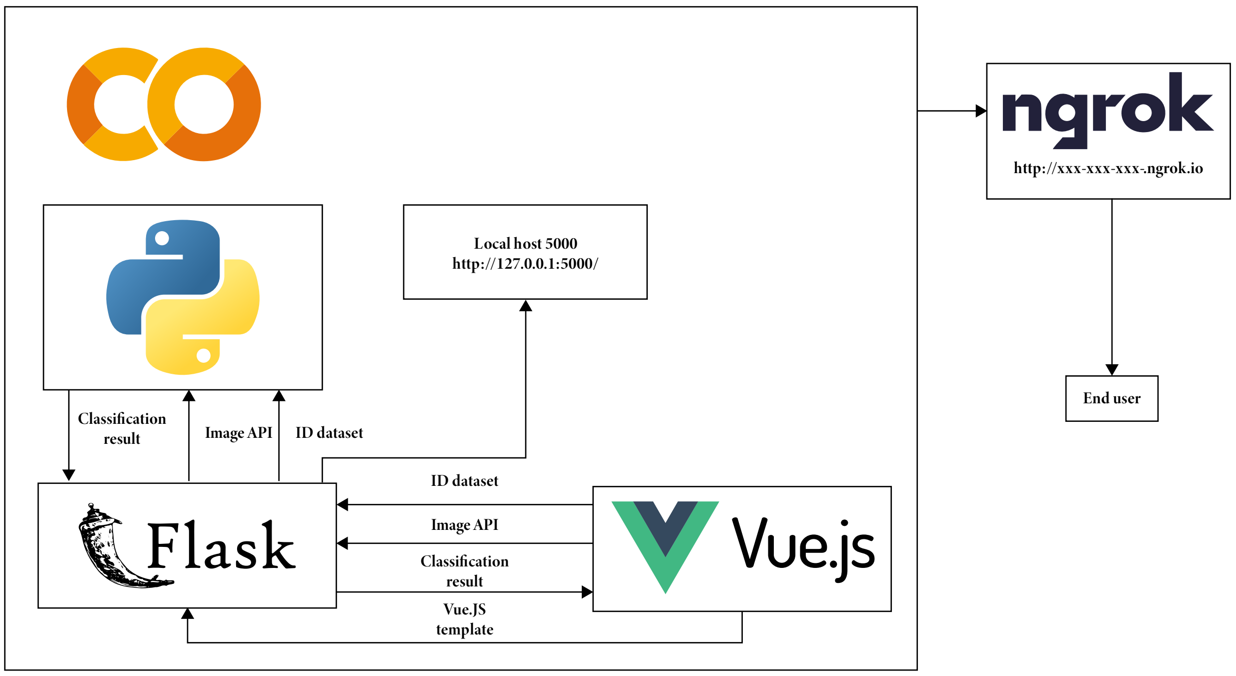


Figure 4.10: Structure of the web application.

Colab: In the Colab notebook we import all the libraries needed (Pytorch, Numpy, Flask, Ngrok...etc), we also have to load the three used pretrained models with their labels: ImageNet can be loaded from Pytorch library and Densenet 10 and 100 can be downloaded from GitHub. The notebook also contains the classification function that transforms the image depending on the selected model and applying our OOD detection method.

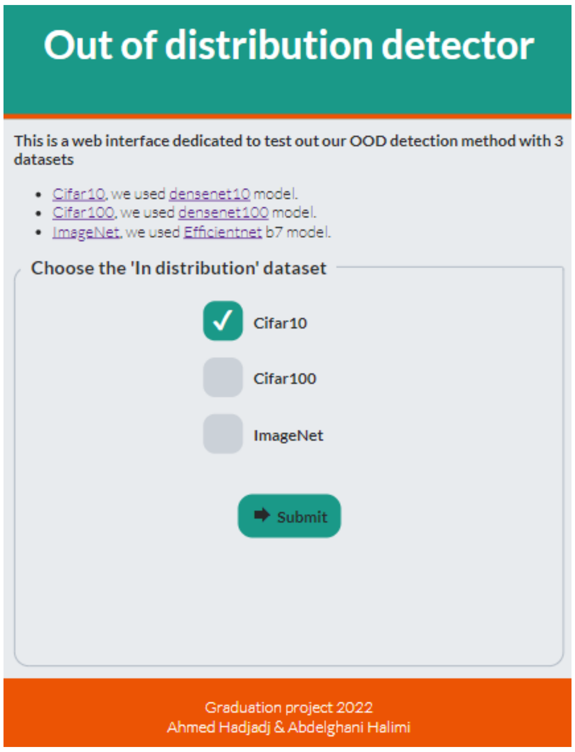
The Flask app: In the last cell we built the Flask app, it contains the index function which gets the In-distribution dataset from the radio-box after pressing submit, and the predict-api function that grabs the image and pass it to the predict function in previous predict function cell, then it returns the prediction results to show it in the Vue.JS application.

Front end: In the Colab notebook, there are two cells dedicated to the front end application, one is for the home page where you select the In-distribution dataset, the second page is for uploading the image and classify it.

Bridging: There is a cell in the notebook dedicated to Ngrok authentication key, adding the key provided by Ngrok allows the local cloud application to be shared with anyone with internet access via Ngrok agent.

4.6.4 Presentation

Our interface uses three different models: Densenet-BC-10, Densenet-BC-100 and Efficientnet-B7, trained with : CIFAR-10, CIFAR-100 and ImageNet respectively. Using this interface is simple as it's supposed to be, at the home page of our web interface you can select one of the in distribution datasets mentioned above then press submit as presented in Figure 4.11.


Out of distribution detector

This is a web interface dedicated to test out our OOD detection method with 3 datasets

- Cifar10, we used densenet10 model.
- Cifar100, we used densenet100 model.
- ImageNet, we used Efficientnet b7 model.

Choose the 'In distribution' dataset

✓ Cifar10

Cifar100

ImageNet

Graduation project 2022
Ahmed Hadjadj & Abdelghani Halimi

Figure 4.11: Web interface homepage

Then upload the image that you want to classify and tap on "Classify image" as shown in Figure 4.12.

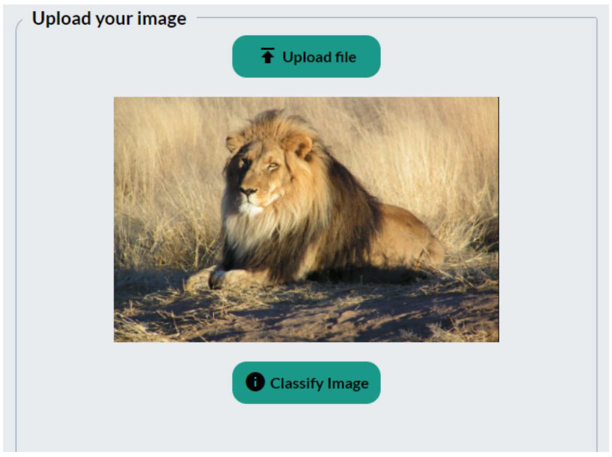


Figure 4.12: Web interface homepage.

The classification output will be the prediction class and its confidence score, with the OOD class in orange and ID class in green as illustrated in Figure 4.13.

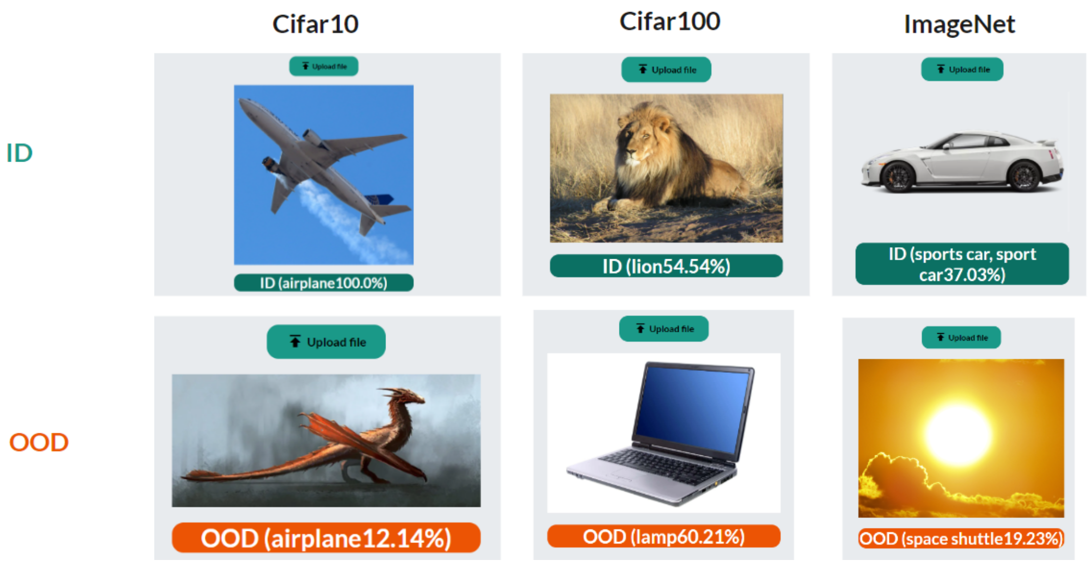


Figure 4.13: Some results from the web interface using CIFAR-10, CIFAR-100 and ImageNet as in-distribution datasets respectively.

4.7 Conclusion

After presenting and analyzing our experimental findings, we discussed the possible causes for these results and pointed up the limitations of the OOD detection techniques considered in this thesis. We concluded by presenting our web user interface implementation of our proposed method.

Conclusions and Future Work

Conclusions

In this work, three out-of distribution detection methods have been implemented, evaluated and compared on several common benchmarks that are used in the literature, as well as on the ImageNet-O dataset, a dataset that is more interesting and important than the previous OOD datasets since it is considered as a more real world OOD scenario. The comparison shows that despite the good performance of ODIN in OOD detection it needs a hyper-parameter tuning and remains a slower method compared to MSP and energy based score function.

In this thesis, we have also investigated the effect of label space size on the OOD detection performance. For that we have used three different ID datasets (CIFAR-10, CIFAR-100 and ImageNet-1K), and we have showed that the performance drastically decreases as the number of ID classes increases. We have also proposed a method that surpassed the three studied methods in detection performances and we created a web user interface to test out this proposed technique on the previous ID datasets.

Obtained results also suggest that none of the evaluated methods are consistently reliable for semantically similar data. The methods show decent results in specific scenarios but they struggles once the out-of-distribution data is too semantically similar to ID data.

Future work

The base classifiers considered in this work show a very high accuracy on the base task, which is rarely the case in a real-world applications. It is therefore of interest to study how the considered methods perform on less accurate classifiers.

Furthermore, it would be of interest to study how these methods perform in a real application and on other domains, such as text data, video, audio...

In this project, we have only worked with three OOD detection methods but in the literature there are many more OOD detection techniques to study. Therefore, it will be very interesting to test other types of OOD detection methods.

Finally, our ambition for the future is to apply these techniques on other inverse problems such as the object detection task, since it is one of the fundamental problems of computer vision.

Bibliography

[1] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” *ICLR*, 2017.

[2] I. Goodfellow, Y. Bengio, and A. Courville, “Deep learning,” *The MIT Press, Cambridge, Massachusetts, 1 edition*, 2016.

[3] K. Murphy, “Machine learning: A probabilistic perspective. mit press,” *Cambridge, Massachusetts, 1 edition*, 2012.

[4] B. Fortuner, “Loss functions,” *ML Glossary documentation*, 2017.

[5] R. Shukla, “L1 vs. l2 loss function,” 7 2015.

[6] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” *In Proceedings - 30th IEEE Conference on Com- puter Vision and Pattern Recognition, CVPR 2017, volume 2017-Janua*, 2017.

[7] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *In British Machine Vision Conference 2016, BMVC 2016, volume 2016-Septe*, 2016.

[8] M. Tan and Q. V, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *ICML*, 2019.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *IEEE*, 1998.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *NeurIPS*, 2012.

[11] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *ICLR*, 2015.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CVPR*, 2015.

[13] K. Hea, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CVPR*, 2016.

[14] L. Ruff, J. R. Kauffmann, Vandermeulen, G. Montavon, W. Samek, Dietterich, and K. Müller, “A unifying review of deep and shallow anomaly detection,” *Proc. IEEE, vol. 109, no. 5*, 2021.

[15] Pimentel, Clifton, and Tarassenko, "A review of novelty detection," *Signal Process., vol. 99*, 2014.

[16] S. Akcay, A. Abarghouei, and T. P. Breckon, "Ganomaly: Semisupervised anomaly detection via adversarial training," *in Computer Vision - ACCV 2018 - 14th Asian Conference on Computer Vision, Perth, Australia*, 2018.

[17] S. Akçay, A. Abarghouei, and T. P. Breckon, "Skip-ganomaly: Skip connected and adversarially trained encoder-decoder anomaly detection," *in International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary*, 2019.

[18] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan, "Do deep generative models know what they don't know?" *NeurIPS*, 2018.

[19] D. Ulmer, L. Meijerink, and G. Cinà, "Trust issues: Uncertainty estimation does not enable reliable ood detection on medical tabular data," *in Proceedings of the Machine Learning for Health NeurIPS Workshop, ser. Proceedings of Machine Learning Research*, 2020.

[20] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *in Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017.

[21] J. Snoek, Y. Ovadia, E. Fertig, B. Lakshminarayanan, S. Nowozin, D. Sculley, J. V. Dillon, J. Ren, and Z. Nado, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," *in Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*, 2019.

[22] T. Pimentel, M. Monteiro, A. Veloso, and N. Ziviani, "Deep active learning for anomaly detection," *in 2020 International Joint Conference References | 79 on Neural Networks (IJCNN)*, 2020.

[23] S. Das, W. Wong, T. G. Dietterich, A. Fern, and A. Emmott, "Incorporating expert feedback into active anomaly discovery," *in IEEE 16th International Conference on Data Mining, ICDM*, 2016.

[24] J. Yang, K. Zhou, Y. Li, and Z. Liu, "Generalized out-of-distribution detection: A survey," *IEEE*, 2021.

[25] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward supervised anomaly detection," *J. Artif. Intell. Res., vol. 46*, 2013.

[26] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, K. M. E. Müller, and M. Kloft, "Deep semi-supervised anomaly detection," *in 8th International Conference on Learning Representations, ICLR*, 2020.

[27] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *NeurIPS*, 2018.

[28] S. P. I. Kobyzev and M. Brubaker, “Normalizing flows: An introduction and review of current methods,” *TPAMI*, 2020.

[29] E. Zisselman and A. Tamar, “Deep residual flow for out of distribution detection,” *CVPR*, 2020.

[30] D. P. Kingma and P. Dhariwal, “Glow: Generative flow with invertible 1x1 convolutions,” *NeurIPS*, 2018.

[31] A. V. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” *ICML*, 2016.

[32] H. Choi, E. Jang, and A. Alemi, “Waic, but why? generative ensembles for robust anomaly detection,” *arXiv preprint arXiv*, 2018.

[33] P. Kirichenko, P. Izmailov, and A. G. Wilson, “Why normalizing flows fail to detect out-of-distribution data,” *NeurIPS*, 2020.

[34] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. A. DePristo, J. V. Dillon, and B. Lakshminarayanan, “Likelihood ratios for out-of-distribution detection,” *NeurIPS*, 2019.

[35] J. Serra, D. A. lvarez, V. Gomez, O. Slizovskaia, and J. Luque, “Input complexity and out-of-distribution detection with likelihood-based generative models,” 2020.

[36] Z. Xiao, Q. Yan, and Y. Amit, “Likelihood regret: An out-ofdistribution detection score for variational auto-encoder,” *NeurIPS*, 2020.

[37] J. Ren, S. Fort, J. Liu, A. G. Roy, S. Padhy, and B. Lakshminarayanan, “A simple fix to mahalanobis distance for improving near-ood detection,” *arXiv preprint arXiv*, 2021.

[38] E. Techapanurak, M. Suganuma, and T. Okatani, “Hyperparameter-free out-of-distribution detection using cosine similarity,” *ACCV*, 2020.

[39] F. S. X. Chen, X. Lan and N. Zheng, “A boundary based outof- distribution classifier for generalized zero-shot learning,” *ECCV*, 2020.

[40] A. Zaeemzadeh, N. Bisagno, Z. Sambugaro, N. Conci, N. Rahnavard, and M. Shah, “Out-of-distribution detection using union of 1-dimensional subspaces,” *CVPR*, 2021.

[41] J. V. Amersfoort, L. Smitha, Y. W. Teh, and Y. Gal, “Uncertainty estimation using a single deep deterministic neural network,” *ICML*, 2020.

[42] H. Huang, Z. Li, L.Wang, S. Chen, B. Dong, and X. Zhou, “Feature space singularity for out-of-distribution detection,” *arXiv preprint arXiv*, 2020.

[43] Y. L. S. Liang and R. Srikant, “Enhancing the reliability of out-ofdistribution image detection in neural networks,” *ICLR*, 2018.

[44] A. Vyas, N. Jammalamadaka, X. Zhu, D. Das, B. Kaul, and T. L. Willke, "Out-of-distribution detection using an ensemble of self supervised leave-out classifiers," *ECCV*, 2018.

[45] D. Hendrycks, M. Mazeika, and T. Dietterich, "Deep anomaly detection with outlier exposure," *ICLR*, 2019.

[46] M. Hein, M. Andriushchenko, and J. Bitterwolf, "Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem," *CVPR*, 2019.

[47] Y.-C, Hsu, Y. Shen, H. Jin, and Z. Kira, "Generalized odin: Detecting out-of-distribution image without learning from out-ofdistribution data," *CVPR*, 2020.

[48] W. Liu, XiaoyunWang, J. D. Owens, and Y. Li, "Energy-based out-of-distribution detection," *NeurIPS*, 2020.

[49] Y. Sun, C. Guo, and Y. Li, "React: Out-of-distribution detection with rectified activations," *NeurIPS*, 2021.

[50] Z. Lin, S. D. Roy, , and Y. Li, "Mood: Multi-level out-of-distribution detection," *CVPR*, 2021.

[51] J. Yang, H. Wang, L. Feng, X. Yan, H. Zheng, W. Zhang, and Z. Liu, "Mood: Multi-level out-of-distribution detection," *ICCV*, 2021.

[52] H. Wang, W. Liu, A. Bocchieri, and Y. Li, "Can multi-label classification networks know what they don't know?" *NeurIPS*, 2021.

[53] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *ICLR*, 2015.

[54] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[55] J. Deng, W. Dong, R. Socher, and Li-Jia, "Imagenet: A large-scale hierarchical image database," *IEEE*, 2009.

[56] J. Wu, Q. Zhang, and G. Xu, "Tiny imagenet visual recognition challenge," 2017.

[57] A. López-Cifuentes, M. Escudero-Viñolo, J. Bescós, and Álvaro García-Martín, "Semantic-aware scene recognition," *Pattern Recognition*, 2019.

[58] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi, "Describing textures in the wild," *CVPR*, 2014.

[59] G. V. Horn, O. M. Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie, "The inaturalist species classification and detection dataset," *CVPR*, 2017.

[60] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," *CVPR*, 2010.

[61] P. Xu, K. A. Ehinger, yinda zhang, A. Finkelstein, S. R. Kulkarni, and J. Xiao, "Turkergaze: Crowdsourcing saliency with webcam based eye tracking," 2015.

[62] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," 2016.

[63] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song, "Natural adversarial examples," *CVPR*, 2021.

Appendices

Appendix A

Figures

Figures A.1 to A.30 show the histograms for each method (MSP, ODIN and energy score) on every considered ID and OOD test set in this thesis. ID data in green and OOD data in orange.

A.1 CIFAR-10

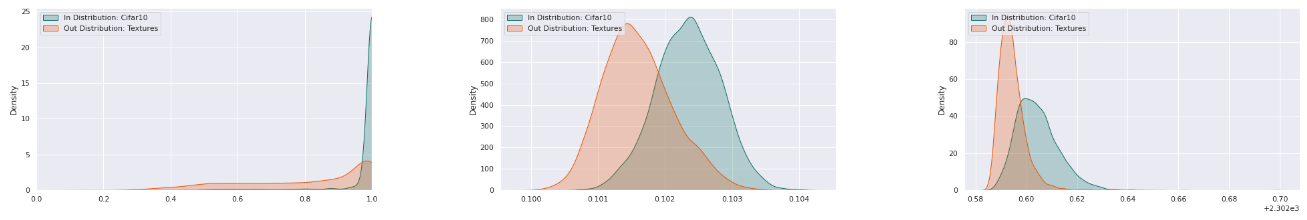


Figure A.1: Distributions over the anomaly scores for CIFAR-10 as ID and Textures as OOD using MSP, ODIN and Energy score respectively

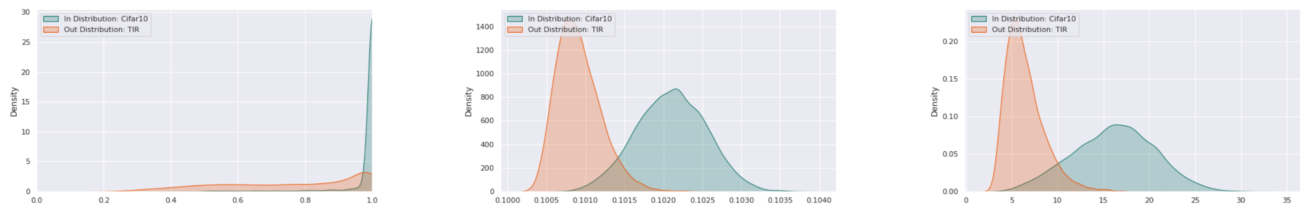


Figure A.2: Distributions over the anomaly scores for CIFAR-10 as ID and TIR as OOD using MSP, ODIN and Energy score respectively

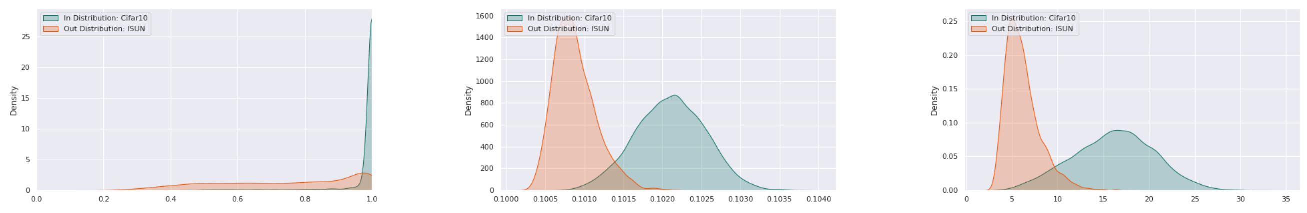


Figure A.3: Distributions over the anomaly scores for CIFAR-10 as ID and ISUN as OOD using MSP, ODIN and Energy score respectively

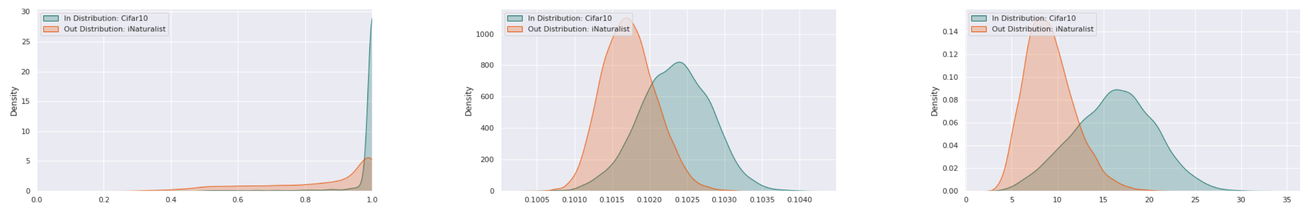


Figure A.4: Distributions over the anomaly scores for CIFAR-10 as ID and iNaturalist as OOD using MSP, ODIN and Energy score respectively

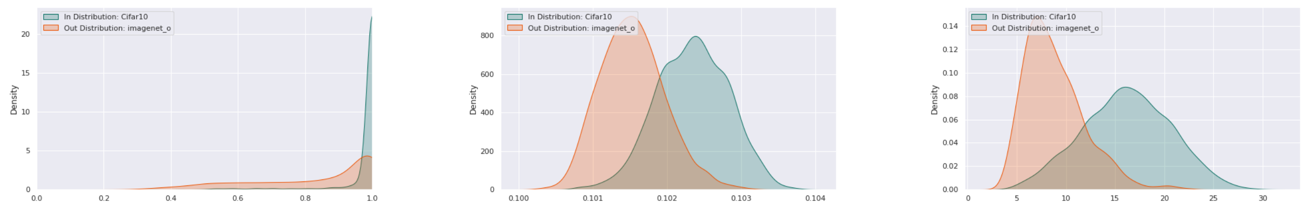


Figure A.5: Distributions over the anomaly scores for CIFAR-10 as ID and ImageNet_o as OOD using MSP, ODIN and Energy score respectively

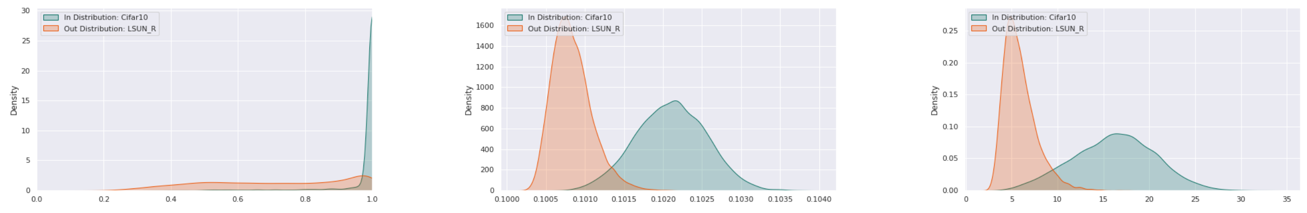


Figure A.6: Distributions over the anomaly scores for CIFAR-10 as ID and LSUN_R as OOD using MSP, ODIN and Energy score respectively

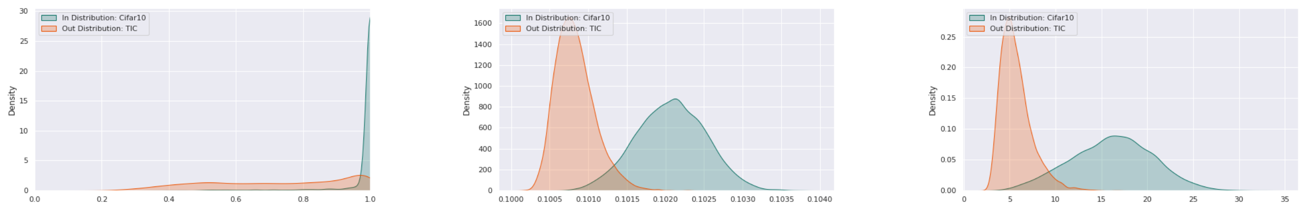


Figure A.7: Distributions over the anomaly scores for CIFAR-10 as ID and TIC as OOD using MSP, ODIN and Energy score respectively

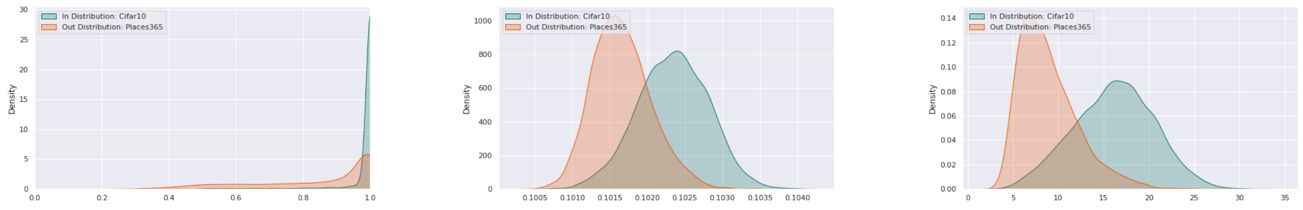


Figure A.8: Distributions over the anomaly scores for CIFAR-10 as ID and Places365 as OOD using MSP, ODIN and Energy score respectively

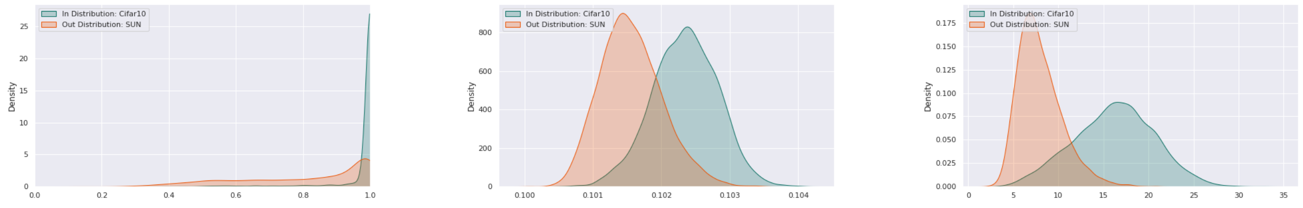


Figure A.9: Distributions over the anomaly scores for CIFAR-10 as ID and SUN as OOD using MSP, ODIN and Energy score respectively

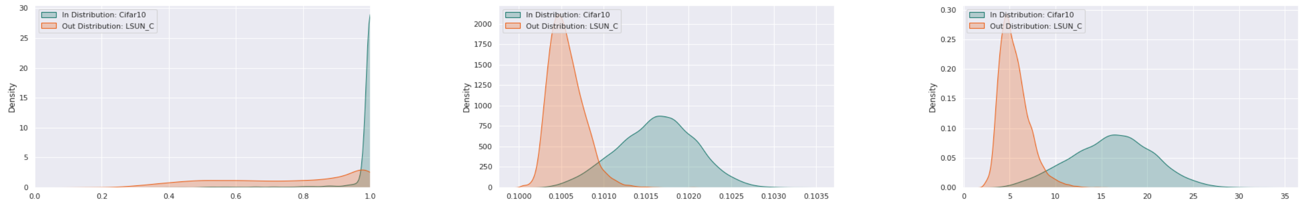


Figure A.10: Distributions over the anomaly scores for CIFAR-10 as ID and LSUN_C as OOD using MSP, ODIN and Energy score respectively

A.2 CIFAR-100

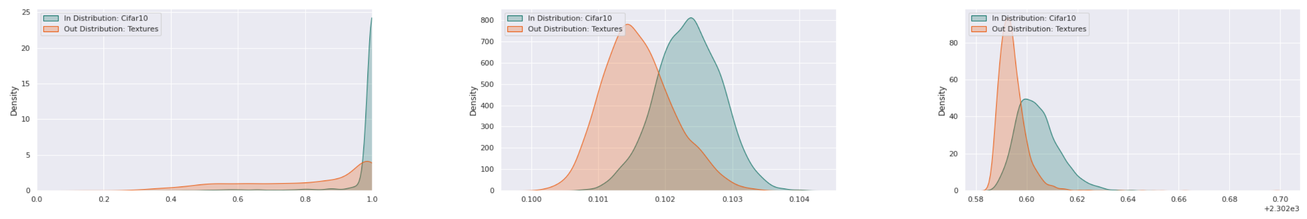


Figure A.11: Distributions over the anomaly scores for CIFAR-100 as ID and Textures as OOD using MSP, ODIN and Energy score respectively

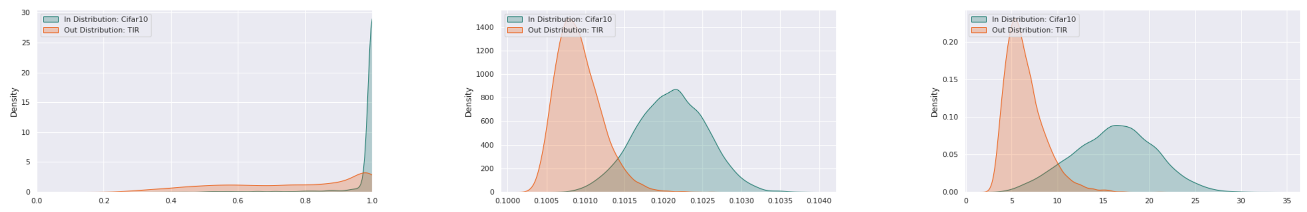


Figure A.12: Distributions over the anomaly scores for CIFAR-100 as ID and TIR as OOD using MSP, ODIN and Energy score respectively

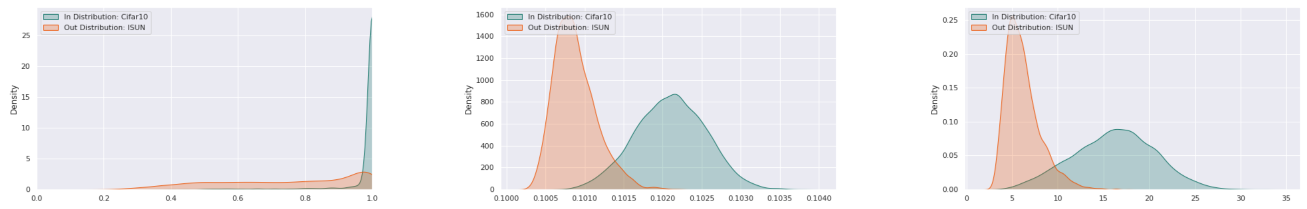


Figure A.13: Distributions over the anomaly scores for CIFAR-100 as ID and ISUN as OOD using MSP, ODIN and Energy score respectively

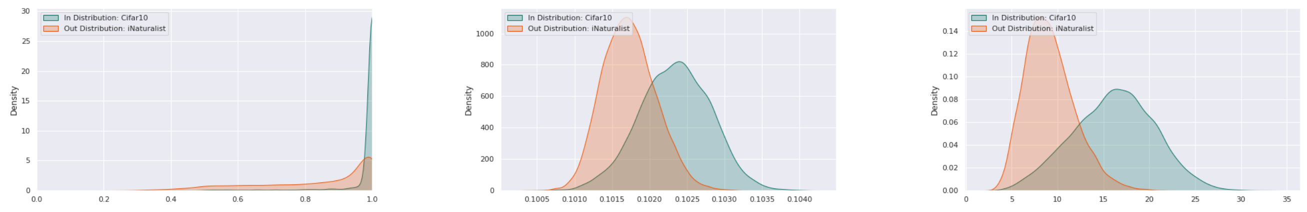


Figure A.14: Distributions over the anomaly scores for CIFAR-100 as ID and iNaturalist as OOD using MSP, ODIN and Energy score respectively

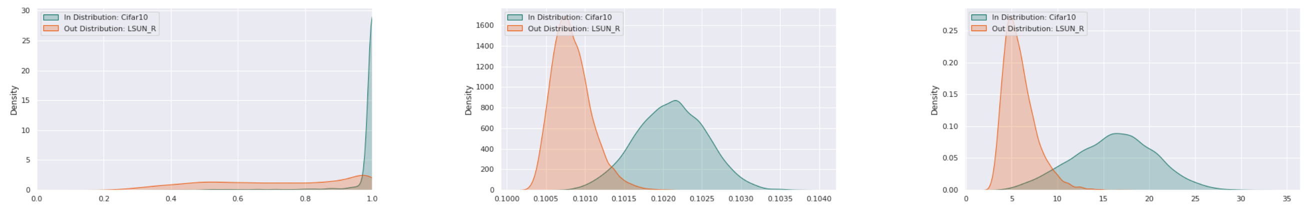


Figure A.15: Distributions over the anomaly scores for CIFAR-100 as ID and LSUN_R as OOD using MSP, ODIN and Energy score respectively

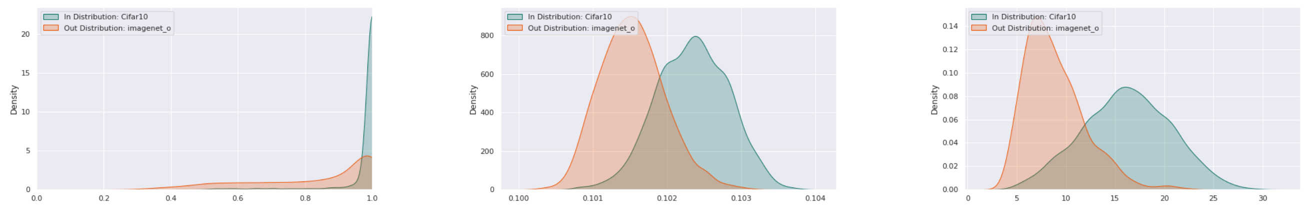


Figure A.16: Distributions over the anomaly scores for CIFAR-100 as ID and ImageNet_o as OOD using MSP, ODIN and Energy score respectively

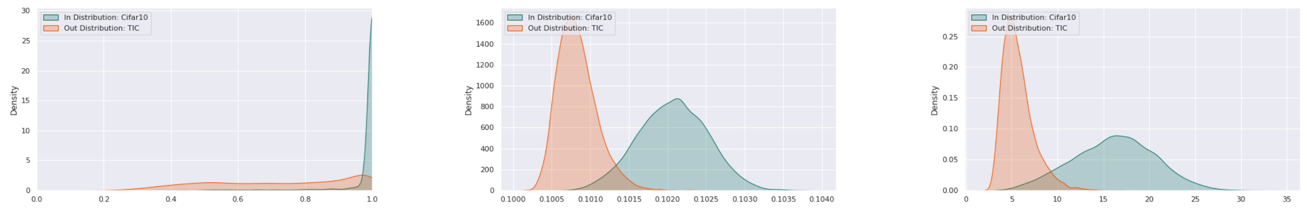


Figure A.17: Distributions over the anomaly scores for CIFAR-100 as ID and TIC as OOD using MSP, ODIN and Energy score respectively

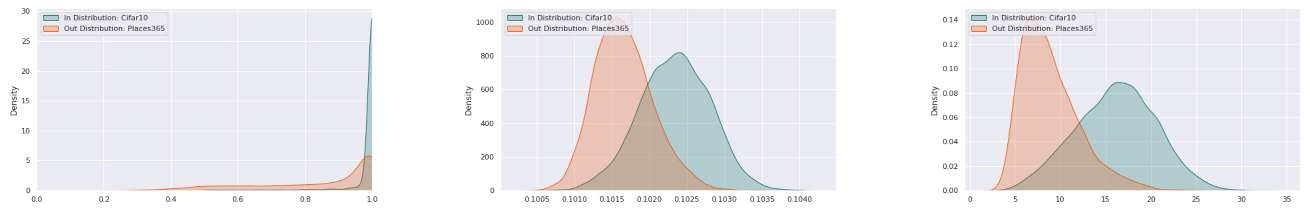


Figure A.18: Distributions over the anomaly scores for CIFAR-100 as ID and Places365 as OOD using MSP, ODIN and Energy score respectively

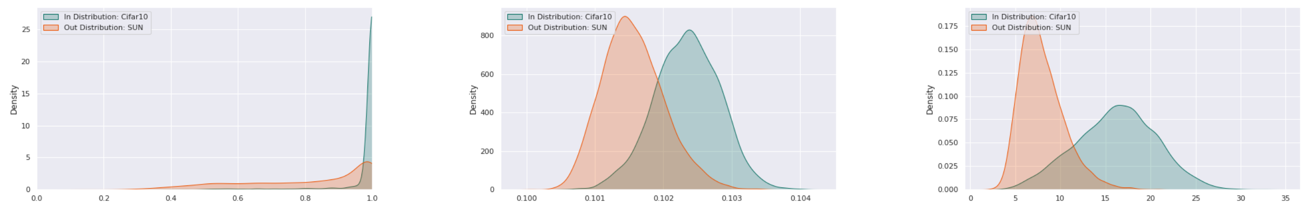


Figure A.19: Distributions over the anomaly scores for CIFAR-100 as ID and SUN as OOD using MSP, ODIN and Energy score respectively

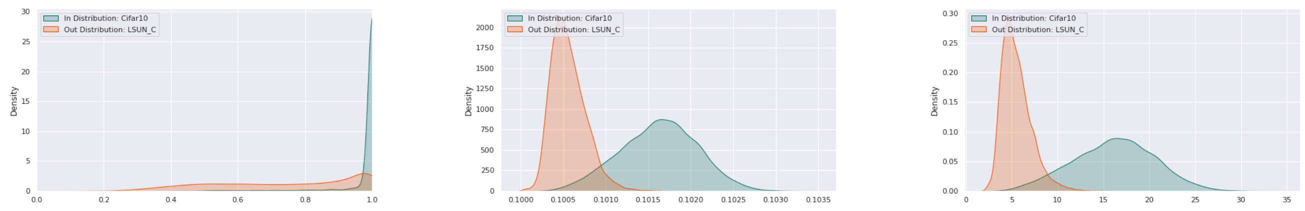


Figure A.20: Distributions over the anomaly scores for CIFAR-100 as ID and LSUN_C as OOD using MSP, ODIN and Energy score respectively

A.3 ImageNet

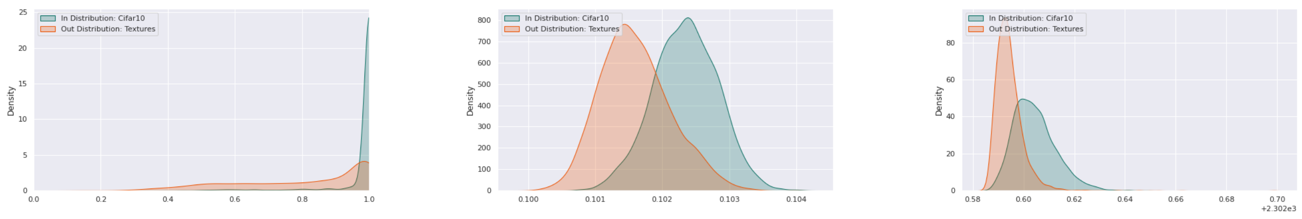


Figure A.21: Distributions over the anomaly scores for ImageNet as ID and Textures as OOD using MSP, ODIN and Energy score respectively

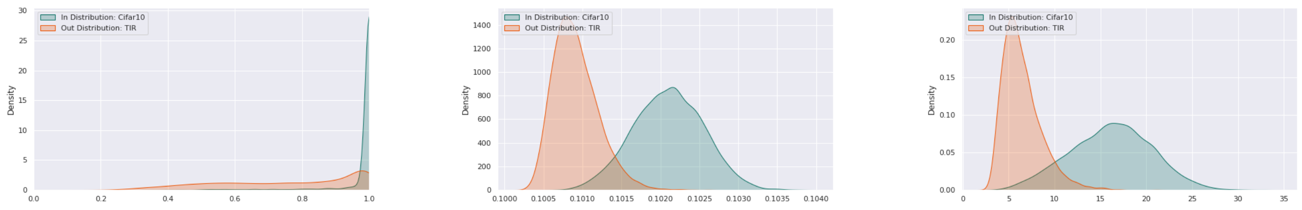


Figure A.22: Distributions over the anomaly scores for ImageNet as ID and TIR as OOD using MSP, ODIN and Energy score respectively

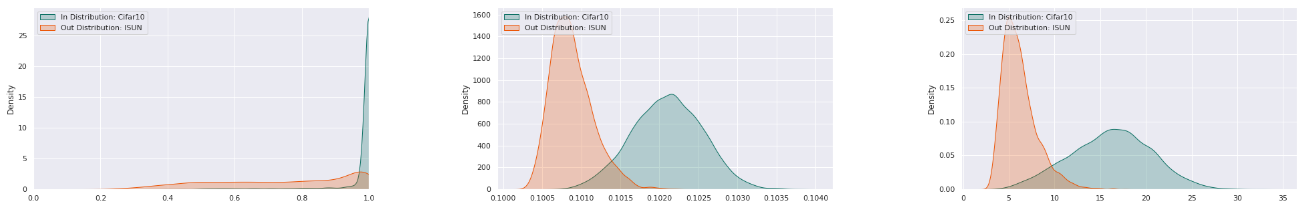


Figure A.23: Distributions over the anomaly scores for ImageNet as ID and ISUN as OOD using MSP, ODIN and Energy score respectively

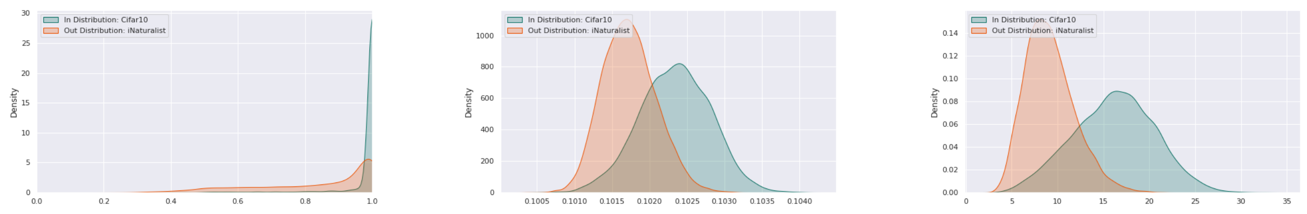


Figure A.24: Distributions over the anomaly scores for ImageNet as ID and iNaturalist as OOD using MSP, ODIN and Energy score respectively

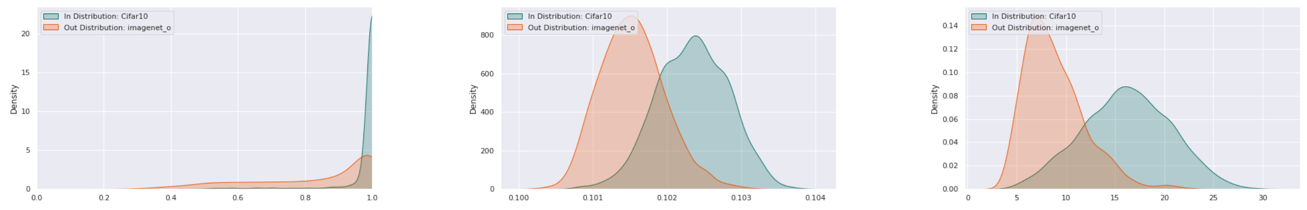


Figure A.25: v for ImageNet as ID and ImageNet_o as OOD using MSP, ODIN and Energy score respectively

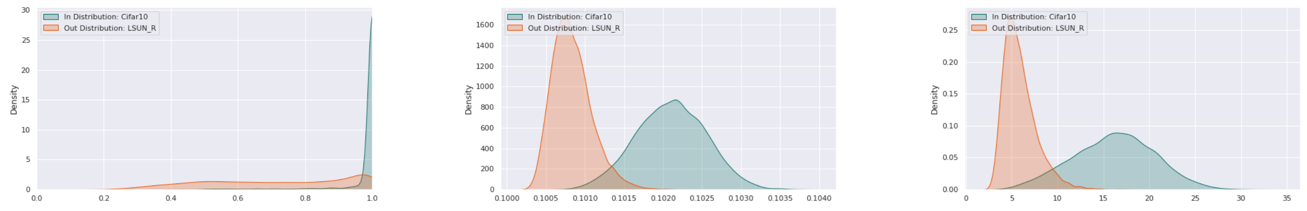


Figure A.26: Distributions over the anomaly scores for ImageNet as ID and LSUN_R as OOD using MSP, ODIN and Energy score respectively

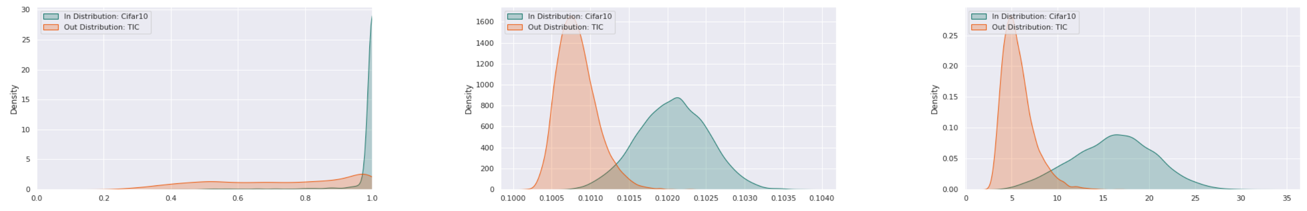


Figure A.27: Distributions over the anomaly scores for ImageNet as ID and TIC as OOD using MSP, ODIN and Energy score respectively

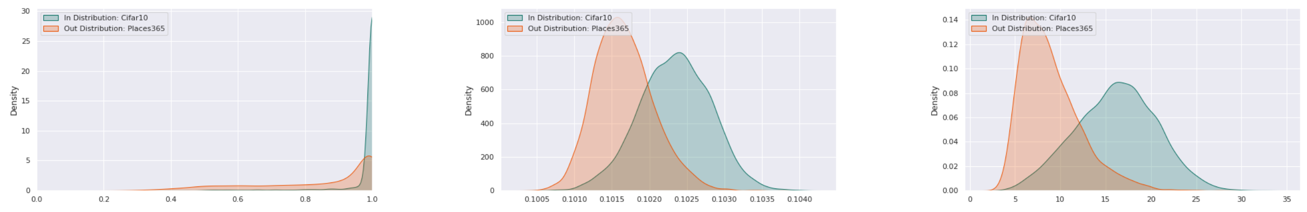


Figure A.28: Distributions over the anomaly scores for ImageNet as ID and Places365 as OOD using MSP, ODIN and Energy score respectively

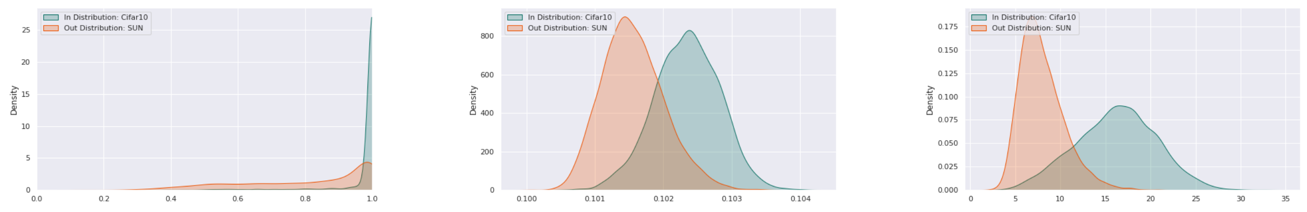


Figure A.29: Distributions over the anomaly scores for ImageNet as ID and SUN as OOD using MSP, ODIN and Energy score respectively

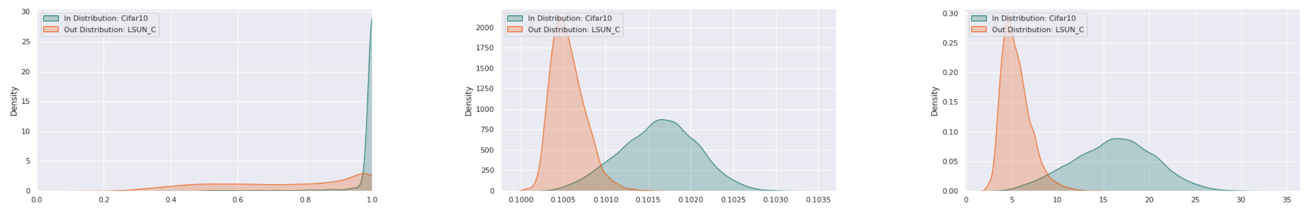


Figure A.30: Distributions over the anomaly scores for ImageNet as ID and LSUN_C as OOD using MSP, ODIN and Energy score respectively