Development and Comparison of Open Set Classification Techniques on ImageNet

Master Thesis

Mike Suter
13-110-671

Submitted on
July 30 2022

Thesis Supervisor
Prof. Dr. Manuel Günther

University of Zurich^UZH
Department of Informatics

Master Thesis

Author: Mike Suter, mike.suter@uzh.ch

Project period: February 01 2022 - August 01 2022

Artificial Intelligence and Machine Learning Group
Department of Informatics, University of Zurich

Acknowledgements

I completed this thesis under the supervision of Prof. Dr. Manuel Günther, Professor at the Artificial Intelligence and Machine Learning (AIML) Group in the Department of Informatics at the University of Zurich, Switzerland. My gratitude goes especially to him for giving me the opportunity to complete my thesis in this exciting field and for continuously providing me with valuable advice and inputs. I am very grateful for the time he took to discuss my questions and review my work. I immensely expanded my knowledge and skills during this period. Furthermore, my thank goes to my brother Remo for continuously supporting me. Lastly, I thank all my colleagues at the University of Zurich, the Professors, as well as the lecturers for providing me with the possibility to learn steadily and grow.

Abstract

In a real-world context, a classifier does not only encounter samples that belong to classes seen during training but also samples whose associated classes are unknown to the model. The task of such a classifier is then to correctly classify samples from known classes and to reject samples that are linked to unknown classes. A classifier that incorporates a mechanism to achieve these two goals is known as an open set classifier/algorithm. For being successful in this task, the dataset that the classifier is trained and evaluated on plays an important role. Ideally, such a dataset is characterized by a variety of classes that are organized in a hierarchical fashion, thereby mimicking a real-world environment. However, since most open set algorithms are developed using small open set datasets with limited diversity, it is unclear how these techniques perform on more challenging datasets. Furthermore, a comparison of the performance of these algorithms is not possible since they are trained on different datasets, using different network topologies and suboptimal evaluation metrics. In this thesis, I conduct a systematic comparison of four relevant open set algorithms: Entropic Open Set Loss (EOS), OpenMax, Extreme Value Machine (EVM), and Placeholders for Open Set Recognition (PROSER). For this comparison, I use three recently developed open set protocols that are based on ImageNet. These ImageNet-based protocols mimic three different open set contexts that systematically vary in their difficulty to perform in. My work shows how these open set algorithms compare to each other in different open set environments and points to the observed strengths and limitations of those approaches.

Zusammenfassung

Ein Bildklassifikationsmodell welches in einer realen Testsituation zur Anwendung gelangt verarbeitet sowohl Bilder von bekannten Klassen, welche Bestandteil des Trainings waren, als auch Bilder von gänzlich unbekannten Klassen. Die Aufgabe des Bildklassifikationsmodell besteht darin, die Bilder von bekannten Klassen der jeweils korrekten Klasse zuzuordnen und die Bilder von unbekannten Klassen zurückzuweisen. Ein Bildklassifikationsmodell welches über einen Mechanismus verfügt der diese beiden Ziele verfolgt, ist auch als Open Set Bildklassifikationmodell/Algorithmus bekannt. Der Datensatz auf welchem das Bildklassifikationsmodell ursprünglich traininert und evaluiert wurde spielt bei der Erreichung dieser Ziele eine wichtige Rolle. Idealerweise verfügt ein solcher Datensatz über eine Vielfalt von hierarchisch organisierten Klassen. Auf diese Weise simuliert der Datensatz eine reale Testumgebung. Es zeigt sich jedoch, dass bei der Entwicklung der meisten Open Set Algorithmen kleine Datensätze verwendet werden, welche durch eine eingeschränkte Vielfalt an Klassen charakterisiert sind. Demnach ist es unklar, ob diese Algorithmen auf umfangreicheren und komplexeren Datensätzen dieselbe Leistung erbringen. Ein Vergleich der Leistungsfähigkeit solcher Algorithmen ist aus verschiedenen Gründen nicht möglich. Einerseits wurden diese auf unterschiedlichen Datensätzen trainiert und evaluiert. Andererseits verwenden diese Algorithmen unterschiedliche Netzwerkarchitekturen und suboptimale Evaluationsmetriken. In dieser Arbeit vergleiche ich vier relevante Open Set Algorithmen auf eine systematisch Weise. Bei den Algorithmen handelt es sich um Entropic Open Set Loss (EOS), OpenMax, Extreme Value Machine (EVM), sowie Placeholders for Open Set Recognition (PROSER). Zur Durchführung dieses Vergleichs verwende ich drei kürzlich entwickelte Open Set Protokolle, welche auf ImageNet Bildern beruhen. Diese Protokolle imitieren drei verschiedene reale Testumgebungen, welche sich systematisch in ihrem Schwierigkeitsgrad unterscheiden. Meine Arbeit zeigt auf, wie sich die ausgewählten Algorithmen in diesen Testumgebungen verhalten. Zudem beleuchtet sie die beobachteten Stärken und Grenzen dieser Techniken.

Acronyms

DNN Deep Neural Network. 1, 8, 9, 11, 12, 14, 16, 19, 32, 38, 41, 42, 62

EVT Extreme Value Theory. 14, 16, 32, 43, 44

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 4, 15, 21, 31, 32, 39

OSCR Open Set Classification Rate. 5, 13, 31, 35–37, 47, 48, 50–53, 55–59, 83–85

OSR Open Set Recognition. 1–3, 5, 7–10, 16–18, 20–22, 28, 31–36, 40, 44

Contents

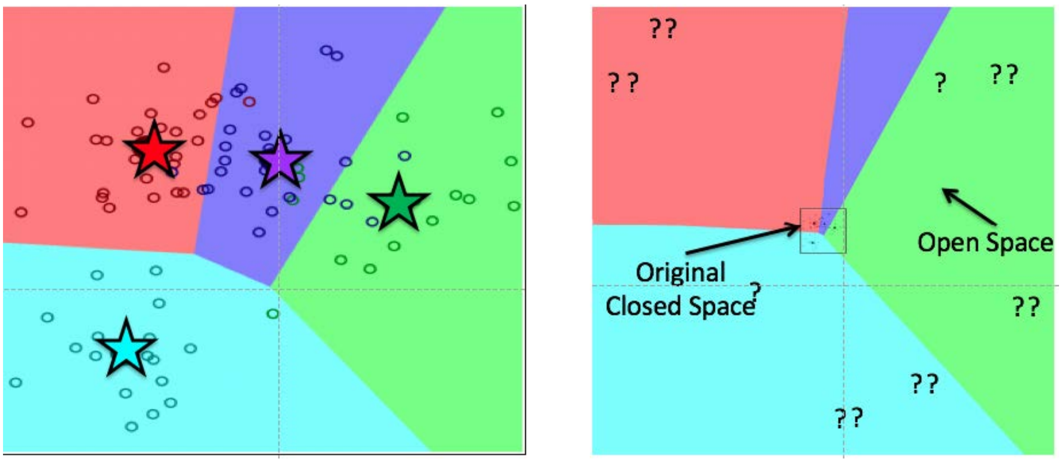
Chapter 1

Introduction

Extensive research in recent years has strongly contributed to the capabilities of deep neural networks (DNNs) in correctly predicting the class that an input image belongs to. This task is generally solvable as long as the network only encounters test samples that belong to categories that have been present during training. However, if the network is required to classify an input image from an unknown class, the task becomes much harder. This scenario is known as open set classification (Scheirer et al., 2013). The task of handling such unknown samples becomes even harder if the classifier has been trained on a dataset that comprises samples from a small number of classes. Therefore, a network can benefit from a dataset that provides a variety of training classes. One such dataset is ImageNet (Deng et al., 2009). This chapter introduces open set classification and points to its associated challenges. Furthermore, it describes the ImageNet database in more detail and argues for its use in open set recognition. The chapter finishes with the problem that this thesis addresses and provides an outline.

1.1 Open Set Classification

A typical Machine Learning-based, supervised classifier is trained on a set of training samples that are associated with a finite set of target classes. Confronted with a test sample from one of these classes, the trained classifier then typically assigns it to the class associated with the highest probability score. Clearly, the task of the classifier is to assign the sample to the correct class. Assuming that the target class of every possible test sample is a member of the set of classes present during training, the model operates under the closed world assumption and is considered to be complete (Rudd et al., 2018; Boult et al., 2019). The classification performance of models on this task increased significantly over the past years (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; Szegedy et al., 2015; He et al., 2016a). However, when the classifier encounters a test sample from a class not present during training, the classification becomes more challenging. As a closed set classifier has no prior knowledge of the novel class, it incorrectly assigns the sample to the most similar class. Encountering novel classes is actually a realistic scenario for classifiers operating in real-world settings. Because of the dynamic and unpredictable nature of real-world contexts, recognition systems must be able to handle novel categories (Bendale and Boult, 2016; Boult et al., 2019). This leads to the notion of designing classifiers that are capable of performing open set recognition (OSR), which refers to the task of recognizing inputs under the (open set) assumption that samples from both known and unknown classes occur during testing. Hence, the knowledge of the model is always incomplete and its design must incorporate a mechanism for recognizing and handling unseen classes (Scheirer et al., 2013; Boult et al., 2019; Geng et al., 2021). According to the formalism of Scheirer et al. (2013), these unknown samples are part of the open space. This open space is characterized by the property that it hosts samples from unknown



(a) Closed set classification

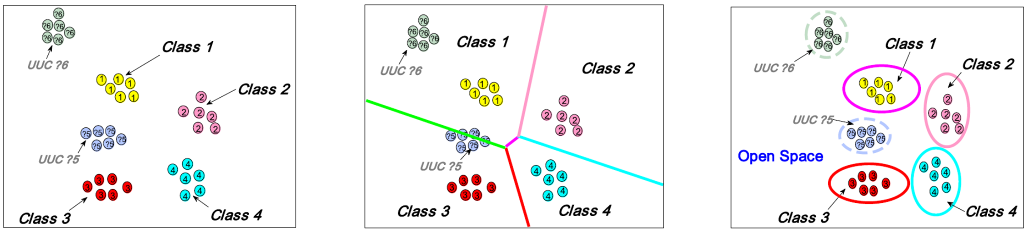
(b) Open set classification

Figure 1.1: CLOSED SET CLASSIFIER OPERATING IN CLOSED VS. OPEN SPACE. This figure illustrates the issue of applying a closed set classifier in an open set context. In (a), the classifier operates in a closed set context and correctly performs the recognition task. In (b), however, the decision boundaries of the same closed set classifier extend into open space and incorrectly partition it. *Source: (Boult et al., 2019)*

classes. Additionally, the distance between these unknown samples and the known samples (that reside in the closed space) is generally far.

When it comes to Machine Learning-based models, Boult et al. (2019) note that most of the systems implicitly operate under the closed world assumption as inputs are assigned to one of the training classes. They further emphasize that this also holds for classifiers that estimate a probability distribution using Bayes' theorem since Bayes' theorem implicitly makes a closed world assumption. Applying such a closed set classifier in an open world setting introduces issues, which Boult et al. (2019) intuitively visualize in figure 1.1. In figure 1.1(a), the learned decision boundaries of a Nearest Class Mean (NCM) model (Mensink et al., 2012) guarantee a decent classification performance in the closed world context. In figure 1.1(b), the closed world assumption is relaxed in favor of an open world assumption, which means that the model may encounter test samples from unknown classes in addition to the samples from the known classes. Now the decision boundaries of the closed set classifier extend into open space and introduce decision regions of unbounded support. As a consequence, samples from emerging unknown classes are incorrectly classified as known classes (Rudd et al., 2018; Boult et al., 2019). If the classification decision is based on the distance of the sample from the class boundaries, the misclassification can be accompanied by a high confidence score (Rudd et al., 2018). Having this issue in mind, Geng et al. (2021) present the idea of loosening the decision boundaries and reserving space for potential unknown classes, as can be seen in figure 1.2. These examples illustrate the need for designing Machine Learning models with a mechanism that allows to recognize and handle samples that are associated with unknown classes. In this sense, the research community identifies two main goals of a classifier operating in an OSR context (Scheirer et al., 2013; Dhamija et al., 2018; Geng et al., 2021):

1. Correctly classifying samples from known classes
2. Rejecting samples from unknown classes



(a) Data distribution

(b) Conventional decision boundaries

(c) Loosened decision boundaries

Figure 1.2: LOOSENING DECISION BOUNDARIES IN OSR. This figure illustrates the idea of a classifier operating in an open set context. In (a), the original data distribution is shown, including the unknown classes ?5 and ?6 that appear only during testing. The decision boundaries of a traditional classifier are illustrated in (b). Furthermore, (c) visualizes the adjusted decision boundaries, reserving space for the unknown classes. *Source: (Geng et al., 2021)*

Referring to the open set context and the two associated problems a classifier needs to master, different formalisms can be found in the literature (Scheirer et al., 2013; Dhamija et al., 2018; Geng et al., 2021). In order to better structure the open set problem, Dhamija et al. (2018) define $\mathcal{Y} \subset \mathbb{N}$ to be the label space that contains all classes. They then divide this space into the subsets:

- $\mathcal{C} = \{1, \ldots, C\} \subset \mathcal{Y}$: the *known classes* that the network needs to identify.
- $\mathcal{U} = \mathcal{Y} \setminus \mathcal{C}$: the *unknown classes* that need to be rejected by the network. $\mathcal{U}$ is further divided into:
  - $\mathcal{B} \subset \mathcal{U}$: the *known unknown classes* as the set of classes that the network can use during training. However, the network does not need to recognize any of these classes since $\mathcal{B} \cap \mathcal{C} = \emptyset$.
  - $\mathcal{A} = \mathcal{U} \setminus \mathcal{B} = \mathcal{Y} \setminus (\mathcal{C} \cup \mathcal{B})$: the *unknown unknown classes* whose samples are not available during training, but during testing. An open set classifier is expected to reject those samples.

The authors then continue to define the different datasets used in the context of OSR as:

- $\mathcal{D}_b^{'}$: set of training samples belonging to $\mathcal{B}$
- $\mathcal{D}_b$: set of test samples belonging to $\mathcal{B}$
- $\mathcal{D}_a$: set of test samples belonging to $\mathcal{A}$
- $\mathcal{D}_c^{'}$: set of training samples belonging to $\mathcal{C}$
- $\mathcal{D}_c$: set of test samples belonging to $\mathcal{C}$
- $\mathcal{D}_u = \mathcal{D}_b \cup \mathcal{D}_a$: set of all unknown test samples

In this thesis, I adopt the aforementioned formalism by Dhamija et al. (2018). When referencing the training and test set as a whole (i.e., irrespective of their composition), I use $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$, respectively. Furthermore, similar to Geng et al. (2021), I utilize the following abbreviations: *KKCs* for the set of known known classes, *KUCs* for the set of known unknown classes, and *UUCs* for the set of unknown unknown classes.

1.2 ImageNet Database

Depending on the context in which an open set classifier is applied, the demands on the dataset the classifier is trained on vary. Ideally, a dataset that is used to train and evaluate the performance of an open set classifier should be similar to a real-world open space. Recalling the two main goals of an open set classifier from section 1.1, correctly classifying samples from known classes and rejecting samples from unknown classes, such a dataset needs to satisfy two criteria. Firstly, the known classes in the dataset should capture the entities a classifier is very likely to encounter in its application domain in order to have an a priori knowledge of possible classes. Secondly, the dataset should also contain unknown classes, which can be encountered in a real-world context by the classifier with a certain likelihood. The state-of-the-art database in research that provides such a required richness of classes and its associated data samples is ImageNet[1] (Deng et al., 2009). ImageNet is a large[2] compilation of images that have been collected with the goal of establishing a benchmark dataset that can be used by researchers to train and evaluate increasingly more sophisticated algorithms. The images are quality controlled and follow the structure of the WordNet hierarchy. WordNet (Miller, 1995; Fellbaum, 1998) represents a large lexical database that consists of English nouns, verbs, adjectives, and adverbs. The words are organized into groups of cognitive synonyms, called synsets. These synsets express distinct concepts and are interlinked by semantic and lexical relations. The relations allow a hierarchical organization of synsets. Deng et al. (2009) design ImageNet such that it utilizes the hierarchical structure of the WordNet synsets and organizes more than 20’000 [3] noun synsets in the form of a tree. In this tree, a synset represents a concept and comes in the form of a node that is linked to hundreds or even thousands of images. While the leaf nodes represent the most distinct concepts, the parent nodes represent increasingly more abstract concepts. For an example, please refer to the paper published by the authors. ImageNet is the data foundation of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC, Russakovsky et al., 2015) 2010-2017, a competition held among researchers to compare the progress of algorithms in the field of computer vision. The ILSVRC uses carefully selected subsets of the ImageNet database and partitions those subsets into a training, a validation, and a test set.

1.3 Problem Formulation

As indicated in sections 1.1 and 1.2, an open set classifier benefits from a training dataset that mimics a real-world scenario. Because of its hierarchical richness, ImageNet forms a suitable basis for designing open set datasets that reproduce real-world contexts. Such datasets can then be used to train and evaluate open set algorithms and to achieve solid benchmarks. However, as Geng et al. (2021) note, most open set algorithms in research are trained on small datasets such as LETTER (Frey and Slate, 1991), MNIST (Lecun et al., 1998), notMNIST (Bulatov, 2011), NIST (Grother and Kayee, 2016), SVHN (Netzer et al., 2011), or CIFAR-10/100 (Krizhevsky, 2009). Due to their size and limited diversity, these datasets are not ideal for mimicking real-world contexts and thus the expressiveness of the performance measures on them can not be readily generalized. Furthermore, existing open set algorithms are not consistently trained on the same datasets and network topologies, which impedes a comparison among them. Finally, the evaluation of these algorithms on the test set is often based on an evaluation metric that does not ideally reflect how

---

[1] `https://www.image-net.org/index.php`

[2] by the time of writing this thesis, the database comprises 14,197,122 images

[3] by the time of writing this thesis, 21’841 noun synsets are indexed

well an algorithm achieves the goals of OSR. In this thesis, I address these issues. Concretely, I address the two following research questions:

1. *How do relevant open set algorithms compare to each other when they are trained and evaluated on a data basis that mimics an instance of a real-world open space?*
2. *How does the performance of such an open set algorithm develop if the algorithm is trained and evaluated on datasets that represent instances of real-world open spaces which are associated with different levels of difficulty of the open set task?*

I address the first research question by conducting a comparison between a selection of open set algorithms that are trained and evaluated on the same data basis, using a uniform network architecture. The comparison is based on an evaluation metric introduced by Dhamija et al. (2018). Concretely, I compare the open set algorithms Entropic Open Set Loss (Dhamija et al., 2018), OpenMax (Bendale and Boult, 2016), EVM (Rudd et al., 2018), and PROSER (Zhou et al., 2021b). These algorithms differ considerably in their approach how to address the OSR task. The data basis comes in the form of three open set protocols that have recently been designed by Bhoumik (2021). These ImageNet-based protocols mimic three different open set contexts that systematically vary in their difficulty to perform in. By comparing the relative (to the other algorithms) performance of a selected algorithm across the three protocols, I address the second research question.

With this comparison, I contribute to a better understanding of how these algorithms compare to each other in a complex and difficult environment. Applying such an algorithm in a more challenging open set context strengthens the generalizability of the results. Furthermore, my work points out the possible strengths and limitations of the evaluated algorithms when it comes to applying them in increasingly more difficult open set contexts. By relying on the OSCR curve (Dhamija et al., 2018) as an evaluation metric, I report performance measures that better capture the nature of OSR than the evaluation metrics usually found in research papers. It is, to my best knowledge, the first time that such a comparison has been performed.

The content of this thesis can be outlined as follows:

- **Related Work (Chapter 2)**: This chapter introduces general techniques proposed by research to address the task of OSR as well as the theoretical background of the algorithms I evaluate. Furthermore, the work by Bhoumik (2021), providing the open set protocols, is introduced.
- **Approach (Chapter 3)**: Here, I show why a direct comparison of the evaluated open set algorithms based on the research papers is difficult. I present my solution proposal to overcome these challenges as well as my contribution.
- **Experimental Setup (Chapter 4)**: This chapter presents the framework of the comparison, which includes the utilized hard- and software, the network architecture, as well as the hyperparameters that need to be specified when running the algorithms. I furthermore offer a justification for my taken decisions.
- **Experiments (Chapter 5)**: Subject of this chapter is the execution of the algorithms, the selection of the final model, and the presentation of the results.
- **Discussion (Chapter 6)**: This chapter discusses and interprets the experimental results and points to potential limitations of the work.
- **Conclusion and Future Work (Chapter 7)**: The focus of this chapter is the summary of the results and gained insights as well as a reflection on possible future research work.

Chapter 2

Related Work

Building on the understanding of the open set recognition problem and the ImageNet database as introduced in chapter 1, this chapter presents important research results associated with the two topics. Specifically, section 2.1 provides a general overview of various open set algorithms as well as a detailed insight into the theoretical background of the algorithms I evaluate in this thesis. Furthermore, the same section draws the line between OSR and related but distinct problems. Section 2.2 then introduces the work of Bhoumik (2021), whose protocols I use for the experiments. Finally, section 2.3 outlines ResNet (He et al., 2016a), a family of network architectures that I utilize in my experiments.

2.1 Open Set Techniques

The origins of OSR date back to Scheirer et al. (2013), which initially provide a formalization of the OSR problem as the minimization of the open set risk:

$$\underset{f \in \mathcal{H}}{\operatorname{argmin}} \left\{ R_{\mathcal{O}}(f) + \lambda_r R_{\mathcal{E}}(f(\hat{V} \cup \hat{K})) \right\} \tag{2.1}$$

Thereby, the authors define an open space risk function $R_{\mathcal{O}}(f)$ that, given a recognition function $f(x)$, returns the fraction of positively classified samples x from the open space with respect to the positively classified samples x from the entire space in consideration (i.e., including the closed space). Hence, the more samples $x \in \mathcal{O}$ are erroneously classified as one of the *KKCs*, the higher the open space risk becomes. Further, $R_{\mathcal{E}}(f)$ denotes the empirical risk function that returns the error measure on the training data with $\hat{V} = D_c^{'}$ and $\hat{K} = D_b^{'}$. The regularization constant λ_r balances the two risk terms. Therefore, the OSR problem is solved by finding a recognition function $f \in \mathcal{H}$ that minimizes the open set risk. Reflecting on this definition, it becomes clear how two main goals of an open set classifier, as indicated by Geng et al. (2021), emerge. Namely, the open space risk $R_{\mathcal{O}}(f)$ represents the demand on the classifier f to correctly reject samples from unknown classes while the empirical risk $R_{\mathcal{E}}(f)$ illustrates the requirement for the classifier f to correctly classify samples from known classes. Based on this formal definition of the OSR problem, Scheirer et al. (2013) propose a model, the 1-versus-Set machine, that includes a mechanism to deal with emerging samples from the open space.

In the decade since the seminal work of Scheirer et al. (2013), the body of research regarding OSR has grown steadily. Associated with it, different models and approaches to tackle the OSR problem have emerged. Recently, Geng et al. (2021) published a comprehensive survey on the research advances in open set recognition. Emanating from the point of view of modeling, the paper provides a categorization of existing models/approaches that all incorporate mechanisms to deal with the OSR problem. This categorization originates from two different types of models that both are divided into subcategories:

- Discriminative Models
    - Traditional Machine Learning Models
    - DNN Models
- Generative Models
    - Instance Generation-based Models
    - Non-Instance Generation-based Models

As Ng and Jordan (2001) explain, discriminative classifiers learn a function that, given a data sample x, computes the probability for a class label $c \in \{1, \ldots, C\}$, i.e., $p(c|x)$. Generative models, in contrast, model the joint probability function $p(x, c)$ that maps a data sample x and a class label c to the associated probability. Based on the above categorization, Geng et al. (2021) give an overview of the published works for every category. When it comes to approaches that belong to the family of discriminative models, they note that the vast majority of them rely on an empirically-set classification threshold. The following paragraphs outline, for every category, some of the approaches.

Traditional Machine Learning Models: As Geng et al. (2021) elucidate, traditional Machine Learning models are generally characterized by the assumption of the training set $\mathcal{D}_{train}$ and the test set $\mathcal{D}_{test}$ being drawn from the same distribution. Since this assumption does not hold anymore in an OSR context, an adaptation of such traditional machine learning models is necessary. In this respect, various works modify a Support Vector Machine (SVM, Cortes and Vapnik, 1995) such that it is able to address the challenges associated with an open set context. Scheirer et al. (2013) introduce the 1-versus-Set machine, an SVM that integrates an open space risk term. The updated model leads to an additional hyperplane that is parallel to the separating hyperplane and introduces a slab in the feature space. Depending on whether a sample $x \in \mathcal{D}_{test}$ is located between the hyperplanes or on a particular side of the slab, the sample is classified as the target class, the non-target class, or is rejected. In a similar manner, Cevikalp (2017) presents the Best Fitting Hyperplane Classifier (BFHC) model that uses the same idea of generating a slab in the feature space. Furthermore, the model is applicable to a non-linear case and is designed for large-scale problems. Motivated by the goal of further reducing the open space risk, Scheirer et al. (2014) propose a Weibull-calibrated SVM (W-SVM) model. This model applies statistical extreme value theory (EVT, Fisher and Tippett, 1928) together with two SVM models. The authors intend to provide a model that takes the distance between a test sample and the space occupied by training samples into account, when estimating the probability of class membership. Specifically, this probability declines in a non-linear fashion when test samples migrate into open space. Another algorithm that relies on EVT, the P_I-SVM, is suggested by Jain et al. (2014). Concretely, for any known training class, the authors apply the statistical EVT to positive training samples at the decision boundary to avoid overfitting to the positive training data and to simplify the rejection of samples from unknown classes. The concepts of the EVT are also utilized by Zhang and Patel (2017), even though their proposed model is not based on an SVM. Specifically, with the sparse representation-based open set recognition model (SROSR), the authors introduce a model that is applicable in the field of computer vision. SROSR makes use of the EVT to model the tails of two reconstruction error distributions. After computing the reconstruction errors for a test sample, the fitted models can then be used to classify or reject the test sample. Motivated by the realization that most OSR methods do not take advantage of distributional information that is present in the training data, Rudd et al. (2018) present the Extreme Value Machine (EVM). This algorithm, which uses EVT as well, is explained in detail in subsection 2.1.4. Yet another approach that falls in the category of traditional Machine Learning models focuses on the distance between a test sample and the means of the known training classes. Specifically, Bendale and Boult (2015) design the Nearest Non-Outlier (NNO) algorithm. This algorithm is linked to the Nearest Class Mean (NCM, Mensink et al., 2013; Ristin et al., 2014) classifier and evaluates a test sample on every classifier that is linked to a training class. In case of all classifiers rejecting the test sample, it is considered to belong to an unknown class. Furthermore, the authors formulate the algorithm in a way such that it is capable of dynamically adding new classes when manually labeled data is provided. A different approach is taken by Vareto et al. (2017) as they address the challenges associated with open set face recognition. In doing so, they devise two algorithms, HPLS and HFCN, that couple hashing functions with classification methods such as partial least squares (Wold, 1985) and fully connected networks.

DNN Models: In contrast to the category of traditional Machine Learning models, the category of DNN models comprises approaches that use a DNN. One such model is OpenMax (Bendale and Boult, 2016). This model, which substitutes the SoftMax layer with an OpenMax layer, is discussed in detail in subsection 2.1.3. Similarly, the Entropic Open Set Loss (Dhamija et al., 2018), a loss function that modifies the probability distribution at the Softmax layer, is examined in subsection 2.1.2. While OpenMax and Entropic Open Set Loss concentrate on image classification, Shu et al. (2017) use a convolutional neural network and substitute the Softmax layer with a 1-vs-rest layer consisting of sigmoid functions to classify text documents in an open set context. The authors name their approach DOC and show via experiments its potential in open text classification. In a similar manner, Kardan and Stanley (2017) present a neural network architecture that incorporates a specialized output layer, called competitive overcomplete output layer (COOL). As the authors explain, the idea of COOL is to associate a collection of competing output units with each training class. This architecture should then prevent the network from overgeneralizing over open space regions.

Instance Generation-based Models: Turning to the family of generative models, the category of instance generation-based models comprises models that are usually also based on a DNN. However, in contrast to models from the category of DNN models, these methods aim at synthesizing samples that belong to *KUCs*, hoping that the distribution of the generated samples is as similar as possible to the distribution of samples from *UUCs*. In general, methods that fall under this category are inspired by the technique of adversarial learning, which has been devised by Goodfellow et al. (2014a). Concretely, the authors design a framework that comprises a generative model and a discriminative model. While the generative model synthesizes samples, the task of the discriminative model is to recognize whether a sample has been generated by the generative model or belongs to the training data. The authors term this architecture a generative adversarial network (GAN). Applying these concepts, Neal et al. (2018) present a data augmentation technique that makes use of GANs and is called counterfactual image generation. With this technique, the authors generate unknown samples that resemble training samples from *KKCs*, but are actually not associated with any of the *KKCs*. These generated samples are then used in the training procedure, which allows a reformulation of the OSR task. Similarly, Yu et al. (2017) design a framework for the OSR task. This framework is known as the adversarial sample generation (ASG) framework and is able to synthesize samples from *KKCs* as well as samples from *KUCs* (i.e., negative samples). As the authors point out, the generated negative samples are similar to the samples from the training set and are used to train an open set classifier. Yet another approach has lately been proposed by Zhou et al. (2021b) and introduces data placeholders as well as classifier placeholders. This approach is examined in detail in subsection 2.1.5.

Non-Instance Generation-based Models: With the collective decision-based OSR model (CD-OSR), Geng and Chen (2022) present an approach that adapts hierarchical Dirichlet process (Teh et al., 2006; Teh, 2010) to the OSR task. As Geng and Chen (2022) point out, this model is characterized by appealing properties. First, CD-OSR is able to circumvent the selection of a decision threshold, a procedure that usually incurs risks since it only takes the information from *KKCs* into account. Furthermore, in contrast to many other existing OSR methods, their model incorporates a mechanism for explicitly modeling samples from *UUCs* that appear during testing, which leads to a new class discovery function.

The preceding paragraphs gave an overview of a broad spectrum of models that all address the OSR challenges. However, my thesis focuses on a particular selection of approaches. Namely, the Entropic Open Set Loss (Dhamija et al., 2018), OpenMax (Bendale and Boult, 2016), the Extreme Value Machine (Rudd et al., 2018), and PROSER (Zhou et al., 2021a). These approaches build the basis for the experiments. In chapter 3, I give a justification for this particular selection. Inspired by the above categorization of Geng et al. (2021), I apply the following typology to the selected algorithms:

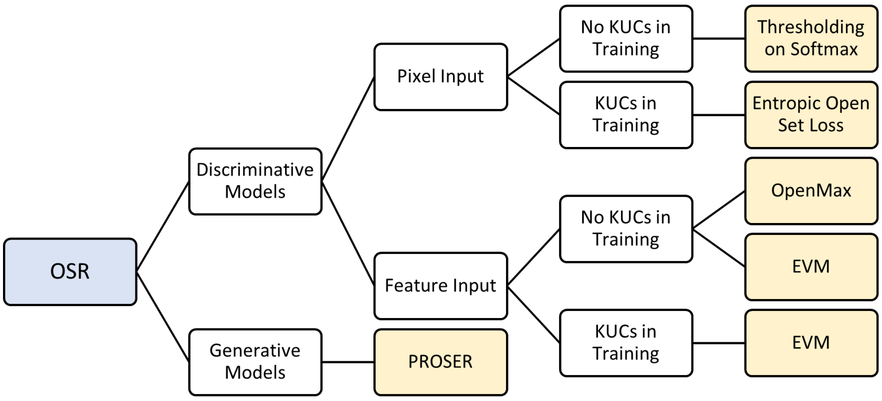


Figure 2.1: TYPOLOGY OF EVALUATED ALGORITHMS.

In the typology depicted in figure 2.1, *Pixel Input* refers to the fact that the input to the model/network comes in the form of an image. In contrast, *Feature Input* denotes the situation where the algorithm builds the model based on deep feature representations of the input images. Furthermore, *KUCs in Training* implies that samples from *KUCs* are used for training, i.e., the model already gains some information on unknown classes. The leaf nodes of the tree refer to the evaluated algorithms. Note that EVM can be trained with and without *KUCs*. However, as explained later in section 5.1, I train the EVM model without *KUCs* for my experiments. The theoretical background of the selected algorithms is presented in subsections 2.1.1 - 2.1.5. Finally, in subsection 2.1.6, I differentiate OSR from related concepts to refine the understanding of the OSR problem.

2.1.1 Thresholding on Traditional Softmax

As noted by Bendale and Boult (2016), many DNNs (Krizhevsky et al., 2012; Chatfield et al., 2014; Szegedy et al., 2015) pass the scores produced by the last fully-connected layer to the Softmax function (Bridle, 1990) that then generates a probability distribution over the C classes present during training. As Goodfellow et al. (2016) illustrate in more detail, the Softmax function takes the network output in the form of real-valued logit values as input and computes the class-wise probability scores. In a closed set context, the class associated with the highest probability score is then considered as the prediction of the network. Formally, the Softmax probability y_c for a training class $c \in \mathcal{C}$ is computed using the logits $z_{c'}$ as:

$$y_c = \frac{e^{z_c}}{\sum\limits_{c'=1}^{C} e^{z_{c'}}} \tag{2.2}$$

Although this definition particularly fits the requirements of a closed set context, it can and has been used to reject samples from unknown classes via thresholding for many years (Matan et al., 1990; De Stefano et al., 2000). In thresholding, first a rejection threshold $\theta \in [0, 1]$ is defined. For a specific input sample in an open set classification context, the maximum Softmax output $y^* = \max y_c$ is then compared against this threshold. If $y^* < \theta$, the input sample is rejected and considered to be unknown. Although being an elegant solution, this approach has some shortcomings. As Matan et al. (1990) and Boult et al. (2019) bring up, relatively small absolute differences between the logit values can lead to much more pronounced differences between the Softmax scores, thereby reflecting a bias toward a particular class. Another issue that can occur in association with the thresholding approach is linked to the underlying model. Assuming a classical deep convolutional neural network whose final layer outputs the logit scores over the C known training classes, a test sample that belongs to the *UUCs* can lead to a high maximum probability score. More concretely, an optimal network recognizes unknown classes, even when these unknown classes share similarities with known training classes. As a consequence, such a network could seek to equalize the logit values, which would then avoid a high maximum probability score. This in turn enables a lower rejection threshold. However, if this previous assumption about the network is not satisfied, the network may produce a relatively high logit value for the most similar class. This in turn increases the likelihood that the unknown samples are not rejected, since the required rejection threshold would not be appealing for operational use. Hence, thresholding on Softmax scores would not be very effective.

As indicated above, the probability distribution over the C known training classes plays an important role in the threshold-based rejection of unknown samples. Dhamija et al. (2018) present an approach that aims at this probability distribution. This approach is the focus of the next subsection.

2.1.2 Entropic Open Set Loss

With Entropic Open Set (EOS) loss and Objectosphere loss, Dhamija et al. (2018) introduce two novel loss functions whose ultimate goal are to make the rejection of unknown samples $x \in \mathcal{D}_u$ based on a threshold more effective by modifying the probability distribution at the Softmax layer (i.e., $P(c|x)$ for $c \in \mathcal{C}$). Concretely, the authors achieve this by formulating the loss functions such that the entropy at the Softmax layer for unknown samples is maximized. Hence, when a test sample $x \in \mathcal{D}_u$ is evaluated, the Softmax scores across the *KKCs* should ideally be balanced out and a rejection based on a threshold θ should therefore become easier. Assuming a DNN architecture in which a fully connected layer maps the convolutional output into a deep feature space

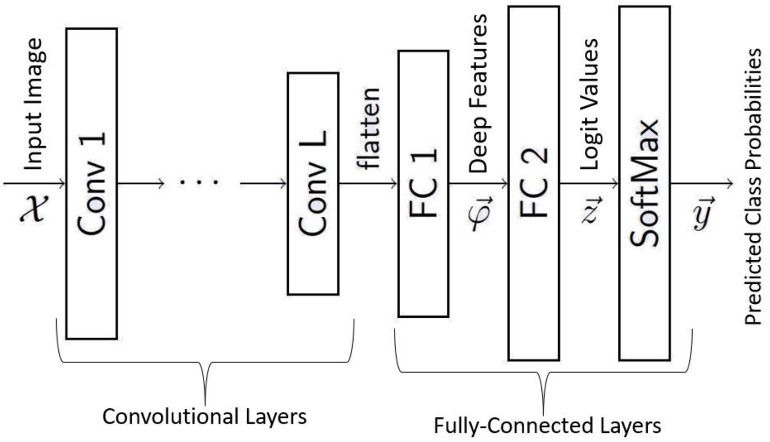


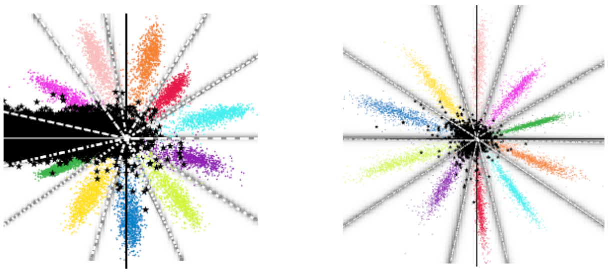
Figure 2.2: NETWORK ARCHITECTURE WITH DEEP FEATURE LAYER. This figure illustrates an exemplary convolutional DNN. The fully connected layer FC1 maps the convolutional output into deep feature space. Subsequently, the fully connected layer FC2 translates the deep features into logits, which are then passed to the Softmax layer. *Source: (Bhoumik, 2021)*

(see figure 2.2), Dhamija et al. (2018) show that their proposed loss functions lead to deep features that better discriminate between known samples and unknown samples.

When it comes to Entropic Open Set loss, Dhamija et al. (2018) decide to use the open space information from known unknown samples $x \in \mathcal{D}_b'$ for training the network. However, they opt against an additional background class for all unknown samples at the Softmax layer to avoid the mapping of unknown samples into a dedicated region of the feature space. Such a dedicated region can be problematic since it suggests a similarity to certain known classes that does not necessarily reflect the actual similarity, thereby providing a suboptimal separation. The authors visualize this situation in figure 2.3(a): while each colored dot represents a sample in 2-d deep feature space from one of the ten known classes, the black dots represent unknown samples $x \in \mathcal{D}_a$. Besides a considerable overlap with the known samples, the unknown samples occupy a certain region in the deep feature space, indicating some similarity with the green and pink samples that might not necessarily be given. In addition to this problematic deep feature space partitioning when using a background class, the authors observe a tendency of a smaller deep feature magnitude for unknown samples $x \in \mathcal{D}_a$ compared to known samples $x \in \mathcal{D}_c$ when applying a network that has been trained using plain Softmax over the C known training classes. Based on this latter observation, Dhamija et al. (2018) hypothesize that the magnitude of the deep feature vector serves as an indicator of a sample being unknown. Consequently, they design the Entropic Open Set loss function in such a way that the entropy of Softmax scores for unknown samples $x \in \mathcal{D}_b'$ is maximized. Formally, the definition of the Entropic Open Set loss function $J_E(x)$ distinguishes between a loss term for known samples and one for unknown samples:

$$J_E(x) = \begin{cases} -\log\ S_c(x) & \text{if } x \in \mathcal{D}_c' \text{ is from class } c \\ -\frac{1}{C}\sum\limits_{c=1}^{C} \log\ S_c(x) & \text{if } x \in \mathcal{D}_b' \end{cases} \tag{2.3}$$

Concretely, Dhamija et al. (2018) assign a known training sample $x \in \mathcal{D}_c'$ from class $c \in \mathcal{C}$ the loss that results from taking the negative logarithm of the standard Softmax output $S_c(x) =$



(a) Background Class (b) Objectosphere Loss

Figure 2.3: DEEP FEATURE REPRESENTATION OF KNOWNS AND UNKNOWNS. In (a), it is illustrated where in 2-d feature space the known (= colored dots) and unknown (= black dots) test samples are mapped to when a LeNet++ network is trained with a background class on MNIST ($\mathcal{D}_c^{'}$) and NIST ($\mathcal{D}_b^{'}$). The idea of reducing the magnitude of the deep feature vector is represented in (b). Assuming a 2-d feature space and a network trained using the Objectosphere loss, the unknown test samples should ideally be mapped to locations close to the origin. *Source: (Dhamija et al., 2018)*

$\frac{e^{l_c(x)}}{\sum_{c' \in \mathcal{C}} e^{l_{c'}(x)}}$, with $l_c(x)$ denoting the logit value for class c. In contrast, the loss of an unknown training sample $x \in \mathcal{D}_b^{'}$ considers the logarithm of the standard Softmax output for all known classes. As formally shown by the authors, the loss $J_E(x)$ for an unknown training sample $x \in \mathcal{D}_b^{'}$ is minimized if the class-wise Softmax scores are identical, i.e., $\forall c \in \mathcal{C} : S_c(x) = \frac{1}{C}$. Consequently, assuming no bias term in the logit layer, identical Softmax scores imply that a deep feature vector $F(x) = \vec{0}$ represents a possible minimum of the associated loss term. This idea of a small magnitude of the deep feature vector is illustrated by the authors in figure 2.3(b): it is recognizable that unknown samples group around the origin in 2-d deep feature space, thereby separating the known from the unknown classes in the feature space without implying a particular similarity to one of the *KKCs*.

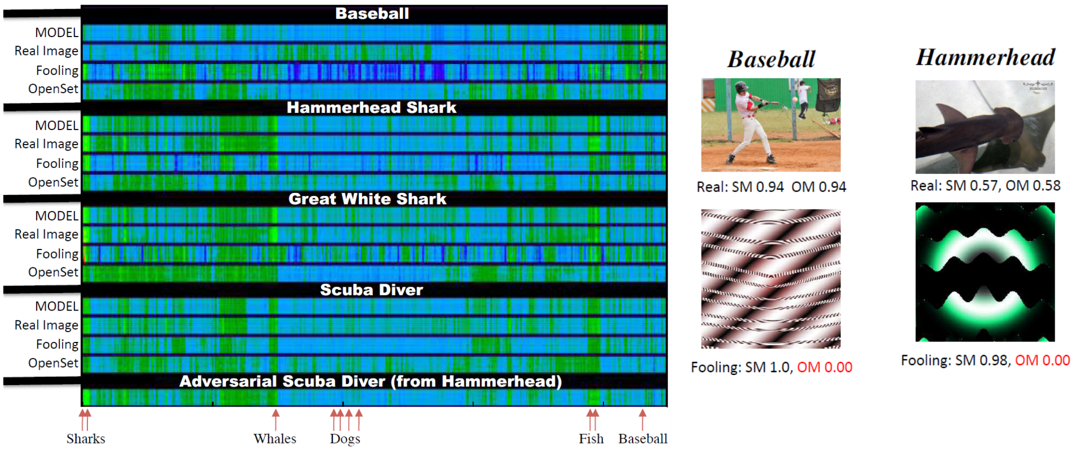
Having laid the theoretical foundation for their approach, Dhamija et al. (2018) then test the capabilities of their approach by conducting two experiments with various datasets. Using the LeNet++ network (Wen et al., 2016), the first experiment relies on MNIST digits as $\mathcal{D}_c^{'}$ and NIST letters as $\mathcal{D}_b^{'}$ in the training procedure. For testing, MNIST letters serve as $\mathcal{D}_c$ and the Devanagari (Pant et al., 2012), CIFAR-10, and notMNIST datasets as $\mathcal{D}_a$. In the second experiment, which uses the ResNet-18 network architecture (He et al., 2016a), CIFAR-10 builds the $\mathcal{D}_c^{'}$ and a subset of CIFAR-100 the $\mathcal{D}_b^{'}$. The test set comprises samples from CIFAR-10 as $\mathcal{D}_c$ and a subset of CIFAR-100 as well as the SVHN dataset as $\mathcal{D}_a$. To assess the performance of the approach, the authors present a novel evaluation metric, the OSCR curve. This metric is treated in detail in section 3.2.2. The results show that Entropic Open Set loss outperforms the Softmax as well as the Background class approach in experiment 1. The same holds true for experiment 2, if the subset of CIFAR-100 represents $\mathcal{D}_a$. Considering the results, the authors see some support for their theory of achieving a more effective distinction between known and unknown samples by reducing the deep feature magnitude of unknown samples. However, they also point to limitations of their approach such as the importance of the selection of unknown training samples $\mathcal{D}_b^{'}$.

While the Entropic Open Set approach, as proposed by Dhamija et al. (2018), integrates unknown samples from the open space into the training procedure, other approaches introduce a new model layer. One such approach is presented in the next subsection.

2.1.3 OpenMax

Instead of applying a threshold θ on the probability scores of a classifier to reject unknown samples, Bendale and Boult (2016) propose another strategy in the form of replacing the Softmax layer with a novel OpenMax layer that additionally produces a probability score for an unknown class. They especially opt against the conventional thresholding mechanism based on research findings that indicate its vulnerability to "fooling" (Nguyen et al., 2015), "rubbish" (Goodfellow et al., 2014b), or adversarial (Goodfellow et al., 2014b; Szegedy et al., 2014) images. As the authors note, fooling and rubbish images are samples that are classified by a DNN as one of the training classes with a high probability, despite being visually far away from the assigned class. In contrast, adversarial images are samples that are visually almost identical to a known training sample but that are particularly engineered to result in a high confidence score for a different training class. Bendale and Boult (2016) therefore conclude that thresholding on Softmax scores does not represent the optimal strategy to recognize and reject unknown classes. As a possible solution to this issue, they apply concepts from Meta-Recognition (Scheirer et al., 2011; Jammalamadaka et al., 2012; Zhang et al., 2014) to gain similarity information and then pass this information to a newly designed algorithm. The algorithm, which they term OpenMax, then generates a probability estimate for a sample being unknown. This probability score can then be used as an argument to reject a test sample.

The approach, as proposed by Bendale and Boult (2016), can be broadly divided into three components. Firstly, a deep convolutional neural network that has been trained in a closed set fashion and that applies the conventional Softmax layer serves as a classifier. Additionally, this network is used to map input samples from pixel space into a deep feature space. Secondly, a multi-class meta-recognition algorithm that fits a probability distribution per training class based on the deep feature representations of the associated training samples. Finally, the OpenMax algorithm whose task is to adjust the probability scores that a test sample produces as well as to estimate a probability score for an unknown class. To achieve this, the OpenMax algorithm uses the probabilistic similarity information that is provided by applying the fitted probability distributions. The key point of the approach by Bendale and Boult (2016) is the research-based suggestion that for quantifying the open space risk, feature space and not pixel space should be used (Scheirer et al., 2013, 2014; Bendale and Boult, 2015). Based on this insight, Bendale and Boult (2016) treat the output of the last layer in the DNN, i.e., the logit values, as the deep feature space representation of the input image and term it Activation Vector (AV). Their underlying assumption is that the associated deep feature space effectively separates known classes from unknown classes and therefore builds a space for obtaining meaningful distance measures. Following this reasoning, the feature space representations of the correctly classified known training samples $x \in \mathcal{D}_c^{'}$ are then extracted and passed to the multi-class meta-recognition algorithm. The algorithm, applying concepts from Extreme Value Theory (EVT, Fisher and Tippett, 1928), fits a Weibull distribution for every training class in *KKCs*. Concretely, for every training class, the mean of the AVs is computed (MAV). Subsequently, a Weibull distribution is fitted to the η largest distances to the MAV on a class basis. These class-wise meta-recognition models can then be used to evaluate the likelihood that a test sample $x \in \mathcal{D}_{test}$ is a member of a training class. This is done by first computing the distance between the AV of the test sample and the MAV for a particular class and then passing this distance value to the associated meta-recognition model. Bendale and Boult (2016) illustrate the idea of assessing similarity in deep feature space via AVs in figure 2.4(a). Considering the activation patterns, a distinction between different classes and types of images is possible. This observation represents the authors' idea of choosing a deep feature space with meaningful separation properties.



(a) Feature space representation of images

(b) Effectiveness of OpenMax

Figure 2.4: CHARACTERISTICS OF OPENMAX. In (a), the activation vectors (AV) for different types of images are visualized. While the original AV consists of 1000 elements and 10 channels per element, the graphic only shows the first 450 vector elements (horizontal pixels) and all the associated channels (vertical pixels). The white headers denote the class, and the labels on the left the type of image. Here, *MODEL* refers to the MAV of the respective class. At the bottom, selected category labels indicate the vector elements that typically show high activation for a sample of the related classes. The pixel colors encode the level of activation of the vector elements/channels, thus revealing different activation patterns for distinct classes and types of images. The robustness of OpenMax against fooling images is exemplified in (b). While *SM* refers to the conventional Softmax probability, *OM* indicates the OpenMax probability. Applying the conventional Softmax approach, the fooling images for the classes Baseball and Hammerhead get unambiguously misclassified. OpenMax, on the other hand, is capable of correctly classifying the fooling images as unknown. *Source: (Bendale and Boult, 2016)*

The class-wise MAVs and meta-recognition models are the main ingredients of the OpenMax algorithm. With this algorithm, Bendale and Boult (2016) introduce a mechanism that is capable of revising the Softmax scores for the *KKCs* by considering the feature space characteristics of a test sample. In detail, OpenMax revises the probability scores for the set of α classes with the highest logit values. The extent to which a selected probability score is adjusted depends on the distance in feature space between the AV of the test sample and the MAV of the class in question. In other words, if the meta-recognition model of the respective class returns a low score for the distance value, the test sample is likely not a member of the class and the high logit value might be unjustified. Therefore, OpenMax reduces the logit value much more significantly than in the case of the meta-recognition model returning a high value. After revising the logit values for all α classes, OpenMax computes an additional pseudo-logit value that represents the unknown class. The amount of this pseudo-logit depends on the total reduction over the α adjusted logit values. In the end, OpenMax computes the Softmax scores over this new logit vector with $C+1$ elements. As indicated by the authors, the decision to reject a test sample can then be based on two criteria: the probability score for the unknown class being highest or the probability score for the unknown class going below a decision threshold θ. Bendale and Boult (2016) then put their system to the test. They use a pretrained AlexNet (Jia et al., 2014) as the network architecture and the ILSVRC 2012 dataset with 1000 *KKCs* as the training dataset. The test dataset comprises the ILSVRC 2012 validation set as $\mathcal{D}_c$ and 15K images from a subset of the ILSVRC 2010 dataset as well as 15K fooling images (Nguyen et al., 2015) as $\mathcal{D}_a$. Using the F-measure, OpenMax is compared to the conventional Softmax approach. For a threshold θ that has been optimized on the test dataset, OpenMax generates an F-score that is around 4.3% higher. Furthermore, the results suggest the potential of OpenMax for recognizing fooling images. In figure 2.4(b), Bendale and Boult (2016) argue that using deep feature space-based similarity information might be a viable strategy to address the issues of fooling and adversarial images. However, despite providing protection against traditional adversarial attacks, it has been shown that OpenMax systems are nonetheless vulnerable to more sophisticated adversarial attacks (Rozsa et al., 2017).

As referred to above, OpenMax relies on a network that maps input samples into a meaningful feature space. These feature space representations are then used to fit probabilistic models per training class. Rudd et al. (2018) propose an approach that is based on similar premises. This approach is subject of the next subsection.

2.1.4 Extreme Value Machine

Considering the OSR problem, Rudd et al. (2018) observe that algorithms based on DNNs do generally not take advantage of distributional information that is present in the training data. Similar to OpenMax (Bendale and Boult, 2016), they use feature space representations of input samples as the basis for building their models because of the indicated significance of feature space in OSR (Scheirer et al., 2013, 2014; Bendale and Boult, 2015). Using the distributional information provided by the training data, Rudd et al. (2018) apply concepts from Extreme Value Theory (Fisher and Tippett, 1928) to fit a set of probabilistic distributions for every known training class. This set of distributions has the property that it can be used to arrive at an expressive probabilistic representation of the class boundary. This in turn allows it to assess the likelihood of a test sample belonging to a training class and therefore to assign or reject it. Referring to the used EVT concepts, the authors name their approach Extreme Value Machine (EVM).

In detail, Rudd et al. (2018) use feature representations of the training samples as the starting point for the EVM. Using the concept of margin distributions (Garg and Roth, 2003; Reyzin and Schapire, 2006; Pelckmans et al., 2007; Aiolli et al., 2008), the authors compute for every training sample x_i of a particular training class $c \in \mathcal{C}$, i.e., $x_i \in \mathcal{D}_c^{'} : y_i = c$, the half-distances to the τ closest training samples belonging to the negative training classes $\mathcal{N} = \mathcal{C} \setminus \{c\}$. The training sample x_i therefore builds the reference point for the distance computation and is termed Extreme Vector (EV). Then, for every EV of the respective class, a Weibull distribution is fitted to this set of τ computed half-distances. Using this distribution, Rudd et al. (2018) continue by inferring a radial inclusion function that models the probability of sample inclusion (*PSI* or Ψ). This Ψ-function can be used to assess the posterior probability that a test sample x' is included in the boundary represented by the radial inclusion function or, equivalently, the likelihood that the test sample x' is associated with the training class that the underlying EV x_i belongs to. Formally, the authors define the Ψ-function as:

$$\Psi(x_i, x'; \kappa_i, \lambda_i) = \exp^{-\left(\frac{\|x_i - x'\|}{\lambda_i}\right)^{\kappa_i}} \tag{2.4}$$

In this definition, $\|x_i - x'\|$ refers to the distance between the test sample x' and the EV x_i. Moreover, κ_i and λ_i denote the Weibull shape and scale parameters that are associated with the EV x_i according to the above-mentioned fitting method. Repeating this procedure for every training class results in one Ψ-model for every training sample (aka EV) per class. Depending on the number of training samples per class, this can lead to a large number of Ψ-models per training class, which in turn can result in a certain level of model redundancy. To address this point, the authors propose a model reduction step that disposes of redundant EVs and their associated inclusion

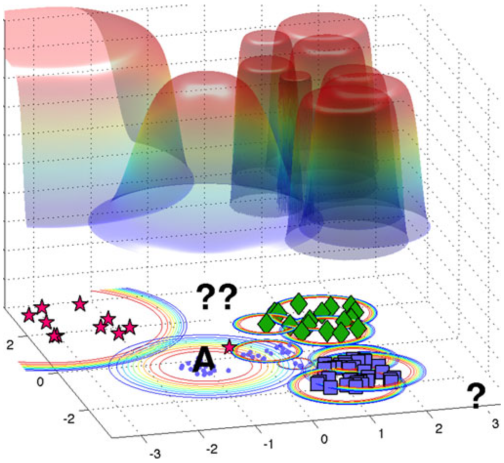


Figure 2.5: FINAL EVM MODELS. The figure shows the elements of the EVM. The training data consists of four different classes (star, dot, diamond, and square). The colored rings represent the probability isocontours of the associated *PSI*/Ψ-model. While red encodes a probability close to 1, blue references low probabilities. The depicted Ψ-models are the result of the reduction step performed by a greedy approximation algorithm. Every Ψ-model is linked to its respective Extreme Vector (EV, e.g., the vector denoted by "A"). Using the final models, unknown samples "?" from the open space can be correctly rejected as the models return low probability scores. *Source: (Rudd et al., 2018)*

functions on a class basis, i.e., (EV, Ψ)-tuples. The idea of this reduction step is to select a representative subset of training samples per class. Concretely, EVs that lie within close proximity in feature space are associated with Ψ-models that produce similar responses to a test sample and therefore generate a certain level of redundancy. Assuming a probability threshold ς and a pair of EVs (x_i, x_j) from the same training class, the tuple (x_j, Ψ_j) is considered to be redundant if $\Psi_i(x_i, x_j) \geq \varsigma$. In contrast, the (x_i, Ψ_i)-tuple is integrated into the final overall model. This reduction task, which mathematically represents a special case of Karp's NP-hard Set Cover problem (i.e., a minimization problem), is tackled using a greedy approximation that runs in polynomial time (Slavík, 1996). After running this greedy algorithm, every training class $c \in \mathcal{C}$ is represented by a set of (x_i, Ψ_i)-tuples with x_i being the EV. Rudd et al. (2018) depict an exemplary solution in figure 2.5. The EVM algorithm is applied to training data from four different classes. The superimposed Ψ-models at the top are the outcome of the model reduction step. Every Ψ-model is associated with one selected EV and its coloration encodes the probability of sample inclusion. As each Ψ-model is fitted individually to the data, the shape and scale parameters can vary considerably. The "?" embody unknown samples from the open space. Using the final Ψ-models, these unknown samples get correctly rejected as a consequence of not generating sufficiently high function responses.

By its structure, the EVM is able to perform the OSR task. Concretely, Rudd et al. (2018) propose the following decision function:

$$y^* = \begin{cases} \operatorname{argmax}_{c \in \{1,\ldots,C\}} \max_{\{i: y_i = c\}} \Psi_i(x_i, x') & \text{if } \max_{\{i: y_i = c\}} \Psi_i(x_i, x') \geq \delta \\ ''unknown'' & \text{otherwise} \end{cases} \tag{2.5}$$

This means that the predicted class y^* of a test sample x' corresponds to the class c whose set of associated Ψ-models hosts the model that returns the maximum probability among the Ψ-models of all classes, given that this probability score reaches at least a probability threshold δ. Otherwise, the test sample x' is rejected and considered to belong to an unknown class.

Rudd et al. (2018) then test the capabilities of their proposed EVM algorithm in an OSR context. Specifically, they follow the OLETTER protocol specified by Scheirer et al. (2014) and randomly choose 15 letters from the LETTER dataset as *KKCs*. The EVM algorithm, including the reduction step, is then run on the training samples belonging to those *KKCs*. During the test procedure, the 11 letters that have not been selected as known classes for the training procedure then build the *UUCs*. Using the F-score as the evaluation metric, the authors find that the EVM performs similarly or better than other state-of-the-art algorithms at the time. However, the EVM is the only approach among the evaluated algorithms that can be used for incremental learning. Specifically, the authors introduce a procedure that can be used to integrate new classes over time without the need for a complete retraining of the collated model. The power of this incremental learning ability is then tested in an Open World Recognition context, leading to a superior performance over the NNO algorithm (Bendale and Boult, 2015). Open World Recognition is a problem that differentiates itself from Open Set Recognition and is elaborated on in subsection 2.1.6.

The introduced approaches up to now all have in common that they fall into the category of discriminative modeling. The algorithm presented in the next section, however, represents a generative approach since it incorporates a mechanism that synthesizes new data instances, thereby anticipating unseen data patterns.

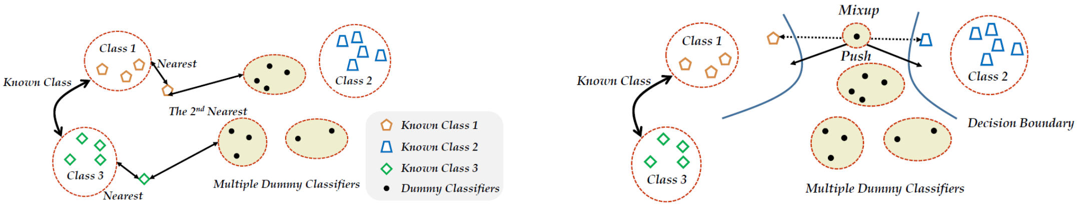
2.1.5 Placeholders for Open Set Recognition

Not convinced by a thresholding approach, Zhou et al. (2021b) pursue the idea of addressing the OSR problem by applying the principles of generative modeling. Inspired by generative models such as G-OpenMax (Ge et al., 2017) and C2AE (Oza and Patel, 2019), they propose a model that is capable of generating unknown samples and then integrating them into the training procedure. The model design closely follows the notion of OSR as a calibration problem consisting of two steps (Guo et al., 2017; Ye et al., 2021). As a first step, making use of known training samples $x \in \mathcal{D}_c^{'}$, new data instances are created. These instances act as data placeholders for unknown classes and are used in the training procedure to prepare the model for encounters with unknown classes during testing/operation. The second step then consists of augmenting the closed set classifier with a classifier placeholder for the unknown classes. This additional classifier aims to enhance the separation between known and unknown samples. In the case of the model introduced by Zhou et al. (2021b), the creation of the data placeholders comes with limited computational cost. Referring to the calibration concept, they call the model Placeholders for Open Set Recognition (PROSER).

As indicated above, PROSER consists of two types of placeholders. On the one hand, Zhou et al. (2021b) assume a pretrained closed set classifier and augment its output layer with an additional dummy classifier. This is formalized as:

$$z = \hat{f}(x) = \left[W^\top \phi(x), \hat{w}^\top \phi(x)\right] \tag{2.6}$$

In this formulation, $W \in \mathbb{R}^{d \times C}$ refers to the linear classifier of the pretrained model that maps the deep feature representation $\phi(x)$ of an input sample x to the C logit values that each refers to a known training class. Furthermore, $\hat{w} \in \mathbb{R}^{d \times 1}$ denotes the linear dummy classifier. Hence, $\hat{f}(x)$ is the augmented model that maps an input sample to $C + 1$ logits, i.e., $z = [z_1, \dots, z_{C+1}]$, which are then passed to the Softmax function to compute the probability scores. Importantly, the PROSER design allows for defining multiple dummy classifiers $\hat{W} \in \mathbb{R}^{d \times P}$ with P denoting the



(a) Classifier placeholders (b) Data placeholders

Figure 2.6: EFFECTS OF PROSER TRAINING. In (a), the consequences of training the classifier placeholders are shown. Importantly, the dummy classifiers are positioned between the target class and the non-target classes. Furthermore, (b) illustrates the impact of generating and using data placeholders on the boundaries of known classes. *Source: (Zhou et al., 2021b)*

number of dummy classifiers/classifier placeholders. However, when an input sample x passes through the network as indicated in equation 2.6, only the dummy classifier $\hat{w}$ that returns the highest score when applied to $\phi(x)$ is actually used. The weights of the dummy classifier $\hat{w}$ are learned during a fine-tuning training procedure. More precisely, Zhou et al. (2021b) introduce a loss function that incentivizes the network to learn dummy classifier weights that are associated with a certain outcome. Specifically, the dummy classifier should generate the highest among the $C+1$ logit values when applied to the feature representation of a generated unknown training sample. This reflects the idea of correctly assigning an unknown sample to the "unknown" class. In contrast, the dummy classifier is expected to map the feature representation $\phi(x_i)$ of a known training sample $x_i \in \mathcal{D}_c^{'}$ to the second highest logit score. Simultaneously, the linear classifier W is responsible for correctly classifying the known training sample x_i, thereby maintaining closed set performance. To address the overconfidence phenomena (Bendale and Boult, 2016) that can be observed in DNNs, Zhou et al. (2021b) introduce an additional calibration step that ensures that the output logit of the dummy classifier is of the same magnitude as the output of the closed set classifier. This is achieved by computing a bias term and adding it to the dummy logit. The authors visualize this idea of introducing dummy classifiers in figure 2.6(a). During the fine-tuning training procedure, the weights of the network are adjusted such that the feature vector of a known training sample $\phi(x_i)$ (e.g., the green diamond close to the class boundary) moves towards its corresponding class cluster. Concurrently, the weights of the associated dummy classifier $\hat{w}$ are adjusted such that the classifier placeholder is located between the target class and the non-target classes. Conceptually, this refers to the idea of reserving classifiers for new classes.

The second placeholder type comes in the form of a data placeholder. As Zhou et al. (2021b) note, generating and using new data instances in the fine-tuning training procedure turns a closed set training into an open set training. They use a process that is capable of efficiently generating data instances whose distribution suggests an unseen class. This process is called manifold mixup and has been presented by Verma et al. (2019). As the inventors illustrate, manifold mixup assumes an embedding module $\phi(\cdot)$ that can be decomposed as:

$$\phi(x) = \phi_{post}(\phi_{pre}(x)) \tag{2.7}$$

Here, $\phi_{pre}(x)$ denotes a sequence of network layers that takes the input sample x from pixel space into a hidden representation. Subsequently, $\phi_{post}(\cdot)$ represents the sequence of layers that transfers the hidden representation into the feature representation $\phi(x)$. The creation of new data instances happens at the level of hidden representations by fusing two known training samples from different classes, $(x_i, x_j) \in \mathcal{D}_c^{'} : c_i \neq c_j$, in an interpolative fashion:

$$\tilde{x}_{pre} = \lambda \cdot \phi_{pre}(x_i) + (1 - \lambda) \cdot \phi_{pre}(x_j) \tag{2.8}$$

Finally, Verma et al. (2019) apply the post-module to the created instance, i.e., $\phi_{post}(\tilde{x}_{pre})$, and get the feature representation of the newly created data instance. With respect to this data creation process, Zhou et al. (2021b) select the first half of the training data in every batch for the creation of new data instances. Assuming an ordered collection of selected training instances, the collection is randomly shuffled to create the required pairs of data instances. By masking pairs whose members are linked to the same class, it is ensured that the data creation process only considers pairs from different classes. As indicated above, Zhou et al. (2021b) use these generated data instances in the fine-tuning training procedure and treat them as representatives of an unknown open set class. They intuitively picture the effects of using data placeholders in the training procedure in figure 2.6(b). The feature representations of the generated instances, depicted as black dots, emerge between the boundaries of the known classes. During training, the decision boundaries of the known classes migrate away from the generated samples and get more compact.

To test the capabilities of the PROSER approach in an OSR context, Zhou et al. (2021b) perform a selection of experiments. For this purpose, they use the protocol designed by Perera et al. (2020). This protocol stipulates that a model is trained on all training instances of a certain closed set dataset. For testing, the test set is augmented by instances of another dataset. These added samples serve as representatives of unknown classes. In the first experiment, Zhou et al. (2021b) choose the ten numbers of the MNIST dataset as the known classes. The unknown classes used during testing come in the form of Omniglot alphabet characters (Lake et al., 2015). Moreover, noisy samples serve as instances of other unknown classes. Comparing the *F*-score achieved by PROSER with the results as reported by Sun et al. (2020) shows that PROSER outperforms methods as traditional Softmax or OpenMax (Bendale and Boult, 2016) significantly on these datasets. A similar picture emerges in a second experiment where the authors use the categories in CIFAR-10 as known classes and categories from ImageNet and LSUN (Fisher et al., 2015) as unknown classes. Considering their work, Zhou et al. (2021b) conclude that PROSER addresses two significant problems in OSR. On the one hand, the generation and use of data placeholders in the PROSER training procedure facilitates the anticipation of new data patterns. On the other hand, introducing classifier placeholders addresses the issue of overconfident closed set classifiers as these placeholders are systematically designed to recognize unknown classes in the future.

2.1.6 Delimitation to Related Concepts

As identified by various researchers (Scheirer et al., 2013; Dhamija et al., 2018; Geng et al., 2021) and introduced in section 1.1, the Open Set Recognition problem consists of two different tasks. Firstly, a classifier in an OSR context is expected to correctly classify samples from known classes. Secondly, it should also be able to reject samples from unknown classes. Considering this definition, research knows related, but delimitable problems. Two of them are particularly notable.

Out-Of-Distribution Detection: Liang et al. (2018) characterize the problem of out-of-distribution detection as the task of a classifier to recognize if an input image is from a data distribution that has been used for training or from a data distribution that has not been part of the training procedure. Concretely, they assume that a classifier is trained on an image dataset that has been drawn from a data distribution that is associated with the *KKCs*. For testing, samples that are drawn from another distribution that is linked to the *UUCs* (the out-distribution) are added to the test dataset containing samples from *KKCs*. The classifier is then expected to recognize if a test sample is from the out-distribution or not. Considering this definition, the binary nature of out-of-distribution detection emerges. Specifically, the task in out-of-distribution detection is to reject a sample from an unknown class and to recognize if a sample is from a known class. The correct classification of the known sample, however, is not part of the objective. Hence, there is a clear distinction to the OSR problem.

Open World Recognition: Bendale and Boult (2015) present Open World Recognition (OWR) as an extension to Open Set Recognition. As the authors describe, OWR additionally takes the concept of incremental learning into account. Concretely, an OWR system is expected to master four tasks. Firstly, the system should reliably detect samples from unknown classes. Then, a model update process is started. This means that the system must select unknown samples that seem to be suited to extend the existing model in a meaningful way. These samples must then be labeled. Finally, the model is updated using the additional information. Assuming such a system, the authors then sketch the different stages in OWR. In the beginning, the model is trained on a dataset. This is followed by the model performing OSR. Having identified the unknown samples, the model update process is started. The last two steps are then repeated. With this concept, Bendale and Boult (2015) propose a solution to a scenario where an open set classifier encounters an increasing number of unknown classes when put in operation. Looking at this definition, it stands out that Open Set Recognition is not a synonym for Open World Recognition. In fact, OSR is a subproblem of OWR. In the literature, OWR is also known as lifelong learning, a paradigm that is characterized by accumulating and updating knowledge in order to solve problems in a dynamic environment (Chen and Liu, 2018).

2.2 Protocols

As mentioned in section 1.2, most open set algorithms in research are trained and evaluated on small-scale datasets such as LETTER, MNIST, SVHN, or CIFAR-10 (Geng et al., 2021). The size and limited diversity of those datasets however affect the generalizability of the reported results. This fact motivated Bhoumik (2021) to design three open set protocols in a way that they mimic instances of a real-world open space and accurately represent the challenges associated with OSR. Concretely, capitalizing on the hierarchical richness of ImageNet, Bhoumik (2021) uses different subsets of the 1′000 classes that constitute the ILSVRC 2012 dataset to design the three protocols. For every protocol, she selects the subset in a systematic manner by using the ImageNet-1k tree structure (Bostock, 2018) as well as the `robustness` library (Engstrom et al., 2019) and partitions this subset into *KKCs*, *KUCs*, and *UUCs*. The data samples that are associated with these clusters of classes then form the open set partition. The allocation of classes is done in a systematic way such that the resulting open set partition captures the difficulties a classifier can encounter in a real-world context where unknown classes may unexpectedly appear. The differences between the three protocols become apparent when considering the distinguishing criteria that Bhoumik (2021) defines:

- Degree of similarity within *KKCs*: in general, it is expected that a higher similarity within classes increases the difficulty of correctly classifying samples from *KKCs*.

- Degree of similarity between *KKCs* and *KUCs*: given that samples of *KUCs* are available during training, a higher degree of similarity between *KKCs* and *KUCs* is associated with a better differentiation capability of the model (Dhamija et al., 2018). Concretely, the trained model should be able to better anticipate the distribution of samples belonging to *UUCs* and thus to reject them.

- Degree of similarity between *KKCs* and *UUCs*: intuitively, the higher the similarity, the harder the rejection task becomes. A high similarity requires the classifier to recognize fine-grained differences between classes.

As it is indicated in table 2.1, Bhoumik (2021) uses these criteria to specify the "hardness" of the protocols regarding the OSR task. Concretely, in protocol 1, the degree of similarity within *KKCs* is high as all known classes are a descendant of the superclass "dog". This makes a correct classification of known classes harder. On the other hand, compared to the *KKCs*, the *UUCs* represent entirely different concepts and the rejection task of a classifier is expected to become easier. This is mainly due to the *UUCs* not sharing many features with the *KKCs*. Protocol 2 includes 31 descendants of the subclass "hunting dog" in the *KKCs*, which increases the difficulty for a model to correctly classify samples. In contrast to protocol 1, the *UUCs* have certain features in common with the *KKCs*. This increases the likelihood that a classifier mistakes an unknown sample for a known sample and therefore the difficulty of the open set rejection task. This holds especially if the *KUCs*, which share many features with the *KKCs*, are not used during training. Protocol 3 is based on the pre-packaged dataset `mixed_13`, which is provided by the `robustness` library and contains a mix of living objects/classes and non-living objects/classes. The *KKCs* consist of some descendants of the superclass set `mixed_13` and the *UUCs* of some descendants of superclasses not contained in `mixed_13`. Concretely, the superclass set that builds the basis for the *UUCs* contains the superclasses "reptile", "clothing", "ungulate", "vegetable", and "aircraft". Since the *UUCs* do represent a mix of living and non-living objects as well, an unknown sample might generate features that are similar to features of known samples from multiple *KKCs*. Therefore, the rejection task for a classifier becomes harder. In contrast, the classification task on known samples is expected to be not as hard as in the other protocols since the similarity within the *KKCs* is relatively low. In sum, Bhoumik (2021) designs the protocols with certain patterns in mind: the level of similarity in appearance between *KKCs* and *UUCs* as well as between *KKCs* and *KUCs* varies across protocols. The same holds for the similarity within *KKCs*. Having specified the open set partitions per protocol, Bhoumik (2021) then continues to assign the samples to a training, validation, and test set. While the samples from *KKCs* and *KUCs* are distributed among all three datasets, the samples from *UUCs* are solely allocated to the test set. Table 2.2 gives an overview of the original protocol splits.

However, protocol 3 has been redesigned in the meantime and does not match the open set partition as originally designed by Bhoumik (2021) and reported in tables 2.1 and 2.2 anymore. Specifically, Palechor et al. (2023) revised protocol 3 in a way that increased the difficulty of the open set task. They achieve this by additionally adding some descendants of the superclass set `mixed_13` to the set of existing *UUCs* as assigned in the original version of protocol 3. Due to the reassignment of classes, the composition of the *KKCs*, *KUCs*, and *UUCs* changes in the revised version. As a consequence, the revised protocol 3 represents the hardest open set task, followed by protocol 2 and protocol 1. It is this revised version of protocol 3 which I use in my experiments. While working with these protocols, I find that they contain an unaddressed source of randomness that subsequently leads to a reproducibility issue. In subsection 3.1.2, I discuss this issue in detail and introduce a solution. The final composition of the protocols can be seen in table 3.1.

2.3 ResNet

Back in 2016, He et al. (2016a) proposed ResNet as a solution to their observation that appending additional layers to a sufficiently deep, plain convolutional neural network increases the training error. They visualize this unexpected phenomenon, which they term degradation problem, in figure 2.7(a). It can be seen that increasing the depth of a network negatively affects the training

Open-set Partition	Protocol 1	Protocol 2	Protocol 3
Known classes	All dog classes (116 descendants)	Some classes of a particular dog subclass (31 descendants)	Some classes of some ancestors (159 descendants)
Known Unknown classes	Some other 4-legged animal classes like zebra, monkey, fox, etc (67 descendants)	Some other classes of the same dog subclass as the known classes (30 descendants)	Other classes of the same ancestors as the known classes (137 descendants)
Unknown Unknown classes	Some non-animal classes like car, food, sunglasses, etc (166 descendants)	Some classes of another dog subclass and some other 4-legged animal classes (55 descendants)	Some classes of other ancestors (116 descendants)

Table 2.1: ORIGINAL OPEN SET IMAGENET PROTOCOLS. The table provides an overview of the three protocols and their respective partition into *KKCs*, *KUCs*, and *UUCs*. *Source: (Bhoumik, 2021)*

error, a phenomenon that contradicts the authors' expectation of overfitting. A possible explanation for this behavior is vanishing gradients, a problem that has already been observed in deep convolutional neural networks (Bengio et al., 1994; Glorot and Bengio, 2010).

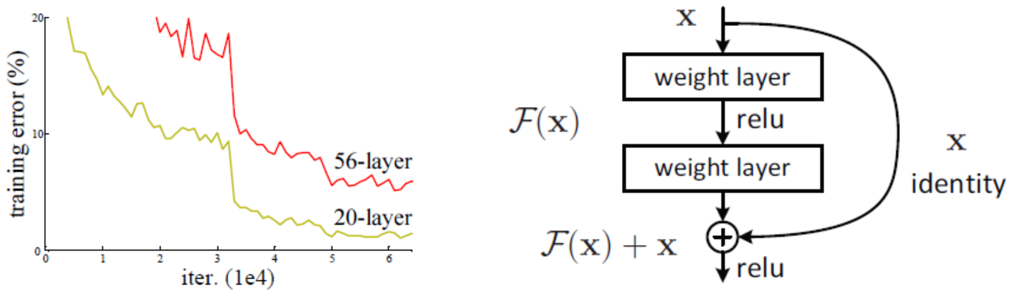
Vanishing gradients in deep networks can be attributed to the application of the chain rule for computing the gradients in the back-propagation method (Rumelhart et al., 1986a). Using the chain rule implies that the gradients for early layers are the result of multiplying a series of derivatives. As the derivatives of the sigmoid and tanh activation functions are small, a repeated multiplication leads to very small numbers that, in the end, cause very small gradients for early layers[1]. This then impedes the network from learning early-layer features. He et al. (2016a) react to this problem by proposing a residual learning framework that builds the basis for the ResNet architecture. Concretely, they push a stack of non-linear layers to fit a residual mapping $\mathcal{F}(x)$ instead of the underlying mapping $\mathcal{H}(x)$, with the residual mapping being formalized as $\mathcal{F}(x) = \mathcal{H}(x) - x$. The original, underlying mapping is given by $\mathcal{H}(x) = \mathcal{F}(x) + x$ and is implemented by an architecture using shortcut connections (Bishop, 1995; Ripley, 1996). As He et al. (2016a) show in figure 2.7(b), the ResNet architecture is based on building blocks that consist of stacked layers and a shortcut connection. The parameterless shortcut connection represents an identity mapping, combining its output with the output of the stacked layers. Since these shortcut connections circumvent a pass through the stacked layers, they provide a path that the gradients can flow through during back-propagation. The authors introduce two types of building blocks: a basic building block in which the shortcut connection skips two convolutional layers and a bottleneck building block in which three convolutional layers are bypassed. Both types additionally apply the *ReLU* activation function (Goodfellow et al., 2016) as well as batch normalization (Ioffe and Szegedy, 2015). As can be seen in table 2.3, the basic building block is used in the 18- and 34-layer variants of ResNet while the bottleneck building block is found in ResNet-50, ResNet-101, and ResNet-152.

---

[1] `https://www.kaggle.com/code/shrutikunapuli/activation-functions-for-neural-networks/notebook`

Protocols	Open-set Partition	Training Size	Validation Size	Test Size
Protocol 1	Known	116,212	29,061	5,800
	Known Unknown	69,680	17,420	3,350
	Unknown Unknown	-	-	8,300
Protocol 2	Known	30,629	7,661	1,550
	Known Unknown	30,055	7,517	1,500
	Unknown Unknown	-	-	2,750
Protocol 3	Known	161,661	40,425	7,950
	Known Unknown	140,477	35,122	6,850
	Unknown Unknown	-	-	5,800

Table 2.2: ORIGINAL PROTOCOL SPLITS. The table illustrates the protocol-wise split of samples into a training, validation, and test set. *Source: (Bhoumik, 2021)*

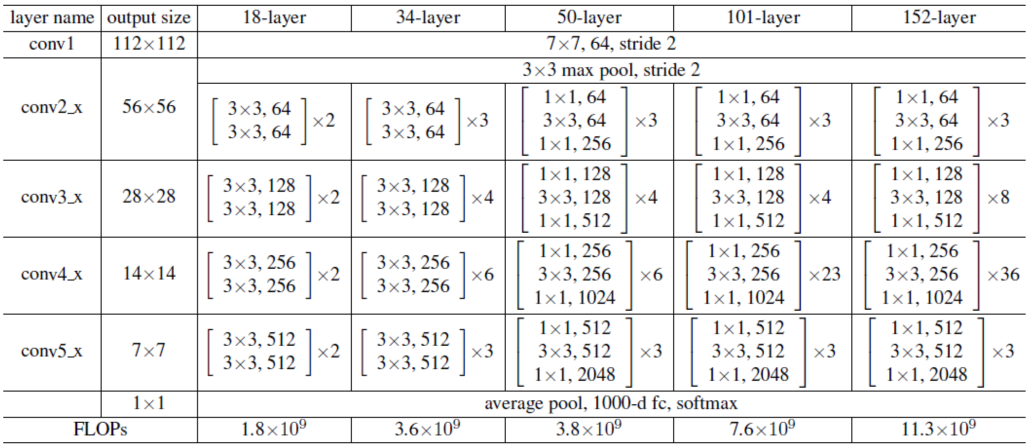


(a) Degradation of training performance

(b) Basic building block

Figure 2.7: DEGRADATION PROBLEM AND SOLUTION APPROACH. In (a), it is illustrated that the training error on CIFAR-10 can increase when additional layers are added to a 20-layer "plain" network. The basic building block is depicted in (b). Residual learning is enabled by a shortcut connection that applies an identity mapping and that is depicted by a curved arc. *Source: (He et al., 2016a)*

Using this residual learning framework, He et al. (2016a) hypothesize that it should be easier for the network to optimize the residual mapping $\mathcal{F}(x)$ than the original, unreferenced mapping. Referring to the degradation problem, they subsequently argue that a deeper network should not produce a higher training error since pushing the residual mapping $\mathcal{F}(x)$ to zero is easier than letting a suite of non-linear layers learn an identity mapping. Due to its results, ResNet quickly gained popularity and laid the foundation for further improvements and architectures (He et al., 2016b; Huang et al., 2016, 2017; Xie et al., 2017).



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 2.3: RESNET ARCHITECTURES. The table illustrates the architectural structure of various variants of ResNet. For every variant, the square brackets refer to the type of building block that is used. While ResNet-18 and ResNet-34 use the basic building block, ResNet-50 and higher use the bottleneck building block that consists of three convolutional layers. The multiplication factor that follows the square brackets denotes the number of blocks that are used in the corresponding layer. *Source:* (He et al., 2016a)

Chapter 3

Approach

The objective of this master thesis lies in developing and comparing open set classification techniques on ImageNet. In this chapter, I analyze the challenges and questions arising in recent research and present the approach I choose for performing the comparison.

3.1 Comparison of Algorithms

This section points out why a comparison of performance results of various open set classification algorithms as reported in research papers is difficult. Concretely, it presents the reasons why the algorithms have been selected, what factors prevent a direct comparison and how these difficulties are overcome.

3.1.1 Algorithm Selection

The goal of this thesis is the comparison of open set algorithms on the ImageNet protocols designed by Bhoumik (2021). To arrive at a selection of algorithms, I define three criteria. Firstly, I consider algorithms whose source code has been released in the form of a software package by the authors of the respective paper. This ensures an integration of the original algorithm. Secondly, only algorithms whose source code is written in the Python programming language (Van Rossum and Drake, 2009) are taken into account. This choice is motivated by the fact that the three ImageNet protocols (Bhoumik, 2021) are written in Python. By using the same programming language, the integration of code is simplified. Thirdly, I select algorithms that focus on image classification. This criterion is justified as I perform the experiments on image data and not on other types of data, such as text data. With respect to the first criterion, I use the list of available software packages by Geng et al. (2021) as a starting point and extend this list by the PROSER algorithm (Zhou et al., 2021b). Applying the second criterion, methods such as the 1-versus-Set machine (Scheirer et al., 2013), the W-SVM (Scheirer et al., 2014), the P_I-SVM (Jain et al., 2014), the Best Fitting Hyperplane Classifier (Cevikalp, 2017), the SROSR model (Zhang and Patel, 2017), and the NNO algorithm (Bendale and Boult, 2015) are excluded since they are written in other programming languages. By applying the third criterion, the DOC model (Shu et al., 2017) is disqualified since it performs text classification. Furthermore, I eliminate the HPLS and the HFCN algorithms (Vareto et al., 2017). Since these algorithms aim at open set face recognition, they require a data basis that is in accordance with this task, a requirement that is not satisfied by the data basis I use. Finally, I exclude the method of counterfactual image generation (Neal et al., 2018) and the ASG framework (Yu et al., 2017). While these algorithms satisfy all criteria, they have been ruled out due to time constraints regarding this thesis. Therefore, I use the remaining algorithms for my comparison: the Entropic Open Set Loss (Dhamija et al., 2018), OpenMax (Bendale and Boult, 2016), the Extreme Value Machine (Rudd et al., 2018), and PROSER (Zhou et al., 2021b). These algorithms build the basis for my experiments, which are introduced in chapters 4 and 5.

3.1.2 Data Basis

Section 1.2 names one challenge of existing OSR models in research: most of them are trained on rather small datasets such as LETTER, MNIST or, CIFAR (Geng et al., 2021). A look at table 3.2 confirms this notion for the algorithms I evaluate in this thesis. Such datasets are associated with a restricted diversity and structural richness, which limits their ability to simulate real-world contexts. As a consequence, it is not readily apparent if the reported performance measures of these algorithms are generalizable to more diverse and challenging contexts. For example: does a classifier, which has been trained and evaluated on an open set partition of the 10-class MNIST dataset, show comparable capabilities in a more diverse context with dozens or hundreds of classes? Another difficulty is that the reported performances of these algorithms are hardly comparable when the underlying data differs. These observations lead Bhoumik (2021) to propose three open set protocols as presented in section 2.2. She designs these protocols in a systematic way, thereby creating three open set partitions that differ in their level of difficulty. For my experiments, I use these protocols as a data basis. This has several reasons. First, since the open set partitions systematically incorporate the structural variety and hierarchical richness of ImageNet, I consider them to build a more appropriate data foundation than datasets such as LETTER, MNIST, or SVHN with respect to the suitability as a proxy for a real-world open space. Furthermore, the open set partitions place different demands on a classifier that tries to meet the OSR goals. In other words, these protocols allow it to systematically vary the environment in which an algorithm operates and thus to better assess possible strengths and weaknesses an algorithm may have. In addition, by choosing the open set partitions as a uniform data basis for all algorithms, I lay the foundation for a meaningful comparison of their performances.

At this point, it has to be noted that some of the examined approaches have also been evaluated based on specifically designed open set protocols and procedures. For example, as can be seen in table 3.2, the evaluation of EVM (Rudd et al., 2018) and PROSER (Zhou et al., 2021b) is based on protocols proposed by Scheirer et al. (2014) and Perera et al. (2020) respectively. However, the OLETTER protocol introduced by Scheirer et al. (2014) constitutes a single instance of an open set environment. This implies that the environment in which the model operates stays constant. The same also holds true for the open set partition used to assess OpenMax (Bendale and Boult, 2016). In contrast, the work of Bhoumik (2021) enables a systematic variation of the degree of similarity within *KKCs*, between *KKCs* and *KUCs*, and between *KKCs* and *UUCs* across the three protocols. To a certain degree, the protocols established by Perera et al. (2020) do allow a systematic variation of the level of similarity within *KKCs* and between *KKCs* and *UUCs*. Nevertheless, in contrast to Bhoumik (2021), these open set partitions do not fully take advantage of the hierarchical richness provided by ImageNet but are based on rather confined datasets. The same applies to the open set partitions employed to assess Entropic Open Set Loss (Dhamija et al., 2018). It is for these reasons that I prefer the protocols designed by Bhoumik (2021) over other protocols and procedures used in research papers.

For the purpose of using the protocols devised by Bhoumik (2021), the University of Zurich, Switzerland, grants me access to the original source code. While setting up the code basis for generating the csv files that are required for instantiating the datasets, I observe a reproducibility issue. Concretely, executing the source code of the revised version of protocol 3 (Palechor et al., 2023, see section 2.2) in `Protocol3_revised.py`, the number of generated *KKCs*, *KUCs*, and *UUCs* vary across different executions of the script. This consequently implies that the number of samples in the training, validation, and test set varies as well across different script executions. To guarantee the reproducibility of the datasets, I inspect the source code and find that the reason for this behavior lies in the iteration process. Listing 3.1 shows an excerpt of one relevant position from the source code of the revised version of protocol 3. Concretely, `in_hier.tree[c].descendants_all` returns a set of identifiers for all descendants of the superclass `c` (Engstrom et al., 2019). As a Python set object is an unordered collection of distinct objects,[1] the iterator that is generated by the **`for`** loop does not return the set objects in a determined order across various script executions. Therefore, I convert the returned set to an ordered list by using Python's built-in function `sorted()`.[2] Listing 3.2 illustrates an excerpt of the modified version of the source code. The same procedure is applied to another line of code in the aforementioned source code of protocol 3, where the first **`for`** loop is defined over a list of manually selected superclass WordNet IDs.

```
# iterate through all subclasses of the selected WordNet IDs
for c in mixed13_superclass_wnid:
   for cnt, wnid in enumerate(in_hier.tree[c].descendants_all):
      # select WordNet IDs at even indices as known classes
      if wnid in in_hier.in_wnids and cnt%2 == 0:
         ...
```

Listing 3.1: Applying a For Loop on an Unordered Set (Protocol 3)

```
# iterate through all subclasses of the selected WordNet IDs
for c in mixed13_superclass_wnid:
   for cnt, wnid in enumerate(sorted(in_hier.tree[c].descendants_all)):
      # select WordNet IDs at even indices as known classes
      if wnid in in_hier.in_wnids and cnt%2 == 0:
         ...
```

Listing 3.2: Applying a For Loop on a Sorted List (Protocol 3)

I find very similar issues in the original source code of protocol 1 and protocol 2 (Bhoumik, 2021). For protocol 2, listing 3.3 illustrates the iteration over the unordered set of all `hunting_dog` subclasses in the nested **`for`** loop. All subclasses that are covered by ImageNet are then added to a dictionary. Finally, the first half of the dictionary entries are selected as known classes. Because of the unordered nature of Python set objects, these known classes vary across different script executions, which also affects the number of samples from *KKCs*. The solution is identical to the one in the revised version of protocol 3, namely by converting the unordered set to a sorted list using Python's built-in function `sorted()`. This conversion procedure is applied to two other lines of code in the original source code of protocol 2 as well as to three lines of code in the original source code of protocol 1. However, in protocol 1, these modifications do not affect the outcome of running the script due to the source code logic. The modifications are only made for reasons of consistency. After these minor code adaptations, running the scripts repeatedly returns identical results across the executions. Consequently, the originally reported protocol splits (Bhoumik, 2021; see also table 2.2) do slightly change regarding the number of samples and can be inspected in table 3.1. I consequently use these modified versions of the protocols for performing my experiments.

---

[1] https://docs.python.org/3/library/stdtypes.html#set

[2] https://docs.python.org/3/library/functions.html#sorted

Protocol	**Open Set Partition**	**Training Size**	**Validation Size**	**Test Size**
Protocol 1	Known Known (116 classes)	116′212	29′061	5′800
	Known Unknown (67 classes)	69′680	17′420	3′350
	Unknown Unknown (166 classes)	-	-	8′300
Protocol 2	Known Known (31 classes)	31′793	7′950	1′550
	Known Unknown (30 classes)	28′891	7′228	1′500
	Unknown Unknown (55 classes)	-	-	2′750
Protocol 3	Known Known (155 classes)	157′744	39′446	7′750
	Known Unknown (91 classes)	93′394	23′349	4′550
	Unknown Unknown (166 classes)	-	-	8′300

Table 3.1: MODIFIED PROTOCOL SPLITS. The table presents the protocol splits, including the number of classes, after modifying the source code in favor of reproducibility.

```
for cnt, (wnid, ndesc_in, ndesc_total) in enumerate(in_hier.wnid_sorted):
   # check if WordNet ID = WordNet ID for Hunting Dog
   if wnid == hunting_dog:
      # iterate through all hunting dog subclasses
      for cnt, wnid in enumerate(in_hier.tree[hunting_dog].descendants_all):
         # check if subclass is a ImageNet Class
         ...
```

Listing 3.3: Applying a For Loop on an Unordered Set (Protocol 2)

Algorithm	**Architecture**	**KUCs in $\mathcal{D}_{train}$**	**Dataset**	**Feature Input**	**Evaluation Metric**
Entropic OSL (Dhamija et al., 2018)	LeNet++/ ResNet18	Yes	Experiment 1: • MNIST ($\mathcal{D}_c'$, $\mathcal{D}_c$) • NIST ($\mathcal{D}_b'$, $\mathcal{D}_b$) • Devanagari (Pant et al., 2012)/ notMNIST/CIFAR-10 ($\mathcal{D}_a$) Experiment 2: • CIFAR-10 ($\mathcal{D}_c'$, $\mathcal{D}_c$) • subset of CIFAR-100 ($\mathcal{D}_b'$, $\mathcal{D}_b$) • SVHN/CIFAR-100 ($\mathcal{D}_a$)	No	OSCR
OpenMax (Bendale and Boult, 2016)	AlexNet	No	• ILSVRC 2012 dataset ($\mathcal{D}_c'$, $\mathcal{D}_c$) • 15 K images of ILSVRC 2010/ 15 K fooling images ($\mathcal{D}_a$)	Yes	F-score
EVM (Rudd et al., 2018)	AlexNet	No	OLETTER protocol by Scheirer et al. (2014): • 15 random labels from LETTER ($\mathcal{D}_c'$, $\mathcal{D}_c$) • remaining 11 labels from LETTER ($\mathcal{D}_a$)	Yes	F-score
PROSER (Zhou et al., 2021b)	Wide ResNet	No	Protocol by Perera et al. (2020) Experiment 1: • MNIST ($\mathcal{D}_c'$, $\mathcal{D}_c$) • Omniglot (Lake et al., 2015)/ MNIST-noise/noise ($\mathcal{D}_a$) Experiment 2: • CIFAR-10 ($\mathcal{D}_c'$, $\mathcal{D}_c$) • ImageNet/ LSUN (Fisher et al., 2015) ($\mathcal{D}_a$) Ablation study: • 15 randomly selected classes from CIFAR-100 ($\mathcal{D}_c'$, $\mathcal{D}_c$) • other CIFAR-100 classes ($\mathcal{D}_a$)	No	F-score

Table 3.2: CHARACTERISTICS OF EVALUATED ALGORITHMS. The table gives an overview of the key characteristics of the examined approaches as originally presented by the authors. While the second column names the relevant architecture for the model, the third column denotes whether the respective approach uses training samples with labels in *KUCs*. The fourth column indicates which datasets have been used for evaluating the corresponding algorithms. Note that for EVM, the dataset used for performing the open world recognition task is not listed as this task is conceptually distinct from the OSR task. The column labeled *Feature Input* states whether the input to the model is given in feature space (=Yes) or in pixel space (=No).

3.1.3 Network Topology

When comparing the performances of the different algorithms, another limiting factor comes in the form of different network architectures. Table 3.2 shows the different network topologies that are used in the original papers. It has to be noted that both the OpenMax as well as the EVM model are EVT models and do not directly consist of a DNN. However, both models use a well-trained DNN in the form of AlexNet as a feature extractor since the input to the models is given in feature space and not in pixel space. Using heterogeneous network architectures across the algorithms impedes the comparability of reported results. In line with Perera et al. (2020), I therefore propose to use the same network architecture for all the evaluated algorithms when performing the comparison.

The question then arises which network topology to use? When it comes to open set classification, research suggests a correlation between the performance of a classifier in the OSR task and its pure classification performance (Yoshihashi et al., 2019; Perera et al., 2020). In the domain of image classification, deep convolutional neural networks are known for their powerful classification abilities (Krizhevsky et al., 2012; Sermanet et al., 2014; Zeiler and Fergus, 2014). Furthermore, the depth of these networks (i.e., the number of stacked convolutional layers) has a significant impact on the performance, a fact that is reflected in the network architectures of many high-performing models in the ILSVRC (Ioffe and Szegedy, 2015; Russakovsky et al., 2015; Simonyan and Zisserman, 2015; Szegedy et al., 2015). The positive effect of deeper networks is even visible in open set classification (Roady et al., 2020). However, due to their vulnerability to the problem of vanishing/exploding gradients (Bengio et al., 1994; Glorot and Bengio, 2010), deep convolutional neural networks apply techniques such as normalization initialization (LeCun et al., 1998; Glorot and Bengio, 2010; He et al., 2015) or batch normalization (Ioffe and Szegedy, 2015) to enable the convergence of stochastic gradient descent during training (LeCun et al., 1989). While enabling convergence, these techniques do not prevent the degradation problem, the phenomenon that adding layers to a deep model can lead to a higher training error (He and Sun, 2015; He et al., 2016a). He et al. (2016a) proposed a solution to this problem by introducing an identity shortcut connection that is combined with layers into a residual building block. The authors then propose a network architecture that is based on these building blocks, called ResNet, and show that this architecture alleviates the problem of vanishing gradients and effectively addresses the degradation problem. After its introduction, ResNet quickly gained popularity among practitioners and researchers, an observation which is reflected in a number of proposed network modifications (He et al., 2016b; Huang et al., 2016) and variants of ResNet such as ResNeXt (Xie et al., 2017) or DenseNet (Huang et al., 2017).

For running the experiments, I choose ResNet-50 as the network architecture for all evaluated algorithms. In my experiments, I train these networks from scratch and do not make use of the pretrained version provided by PyTorch.[3] By using a homogeneous architecture, I ensure better comparability of results. The choice is justified by the advantages of ResNet as described above. Furthermore, ResNet-based networks show a good classification performance on ImageNet,[4] an important observation as the used protocols are based on ImageNet.

---

[3] https://pytorch.org/vision/stable/models.html

[4] https://pytorch.org/vision/stable/models.html#classification

3.2 Evaluation Metrics

In section 1.1, the two main goals of an open set classifier are introduced: firstly, a correct classification of samples that are associated with a known class $c \in$ *KKCs* seen during training. Secondly, the classifier should reject samples whose class is a member of *KUCs* $\cup$ *UUCs* (Geng et al., 2021). A correct rejection in this light means not to assign such a sample to one of the classes in *KKCs*, but to recognize the "unknownness" of the sample. Depending on the approach, recognizing the "unknownness" can mean, for example, assigning the sample to a dedicated background class as done in PROSER (Zhou et al., 2021b) or rejecting the sample because of the maximum Softmax score being below a defined probability threshold as in Entropic Open Set Loss (Dhamija et al., 2018). Having this in mind, an appropriate evaluation metric should accurately reflect these two goals such that the performance of classifiers can be assessed and compared. However, in contrast to the closed set context, the additional task of handling the unknown classes poses an additional challenge to the evaluation metric. This challenge is usually met by adapting established multi-class evaluation metrics (Dhamija et al., 2018; Geng et al., 2021). However, a closer look reveals that many metrics used in research papers have notable drawbacks.

3.2.1 Common Metrics and Their Drawbacks

In their survey on the advances in OSR, Geng et al. (2021) list some frequently used evaluation metrics in the OSR context:

- **Accuracy for Open Set Recognition**: Rooted in the closed set multi-class version, accuracy is defined as

  $$\mathcal{A}_O = \frac{\sum_{i=1}^{C}(TP_i + TN_i) + TU}{\sum_{i=1}^{C}(TP_i + TN_i + FP_i + FN_i) + (TU + FU)} \tag{3.1}$$

  Here, TP_i, TN_i, FP_i, and FN_i refer to the true positive, true negative, false positive, and false negative for the i^{th} known known class, with the class index $i \in \{1, 2, \dots, C\}$ and C denoting the size of *KKCs*. Moreover, the number of samples whose class label is in *KUCs* $\cup$ *UUCs* and that are correctly rejected is represented by TU. Similarly, FU refers to the number of samples whose class label is in *KKCs* and that are incorrectly rejected.

- **F-Measure for Open Set Recognition**: Taking the precision P as well as the recall R into account, the F-measure is defined as

  $$F = 2 \times \frac{P \times R}{P + R} \tag{3.2}$$

  Following a One-vs-Rest (OvR) approach based on a confusion matrix, the F score in closed set multi-class classification is computed for every class. Afterward, the set of class-wise F scores is subject to an averaging technique, leading to a single F score.[5] Extended to the open set context, this leads to all samples with labels $\in$ *KUCs* $\cup$ *UUCs* being treated as members of one additional class. This class then leads to another row entry in the confusion matrix (Geng et al., 2021).

---

[5] https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

Although these evaluation metrics account for the unknown classes, they insufficiently mirror the ability of an open set classifier to meet the OSR goals mentioned above. For example, Geng et al. (2021) point out that in the F-score, the correctly rejected unknown samples count as TP, which is misleading since TP now refers to samples that have not been seen during training. This shortcoming is addressed by Júnior et al. (2017), which provide a modified F-score that only considers Precision and Recall for classes in *KKCs*. Concretely, equation 3.2 uses the following versions of Precision and Recall for computing the macro-F-score and the micro-F-score, respectively:

$$P_{macro} = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FP_i}, R_{macro} = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FN_i} \tag{3.3}$$

$$P_{micro} = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C}(TP_i + FP_i)}, R_{micro} = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C}(TP_i + FN_i)} \tag{3.4}$$

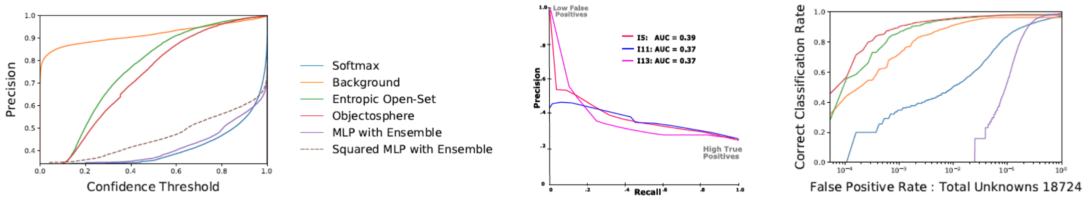
However, there are other reasons why neither the accuracy as in equation 3.1 nor the various forms of the F-score as described above are suitable evaluation metrics for OSR. On the one hand, the accuracy metric as given in equation 3.1 is sensitive to imbalanced datasets. For example, assuming a testing dataset that consists predominantly of unknown samples that are correctly rejected, a high accuracy may be achieved even if the classifier itself shows a very poor classification performance on known samples (Geng et al., 2021). While a weighted accuracy metric as proposed by Júnior et al. (2017) deals with this difficulty, there is a more serious limitation. Namely, the single value that results from applying either the accuracy or the F metric does not allow for a differentiated assessment of the performance of an open set classifier regarding the two OSR goals. Concretely, an isolated statement on either the classification performance or the rejection performance of a classifier is not possible since both key figures are unified in the returned single value. In short, these metrics do not adequately measure what they are supposed to measure in the OSR context.

Besides accuracy and the F-score, other metrics are frequently used in research papers to evaluate the performance of open set models. Dhamija et al. (2018) especially highlight the following metrics:

- **Accuracy/Precision vs. Confidence Curve**: This plot, applied by Lakshminarayanan et al. (2017), shows the performance of an open set classifier at various Softmax thresholds. The threshold, plotted on the horizontal axis, thereby takes on the role of a rejection threshold θ that determines when a sample is rejected due to the maximum Softmax score being $< \theta$.

- **Area Under the Curve (AUC) of a Precision Recall Curve**: The basis of this metric is a plotted curve that represents various Precision-Recall combinations at a given rejection threshold. Such a Precision-Recall combination can be interpreted as an operating point when selecting the associated rejection threshold. Depending on the application domain, the chosen rejection threshold naturally leads to either a high precision or a high recall. The metric itself then computes the area under this curve.

In their analysis, Dhamija et al. (2018) then point out the disadvantages of these metrics when used to evaluate the performance of a classifier in an open set context. Assuming a One-vs-Rest approach (as described above), the *Accuracy vs. Confidence Curve* suffers from high sensitivity to an imbalanced dataset. Furthermore, the curve is not suited for comparing algorithms since threshold values are not relatable in a meaningful way across different algorithms. Also, the authors illustrate in figure 3.1(a) that the *Accuracy vs. Confidence Curve* is vulnerable to scaling errors.



(a) Precision vs. Confidence Curve (b) AUC of PR (c) Open Set Classification Rate Curve

Figure 3.1: EVALUATION METRICS IN OSR. This figure illustrates different evaluation metrics that are used in research papers to assess the performance of an open set algorithm. For detailed information on the algorithms and the data sources associated with the figures, see Dhamija et al. (2018). While (a) plots the precision against an increasing confidence threshold for a collection of algorithms, (b) shows the Precision-Recall curves with their associated AUC values. Finally, (c) illustrates the OSCR curves for the same algorithms as used in (a). *Source: (Dhamija et al., 2018)*

According to the figure, it seems that the curve with the label *Squared MLP with Ensemble* seems to be superior to the one labeled *MLP with Ensemble*, despite being the same curve with squared Softmax scores. This contradicts the idea that monotonically re-normalizing the Softmax scores should not affect an evaluation metric. Regarding the AUC, Dhamija et al. (2018) note that the score itself does not allow to draw conclusions on the possible operating points. They exemplify this observation in 3.1(b): choosing an operating point at the precision of 0.8, algorithm *I13* outperforms algorithm *I5* with respect to recall, despite *I5* having the highest AUC score. This implies that an algorithm with the highest AUC score does not necessarily provide the best operating points for a specific application domain. The authors conclude that, despite being a widely used measure in research papers, the AUC of a PR curve is not a reliable metric for selecting the best-performing algorithm.

Besides the criticism offered by Dhamija et al. (2018), similar points as for the above-mentioned OSR-versions of Accuracy and the F-score can be made for the Precision vs. Confidence curve as well as the AUC score: the very definition of these metrics does not allow for a differentiated assessment of the performance of an open set classifier regarding the two OSR goals. Hence, they do not adequately measure what they are supposed to measure. This realization motivated Dhamija et al. (2018) to design a new evaluation metric: the Open Set Classification Rate (OSCR) curve.

3.2.2 Open Set Classification Rate Curve

To address the issues mentioned in subsection 3.2.1, Dhamija et al. (2018) introduce the OSCR curve, which is created by plotting the *Correct Classification Rate* (CCR) against the *False Positive Rate* (FPR) at different probability score thresholds θ:

$$
\begin{aligned}
\mathrm{CCR}(\theta) &= \frac{\big|\{x \,|\, x \in \mathcal{D}_c \wedge \operatorname{argmax}_c P(c \,|\, x) = \hat{c} \wedge P(\hat{c} \,|\, x) > \theta\}\big|}{\big|\mathcal{D}_c\big|}, \\
\mathrm{FPR}(\theta) &= \frac{\big|\{x \,|\, x \in \mathcal{D}_u \wedge \max_c P(c \,|\, x) \geq \theta\}\big|}{\big|\mathcal{D}_u\big|}
\end{aligned}
\tag{3.5}
$$

According to the authors' definition, the CCR represents the fraction of known samples in $\mathcal{D}_c$ where the actual class $\hat{c}$ is assigned the highest probability score by the model and where this probability score exceeds the threshold θ. The FPR on the other hand refers to the fraction of unknown samples in $\mathcal{D}_u$ where the highest probability score over all known classes in $\mathcal{C}$ as assigned by the model is greater than or equal to the threshold θ. In figure 3.1(c), Dhamija et al. (2018) provide an example of the OSCR metric being applied to evaluate the performance of various algorithms.

Reflecting on the definition of the OSCR metric, some advantages over the metrics elucidated in subsection 3.2.1 become apparent. First, the metric allows a performance comparison as the FPR has the same meaning across different algorithms. Furthermore, the OSCR metric represents the two OSR goals much better than the aforementioned metrics. Concretely, the CCR mirrors the ability of a classifier to correctly classify samples from known classes, while the FPR reflects the ability to correctly reject unknown samples. Importantly, the OSCR metric does not merge these two key figures in one single value, therefore allowing a differentiated assessment of the OSR goals at various threshold levels.

3.2.3 Evaluation Metric Used in Experiments

To compare the performance of the algorithms that I examine in this thesis, an evaluation metric is necessary. Looking at table 3.2, it becomes clear that all approaches but Entropic Open Set Loss use the F-score to measure and compare the performance. For the reasons set out in the subsections 3.2.1 and 3.2.2, I consider the OSCR metric (Dhamija et al., 2018) to be the best currently available fit for the experiments. I therefore use this metric to evaluate the performance of the algorithms on the test set. Subsequently, plotting the OSCR curves for the respective algorithms allows a consistent performance comparison.

Chapter 4

Experimental Setup

A carefully thought-out experimental setup facilitates the comparison of the algorithms. This chapter introduces the utilized hard- as well as the software. This is followed by a description of the data preprocessing and the network architecture that builds the basis of the comparison. Finally, the chapter presents the configuration decisions as well as the hyperparameter specifications that are required when working with the algorithms.

4.1 General Setup

To conduct the comparison between the open set algorithms Entropic Open Set (Dhamija et al., 2018), OpenMax (Bendale and Boult, 2016), EVM (Rudd et al., 2018), and PROSER (Zhou et al., 2021b), I use the Python programming language (Van Rossum and Drake, 2009) and the open source machine learning framework PyTorch (Paszke et al., 2019). The Artificial Intelligence and Machine Learning (AIML) Group in the Department of Informatics at the University of Zurich, Switzerland, provides me access to a server that is equipped with eight GeForce RTX™ 2080 Ti Turbo 11G GPUs. Since my experiments are computationally expensive, I make use of this server.

For reasons of manageability, I create two Python packages that structure my code. On the one hand, the package `imageNet_protocols` contains all the functionality that is needed to access the data associated with the three ImageNet protocols (see section 3.1.2). The functionality of this package is discussed in more detail in the next section. On the other hand, the package `eosa` (Evaluation of Open Set Algorithms) organizes the code that is required to evaluate the algorithms. Hereafter, the crucial modules of this package are briefly explained.

`approaches`: This module maintains a separate Python class for every approach, including the baseline approach in the form of thresholding on traditional Softmax (see subsection 2.1.1). These classes consist of attributes and methods that allow the training, validation, and testing of the approaches. This includes the evaluation of the trained models on the validation set, the storage of the best training model, as well as the computation and storage of the OSCR metric. Importantly, these classes directly access or integrate the source code of the algorithms. This source code is open source and hosted on GitHub (Dhamija, 2021; Zhou et al., 2021a). In case of Entropic Open Set Loss, OpenMax, and EVM, the source code is provided in a Python package termed `VAST` (Dhamija, 2021).

`architectures`: Since I use a specific version of the ResNet-50 architecture for my experiments (see section 4.3), the PyTorch-based implementation requires a slight adaptation. This module provides classes that implement this adjustment for a series of ResNet variants (i.e., ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152). These classes directly inherit from the ResNet class that is provided by PyTorch[1] and therefore take the correct architecture into account. An instance of a class of this module has the property that it returns the logit values as well as the deep feature representation when called on a data sample. Having access to the deep features is necessary since some of the evaluated approaches operate on deep feature representations of data samples.

`data_prep`: As pointed out in chapter 2, OpenMax and EVM rely on deep feature representations of the data samples in order to build, evaluate, and test their models. Since the datasets consist of samples in pixel space, a mapping into feature space is required. This module contains two classes, `EVMExtractor` and `OpenMaxExtractor`, that perform this task. While both classes inherit from the same parent class, `OpenMaxExtractor` additionally incorporates functionality which ensures that only the correctly classified known training samples $x \in \mathcal{D}_c^{'}$ are considered for feature extraction (see section 2.1.3). Concretely, a fully trained DNN can be passed as an argument to the constructor of such a class. After instantiation, a PyTorch `DataLoader`[2] object managing the data samples of a specific dataset is handed over to certain class methods. The DNN is then used to map the data samples to their feature representations. The class finally returns the feature representations and, optionally, the logit values of the data samples. More specifically, the class provides these features and logit values in the data structure that is requested by the open source algorithms (Dhamija, 2021).

`tools`: As discussed in detail in section 4.4, the evaluated algorithms rely on various hyperparameters and can be configured in various ways. This increases the importance of having a mechanism that allows conducting the experiments in a structured and guided way. This module hosts a function that instantiates and configures a Python argument parser[3] object and performs the necessary checks on the passed arguments. The basic idea of this function is to provide a mechanism that allows to run the experiments from a command line interface/interpreter (CLI) in a guided way. Specifically, for a selected algorithm, this function ensures that all relevant hyperparameters are specified and that the algorithm can only be configured in a valid way. Importantly, the argument parser object systematically guides the CLI user towards a correct configuration by pointing to the optional and required arguments.

4.2 Dataset

As clarified in section 3.1.2, I use the modified versions of the three ImageNet protocols developed by Bhoumik (2021) as the data basis for the comparison of the selected algorithms. This is possible as the University of Zurich, Switzerland, grants me full access to the source code associated with these protocols. To work with these protocols, I set up a Python package, named `imageNet_protocols`, that hosts all functionality that is needed to generate the PyTorch `Dataset`[4] objects that manage the data associated with the protocols. This functionality can be broadly divided into two categories.

---

[1] `https://pytorch.org/vision/stable/_modules/torchvision/models/resnet.html`

[2] `https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader`

[3] `https://docs.python.org/3/library/argparse.html`

[4] `https://pytorch.org/docs/stable/data.html#map-style-datasets`

Generation of Access Files: The original source code (Bhoumik, 2021) generates, for every protocol, CSV files that hold the file names of the needed images from the ILSVRC 2012 dataset as well as the assigned class targets. Every protocol script generates separate CSV files for the training, the validation, and the test dataset (for details, see Bhoumik (2021)). This information can then be used by the custom `Dataset` class to load the images. It is assumed that access to the ILSVRC 2012 dataset is given. The `imageNet_protocols` package provides a function that can be used to (re)generate these access files for the desired protocol(s).

Generation of Datasets: The `imageNet_protocols` package offers a function that returns `Dataset` objects for a specified protocol. Concretely, the function takes the file system paths to a selection of the generated CSV access files as arguments. Using these arguments, it then instantiates an object of a custom `Dataset` class (Bhoumik, 2021) for every CSV file. Relying on the information stored in the CSV file, this custom `Dataset` class loads the images and prepares the associated integer labels. Importantly, the class allows to apply certain transformations[5] to the images. Regarding the protocols, Bhoumik (2021) makes use of various transformations. First, the images are resized[6] to a uniform size of 300×300 to account for the different shapes and sizes of images that are part of the ImageNet database. This is followed by cropping[7] the images at the center, using an output size of 224×224. Then, the images are converted to PyTorch tensors[8] and subsequently normalized,[9] using channel-wise means of $[0.485, 0.456, 0.406]$ and standard deviations of $[0.229, 0.224, 0.225]$. These aforementioned transformations are applied to the training, validation, and test images. Only to the training images, additional data augmentation techniques in the form of a random horizontal flip[10] with a default probability of 0.5 as well as a random rotation[11] of ± 10 degrees are applied after the cropping operation. Data augmentation techniques are known measures against overfitting on image data (Simard et al., 2003; Cireşan et al., 2011, 2012). For my work, I adopt all these transformations, which are defined in the source code, without modifications.

To conduct my experiments, I make use of the above-described functionality provided by the `imageNet_protocols` package. Specifically, for a chosen protocol, I generate the required training, validation, and test `Dataset` objects and pass them individually to the constructor of PyTorch's `DataLoader` class. Additionally, I define a default batch size of 64 and ensure that the `DataLoader` instance for the training data applies random shuffling.

4.3 Network Architecture

For the reasons stated in section 3.1.3, I use the ResNet-50 architecture for my experiments. As listed in table 2.3, ResNet-50 consists of 16 bottleneck building blocks. Its depth provides a reasonable trade-off between performance[12] and the number of trainable parameters and thus the required training time (He et al., 2016a). Using a deeper architecture does not improve the performance substantially but does lead to longer training times (He et al., 2016a; Bianco et al., 2018; Leong et al., 2020). However, instead of directly loading the ResNet-50 implementation from Py-

---

[5] https://pytorch.org/vision/stable/transforms.html

[6] https://pytorch.org/vision/main/generated/torchvision.transforms.Resize.html

[7] http://pytorch.org/vision/main/generated/torchvision.transforms.functional.crop.html

[8] https://pytorch.org/vision/main/generated/torchvision.transforms.ToTensor.html

[9] http://pytorch.org/vision/main/generated/torchvision.transforms.Normalize.html

[10] http://pytorch.org/vision/master/generated/torchvision.transforms.RandomHorizontalFlip.html

[11] http://pytorch.org/vision/main/generated/torchvision.transforms.RandomRotation.html

[12] https://pytorch.org/vision/stable/models.html#classification

Torch, I use a slightly altered version of the network. Concretely, I add a fully connected layer[13] that maps the pooled and flattened output of the last convolutional layer into the deep feature space. From there, another fully connected layer maps the deep feature representation to the logit values. This adaptation is motivated by work in the domain of face verification. In this context, Ranjan et al. (2017) present the architecture of a typical deep convolutional neural network for face verification, where a fully connected layer maps the output of the last convolutional layer to its feature representation. Then, another fully connected layer is applied to these features, resulting in the logit values. Hence, the penultimate layer of such a network generates deep features. As the authors argue, the idea of such a penultimate layer is to produce distinctive features. More precisely, features from multiple samples of the same class should be located in close proximity to each other in the deep feature space. Moreover, this cluster of features should ideally be distinguishable from feature clusters that are associated with other classes. Having features that satisfy these requirements encourages their use in other tasks. However, it has to be noted that such a deep feature layer by itself does not guarantee that the generated features are actually distinctive. In fact, Ranjan et al. (2017) point to the link between a powerful deep feature layer and the used loss function. Since OpenMax (Bendale and Boult, 2016) and the EVM (Rudd et al., 2018) operate on deep features that are assumed to be distinctive, I integrate this deep feature layer into the ResNet-50 architecture. The modified architectures are provided by the `architectures` module that has been introduced in section 4.1.

4.4 Algorithm Configuration and Hyperparameters

As indicated in the relevant research papers, the performance of the evaluated algorithms depends on various hyperparameters (Bendale and Boult, 2016; Rudd et al., 2018; Dhamija et al., 2018; Zhou et al., 2021b). In addition, using these algorithms may require some further configurations. The following subsections discuss these hyperparameters and configurations for every algorithm. Furthermore, I present the configurations and hyperparameters that build the basis for the experiments (which are described in the next chapter) and justify these decisions.

4.4.1 Thresholding on Traditional Softmax

As introduced in subsection 2.1.1, this approach defines a probability threshold θ that is used to decide if an input sample x is classified as a known class or if it is considered to be a member of an unknown class and therefore rejected. In sum, the sample x is passed as an input to the trained network, resulting in a logit score for every of the C training classes. These logit values are subsequently passed to the Softmax function and the maximum resulting probability score is then compared against the threshold θ. This brief description points to the essential elements of the algorithm, which are outlined below. Since this approach represents the most basic way to address the task of OSR, I use its performance results as the baseline.

Network Architecture & Loss Function: As justified in section 4.3, I opt for the ResNet-50 network architecture. Associated with this choice, the question of the deep feature dimensionality arises. According to Donahue et al. (2014), powerful deep features incorporate semantic knowledge that allows assigning the associated input images to categories with a semantic meaning. In other words, grouping similar deep feature representations should lead to a situation where the associated input samples build a semantically meaningful category. More importantly, this emerging category should not be obvious when only looking at the pixel space representation

---

[13] https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

of these samples. However, as Donahue et al. (2014) further note, a very high dimensional deep feature space might fail to achieve this goal. Intuitively, the dimensionality of this feature space should not be too small in order to not be overly restrictive regarding the possible semantic clusters. Taking these findings and considerations into account, I configure the ResNet-50 network with a 512-dimensional feature space. The same dimensionality has also been used by Donahue et al. (2014) and can be observed in the ResNet-18 architecture (He et al., 2016a). Furthermore, I optimize the cross entropy loss function[14] since it is associated with a fast training time and a robust generalization (Bishop, 2006).

Optimizer: The training of a DNN such as ResNet-50 in Pytorch requires an optimizer.[15] In this regard, Stochastic Gradient Descent (SGD) with momentum is a common choice. Introduced by Rumelhart et al. (1986b), the main idea of this method is to speed up the learning process by taking the gradients of the previous update steps into account when computing the parameter weights of the current step. However, there exist alternative optimizers such as Adam. This optimizer relies on the Adam optimization algorithm that has been presented by Kingma and Ba (2015). With Adam, the authors design an optimization algorithm that integrates the benefits of two other optimization methods: AdaGrad (adaptive gradient algorithm, Duchi et al., 2011) and RMSProp (root mean square propagation, Hinton et al., 2012). As Kingma and Ba (2015) demonstrate, Adam adapts the learning rate per parameter, is able to deal with sparse and noisy gradients, is efficient in terms of computation and memory requirements, and focuses especially on machine learning problems that include large datasets or networks with a large number of parameters.

To find a solid baseline model, I try the SGD as well as the Adam optimizer in the training procedure. Both optimizers require some arguments. In case of SGD,[16] the most important parameters are the learning rate α and the momentum term β. While the learning rate α specifies the update step (Goodfellow et al., 2016), the momentum term β determines the number of gradients from preceding update steps that are taken into account for computing the weights at the current update step (Rumelhart et al., 1986b). Adam,[17] on the other hand, uses the learning rate α and two beta coefficients β_1, β_2. While the role of the learning rate and the β_1 coefficient is the same as in SGD, the β_2 coefficient plays a role in adapting the learning rate on the basis of an average of magnitudes of recent gradients (Hinton et al., 2012). For more details, refer to Kingma and Ba (2015). Regarding the training procedure, these parameters become hyperparameters since optimal optimization parameters such as the learning rate can generally not be determined analytically (Reed and Marksll, 1999). Therefore, for both optimizers, I specify a selection of hyperparameter combinations that are used to train various models. I then select the model that performs best on the validation set as the baseline model. While the specific details of the training procedure are presented in subsection 5.1.1, at this point, I introduce the used hyperparameter combinations:

- **Adam ($\boldsymbol{\alpha, \beta_1, \beta_2}$)**: [(0.001, 0.9, 0.999), (0.01, 0.9, 0.999), (0.0001, 0.9, 0.999), (0.00001, 0.9, 0.999), (0.001, 0.8, 0.999), (0.001, 0.7, 0.999), (0.001, 0.6, 0.999), (0.001, 0.7, 0.9)]
- **SGD ($\boldsymbol{\alpha, \beta}$)**: [(0.1, 0.9), (0.01, 0.9), (0.001, 0.9), (0.0001, 0.9), (0.01, 0.6)]

Regarding Adam, I use the default settings (0.001, 0.9, 0.999) as suggested by Kingma and Ba (2015) as the starting point and then vary the learning rate as well as, to a certain degree, the beta coefficients. In case of SGD, I proceed similarly. Focusing primarily on the learning rate α is motivated by the importance of this parameter in gradient-based techniques (Bengio, 2012; Goodfellow et al., 2016).

---

[14] `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`

[15] `https://pytorch.org/docs/stable/optim.html`

[16] `https://pytorch.org/docs/stable/generated/torch.optim.SGD.html`

[17] `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`

4.4.2 Entropic Open Set Loss

Since Entropic Open Set Loss (Dhamija et al., 2018) trains a DNN, similar configuration decisions as in the Softmax approach need to be taken.

Network Architecture & Loss Function: As argued in subsection 3.1.3, I use the same network architecture for all the evaluated algorithms to guarantee comparability. Therefore, ResNet-50 with a 512-dimensional deep feature space is the subject of the training procedure. The loss function comes in the form of the Entropic Open Set Loss as presented in subsection 2.1.2. Concretely, I load its original open source implementation as provided by Dhamija (2021).

Optimizer: For the same comparability reasons, I take the identical optimizers as in the Softmax approach into account. Namely, SGD and Adam. However, instead of experimentally determining the optimal hyperparameter combination as done for the Softmax approach, I adopt the optimizer and the associated hyperparameters that lead to the selected baseline model as explained in subsection 5.1.1. In other words, the basic idea is to first find the baseline model and then use its associated optimizer and hyperparameters to train the Entropic Open Set model from scratch. This procedure is explained in detail in subsection 5.1.2.

4.4.3 OpenMax

The fact that Bendale and Boult (2016) integrate Extreme Value Theory (Fisher and Tippett, 1928) into the OpenMax design has some implications for the configuration of the algorithm.

Network Architecture: The OpenMax algorithm relies on a DNN that generates and returns deep feature representations of the input samples as well as the samples' logit values (Bendale and Boult, 2016). For reasons of comparability (see subsection 3.1.3), I use the baseline model that has been selected for a specific protocol (see subsection 5.1.1) as the extracting network.

Tail Size: Since Bendale and Boult (2016) fit a Weibull distribution for every training class, a tail size must be specified. Concretely, the source code (Dhamija, 2021) provides a training function that accepts the relevant hyperparameters (i.e., the tail size and the distance multiplier) in the form of lists. Applying a grid search method, the function then computes a model for every possible combination of the hyperparameters. I therefore pass the following list of tail sizes to this function:

- Tail Size: [10, 100, 250, 500, 750, 1000]

The selection traces back to a preliminary experiment that I conduct. This trial run suggests better validation results for larger tail sizes. To highlight this point, I keep a relatively low tail size of 10 in the list. The fact that the OpenMax source code (Dhamija, 2021) uses the number of training samples per class as the default tail size supports my choice of evaluating large tail sizes.

Distance Multiplier: Besides the tail size, the aforementioned training function (Dhamija, 2021) treats the distance multiplier as its second hyperparameter. The distance multiplier, a float value, is applied to the η largest feature space distances from training samples of a particular training class (i.e., the positive samples) to the Mean Activation Vector (MAV) of the same class. Subsequently, a Weibull distribution is fitted on these modified distances. Hence, the distance multiplier acts as a scaling factor. To understand the implications of this distance multiplier, recall that OpenMax learns the class-specific (probabilistic) decision boundary in the feature space based on distance information from the positive training samples. However, if these positive data points are not a representative sample of the class' data distribution, there might be a considerable amount of test samples from the same class that show a higher distance to the MAV. These samples would subsequently not be associated with the class at all. To account for this scenario, the distance multiplier is set to values ≥ 1.0. At this point, it has to be mentioned that the original version of OpenMax, as presented by the authors and described in subsection 2.1.3, does not make use of the concept of a distance multiplier. In fact, this hyperparameter has been introduced in the source code (Dhamija, 2021). Therefore, keep in mind that, unless applying a distance multiplier of 1.0, a slightly modified version of OpenMax is used. Making use of the mentioned grid search capabilities, I pass the following list to the OpenMax training function:

- Distance Multiplier = [1.5, 1.7, 2.0, 2.3]

This choice is based on the preliminary experiment mentioned in the previous paragraph on the tail size. Concretely, the trial run suggests better models for distance scaling factors between 1.3 and 2.3.

Distance Metric: Since Bendale and Boult (2016) compute distances between feature space representations of data samples and MAVs in the training as well as in the testing phase, a distance metric is required. Specifically, the source code accepts the Euclidean as well as the cosine distance metric (Dhamija, 2021). For the experiments, I use the cosine distance metric. This choice is motivated by previous research results which suggest that Euclidean distance is not an ideal metric to measure the distances between deep feature representations of samples (Aggarwal et al., 2001; Rudd et al., 2018). Moreover, Rudd et al. (2018) achieve their results in a similar task using the cosine metric.

Alpha Value: With this value, Bendale and Boult (2016) specify the number of classes whose probability score is revised by the OpenMax algorithm. More precisely, OpenMax revises the α highest logit values of a test sample by taking its feature space representation and the fitted probability models into account. However, the research paper does not go into great detail regarding the choice of this hyperparameter but refers to a grid search procedure Bendale and Boult (2016). Therefore, I use the following values in the training procedure:

- Alpha = [2, 3, 5, 10]

4.4.4 Extreme Value Machine

Similar to the OpenMax algorithm (Bendale and Boult, 2016), the Extreme Value Machine (EVM, Rudd et al., 2018) depends to a large degree on EVT concepts (Fisher and Tippett, 1928). Since the EVM fits its distribution models on distance measures in deep feature space, similar design decisions and hyperparameters as in case of OpenMax emerge. Concretely, these EVM hyperparameters come in the form of the tail size, the distance multiplier, and the cover threshold.

Network Architecture: As indicated before and similar to OpenMax, Rudd et al. (2018) start the training and testing procedure by extracting deep feature representations of data samples. To perform this task, I yet again apply the baseline model that has been selected for a specific protocol (see subsection 5.1.1) as the extracting network. This decision is yet again based on the same reasons as explained in subsection 3.1.3.

Tail Size: As described in detail in subsection 2.1.4, Rudd et al. (2018) compute for every training sample of a particular class the half-distances to the τ closest negative training samples and subsequently fit a Weibull distribution on this collection of distances. In this case, τ refers to the tail size of the distribution. As stated by the authors, EVT does not provide detailed instructions on how to find the optimal tail size τ. Therefore, they fall back on cross-class validation to find the optimal parameter. Since the source code of the EVM training function (Dhamija, 2021) provides, identical to the OpenMax training function, a grid search functionality, I define a list of tail sizes. The function then computes the probability distributions for every possible hyperparameter combination.

- Tail Size = [10, 25, 50, 75, 100, 150, 200, 300, 500, 1000]

The selection of tail size values is motivated by a preliminary experiment that I conduct. This trial run allows me to specify meaningful stepsizes. A tail size of 75 served as a starting point since this value corresponds to the optimal tail size identified by Rudd et al. (2018) on a 15 K training set. Regarding the higher values, I refer to the notion of a positive correlation between the feature space dimensionality and the number of "boundary points" (Rudd et al., 2018).

Distance Multiplier: This hyperparameter comes in the form of a float value and is applied to the distances from a specific EV to its τ closest negative training samples (see subsection 2.1.4) before fitting the distribution. To understand the implications of this hyperparameter, note that EVM learns the (probabilistic) decision boundary for a specific EV in the feature space based on distance information from negative training samples. If the known training classes are well separated in feature space, i.e., there is a considerable space between different clusters of class samples, inaccurate boundaries with respect to unknown classes can be the consequence. As an example, assume such a distinctive feature space as outlined above and a (probabilistic) decision boundary that is associated with an EV. An unknown test sample $x \in$ *UUCs* that is located between the EV and its closest negative training samples is then likely but erroneously associated with the known class of the EV. To avoid this scenario, a distance multiplier with a value ≤ 1.0 can be applied to adjust the boundary of the EV. Taking these considerations into account, I scale the distances by the following factors for my experiments:

- Distance Multiplier = [0.10, 0.20, 0.30, 0.40, 0.50, 0.70, 0.90, 1.00]

The selection of relatively low values is based on the preliminary experiment mentioned above. This trial run encourages the investigation of low values.

Cover Threshold: This hyperparameter traces back to the model reduction step that has been introduced by Rudd et al. (2018) to remove redundant EVs and their associated Ψ-models. Furthermore, this step reduces computational demands during testing/operation. However, since the hardware I am operating on provides plenty of computational resources, I do choose a cover threshold of 1.0. This basically means that all samples of a particular training class provide a usable Ψ-model.

Distance Metric: Following the rationale of the previous subsection on the OpenMax distance metric, as well as the recommendation by Rudd et al. (2018), I use the cosine distance.

4.4.5 PROSER

Taking the viewpoint of OSR being a calibration problem, PROSER synthesizes and subsequently uses unseen data samples and augments the closed set classifier with additional classifiers (Zhou et al., 2021b). The algorithmic structure of PROSER entails different hyperparameter specifications as well as design decisions.

Network Architecture: Zhou et al. (2021b) perform their experiments using a Wide ResNet architecture. Following the line of reasoning in the preceding subsections regarding comparability, I make use of a ResNet-50 architecture that maps input samples into a 512-dimensional deep feature space. Concretely, for a specific protocol, I use the baseline model that has been selected according to subsection 5.1.1. This choice is in line with the notion of extending a pretrained closed set classifier by dummy classifiers (Zhou et al., 2021b). As indicated in subsection 2.1.5, the authors generate new data instances using hidden representations of known training samples from different classes. As the official source code (Zhou et al., 2021a) reveals, this creation process is performed after the second (of three) group of blocks. However, the ResNet-50 architecture consists of four groups of blocks. To stay as consistent as possible with the logic applied in the Wide ResNet, I use the output of the third (i.e., the penultimate) group of blocks as the hidden representations. Therefore, I extend the original source code with customized code that implements this described logic.

Optimizer: Since the original source code that implements the PROSER training logic relies on the SGD optimizer and does not allow for specifying an alternative (Zhou et al., 2021a), the network is trained using the SGD optimizer.

Coefficients α, λ_1, and λ_2: According to the original source code (Zhou et al., 2021a), the training procedure needs various coefficients. On the one hand, α is used to initialize the Beta distribution. Furthermore, λ_1 and λ_2 act as weighting factors in the composite loss function. Since neither the research paper (Zhou et al., 2021b) nor the original source code elaborates on the handling and choice of these coefficients, I use the default values as defined in the original source code. These values are $\alpha = 1.0$ and $\lambda_1 = \lambda_2 = 1.0$.

Number of Dummy Classifiers: As illustrated in subsection 2.1.5, the fine-tuning procedure places the dummy classifiers between the target class and non-target classes, which reflects the idea of introducing classifiers for emerging unknown classes (Zhou et al., 2021b). The question therefore is by how many dummy classifiers should the model be augmented? In their paper, Zhou et al. (2021b) report results of their conducted experiments that suggest that a higher number of dummy classifiers can improve performance. However, the highest number of reported dummy classifiers is 3. Therefore, I augment the baseline model with the following number of dummy classifiers over the course of the training procedure:

- No. Dummy Classifiers = [1, 2, 3, 4, 5, 7, 9, 10, 15, 20, 30]

More precisely, for every element of this list, I train a model and evaluate it on the validation set. The best-performing model is then compared against the other algorithms on the test set. Details on the training procedure are presented in the next chapter.

Chapter 5

Experiments

Given a thoroughly designed experimental setup, the experiments can eventually be performed. This chapter highlights the important steps in executing the algorithms on all three protocols. For each algorithm, it illustrates which evaluation metric is used to measure the performance of the model on the validation set and how to arrive at a final model in the presence of hyperparameters. The chapter closes with a presentation and analysis of the experimental results.

5.1 Execution of Algorithms

In general, a certain pattern is common to the evaluation of all algorithms. Concretely, the protocol is specified and the algorithm of interest is selected and configured. Then, the required training and validation datasets are prepared and loaded. This is followed by instantiating the class associated with the algorithm and starting the training procedure. As presented in section 4.1, the mentioned class is loaded from the module `approaches`. The training procedure covers the training of the model as well as the evaluation of the trained model on the validation set. The model that performs best on the validation set is selected as the optimal model and subsequently exposed to the testing procedure, which measures the performance of the model on the loaded test set in terms of the OSCR metric (Dhamija et al., 2018). Generally, the classes loaded from the module `approaches` access or integrate the original source code of the algorithms (Dhamija, 2021; Zhou et al., 2021a). Deviations from this principle are explicitly mentioned and justified. Moreover, the basic idea is to find the best configurations and hyperparameters for a model on protocol 2 and then transfer these settings to protocols 1 and 3. Performing the optimization on protocol 2 is based on the understanding that this protocol represents an open set context of medium difficulty when compared to protocol 1 (easy) and protocol 3 (hard). The following subsections describe the training and testing procedure for every algorithm in detail. Additionally, the most important characteristics of the algorithm experiments are summarized in table 5.1. Note that the algorithm-specific configurations and hyperparameters, as introduced in section 4.4, are integral parts of these experiments.

5.1.1 Thresholding on Traditional Softmax

As explained in subsection 4.4.1, the utilized ResNet-50 network is optimized via the cross entropy loss function and represents a closed set classifier. This implies that the training and the validation dataset consist of samples from the *KKCs*. Starting with protocol 2, I train the model for 100 epochs, using one specific optimizer configuration as defined in subsection 4.4.1. After every epoch, I validate the current model on the validation dataset, using *accuracy* as the performance

	Softmax (Baseline)	**Entropic Open Set Loss**	**OpenMax**	**EVM**	**PROSER**
Composition of Training Set	*KKCs*	*KKCs*, *KUCs*	*KKCs*	*KKCs*	*KKCs*
Composition of Validation Set	*KKCs*	*KKCs*, *KUCs*	*KKCs*, *KUCs*	*KKCs*, *KUCs*	*KKCs*, *KUCs*
Composition of Test Set	*KKCs*, *UUCs*	*KKCs*, *UUCs*	*KKCs*, *UUCs*	*KKCs*, *UUCs*	*KKCs*, *UUCs*
Input Space	Pixel	Pixel	Feature	Feature	Pixel
Optimizer	Adam	Adam	-	-	SGD
evaluation metric	Accuracy	Weighted Average Confidence	CCR@FPR	CCR@FPR	ROC AUC

Table 5.1: KEY POINTS OF EXPERIMENTS. The table gives an overview of important characteristics of the conducted experiments. The first three rows refer to the classes that can be found in the respective datasets. The row *Input Space* denotes whether the input to the model is given in pixel space or in feature space.

metric. Accuracy is defined as the number of correct classifications, i.e., $Acc = \frac{TP+TN}{TP+FN+FP+TN}$, and is a frequently used metric for assessing the performance of classifiers (Branco et al., 2016). Having a balanced dataset further justifies its use (Batista et al., 2004). The model setup that leads to the best validation performance over all the defined epochs is then stored in memory for later retrieval. This procedure is then repeated for every of the remaining optimizer configurations. Then, the same is done for 200 and 300 epochs. Concretely, the model is trained from scratch for 200 and 300 epochs, respectively, for every defined optimizer configuration. This means that, in the end, there is one model for every optimizer-associated hyperparameter combination for 100, 200, and 300 epochs. More specifically, table A.1 provides an overview of all the resulting models and their associated validation performances on protocol 2. Hence, this approach provides me with a set of possible baseline models. Considering the accuracy measures, I opt for the model that has been trained over 200 epochs, using an Adam optimizer (Kingma and Ba, 2015) with hyperparameters $\alpha = 0.001$, $\beta_1 = 0.7$, $\beta_2 = 0.999$ as the baseline model. I prefer this model to the best-performing overall model (Adam-$(0.001, 0.6, 0.999)$ over 300 epochs) since the latter comes with little performance increase for a considerable amount of additional training time. Referring to the general procedure as set out in the introduction to this chapter, I transfer this optimizer configuration to protocol 1 and protocol 3. This means that, for every protocol, the baseline model is trained over 200 epochs, using an Adam optimizer with parameters $\alpha = 0.001$, $\beta_1 = 0.7$, $\beta_2 = 0.999$.

Having found the baseline model for a specific protocol, it can be passed to the testing procedure. As argued in chapter 3 and mentioned above, I rely on the OSCR curve to compare the algorithms. Therefore, the testing procedure loads the baseline model and computes the CCR as well as the FPR value for every possible threshold θ, using functionality provided by the `VAST` package[1] (Dhamija, 2021). These performance scores serve as the baseline against which the other algorithms are compared.

5.1.2 Entropic Open Set Loss

As has been introduced in subsection 2.1.2, this approach uses samples from *KKCs* as well as from *KUCs* to train the model. The training procedure therefore starts by loading the relevant training and validation datasets as well as the Entropic Open Set Loss function (Dhamija, 2021) and passing them to the training function. Taking on the optimizer configuration of the baseline model (i.e., Adam optimizer with $\alpha = 0.001$, $\beta_1 = 0.7$, $\beta_2 = 0.999$, see previous subsection), the loss function is minimized over 200 epochs. After every epoch, the current model is validated on the validation dataset, and the best-performing model over all epochs is stored in memory. Concretely, the model performance is measured in terms of the average confidence. This evaluation metric has been introduced by Schnyder (2021) and measures the confidence of a model in correctly classifying samples from known classes and rejecting samples from unknown classes. Let $\mathcal{D}_c^{''}$ be the set of validation samples belonging to $\mathcal{C}$ and $\mathcal{D}_b^{''}$ be the set of validation samples belonging to $\mathcal{B}$. Further, assume a probability distribution over the C known training classes. Then, the confidence associated with a single sample x is formally defined as:

$$Conf(x) = \begin{cases} y_t & \text{if } x \in \{\mathcal{D}_c^{'}, \mathcal{D}_c^{''}, \mathcal{D}_c\} \\ 1 - \max_c \; y_c + \frac{1}{C} & \text{if } x \in \{\mathcal{D}_b^{'}, \mathcal{D}_b^{''}, \mathcal{D}_u\} \end{cases} \tag{5.1}$$

In words, the confidence of a sample that belongs to a known class is given by the Softmax score of its target class t. For an unknown sample, the confidence is computed by subtracting the highest Softmax score $\max_c \; y_c$ over all known training classes C from 1 and then adding an offset of $\frac{1}{C}$. This offset, which is also applied by Dhamija et al. (2018) in the loss function, accounts for the fact that the model does not know a separate class for the unknown samples but has only the possibility to compute probability scores that are linked to known classes. In the end, the average confidence is computed by first summing up the confidence scores of all samples from the validation set and then dividing this sum by the number of samples in the validation set. However, the average confidence is generally vulnerable to class imbalance. For this reason, I use the weighted average confidence as the evaluation metric. This is achieved by computing the sum of confidences for the known samples and the unknown samples of the validation set separately. Based on these two sums, the average confidence of the known samples and the unknown samples can be calculated. Finally, I average these two averaged scores to get the weighted average confidence. Assuming $x^+ \in \mathcal{D}_c^{''}$, $x^- \in \mathcal{D}_b^{''}$, $|\mathcal{D}_c^{''}| = M$, and $|\mathcal{D}_b^{''}| = N$, this can formally be expressed as:

$$Conf_{w\_avg} = \frac{1}{2} \times \left(\frac{\sum_{i=1}^{M} Conf(x_i^+)}{M} + \frac{\sum_{j=1}^{N} Conf(x_j^-)}{N} \right) \tag{5.2}$$

Following this procedure, I train a model on every protocol. Importantly, for every protocol, I adopt the optimizer setup from the respective baseline model. The best-performing model on protocol 2 achieves an average weighted confidence of 0.6782. Similarly, protocols 1 and 3 result in average weighted confidences of 0.7670 and 0.7056, respectively. After finishing the protocol-specific training, the testing procedure begins with loading the test dataset as well as the best training model. Then, analogously to the Softmax approach, the CCR as well as the FPR values are computed. Finally, these scores are plotted to compare them against the baseline model.

---

[1] `https://github.com/Vastlab/vast/blob/2ad7fa23cb8f0f1acb71d6b6d9aced6a70562e0c/vast/eval/eval.py#L71`

5.1.3 OpenMax

Since the OpenMax approach works with deep feature representations of the data samples, I start the training procedure by extracting the features as well as the logit values from the required datasets, using the protocol-specific baseline model (see subsection 5.1.1) together with the `OpenMaxExtractor` class from the `data_prep` module (see section 4.1). Then, the OpenMax training function[2] is loaded from the `VAST` package (Dhamija, 2021) and subsequently executed. For a specific hyperparameter combination (i.e., tail size and distance multiplier), the function computes the parameters of a Weibull distribution for every training class. Since this process applies to every possible hyperparameter combination, a collated model for every hyperparameter combination can be stored in memory. Such a collated model consists of the distributional parameters and the MAV of every training class. More specifically, the distributional parameters are the result of a fitting process that considers the hyperparameters associated with the collated model. Having a set of collated models, the task is now to find the one that leads to the best validation performance on the validation set (consisting of the above-mentioned features and logit values). Concretely, a dedicated inference function[3] computes for every data sample of the validation set the probabilities of being associated with any of the known training classes. These probability scores are then used together with the logit values and a specific α value to revise the Softmax scores as described in detail in subsection 2.1.3. However, instead of using the original revision function,[4] I use a slightly customized version that returns the revised Softmax scores of only the known classes, but not the "pseudo" class. The reason for this decision will soon become apparent. Hence, considering the additional hyperparameter α, there is a need to evaluate the performance of the models associated with every possible *tail size-distance multiplier-alpha* hyperparameter combination. Specifically, referring to subsection 4.4.3 on the OpenMax hyperparameters, a total of $6 \times 4 \times 4$ models need to be evaluated on the validation set. This course of action is in line with the one applied by Bendale and Boult (2016), which perform a grid search to find the optimal hyperparameter combination.

At this point, the question of the evaluation metric arises. Here, I turn to the procedure chosen by Dhamija et al. (2018). They compute the CCR and FPR values and then define a series of FPR thresholds. For each of these thresholds, the associated CCR is then identified and reported. Basically, this procedure reports the performance in terms of the CCR at some predefined FPR values. While Dhamija et al. (2018) set these threshold values in their experiments at 0.0001, 0.001, 0.01, and 0.1, I choose values of 0.001, 0.01, 0.1, 0.15, and 0.2. Discarding the value of 0.0001 can be traced back to my observations that an FPR of 0.0001 is not a meaningful threshold in the context of these experiments. Furthermore, referring to the customized revision function mentioned above, I only use the revised Softmax scores of the known classes for the computation of the CCR and FPR values. The reason for this can be found in the definition of the OSCR metric (see equation 3.5). Namely, the CCR as well as the FPR are only defined for a probability distribution over the C known training classes. Table A.2 illustrates the outcome of the validation procedure on the validation set of protocol 2. For every evaluated model configuration, the CCR value at every defined FPR threshold is presented. In the optimal case, there would be one single model that performs best (i.e., reaches the highest CCR value) at every FPR. However, this situation does not occur in the present case, which makes the selection of a final model more challenging. Ultimately, I decide in favor of the model that is trained with a tail size of 1000, a distance multiplier

---

[2] `https://github.com/Vastlab/vast/blob/2ad7fa23cb8f0f1acb71d6b6d9aced6a70562e0c/vast/openset Algos/openmax.py#L69`

[3] `https://github.com/Vastlab/vast/blob/2ad7fa23cb8f0f1acb71d6b6d9aced6a70562e0c/vast/openset Algos/openmax.py#L105`

[4] `https://github.com/Vastlab/vast/blob/2ad7fa23cb8f0f1acb71d6b6d9aced6a70562e0c/vast/openset Algos/heuristic.py#L5`

value of 2.0 and an alpha value of 5. As done in table A.2, I abbreviate this model as (*1000_2.0_5*). Despite not scoring the highest at any FPR threshold, the selected model exhibits CCR values that are among the highest at any FPR threshold. In fact, it is associated with the highest sum of available CCR values over the specified FPR thresholds among all evaluated models. Following the general procedure, I then use this protocol-2-based optimal hyperparameter combination to train the model on protocol 1 and protocol 3. To avoid an inferior model due to non-transferable hyperparameters, I train various models and compare their performance metrics on the validation set. As can be seen in tables A.3 and A.4, the model (*1000_2.0_5*) still reaches competitive CCR values in both protocols and is therefore selected as the final model for the respective protocol. In sum, the final model on all three protocols is trained using a tail size of 1000, a distance multiplier value of 2.0 and an alpha value of 5.

Having determined the final model associated with a specific protocol, the testing procedure, which is very similar to the validation procedure, is started. This means that the test dataset in the form of the features and the logit values is prepared and the final model is loaded. Then, the aforementioned inference function computes the probability scores, which are subsequently used to establish the revised Softmax scores of the known classes. Ultimately, these revised probability scores build the basis for calculating the CCR and FPR values that represent the OSCR curve. The testing procedure finishes by storing these CCR and FPR values.

5.1.4 Extreme Value Machine

As in the case of OpenMax, the Extreme Value Machine (EVM) fits probabilistic distributions based on distances measured in feature space. Therefore, the training procedure begins with the sample-wise extraction of deep feature representations from the training and validation datasets. This is achieved by using the protocol-specific baseline model (see subsection 5.1.1) together with the `EVMExtractor` class from the `data_prep` module (see section 4.1). Subsequently, the EVM training function,[5] which is loaded from the `VAST` package (Dhamija et al., 2018), comes into play. For a specific hyperparameter combination (consisting of a tail size, a distance multiplier, and a cover threshold), this function fits various Weibull distributions for every training class. As illustrated in detail in subsection 2.1.4, every distribution is associated with an EV that belongs to the respective class. This fitting procedure is then repeated for every of the remaining hyperparameter combinations. Analogously to OpenMax, the training procedure continues by compiling a collated model for every hyperparameter combination. The collated model includes the EVs of every training class, as well as the distributional parameters that are associated with these EVs. Once again, the task is to find the collated model that performs best when evaluated on the validation dataset. The validation procedure therefore computes for every data sample of the validation set the probabilities of being associated with any of the known training classes. This computation is performed by the `VAST`-provided EVM inference function.[6] Then, the same method as described in the previous subsection on OpenMax is applied. This means that the selection of the final model is based on the comparison of CCR values at predefined FPR thresholds. Here, I use the same FPR threshold values, namely 0.001, 0.01, 0.1, 0.15, and 0.2. Therefore, the abovementioned probability scores serve as the basis for computing the required CCR and FPR values. Table A.5 summarizes the evaluation of the $10 \times 8 \times 1$ models on the validation set of protocol 2. Since no model performs best at every FPR threshold, the selection becomes a trade-off. Applying the same selection strategy as for OpenMax, I opt for the model that is trained with a tail

---

[5] https://github.com/Vastlab/vast/blob/2ad7fa23cb8f0f1acb71d6b6d9aced6a70562e0c/vast/opensetAlgos/EVM.py#L130

[6] https://github.com/Vastlab/vast/blob/2ad7fa23cb8f0f1acb71d6b6d9aced6a70562e0c/vast/opensetAlgos/EVM.py#L311

size of 75, a distance multiplier value of 0.20, and a cover threshold value of 1.0. Concretely, this model shows the highest sum of available CCR values over the specified FPR thresholds among all evaluated models. As done in table A.5, I abbreviate the model as (*75_0.20_1.00*). Having the general procedure in mind, I then try to apply the hyperparameter combination that is linked to the final model to protocol 1 and protocol 3. However, I observe that this transfer does not work in the case of EVM since these hyperparameters lead to inferior models on both protocols. Conducting a thorough analysis, I find that the distance multiplier is mainly responsible for this result. Furthermore, the gradation of the tail size values seems to be too fine-grained. Based on these insights, I specify slightly altered lists of hyperparameters and repeat the entire training procedure on protocols 1 and 3. The results of the model validation on both protocols are shown in tables A.6 and A.7. Applying the same selection strategy as on protocol 2, I select the model (*1000_0.50_1.00*) as the final one on protocol 1. On protocol 3, I choose the model (*150_0.40_1.00*). A possible explanation for the higher distance multipliers in protocols 1 and 3 can be found in the considerations presented in subsection 4.4.4. Recalling the number of known classes linked to the three protocols (see table 3.1), it is possible that the feature space associated with protocol 2 is more distinctive since there is a comparably low number of known training classes present. As a consequence, the distance values need to be downscaled more pronouncedly compared to protocols 1 and 3.

As the last step, the final model associated with a specific protocol is passed to the testing procedure. Then, the aforementioned inference function calculates the probability scores for every sample of the prepared test set. Finally, these scores serve as the basis for computing the OSCR-associated CCR and FPR values.

5.1.5 PROSER

As in the case of the other approaches, I integrate the released open source code (Zhou et al., 2021a) into the training and testing procedure. However, I additionally make use of some improvements that have recently been implemented in the context of a project by Elsaadany (2022). This project has been carried out with the Artificial Intelligence and Machine Learning (AIML) Group in the Department of Informatics at the University of Zurich, Switzerland. Most importantly, the project implements functionality that is not provided in the released source code but has been mentioned in the paper (Zhou et al., 2021b). Concretely, apart from some small bug fixes, two important features are introduced. On the one hand, the modified code ensures that only hidden representations from two samples that belong to different classes are mixed in the course of the instance creation. On the other hand, an additional function implements the computation of the bias term (see subsection 2.1.5). Thus, in order to run the algorithm with all the features proposed in the paper (Zhou et al., 2021b), I rely on the mentioned project code base (Elsaadany, 2022) to which the University of Zurich, Switzerland, grants me access to.

The training procedure starts by loading the necessary datasets as well as the baseline model, which is subsequently augmented by the specified number of dummy classifiers. The model is then fine-tuned for 10 epochs, which is the number that is defined in the original source code (Zhou et al., 2021a). Since the source code does not parameterize this variable,[7] I inherit this value. During this process, new data instances are generated and used together with original training samples to learn the weights of the dummy classifiers. After each training epoch, the bias term is computed and the current model is evaluated on the validation set. During validation, the current model is used to predict the $C + 1$ logit values for every sample of the validation set. The

---

[7] `https://github.com/zhoudw-zdw/CVPR21-Proser/blob/f5b53b90509e6460a783a0baa72bda52364810fd/proser_unknown_detection.py#L190`

bias term is then added to the logit representing the unknown class. Based on these logit values, the performance of the model is measured in terms of the ROC AUC (Area Under the Receiver Operating Characteristic Curve, Hanley and McNeil, 1982), which is the evaluation metric that is used by Zhou et al. (2021b).[8] The best-performing model over all epochs is stored in memory. Considering that the number of dummy classifiers represents a hyperparameter, I end up with eleven different models that integrate the bias term into their model. Table A.8 shows the performances of these models on the validation set of protocol 2. As can be seen, the best-performing model that makes use of the bias computation incorporates three dummy classifiers. For protocol 2, I therefore choose this model as the final one. However, when I transfer this optimal hyperparameter to protocols 1 and 3, I end up with uncompetitive models. Comparing the number of (known known) training classes across the three protocols, I observe that protocol 2 trains on 31 classes, while protocols 1 and 3 train on 116 and 155 classes, respectively (see table 3.1). As a consequence, I alter the hyperparameter list of trainable dummy classifiers and repeat the training procedure on protocols 1 and 3. Tables A.9 and A.10 illustrate the associated validation results. For protocol 1, these results lead me to choose the model with five dummy classifiers as the final one. When it comes to protocol 3, the final model incorporates twenty dummy classifiers.

Following the general procedure, the final model for a specific protocol is then assessed using the test set. The testing procedure therefore loads the final model and computes the logit values of every test sample in the same way as in the training procedure. The logit values are subsequently passed to the Softmax function, which computes the probability scores. The probability scores of only the known training classes are then used to compute the CCR and FPR values. The reason for ignoring the probability score linked to the dummy classifier is the same as mentioned in subsection 5.1.3. Namely, the OSCR metric is only defined for a probability distribution over the C known training classes.

5.2 Results

Conducting the experiments according to the procedures described in the previous section leads to a collection of CCR and FPR values for every algorithm. These values actually represent the OSCR curve (Dhamija et al., 2018) and are plotted to compare the performances of the algorithms on the test datasets against each other. Referring to table 5.1, the test datasets of all protocols comprise samples from *KKCs* as well as from *UUCs*. The resulting plots are depicted in figures 5.1, 5.2, and 5.3. Referring back to subsection 3.2.2, note that every plotted point represents a pair that consists of one CCR value and one FPR value. Each such pair is associated with a rejection threshold θ, which itself is not derivable from the plot. Hence, such a pair can be considered as an operating point when the associated rejection threshold is applied. Depending on the domain requirements, i.e., a focus on a high CCR, a low FPR, or a trade-off, a suitable operating point can be chosen. An FPR of $10^0 = 1.0$ assesses the closed set performance of an algorithm. Assuming a probability-based system, such an FPR value can always be achieved by setting the threshold θ to 0. Based on these considerations, an algorithm completely outperforms another algorithm if it achieves higher CCR values at any FPR value. Regarding the OSCR plots, tables A.11, A.12, and A.13 simplify their numerical interpretation. These tables give an overview of the achieved CCR values at various FPR values for all algorithms. In the following, I present the experimental results on all protocols.

---

[8] https://github.com/zhoudw-zdw/CVPR21-Proser/blob/f5b53b90509e6460a783a0baa72bda52364810fd/proser_unknown_detection.py#L82

Protocol 1: Considering figure 5.1 as well as table A.11, it can be seen that the Entropic Open Set Loss (EOS, Dhamija et al., 2018) outperforms all other algorithms at FPR values in the range of around 0.001 to 1.0. However, the curve levels off as it approaches the FPR value of 1.0. Similarly, the OpenMax model (Bendale and Boult, 2016) beats the Extreme Value Machine (EVM, Rudd et al., 2018), the PROSER algorithm (Zhou et al., 2021b), and the baseline model at FPR values between around 0.003 and 0.9. In fact, the baseline model shows a fractionally higher closed set performance than OpenMax. Moreover, the PROSER algorithm performs consistently worse than the baseline model at all FPR values where the latter is operational. In contrast, the EVM surpasses the baseline model in an FPR-value range of around 0.08 to 0.6. Regarding the closed set performance of the algorithms, the baseline model beats all other algorithms but Entropic Open Set Loss. With respect to the left tails of the curves, PROSER is able to perform at lower FPR values, although the associated CCR values are at a low level.

Protocol 2: Figure 5.2 and table A.12 reflect the performances on protocol 2. At FPR values in the range of around 0.006 to 0.4, OpenMax outperforms all other algorithms. However, this situation changes as the FPR value is raised above the threshold of 0.4. Furthermore, EOS surpasses the baseline model in the FPR-value range of around 0.02 to 0.4. Then, similarly to protocol 1, the curve flattens as the FPR approaches the value of 1.0. In contrast to protocol 1, EVM performs better than the baseline model over a considerable range of FPR values. Concretely, EVM tops the baseline model in the range of around 0.03 to 0.9. A similar observation can be made for PROSER. While beaten by the baseline model on protocol 1, PROSER achieves higher CCR values than the baseline model at FPR values between around 0.02 and 0.05 as well as around 0.06 to 0.4. When it comes to the closed set performance of the algorithms, the baseline model reaches the highest CCR value, followed by EVM. On the other hand, EOS demonstrates the lowest closed set performance among all algorithms. Regarding the left tails of the curves, EVM and PROSER are able to produce the lowest FPR values among all evaluated algorithms.

Protocol 3: As can be seen from figure 5.3 and table A.13, OpenMax achieves the highest CCR values among all compared algorithms at FPR values between around 0.02 and 0.7. However, it performs almost on a par with the baseline model, and a statistical significance test would be necessary to demonstrate OpenMax's superiority. Furthermore, at FPR values in the range of around 0.03 to 1.0, EOS, EVM, and PROSER perform worse than the baseline model. Approaching the FPR value of 1.0, their curves begin to level off. Assessing the closed set performance of the algorithms, the baseline model sets the benchmark. OpenMax, coming closest to this benchmark, outperforms PROSER, EVM, and EOS. Taking a look at the left tails of the curves, a familiar picture emerges. Specifically, PROSER is able to operate at lower FPR values than the other algorithms.

In sum, EOS sets the benchmark on protocol 1. However, with the increasing difficulty of the open set task, it loses its advantage over the other algorithms. In fact, on protocol 3, EOS is not competitive anymore with the baseline model. OpenMax, on the other hand, is able to keep up its performance. Specifically, it performs better than the baseline model at FPR values between around 0.02 to 0.6 over all three protocols. However, it can be observed that its advantage over the next best algorithm diminishes with increasing difficulty of the open set task. When it comes to EVM and PROSER, a mixed picture emerges. On the one hand, on protocols 1 and 3, PROSER shows lower CCR values than the baseline model at all FPR values at which the latter model is operational. However, this observation does not apply anymore to protocol 2, where PROSER outperforms the baseline model at FPR values in the range of around 0.02 and 0.05 as well as around 0.06 to 0.4. On the other hand, EVM loses out to the baseline model on protocol 3 at all FPR values that are achievable by the baseline model. On protocol 2, however, EVM yields higher

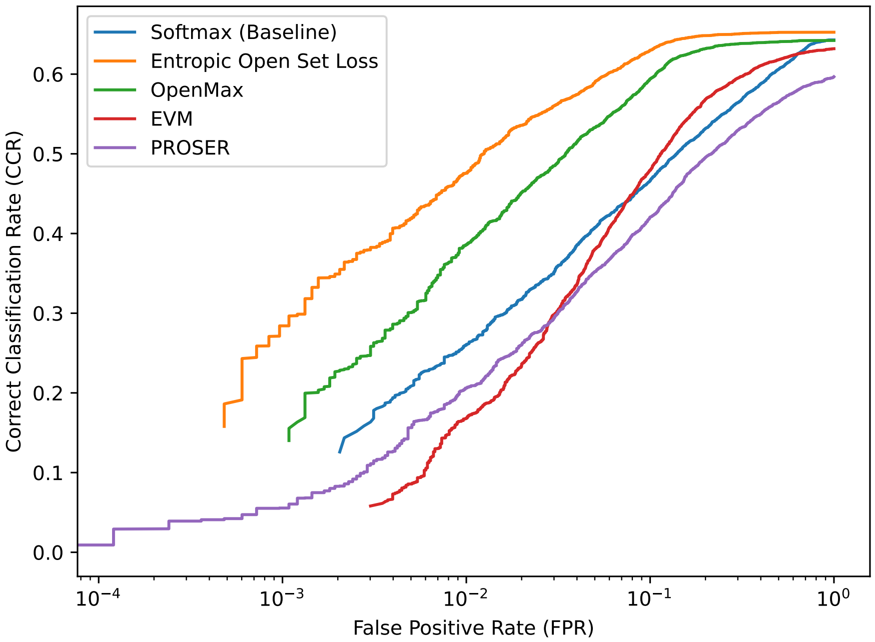


Figure 5.1: OSCR CURVES FOR PROTOCOL 1. This figure depicts the OSCR curves that result when the examined algorithms are evaluated on the test dataset of protocol 1.

CCR values for FPR values in the range of around 0.03 to 0.9. On protocol 1, EVM surpasses the baseline model within an interval consisting of FPR values between around 0.08 to 0.6. Regarding the closed set performances of the algorithms across the three protocols, the results show that only the baseline model consistently ranks top. In fact, the baseline model reaches the second highest CCR value on protocol 1, and the highest CCR value on the remaining protocols at an FPR value of 1.0. In contrast, compared to the other algorithms, PROSER is able to operate at lower FPR values across all three protocols. In the next chapter, I discuss these findings.

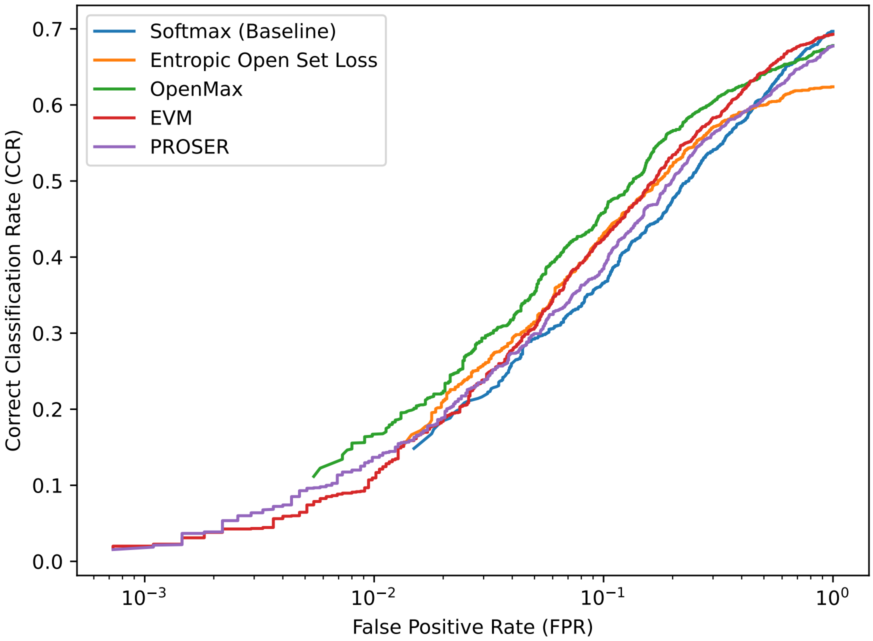


Figure 5.2: OSCR Curves for Protocol 2. This figure depicts the OSCR curves that result when the examined algorithms are evaluated on the test dataset of protocol 2.

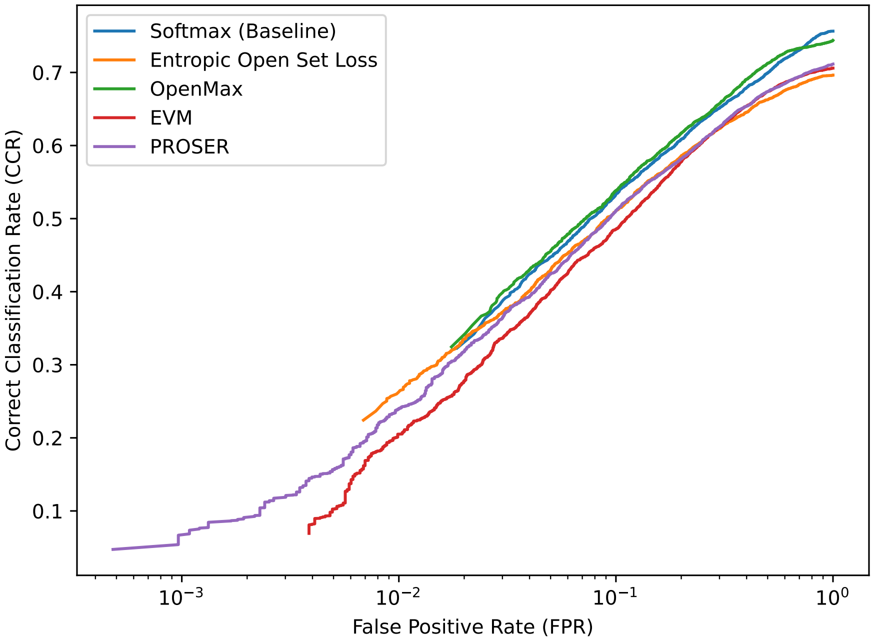


Figure 5.3: OSCR Curves for Protocol 3. This figure depicts the OSCR curves that result when the examined algorithms are evaluated on the test dataset of protocol 3.

Chapter 6

Discussion

The realization of a systematic comparison among the open set algorithms Entropic Open Set Loss (EOS, Dhamija et al., 2018), OpenMax (Bendale and Boult, 2016), EVM (Rudd et al., 2018), and PROSER (Zhou et al., 2021b) on three ImageNet protocols (Bhoumik, 2021) provides me with various results. In this chapter, I draw insights from these results, discuss their implications and point to potential limitations of my work.

6.1 Interpretation of Results

Considering the OSCR plots (Dhamija et al., 2018) depicted in section 5.2 as well as tables A.11, A.12, A.13, it can be seen that EOS loses its relative advantage over the other algorithms as the difficulty of the open set task increases. Concretely, EOS outperforms all remaining algorithms on protocol 1, which is associated with a comparably easy open set task (see section 2.2 and Bhoumik, 2021). On protocol 2, EOS still beats the baseline model over a considerable range of FPR values but clearly loses out to OpenMax. On protocol 3, which is associated with the most difficult open set task among the three protocols, EOS can not keep up with the baseline model anymore. These results indicate that the performance of EOS does not generalize well to more complex open set contexts. This observation is also supported by considering the findings by Dhamija et al. (2018). Figure 6.1 illustrates the results of their experiment in which the performance of various algorithms on an open set dataset is assessed. This dataset consists of MNIST (=$\mathcal{D}_c$) and notMNIST (= $\mathcal{D}_a$) samples. The MNIST dataset as well as the notMNIST dataset comprises 10 classes (LeCun et al., 1989; Bulatov, 2011). Hence, compared to the protocols used in the experiments, this dataset comes with limited diversity and hierarchical richness and can be considered an easier open set context. Looking at figure 6.1, it can be seen that the achieved CCR values are much higher at various FPR values in comparison to the values achieved in my experiments. Although one must be careful when comparing those values as the underlying network architecture differs, the differences hint at the difficulty that EOS experiences when performing in more challenging open set contexts. According to my results, these difficulties occur despite the fact that EOS is the only tested algorithm that takes advantage of additional knowledge in the form of samples from *KUCs*. However, for protocol 2 and protocol 3, it is possible that the selected *KUCs* are not sufficiently suited to prepare the model for the successful recognition of *UUCs*. This assumption is supported by Dhamija et al. (2018), which stress that the choice of *KUCs* is important when it comes to the open set performance of EOS.

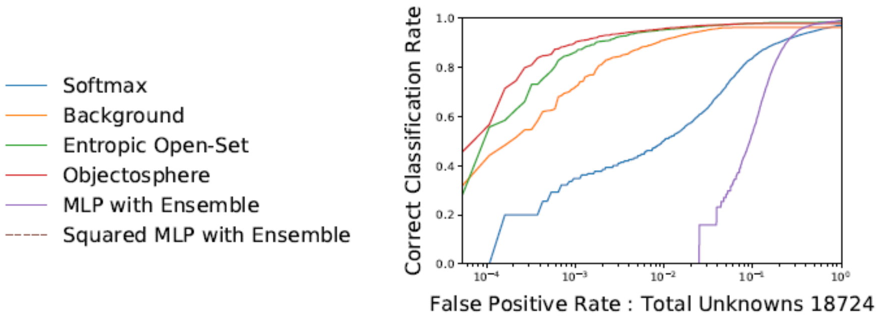


Figure 6.1: PERFORMANCE COMPARISON ON OPEN SET DATASET. This figure contrasts the OSCR curves of various algorithms, which have been evaluated on a test dataset consisting of MNIST (=$\mathcal{D}_c$) and notMNIST (= $\mathcal{D}_a$) samples. The used network is based on a LeNet++ architecture and has been trained using MNIST digits as $\mathcal{D}_c^{'}$ and NIST letters as $\mathcal{D}_b^{'}$. *Source: (Dhamija et al., 2018)*

In contrast to EOS, my results suggest that OpenMax is more robust against a changing open set dataset. While OpenMax establishes itself as the second best algorithm over a large range of FPR values, it achieves the highest CCR values over a considerable range of FPR values on protocols 2 and 3. On protocol 3, however, it only beats the baseline model by a small margin. This observation is valorized by the fact that OpenMax uses the same hyperparameters on all three protocols. As tables A.3 and A.4 indicate, the optimal hyperparameters on protocol 2 seem, to a certain degree, to be robust against a changing open set context. Avoiding the need to recalibrate the model on every new data basis increases the usability of this algorithm.

When it comes to EVM and PROSER, the results are equivocal. On the one hand, EVM can not keep up with the baseline model on protocol 1 at low FPR values where the latter model is operational. This changes on protocol 2, where EVM achieves higher CCR values than the baseline model at low FPR values. On protocol 3, EVM is not competitive against the baseline model at all FPR values that are achievable by the latter. When considering these findings, one has to keep in mind that the EVM model uses distances that are measured in deep feature space. This basically suggests that the success of this model partly depends on distinctive features. The importance of having salient features has been pointed out by Donahue et al. (2014) and Ranjan et al. (2017). According to them, features from data samples that belong to the same class should ideally be located in close proximity to each other in deep feature space, thereby building a cluster. Furthermore, this cluster should be delimitable from clusters that represent other classes. Coming back to EVM, if the baseline model used for feature extraction does not produce meaningful features, the quality of the EVM model could suffer. This in turn could be one explanation for the observed results. However, one could point to OpenMax, whose model operates on features as well. But while EVM computes the distances to the τ closest samples from negative training classes, OpenMax only considers positive training samples for the distance computation. This might lead to a lower susceptibility to an imperfect feature space. On the other hand, PROSER does not achieve the benchmark set by the baseline model on protocols 1 and 3 at all FPR values at which the baseline model is operational. However, this changes on protocol 2. A possible explanation of this behavior could lie in the hyperparameters. As pointed out in subsection 4.4.5, I use the default settings for the coefficients α, λ_1, and λ_2. I do this since neither the research paper (Zhou et al., 2021a) nor the source code provides information on the handling and choice of these coefficients. However, it is possible that there exist other coefficients that lead to a more competitive PROSER model. Furthermore, the PROSER fine-tuning procedure runs over 10 epochs. Since the source code (Zhou et al., 2021a) does not parameterize this variable,[1] I inherit this value. Hence, increasing the number of fine-tuning epochs might result in a more powerful model. In fact, my results encourage the modification of the numbers of epochs. As can be seen in tables A.8, A.9, and A.10, the optimal model on all three protocols is associated with a high fine-tuning epoch, which suggests that the network training might not be completed. Also, it is possible that the evaluation metric that is chosen by Zhou et al. (2021b) might not be the optimal one. According to the aforementioned tables, the two alternative evaluation metrics suggest selecting other models. In particular, the two alternative evaluation metrics show the lowest score for the selected model on protocol 1 (see table A.9). Importantly, these observations do not show that the *AUC* metric is inappropriate, but they encourage further investigation in the future. Another point that has to be mentioned with respect to EVM and PROSER is the transferability of hyperparameters. As described in subsections 5.1.4 and 5.1.5, I repeat the entire training procedure on protocols 1 and 3 since the transfer of the optimal hyperparameters on protocol 2 to these two protocols leads to substantially inferior models. Hence, in contrast to OpenMax, applying EVM and PROSER to new open set contexts requires finding a new set of hyperparameters via a complete training procedure. This makes the configuration of these models more expensive and might affect their applicability.

When it comes to the left side of the curves, PROSER is able to operate at lower FPR values than most of the other algorithms. In fact, only EVM provides such low FPR values on protocol 2. These values point to an interesting characteristic of the other algorithms. Considering, for example, OpenMax on protocol 3, its lowest achievable FPR value is around 0.02. Reflecting upon the OSCR definition in equation 3.5, this indicates that the FPR can not be lowered anymore by raising the threshold θ. Such a situation can occur if a certain number of samples that belong to the *UUCs* (in this case, around 2% of them) show a probability score of 1.0 for one of the C known training classes. Therefore, one can conclude that OpenMax produces more such samples than PROSER on protocol 3. This observation can be applied to the other protocols as well.

Finally, taking a look at the closed set performance, my results reflect the nature of the protocols as well as the models. Specifically, the baseline model achieves a high closed set performance across all three protocols. Since the baseline model is selected on the basis of its closed set performance, these results are of no surprise. Furthermore, considering figures A.1, A.2, A.3, A.4, and A.5, as well as tables A.11, A.12, and A.13, all algorithms achieve a higher closed set performance on protocol 3 than on protocol 2 or protocol 1. This finding mirrors the composition of these protocols. While protocols 1 and 2 are associated with a relatively difficult closed set task, protocol 3 is linked to a comparably easier closed set task (Bhoumik, 2021). Additionally, two interesting findings are associated with EOS. Firstly, EOS shows a higher closed set performance on protocol 1 than the baseline model. This result is remarkable since open set classifiers are considered to achieve a trade-off between the performance on the open set task and the performance on the closed set task (see equation 2.1 by Scheirer et al., 2013). However, in my experiments, EOS improves the performance on the closed set task as well as on the open set task when compared to a closed set classifier as the baseline model. Secondly, my results demonstrate that EOS shows the lowest closed set performance among all evaluated algorithms on protocol 2. A recent study by Palechor et al. (2023) serves as a possible explanation for this finding. Specifically, the authors apply the same three protocols in their experiments as I do. By making use of their newly introduced evaluation metric, they show that EOS focuses more on rejecting samples from *KUCs* during training than on correctly classifying samples from *KKCs*.

---

[1] `https://github.com/zhoudw-zdw/CVPR21-Proser/blob/f5b53b90509e6460a783a0baa72bda52364810fd/proser_unknown_detection.py#L190`

6.2 Limitations

While my work addresses existing issues in open set research, some limitations need to be mentioned. The first limitation relates to the preprocessing of the images, as mentioned in section 4.2. Concretely, I first resize the input samples to a uniform size of 300 × 300. Subsequently, a patch of size 224 × 224 is cropped out at the center. However, Krizhevsky et al. (2012) recommend another procedure. They first scale the shorter side of the image down to a size of 256. Then, the resulting image is cropped out at the center, using a crop size of 256 × 256. In this way, in contrast to my procedure, they respect the aspect ratio. Furthermore, after the center crop, I apply a data augmentation technique in the form of a random rotation to the training samples. However, PyTorch (Paszke et al., 2019) recommends[2] a `RandomResizedCrop`,[3] which crops a random part of the image and resizes it. This procedure aims at counteracting possible overfitting to the training data. Another possible improvement is associated with the network architecture. Concretely, my experiments are only based on the ResNet-50 architecture. While ResNet-50 is considered to provide a good trade-off between performance and the number of trainable parameters (He et al., 2016a), it is feasible that other network architectures have their own strengths when it comes to dealing with unknown samples (Roady et al., 2020). This is of special importance when considering the importance of the deep feature representations when they are used in subsequent tasks (Donahue et al., 2014; Ranjan et al., 2017). Therefore, there might be other network architectures that yield more meaningful deep features and results. When it comes to PROSER, as justified in subsections 4.4.5 and 5.1.5, I work with the provided default hyperparameters. However, as indicated in section 6.1, there might be other hyperparameter combinations that lead to a more competitive PROSER model. Finally, I perform my comparison on three instances of an open set environment. While these instances are designed in a systematic way, varying the difficulty of the open set as well as the closed set task (Bhoumik, 2021), they are by no means an exhaustive set of possible open set contexts. Hence, other protocols might further expand the knowledge about the strengths and weaknesses of open set algorithms.

---

[2] `https://github.com/pytorch/examples/blob/7ed7ac7b01add7ca29d45f25700e73a4b517ccea/imagenet/main.py#L217`

[3] `http://pytorch.org/vision/main/generated/torchvision.transforms.RandomResizedCrop.html#randomresizedcrop`

Chapter 7

Conclusion and Future Work

In this thesis, I address the issue of unsatisfactory comparability of prominent open set algorithms due to the use of heterogeneous data foundations and network characteristics, as well as suboptimal evaluation metrics. Furthermore, since such algorithms are usually evaluated on small datasets (Geng et al., 2021) that are associated with limited diversity and hierarchical richness, it is not clear how their performances develop when the difficulty of the closed set task is increased. By conducting a systematic comparison of four algorithms, Entropic Open Set Loss (EOS, Dhamija et al., 2018), OpenMax (Bendale and Boult, 2016), EVM (Rudd et al., 2018), and PROSER (Zhou et al., 2021a), on recently developed ImageNet protocols (Bhoumik, 2021), I enable a direct contrast of these methods and contribute to a better understanding of their behavior in different open set contexts. Referring to the research questions stated in section 1.3, my work shows that OpenMax outperforms EVM as well as PROSER on all three protocols over a considerable range of FPR values. Furthermore, OpenMax is defeated by EOS only on protocol 1. In fact, EOS loses its advantage over the other algorithms as the difficulty of the open set task increases. Apart from that, EVM and PROSER have difficulties keeping up with the baseline model on protocols 1 and 3. The results further point to the robustness of OpenMax when it comes to the transferability of hyperparameters to other open set environments. However, with increasing difficulty of the open set task, the difference between the OpenMax performance and the performance of the competing algorithms diminishes. In contrast, EVM and PROSER require a new hyperparameter configuration on every protocol. This restricted transferability of hyperparameters to other open set contexts might make them less appealing.

The findings from my experiments open up many possible directions for further investigations. First, since OpenMax and EVM operate on feature representations of data samples, alternative network architectures could be considered for feature extraction. In fact, trying out networks that are associated with other mappings into feature space is encouraged by findings from Donahue et al. (2014) and Ranjan et al. (2017) that stress the importance of deep feature representations when used in subsequent tasks. As a possible approach, the trained EOS network could be used for feature extraction instead of the used baseline model. Due to time constraints, I could not test this option. Similarly, the dimensionality of the feature space could be varied to inspect possible effects on the results. This is encouraged by Donahue et al. (2014), who point out the role of the feature space dimensionality. Moreover, the focus could be laid on the used training data. Specifically, EVM has the possibility to utilize samples from *KUCs* in the training procedure (Dhamija, 2021). On the one hand, EVM could consider known unknown samples $x \in \mathcal{D}_b'$ for the computation of the pairwise distances, which might in turn have an effect on the distribution fitting procedure for the known classes. On the other hand, EVM could treat all known unknown samples $x \in \mathcal{D}_b'$ as a single class and then estimate models for this class. Either way, it would be interesting to investigate the impact of this additional knowledge on the EVM performance. In a similar vein, new open set partitions that alter the level of similarity between *KKCs* and *KUCs* could be composed. This would be of interest as the success of EOS is affected by the choice of *KUCs* (Dhamija et al., 2018). When it comes to PROSER, performing a grid search over the hyperparameters could unlock potential that has not been exposed in my experiments. Also, to get a more complete picture, the comparison could be expanded to other open set algorithms such as those mentioned in subsection 3.1.1. Finally, it would be interesting to include "fooling" (Nguyen et al., 2015), "rubbish" (Goodfellow et al., 2014b), and adversarial (Goodfellow et al., 2014b; Szegedy et al., 2014) images into the protocols and then perform the comparison, as DNNs are generally vulnerable to those types of images (Goodfellow et al., 2014b; Nguyen et al., 2015; Rozsa et al., 2017).

In sum, my work gives an idea of how four relevant open set algorithms compare to each other in different open set environments. It further points to the observed strengths and limitations of those approaches. These insights can be used for further investigations or possible improvements of the algorithms. The task of Open Set Recognition is a difficult one, as my experiments show. However, expressed in a different way:

> *"For any scientist the real challenge is not to stay within the secure garden of the known but to venture out into the wilds of the unknown."* (Du Sautoy, 2017)

Appendix A

Attachments

Optimizer	**Hyperparameters**	**Training Epochs**		
		100	**200**	**300**
Adam	(0.01, 0.9, 0.999)	0.54969 (98)	0.58340 (187)	0.59660 (244)
	(0.001, 0.9, 0.999)	0.60491 (91)	0.61358 (169)	0.60415 (287)
	(0.0001, 0.9, 0.999)	0.54767 (86)	0.57233 (195)	0.57333 (262)
	(0.00001, 0.9, 0.999)	0.42541 (76)	0.42075 (197)	0.42403 (297)
	(0.001, 0.8, 0.999)	**0.63862 (96)**	0.63635 (153)	0.61157 (274)
	(0.001, 0.7, 0.999)	0.61082 (98)	**0.64000 (192)**	0.63962 (258)
	(0.001, 0.6, 0.999)	0.61157 (63)	0.62779 (100)	**0.64264 (233)**
	(0.001, 0.7, 0.9)	0.60855 (56)	0.59862 (175)	0.60491 (228)
SGD	(0.1, 0.9)	0.42403 (87)	0.42013 (199)	0.50780 (292)
	(0.01, 0.9)	**0.59270 (68)**	**0.61472 (162)**	**0.63711 (230)**
	(0.001, 0.9)	0.50000 (99)	0.52579 (195)	0.52679 (263)
	(0.0001, 0.9)	0.30679 (98)	0.42465 (192)	0.46252 (269)
	(0.01, 0.6)	0.56704 (98)	0.57019 (167)	0.57484 (289)

Table A.1: Validation of Softmax Model on Protocol 2. The table lists the hyperparameter combinations used to train the Softmax model on protocol 2 as well as the corresponding performance measures on the validation dataset. The column *Training Epochs* reflects the maximum accuracy and, in parentheses, the associated training epoch where this performance has been achieved. Moreover, the column *Hyperparameters* contains the arguments that are passed to the optimizer. In case of Adam, these arguments refer to the learning rate and the two beta coefficients. On the other hand, the Stochastic Gradient Descent (SGD) optimizer requires the learning rate as well as the momentum term. For both optimizers, the maximum accuracy value for a specific number of training epochs is marked in red. Furthermore, the hyperparameter combination that is associated with the selected model is marked in italics.

Table A.2 illustrates the outcome of the OpenMax model validation on protocol 2 for different hyperparameter combinations. Concretely, the training procedure fits a model for every hyperparameter combination consisting of the tail size (TS), the distance multiplier (DM) as well as the number of revisable classes (Alpha). Every model is then evaluated on the validation set, resulting in the computation of the CCR value for a selection of FPRs (0.001, 0.01, 0.1, 0.15, and 0.2). Hence, the cells list the CCR values for the FPR indicated by the respective column. The maximum CCR value for a specific FPR value over all models is marked in red. Furthermore, the hyperparameter combination that is associated with the selected model is marked in italics. Cells that contain a " - " refer to the situation where the FPR value is not achievable by the associated model. For example, the model associated with the hyperparameter combination *(10_1.5_2)* generates a minimum FPR value that is higher than 0.01. Therefore, there is no CCR value available.

Hyperparameters (TS_DM_Alpha)	**CCR@FPR**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(10_1.5_2)	-	-	0.27862	0.34226	0.39019
(10_1.5_3)	-	-	0.27799	0.34138	0.38918
(10_1.5_5)	-	-	0.27610	0.34025	0.38503
(10_1.5_10)	-	-	0.28352	0.34201	0.38843
(10_1.7_2)	-	-	0.27862	0.34164	0.39019
(10_1.7_3)	-	-	0.27862	0.34176	0.38943
(10_1.7_5)	-	-	0.27748	0.34138	0.38893
(10_1.7_10)	-	-	0.27786	0.34063	0.38843
(10_2.0_2)	-	-	0.27862	0.34164	0.39019
(10_2.0_3)	-	-	0.27874	0.34176	0.38969
(10_2.0_5)	-	-	0.27887	0.34189	0.38956
(10_2.0_10)	-	-	0.27912	0.34189	0.38969
(10_2.3_2)	-	-	0.27862	0.34164	0.39019
(10_2.3_3)	-	-	0.27862	0.34164	0.39019
(10_2.3_5)	-	-	0.27874	0.34164	0.38994
(10_2.3_10)	-	-	0.27875	0.34164	0.38969
(100_1.5_2)	-	-	0.28352	0.34767	0.39811
(100_1.5_3)	-	-	0.28352	0.34830	0.39170
(100_1.5_5)	-	-	0.28365	0.34579	0.39484
(100_1.5_10)	-	0.09270	0.30742	0.36516	0.40503

Hyperparameters (Cont.) (TS_DM_Alpha)	**CCR@FPR (Cont.)**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(100_1.7_2)	-	-	0.28164	0.34478	0.39421
(100_1.7_3)	-	-	0.28264	0.34377	0.39195
(100_1.7_5)	-	-	0.28189	0.34113	0.38579
(100_1.7_10)	-	0.09748	0.30591	0.36642	0.40969
(100_2.0_2)	-	-	0.28000	0.34440	0.39308
(100_2.0_3)	-	-	0.28063	0.34365	0.39019
(100_2.0_5)	-	-	0.27925	0.33962	0.38679
(100_2.0_10)	-	-	0.28541	0.34327	0.39119
(100_2.3_2)	-	-	0.27912	0.34327	0.39258
(100_2.3_3)	-	-	0.27975	0.34377	0.39145
(100_2.3_5)	-	-	0.27862	0.34151	0.38956
(100_2.3_10)	-	-	0.27761	0.34025	0.38654
(250_1.5_2)	-	-	0.29195	0.35409	0.40189
(250_1.5_3)	-	-	0.29585	0.35862	0.40516
(250_1.5_5)	-	-	0.30277	0.37170	0.41686
(250_1.5_10)	-	0.07484	0.29044	0.36151	0.39711
(250_1.7_2)	-	-	0.28692	0.35019	0.39899
(250_1.7_3)	-	-	0.29107	0.35346	0.39811
(250_1.7_5)	-	-	0.29396	0.35472	0.40491
(250_1.7_10)	-	0.08956	0.31006	0.36628	0.41208
(250_2.0_2)	-	-	0.28340	0.34692	0.39723
(250_2.0_3)	-	-	0.28616	0.34767	0.39358
(250_2.0_5)	-	-	0.28692	0.34730	0.39057
(250_2.0_10)	-	0.09698	0.31585	0.37308	0.41723
(250_2.3_2)	-	-	0.28176	0.34516	0.39358
(250_2.3_3)	-	-	0.28428	0.34415	0.39308
(250_2.3_5)	-	-	0.28264	0.34352	0.38855
(250_2.3_10)	-	-	0.29484	0.35736	0.40038

Hyperparameters (Cont.) (TS_DM_Alpha)	**CCR@FPR (Cont.)**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(500_1.5_2)	-	-	0.29874	0.35811	0.40075
(500_1.5_3)	-	-	0.30830	0.36767	0.41069
(500_1.5_5)	-	-	0.32075	**0.38579**	0.42616
(500_1.5_10)	0.01472	0.05786	0.28050	0.34038	0.38692
(500_1.7_2)	-	-	0.29673	0.35585	0.40201
(500_1.7_3)	-	-	0.30164	0.36239	0.40818
(500_1.7_5)	-	-	0.31283	0.37472	0.42088
(500_1.7_10)	0.01786	0.06994	0.29371	0.35610	0.39937
(500_2.0_2)	-	-	0.29132	0.35031	0.39887
(500_2.0_3)	-	-	0.29585	0.35761	0.40201
(500_2.0_5)	-	-	0.30013	0.36075	0.41082
(500_2.0_10)	-	0.08654	0.31296	0.37195	0.41547
(500_2.3_2)	-	-	0.28767	0.34893	0.39585
(500_2.3_3)	-	-	0.29132	0.35396	0.39761
(500_2.3_5)	-	-	0.29434	0.35535	0.39975
(500_2.3_10)	-	**0.10226**	0.31799	0.37623	0.42214
(750_1.5_2)	-	-	0.29774	0.35698	0.38906
(750_1.5_3)	-	-	0.30981	0.37283	0.41711
(750_1.5_5)	-	0.09069	**0.32239**	0.38201	**0.42780**
(750_1.5_10)	0.01321	0.05333	0.26138	0.31698	0.35358
(750_1.7_2)	-	-	0.29774	0.35585	0.39660
(750_1.7_3)	-	-	0.30755	0.36642	0.41233
(750_1.7_5)	-	0.09774	0.31786	0.38440	0.42667
(750_1.7_10)	0.01270	0.05484	0.27522	0.33170	0.37685
(750_2.0_2)	-	-	0.29585	0.35421	0.39912
(750_2.0_3)	-	-	0.30541	0.36277	0.40654
(750_2.0_5)	-	-	0.31069	0.37509	0.42063
(750_2.0_10)	**0.01799**	0.07069	0.29560	0.35371	0.39786

Hyperparameters (Cont.) (TS_DM_Alpha)	**CCR@FPR (Cont.)**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(750_2.3_2)	-	-	0.29371	0.35069	0.39824
(750_2.3_3)	-	-	0.30000	0.35799	0.40302
(750_2.3_5)	-	-	0.30667	0.36377	0.41094
(750_2.3_10)	-	0.08906	0.31220	0.37094	0.41610
(1000_1.5_2)	-	-	0.29233	0.34126	0.37346
(1000_1.5_3)	-	-	0.31547	0.37962	0.41660
(1000_1.5_5)	-	0.08541	0.31434	0.37472	0.41585
(1000_1.5_10)	0.01421	0.05484	0.23774	0.28176	0.31899
(1000_1.7_2)	-	-	0.29597	0.35157	0.38566
(1000_1.7_3)	-	-	0.30830	0.37258	0.41484
(1000_1.7_5)	-	0.08830	0.31950	0.38101	0.42365
(1000_1.7_10)	0.01170	0.05535	0.25648	0.30528	0.33987
(1000_2.0_2)	-	-	0.29597	0.35346	0.39308
(1000_2.0_3)	-	-	0.30767	0.36742	0.41119
(1000_2.0_5)	-	0.10000	0.31836	0.38566	0.42440
(1000_2.0_10)	0.01383	0.06516	0.273834	0.32906	0.36780
(1000_2.3_2)	-	-	0.29484	0.35094	0.39686
(1000_2.3_3)	-	-	0.30516	0.36226	0.40503
(1000_2.3_5)	-	-	0.31522	0.37736	0.42038
(1000_2.3_10)	**0.01799**	0.07082	0.29509	0.35107	0.39208

Table A.2: Validation of OpenMax Model on Protocol 2

Hyperparameters (TS_DM_Alpha)	**CCR@FPR**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(1000_1.5_2)	-	0.18964	0.34293	0.37108	0.39176
(1000_1.5_3)	-	0.20364	0.39245	0.41681	0.43811
(1000_1.5_5)	0.10729	0.22649	**0.44789**	0.47180	0.48605
(1000_1.5_10)	0.086851	0.20161	0.35770	0.38856	0.41062
(1000_1.7_2)	-	0.18306	0.36320	0.39352	0.41471
(1000_1.7_3)	-	0.19249	0.39703	0.42820	0.45057
(1000_1.7_5)	-	0.20760	0.42917	**0.47775**	**0.50308**
(1000_1.7_10)	**0.11242**	0.22542	0.40456	0.43691	0.45866
(1000_2.0_2)	-	0.17907	0.37469	0.40756	0.43068
(1000_2.0_3)	-	0.18365	0.38949	0.42645	0.45353
(1000_2.0_5)	-	0.18988	0.39252	0.43746	0.46846
(1000_2.0_10)	0.10306	**0.23323**	0.43312	0.46516	0.48715
(1000_2.3_2)	-	0.17484	0.37081	0.40959	0.43601
(1000_2.3_3)	-	0.18062	0.37824	0.41785	0.44544
(1000_2.3_5)	-	0.18241	0.37886	0.41743	0.44809
(1000_2.3_10)	-	0.21014	0.42225	0.46753	0.49812

Table A.3: Validation of OpenMax Model on Protocol 1. The table illustrates the outcome of the OpenMax model validation on protocol 1 for different hyperparameter combinations. For a description of the meaning of the columns, see table A.2

.

Hyperparameters (TS_DM_Alpha)	**CCR@FPR**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(1000_1.5_2)	-	-	0.38769	0.44481	0.47845
(1000_1.5_3)	-	-	0.38904	0.45434	0.49866
(1000_1.5_5)	-	-	0.35732	0.42205	0.46367
(1000_1.5_10)	**0.00537**	0.03179	0.18876	0.25174	0.30553
(1000_1.7_2)	-	-	0.39010	0.45264	0.49498
(1000_1.7_3)	-	-	0.38932	**0.45541**	**0.50294**
(1000_1.7_5)	-	-	0.37340	0.43832	0.48375
(1000_1.7_10)	-	**0.03871**	0.20157	0.26614	0.31874
(1000_2.0_2)	-	-	0.39031	0.45335	0.50048
(1000_2.0_3)	-	-	0.38825	0.45490	0.50279
(1000_2.0_5)	-	-	0.38278	0.44826	0.49673
(1000_2.0_10)	-	-	0.27468	0.33671	0.37755
(1000_2.3_2)	-	-	**0.39056**	0.45269	0.49975
(1000_2.3_3)	-	-	0.38916	0.45310	0.50127
(1000_2.3_5)	-	-	0.38491	0.44996	0.49817
(1000_2.3_10)	-	-	0.34049	0.40564	0.45074

Table A.4: Validation of OpenMax Model on Protocol 3. The table illustrates the outcome of the OpenMax model validation on protocol 3 for different hyperparameter combinations. For a description of the meaning of the columns, see table A.2.

Table A.5 illustrates the outcome of the EVM model validation on protocol 2 for different hyperparameter combinations. Concretely, the training procedure fits a model for every hyperparameter combination consisting of the tail size (TS), the distance multiplier (DM) as well as the cover threshold (CT). Every model is then evaluated on the validation set, resulting in the computation of the CCR value for a selection of FPRs (0.001, 0.01, 0.1, 0.15, and 0.2). Hence, the cells list the CCR values for the FPR indicated by the respective column. The maximum CCR value for a specific FPR value over all models is marked in red. Furthermore, the hyperparameter combination that is associated with the selected model is marked in italics. Cells that contain a " - " refer to the situation that the FPR value is not achievable by the associated model. For example, the models associated with the hyperparameter combinations *(10_0.70_1.00)*, *(10_0.90_1.00)*, and *(10_1.00_1.00)* all generate a minimum FPR value that is higher than 0.2. Therefore, there is no CCR value available.

Hyperparameters (TS_DM_CT)	**CCR@FPR**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(10_0.10_1.00)	0.00453	**0.05912**	0.28805	0.34264	0.39622
(10_0.20_1.00)	-	0.05220	0.30226	0.36893	0.41308
(10_0.30_1.00)	-	-	0.31082	0.37660	0.41786
(10_0.40_1.00)	-	-	0.31157	0.37786	0.42025
(10_0.50_1.00)	-	-	0.29132	0.36516	0.41673
(10_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-
(25_0.10_1.00)	0.00453	0.05421	0.29824	0.36340	0.40679
(25_0.20_1.00)	-	0.05182	0.31258	0.37560	0.41862
(25_0.30_1.00)	-	-	0.31371	0.37849	0.42189
(25_0.40_1.00)	-	-	0.30805	0.37396	0.42126
(25_0.50_1.00)	-	-	0.29459	0.36591	0.41635
(25_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-
(50_0.10_1.00)	0.00453	0.05811	0.30642	0.36717	0.41082
(50_0.20_1.00)	-	0.05245	0.32000	0.37635	0.42403
(50_0.30_1.00)	-	-	0.31472	0.37773	0.42390
(50_0.40_1.00)	-	-	0.30981	0.37296	0.41937
(50_0.50_1.00)	-	-	0.29623	0.36642	0.41572
(50_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-

Hyperparameters (Cont.) (TS_DM_CT)	**CCR@FPR (Cont.)**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(75_0.10_1.00)	0.00440	0.05572	0.31195	0.36969	0.41522
(75_0.20_1.00)	-	0.05270	**0.32025**	0.37899	0.42289
(75_0.30_1.00)	-	-	0.31459	0.37610	0.42264
(75_0.40_1.00)	-	-	0.30969	0.37346	0.41911
(75_0.50_1.00)	-	-	0.29811	0.36742	0.41522
(75_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-
(100_0.10_1.00)	0.00440	0.05409	0.31208	0.37006	0.41748
(100_0.20_1.00)	-	0.05258	0.31987	**0.38012**	0.42088
(100_0.30_1.00)	-	-	0.31421	0.37711	0.42327
(100_0.40_1.00)	-	-	0.30955	0.37434	0.41937
(100_0.50_1.00)	-	-	0.29886	0.36730	0.41522
(100_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-
(150_0.10_1.00)	0.00428	0.05535	0.31044	0.37057	0.41748
(150_0.20_1.00)	-	0.05233	0.31748	0.37799	0.42201
(150_0.30_1.00)	-	-	0.31371	0.37597	0.42252
(150_0.40_1.00)	-	-	0.30994	0.37472	0.41899
(150_0.50_1.00)	-	-	0.29950	0.36792	0.41572
(150_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-
(200_0.10_1.00)	0.00428	0.05560	0.31031	0.37107	0.41774
(200_0.20_1.00)	-	0.05182	0.31849	0.37710	0.42126
(200_0.30_1.00)	-	-	0.31270	0.37660	0.42239
(200_0.40_1.00)	-	-	0.30893	0.37472	0.41849
(200_0.50_1.00)	-	-	0.29962	0.36730	0.41572
(200_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-
(300_0.10_1.00)	0.00415	0.05623	0.30943	0.37044	0.41711
(300_0.20_1.00)	-	0.05132	0.31937	0.37887	0.42151
(300_0.30_1.00)	-	-	0.31308	0.37761	0.42238
(300_0.40_1.00)	-	-	0.30842	0.37434	0.41811
(300_0.50_1.00)	-	-	0.29597	0.36692	0.41497
(300_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-

Hyperparameters (Cont.) (TS_DM_CT)	**CCR@FPR (Cont.)**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(500_0.10_1.00)	0.00428	0.05308	0.30730	0.37019	0.41673
(500_0.20_1.00)	-	0.05170	0.31572	0.37937	0.42252
(500_0.30_1.00)	-	-	0.31409	0.37748	**0.42440**
(500_0.40_1.00)	-	-	0.30818	0.37409	0.41874
(500_0.50_1.00)	-	-	-	0.36088	0.41447
(500_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-
(1000_0.10_1.00)	**0.00553**	0.05447	0.30780	0.36843	0.41371
(1000_0.20_1.00)	-	0.05320	0.31472	0.37849	0.42352
(1000_0.30_1.00)	-	-	0.31522	0.37786	0.42377
(1000_0.40_1.00)	-	-	0.30264	0.37082	0.41811
(1000_0.50_1.00)	-	-	-	0.35886	0.40893
(1000_{0.70, 0.90, 1.00}_1.00)	-	-	-	-	-

Table A.5: Validation of EVM Model on Protocol 2

Table A.6 illustrates the outcome of the EVM model validation on protocol 1 for different hyperparameter combinations. For a description of the meaning of the columns, see table A.5.

Hyperparameters (TS_DM_CT)	**CCR@FPR**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(10_0.50_1.00)	**0.03414**	0.12119	0.28843	0.31286	0.33151
(10_0.60_1.00)	-	0.14074	0.34410	0.36843	0.38478
(10_0.70_1.00)	-	0.15801	0.38753	0.41860	0.43622
(10_0.80_1.00)	-	-	0.39702	0.43777	0.46365
(10_0.90_1.00)	-	-	0.37948	0.42449	0.45287
(10_1.00_1.00)	-	-	-	-	-
(50_0.50_1.00)	0.03180	0.12629	0.37032	0.40264	0.42242
(50_0.60_1.00)	-	0.14397	0.39273	0.43168	0.45625
(50_0.70_1.00)	-	**0.16021**	0.40030	0.44124	0.46839
(50_0.80_1.00)	-	-	0.39830	0.44011	0.46695
(50_0.90_1.00)	-	-	0.38092	0.42583	0.45398
(50_1.00_1.00)	-	-	-	-	-
(75_0.50_1.00)	0.03197	0.12790	0.38251	0.41898	0.44145
(75_0.60_1.00)	-	0.14370	0.39630	0.43921	0.46457
(75_0.70_1.00)	-	0.15736	0.40129	0.44252	0.46974
(75_0.80_1.00)	-	-	0.39868	0.44093	0.46767
(75_0.90_1.00)	-	-	0.37848	0.42538	0.45305
(75_0.1.00_1.00)	-	-	-	-	-
(100_0.50_1.00)	0.03228	0.12787	0.38849	0.42769	0.45133
(100_0.60_1.00)	-	0.14270	0.39782	0.44272	0.46657
(100_0.70_1.00)	-	0.15502	0.40219	0.44351	0.47056
(100_0.80_1.00)	-	-	0.39830	0.44107	0.46771
(100_0.90_1.00)	-	-	-	0.42387	0.45284
(100_1.00_1.00)	-	-	-	-	-

Hyperparameters (Cont.) (TS_DM_CT)	**CCR@FPR (Cont.)**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(150_0.50_1.00)	0.03152	0.12993	0.39500	0.43749	0.46351
(150_0.60_1.00)	-	0.14177	0.40064	0.44599	0.47063
(150_0.70_1.00)	-	0.15178	0.40236	0.44434	0.47287
(150_0.80_1.00)	-	-	0.39830	0.44096	0.46781
(150_0.90_1.00)	-	-	-	0.42077	0.45047
(150_1.00_1.00)	-	-	-	-	-
(300_0.50_1.00)	-	0.13200	0.40133	0.44500	0.47311
(300_0.60_1.00)	-	0.13922	0.40363	0.44771	0.47476
(300_0.70_1.00)	-	-	0.40356	0.44682	0.47445
(300_0.80_1.00)	-	-	0.39655	0.44004	0.4678
(300_0.90_1.00)	-	-	-	0.41458	0.44444
(300_1.00_1.00)	-	-	-	-	-
(500_0.50_1.00)	-	0.13224	0.40467	0.44902	0.47528
(500_0.60_1.00)	-	0.13699	0.40563	0.44881	0.47744
(500_0.70_1.00)	-	-	0.40370	0.44761	0.47545
(500_0.80_1.00)	-	-	0.39255	0.43794	0.46640
(500_0.90_1.00)	-	-	-	-	0.43763
(500_1.00_1.00)	-	-	-	-	-
(1000_0.50_1.00)	-	0.13186	**0.40914**	**0.45367**	**0.48047**
(1000_0.60_1.00)	-	0.12887	0.40852	0.45143	0.47985
(1000_0.70_1.00)	-	-	0.40284	0.44737	0.47621
(1000_0.80_1.00)	-	-	0.38750	0.43453	0.46323
(1000_0.90_1.00)	-	-	-	-	-
(1000_1.00_1.00)	-	-	-	-	-

Table A.6: Validation of EVM Model on Protocol 1

Table A.7 illustrates the outcome of the EVM model validation on protocol 3 for different hyperparameter combinations. For a description of the meaning of the columns, see table A.5.

Hyperparameters (TS_DM_CT)	**CCR@FPR**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(10_0.40_1.00)	-	0.10599	0.29960	0.33463	0.35968
(10_0.50_1.00)	-	-	0.34465	0.38404	0.41206
(10_0.60_1.00)	-	-	0.37081	0.42400	0.45903
(10_0.70_1.00)	-	-	0.37489	0.43733	0.48010
(10_0.80_1.00)	-	-	-	0.43297	0.47967
(10_{0.90, 1.00}_1.00)	-	-	-	-	-
(50_0.40_1.00)	-	0.10696	0.34647	0.40136	0.43621
(50_0.50_1.00)	-	-	0.36219	0.41981	0.46017
(50_0.60_1.00)	-	-	0.37076	0.43003	0.47282
(50_0.70_1.00)	-	-	0.37540	0.43690	0.48002
(50_0.80_1.00)	-	-	-	0.43698	0.48025
(50_{0.90, 1.00}_1.00)	-	-	-	-	-
(75_0.40_1.00)	-	**0.10739**	0.34952	0.40668	0.44588
(75_0.50_1.00)	-	-	0.36244	0.42014	0.46175
(75_0.60_1.00)	-	-	0.37111	0.42983	0.47320
(75_0.70_1.00)	-	-	0.37494	0.43672	0.47980
(75_0.80_1.00)	-	-	-	0.43794	0.48058
(75_{0.90, 1.00}_1.00)	-	-	-	-	-
(100_0.40_1.00)	-	0.10713	0.35005	0.40876	0.44940
(100_0.50_1.00)	-	-	0.36297	0.42184	0.46256
(100_0.60_1.00)	-	-	0.37129	0.42975	0.47343
(100_0.70_1.00)	-	-	0.37532	0.43721	0.47964
(100_0.80_1.00)	-	-	-	**0.43964**	**0.48073**
(100_{0.90, 1.00}_1.00)	-	-	-	-	-

Hyperparameters (Cont.) (TS_DM_CT)	**CCR@FPR (Cont.)**				
	0.001	**0.01**	**0.1**	**0.15**	**0.2**
(150_0.40_1.00)	-	0.106855	0.34997	0.40970	0.45226
(150_0.50_1.00)	-	-	0.36260	0.42202	0.46261
(150_0.60_1.00)	-	-	0.37106	0.43016	0.47308
(150_0.70_1.00)	-	-	0.37631	0.43665	0.47974
(150_0.80_1.00)	-	-	-	-	0.48111
(150_{0.90, 1.00}_1.00)	-	-	-	-	-
(300_0.40_1.00)	-	0.10658	0.34896	0.40932	0.45389
(300_0.50_1.00)	-	-	0.36224	0.42118	0.46423
(300_0.60_1.00)	-	-	0.37122	0.43028	0.47338
(300_0.70_1.00)	-	-	0.37710	0.43644	0.47990
(300_0.80_1.00)	-	-	-	-	0.48063
(300_{0.90, 1.00}_1.00)	-	-	-	-	-
(500_0.40_1.00)	-	0.10625	0.34865	0.40828	0.45485
(500_0.50_1.00)	-	-	0.36156	0.42103	0.46433
(500_0.60_1.00)	-	-	0.37142	0.43039	0.47348
(500_0.70_1.00)	-	-	**0.37763**	0.43586	0.47967
(500_{0.80, 0.90, 1.00}_1.00)	-	-	-	-	-
(1000_0.40_1.00)	-	0.10521	0.34810	0.40638	0.45353
(1000_0.50_1.00)	-	-	0.36141	0.42078	0.46405
(1000_0.60_1.00)	-	-	0.37243	0.43026	0.47341
(1000_0.70_1.00)	-	-	-	0.43655	0.47919
(1000_{0.80, 0.90, 1.00}_1.00)	-	-	-	-	-

Table A.7: Validation of EVM Model on Protocol 3

Number of Dummy Clfs.	No Bias Computation			Bias Computation		
	AUC	**Avg. Conf. (VAST)**	**Avg. Conf. (Dummy)**	**AUC**	**Avg. Conf. (VAST)**	**Avg. Conf. (Dummy)**
1	0.6994 (8)	**0.5761 (8)**	**0.5519 (8)**	0.6953 (10)	**0.5427 (10)**	**0.4543 (10)**
2	0.6989 (8)	0.5728 (8)	0.5477 (8)	0.6956 (10)	0.5418 (10)	0.4475 (10)
3	0.6974 (7)	0.5694 (9)	0.5378 (10)	**0.6970 (10)**	0.5425 (10)	0.4517 (10)
4	0.6931 (7)	0.5682 (9)	0.5364 (10)	0.6935 (10)	0.5408 (10)	0.4457 (10)
5	0.7010 (10)	0.5717 (9)	0.5433 (10)	0.6945 (10)	0.5411 (10)	0.4466 (10)
7	**0.7013 (7)**	0.5739 (6)	0.5478 (10)	0.6950 (10)	0.5409 (10)	0.4470 (10)
9	0.6968 (7)	0.5700 (9)	0.5399 (10)	0.6913 (10)	0.5379 (10)	0.4380 (10)
10	0.6948 (7)	0.5614 (6)	0.5257 (10)	0.6940 (10)	0.5413 (10)	0.4472 (10)
15	0.6993 (7)	0.5720 (9)	0.5448 (10)	0.6952 (10)	0.5417 (10)	0.4486 (10)
20	0.6977 (7)	0.5694 (9)	0.5376 (10)	0.6925 (10)	0.5378 (10)	0.4393 (10)
30	0.6932 (7)	0.5678 (9)	0.5351 (10)	0.6946 (10)	0.5400 (10)	0.4452 (10)

Table A.8: VALIDATION OF PROSER MODEL ON PROTOCOL 2. The table lists the number of dummy classifiers used to train the PROSER model on protocol 2, as well as the corresponding performance measures on the validation dataset. For the purpose of comparison, the model is trained in two different ways: with and without computing the bias term. However, the selection of the best training model is only based on models that include bias computation as it is introduced in the paper by Zhou et al. (2021b). The validation procedure computes three different metrics: the *AUC*, as suggested by the authors, as well as two different variations of the average confidence (Schnyder, 2021, see equation 5.1). Concretely, the *Avg. Conf. (VAST)* refers to a function in the `VAST` package (Dhamija, 2021) that covers the confidence computation for unknown samples in the case of having a probability distribution over $C + 1$ classes. While the computation for known samples remains identical to equation 5.1, the confidence for unknown samples is given as $1 - \max_{c:c\in\{1,\ldots,C\}} y_c$. In other words, the confidence is computed by subtracting the highest probability among all known classes C from 1. On the other hand, the version *Avg. Conf. (Dummy)* is applied by Elsaadany (2022) and simply considers the probability score of the unknown class, y_{C+1}, as the confidence of an unknown sample. Again, the confidence definition for known samples is identical to the one in the equation 5.1. Moreover, the computation of the average follows the same steps as described in subsection 5.1.2. While these two confidence measures only serve as alternatives, the selection of the best training model is based on the *AUC* score as done by Zhou et al. (2021b). In addition to the maximum performance score, a cell contains the associated epoch (in parentheses). For every evaluation metric, the highest score among all validated models is marked in red. Furthermore, the number of dummy classifiers that are associated with the selected model is marked in italics.

Number of Dummy Clfs.	**Bias Computation**		
	AUC	**Avg. Conf. (VAST)**	**Avg. Conf. (Dummy)**
1	0.7525 (8)	0.5632 (10)	0.3935 (8)
3	0.7579 (8)	0.5633 (10)	**0.4051 (10)**
5	**0.7618** (8)	0.5576 (8)	0.3885 (10)
10	0.7573 (8)	0.5636 (10)	0.4017 (8)
20	0.7588 (8)	**0.5657 (10)**	0.4001 (8)
30	0.7534 (8)	0.5632 (10)	0.3887 (8)
50	0.7557 (8)	0.5653(10)	0.3969 (8)
75	0.7558 (8)	0.5634 (10)	0.3985 (8)
100	0.7569 (8)	0.5642 (10)	0.4046 (8)

Table A.9: VALIDATION OF PROSER MODEL ON PROTOCOL 1. The table lists the number of dummy classifiers used to train the PROSER model on protocol 1 as well as the corresponding performance measures on the validation dataset. The validation procedure computes three different metrics: the *AUC* as suggested by the authors, as well as two different variations of the average confidence (Schnyder, 2021, see equation 5.1). For a description of these confidence metrics, see table A.8. The selection of the best training model is based on the *AUC* score as done by Zhou et al. (2021b). In addition to the maximum performance score, a cell contains the associated epoch (in parentheses). For every evaluation metric, the highest score among all validated models is marked in red. Furthermore, the number of dummy classifiers that is associated with the selected model is marked in italics.

Number of Dummy Clfs.	**Bias Computation**		
	AUC	**Avg. Conf. (VAST)**	**Avg. Conf. (Dummy)**
1	0.7040 (9)	**0.5694 (10)**	0.4294 (9)
3	0.6994 (5)	0.5663 (6)	0.4116 (8)
5	0.7042 (9)	0.5686 (10)	0.4371 (9)
10	0.7048 (9)	0.5691 (10)	0.4331 (9)
20	**0.7049** (9)	0.5690 (10)	**0.4446 (9)**
30	0.7022 (8)	0.5674 (10)	0.4225 (8)
50	0.7000 (9)	0.5626 (4)	0.4147 (9)
75	0.7027 (8)	0.5673 (7)	0.4235 (8)
100	0.7026 (9)	0.5675 (10)	0.4356 (9)

Table A.10: Validation of PROSER Model on Protocol 3. The table lists the number of dummy classifiers used to train the PROSER model on protocol 3 as well as the corresponding performance measures on the validation dataset. For a description of the meaning of the columns and metrics, see table A.8.

FPR	Softmax	EOS	OpenMax	EVM	PROSER
0.0010	-	**0.2710**	-	-	0.0552
0.0030	0.1634	**0.3795**	0.2469	-	0.1091
0.0060	0.2276	**0.4347**	0.3160	0.1029	0.1660
0.0090	0.2514	**0.4681**	0.3731	0.1631	0.1971
0.0100	0.2598	**0.4757**	0.3855	0.1678	0.2060
0.0200	0.3172	**0.5357**	0.4512	0.2336	0.2600
0.0300	0.3514	**0.5602**	0.4848	0.2981	0.2936
0.0400	0.3850	**0.5745**	0.5145	0.3359	0.3272
0.0500	0.4071	**0.5890**	0.5329	0.3800	0.3519
0.0600	0.4226	**0.6017**	0.5474	0.4071	0.3672
0.0700	0.4359	**0.6093**	0.5590	0.4288	0.3814
0.0800	0.4459	**0.6178**	0.5724	0.4491	0.3978
0.0900	0.4572	**0.6243**	0.5822	0.4678	0.4071
0.1000	0.4660	**0.6300**	0.5940	0.4795	0.4205
0.2000	0.5317	**0.6484**	0.6319	0.5683	0.4953
0.3000	0.5641	**0.6503**	0.6379	0.5952	0.5297
0.4000	0.5878	**0.6519**	0.6397	0.6109	0.5498
0.5000	0.6060	**0.6524**	0.6407	0.6179	0.5648
0.6000	0.6188	**0.6526**	0.6414	0.6236	0.5764
0.7000	0.6321	**0.6526**	0.6421	0.6272	0.5847
0.8000	0.6398	**0.6526**	0.6421	0.6295	0.5897
0.9000	0.6421	**0.6526**	0.6422	0.6305	0.5934
1.0000	0.6429	**0.6526**	0.6422	0.6319	0.5967

Table A.11: TEST PERFORMANCE ON PROTOCOL 1. This table illustrates the performances of the evaluated algorithms on the test dataset of protocol 1 at selected FPR values. The columns *Softmax*, *EOS*, *OpenMax*, *EVM*, and *PROSER* contain the performance measures in the form of CCR values. In contrast, the column *FPR* contains a selection of FPR thresholds at which the performance can be compared. Cells that comprise a " - " refer to the fact that the algorithm could not achieve the associated FPR value at any rejection threshold θ. The maximum CCR value at every FPR threshold is marked in red.

FPR	Softmax	EOS	OpenMax	EVM	PROSER
0.0010	-	-	-	**0.0200**	0.0187
0.0030	-	-	-	0.0426	**0.0600**
0.0060	-	-	**0.1226**	0.0826	0.0981
0.0090	-	-	**0.1561**	0.0923	0.1252
0.0100	-	-	**0.1645**	0.1097	0.1368
0.0200	0.1839	0.2097	**0.2213**	0.1839	0.1884
0.0300	0.2181	0.2587	**0.2935**	0.2387	0.2355
0.0400	0.2606	0.2890	**0.3174**	0.2774	0.2735
0.0500	0.2923	0.3148	**0.3529**	0.3071	0.2994
0.0600	0.3058	0.3445	**0.3935**	0.3413	0.3245
0.0700	0.3245	0.3742	**0.4155**	0.3735	0.3400
0.0800	0.3355	0.3923	**0.4271**	0.3923	0.3626
0.0900	0.3548	0.4129	**0.4387**	0.4103	0.3723
0.1000	0.3658	0.4303	**0.4587**	0.4239	0.3845
0.2000	0.4748	0.5206	**0.5652**	0.5342	0.5013
0.3000	0.5400	0.5703	**0.6039**	0.5832	0.5619
0.4000	0.5768	0.5903	**0.6245**	0.6168	0.5884
0.5000	0.6103	0.6000	0.6400	**0.6426**	0.6077
0.6000	0.6413	0.6090	0.6510	**0.6626**	0.6277
0.7000	0.6613	0.6187	0.6587	**0.6768**	0.6471
0.8000	0.6755	0.6213	0.6652	**0.6819**	0.6574
0.9000	0.6884	0.6232	0.6729	**0.6890**	0.6684
1.0000	**0.6968**	0.6239	0.6774	0.6929	0.6774

Table A.12: TEST PERFORMANCE ON PROTOCOL 2. This table illustrates the performances of the evaluated algorithms on the test dataset of protocol 2 at selected FPR values. For a description of the columns, see table A.11.

FPR	Softmax	EOS	OpenMax	EVM	PROSER
0.0010	-	-	-	-	**0.0539**
0.0030	-	-	-	-	**0.1186**
0.0060	-	-	-	0.1373	**0.1769**
0.0090	-	**0.2550**	-	0.1938	0.2289
0.0100	-	**0.2630**	-	0.2050	0.2392
0.0200	-	0.3352	**0.3430**	0.2774	0.3177
0.0300	0.3866	0.3719	**0.3983**	0.3361	0.3639
0.0400	0.4239	0.4013	**0.4298**	0.3701	0.3929
0.0500	0.4479	0.4293	**0.4552**	0.4012	0.4240
0.0600	0.4686	0.4526	**0.4786**	0.4257	0.4439
0.0700	0.4877	0.4686	**0.4961**	0.4461	0.4644
0.0800	0.5034	0.4815	**0.5101**	0.4599	0.4812
0.0900	0.5185	0.4986	**0.5230**	0.4706	0.4963
0.1000	0.5335	0.5103	**0.5381**	0.4854	0.5108
0.2000	0.6083	0.5859	**0.6170**	0.5778	0.5818
0.3000	0.6511	0.6231	**0.6573**	0.6249	0.6258
0.4000	0.6792	0.6457	**0.6905**	0.6539	0.6538
0.5000	0.6995	0.6626	**0.7126**	0.6739	0.6739
0.6000	0.7187	0.6755	**0.7265**	0.6861	0.6848
0.7000	0.7306	0.6839	**0.7332**	0.6942	0.6945
0.8000	**0.7444**	0.6905	0.7361	0.7000	0.7023
0.9000	**0.7532**	0.6952	0.7395	0.7034	0.7066
1.0000	**0.7565**	0.6964	0.7439	0.7059	0.7115

Table A.13: TEST PERFORMANCE ON PROTOCOL 3. This table illustrates the performances of the evaluated algorithms on the test dataset of protocol 3 at selected FPR values. For a description of the columns, see table A.11.

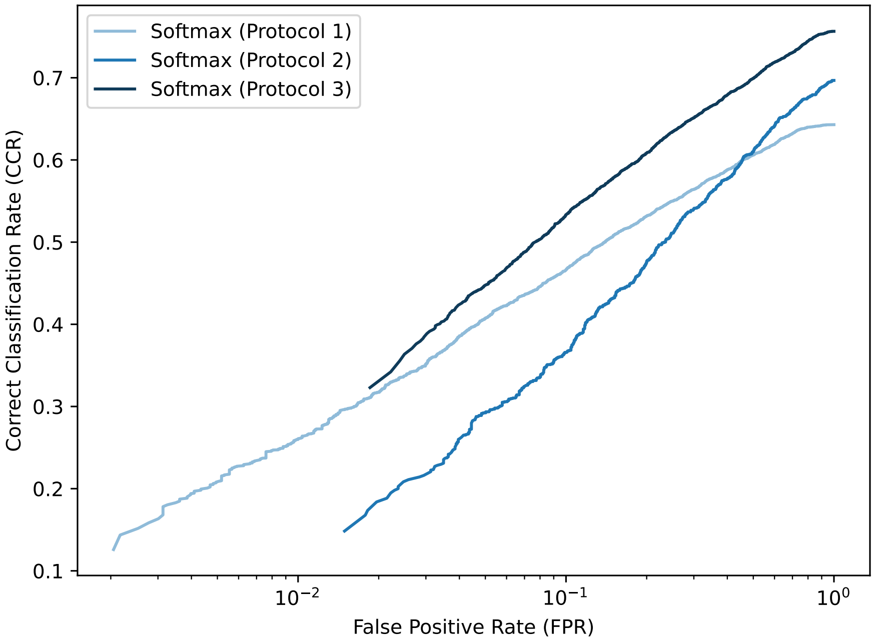


Figure A.1: OSCR CURVES FOR SOFTMAX. This figure depicts, for all three protocols, the OSCR curves associated with the baseline model.

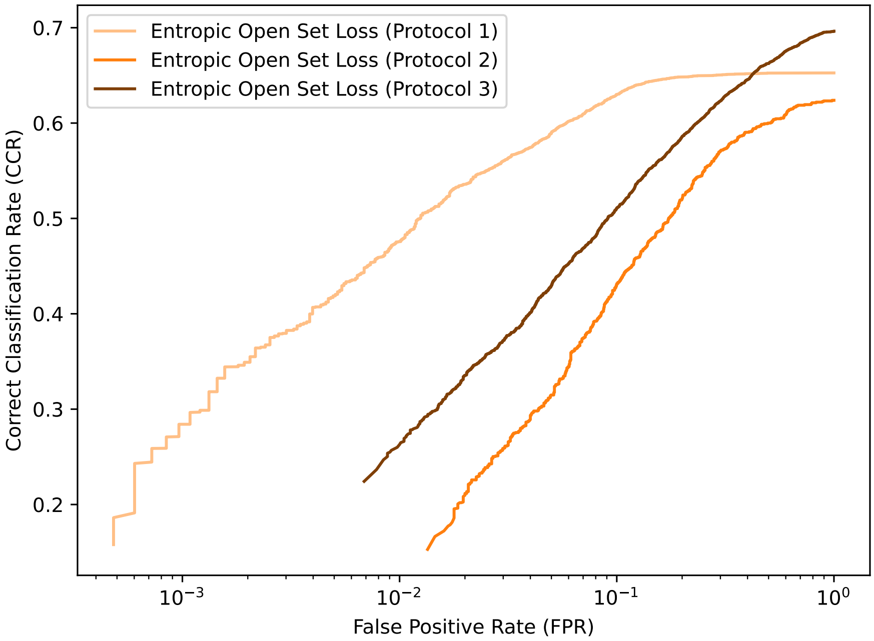


Figure A.2: OSCR CURVES FOR EOS. This figure depicts, for all three protocols, the OSCR curves associated with EOS.

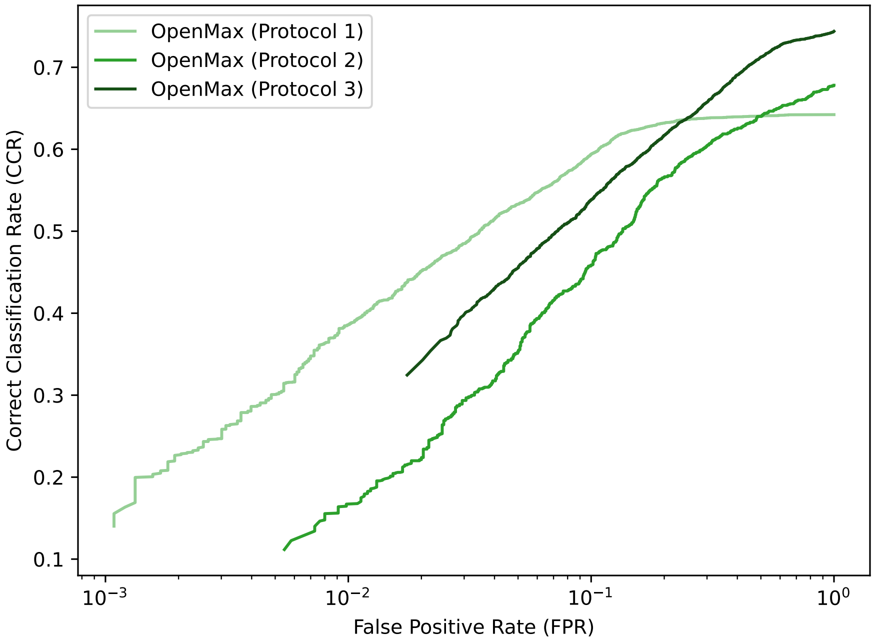


Figure A.3: OSCR CURVES FOR OPENMAX. This figure depicts, for all three protocols, the OSCR curves associated with OpenMax.

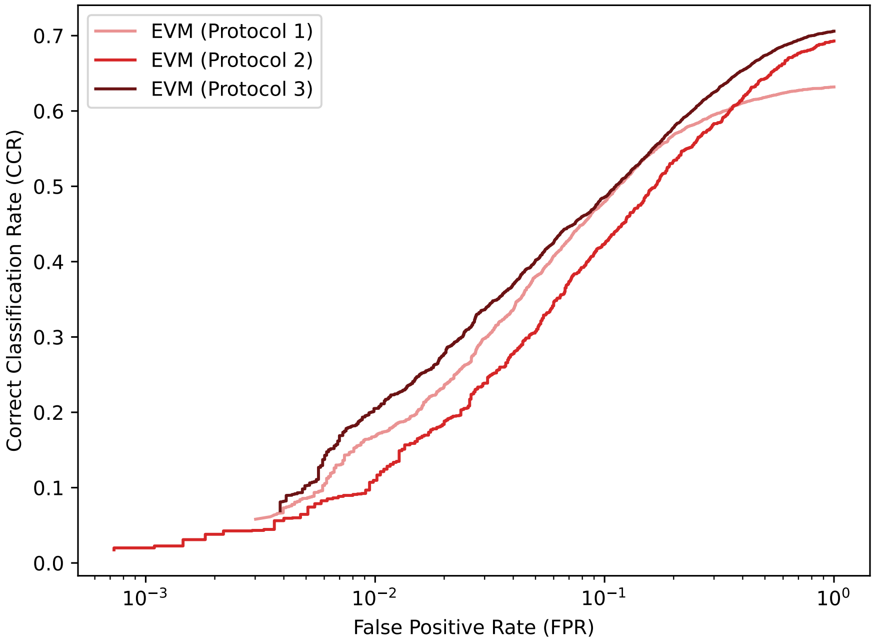


Figure A.4: OSCR CURVES FOR EVM. This figure depicts, for all three protocols, the OSCR curves associated with EVM.

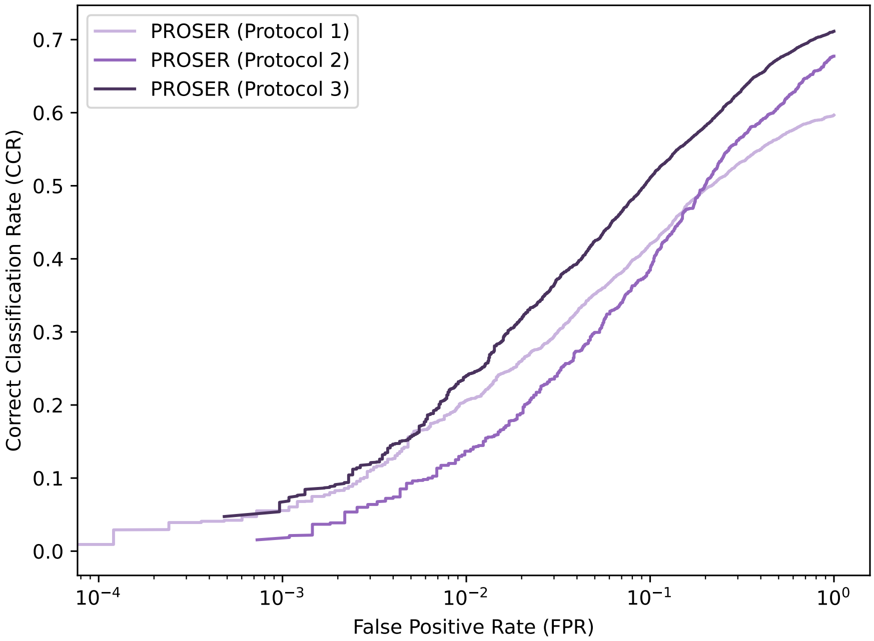


Figure A.5: OSCR CURVES FOR PROSER. This figure depicts, for all three protocols, the OSCR curves associated with PROSER.

List of Figures

List of Tables

List of Listings

Bibliography

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th International Conference on Database Theory*, page 420–434, Berlin, Heidelberg. Springer.

Aiolli, F., San Martino, G., and Sperduti, A. (2008). A kernel method for the optimization of the margin distribution. In *Proceedings of the 18th International Conference on Artificial Neural Networks*, page 305–314, Berlin, Heidelberg. Springer.

Batista, G. E. A. P. A., Prati, R. C., and Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Exploration Newsletter*, 6(1):20–29.

Bendale, A. and Boult, T. (2015). Towards open world recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1893–1902.

Bendale, A. and Boult, T. E. (2016). Towards open set deep networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1563–1572.

Bengio, Y. (2012). *Practical Recommendations for Gradient-Based Training of Deep Architectures*, pages 437–478. Springer, Berlin, Heidelberg.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bhoumik, A. (2021). Open-set classification on ImageNet. Master's thesis, University of Zurich.

Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer, New York, NY.

Bostock, M. (2018). ImageNet Hierarchy. `https://observablehq.com/@mbostock/imagenet-hierarchy`. Accessed: 2022-05-04.

Boult, T. E., Cruz, S., Dhamija, A., Gunther, M., Henrydoss, J., and Scheirer, W. (2019). Learning and the unknown: Surveying steps toward open world recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9801–9807.

Branco, P., Torgo, L., and Ribeiro, R. P. (2016). A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys*, 49(2).

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236, Berlin, Heidelberg. Springer.

Bulatov, Y. (2011). notMNIST dataset. `http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html`. "Accessed: 2022-07-22".

Cevikalp, H. (2017). Best fitting hyperplanes for classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1076–1088.

Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*.

Chen, Z. and Liu, B. (2018). *Lifelong Machine Learning*. Springer International Publishing, Cham.

Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649.

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). High-performance neural networks for visual object classification. *CoRR*, abs/1102.0183.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Language*, 20(3):273–297.

De Stefano, C., Sansone, C., and Vento, M. (2000). To reject or not to reject: that is the question-an answer in case of neural classifiers. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(1):84–94.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Dhamija, A. (2021). Various Algorithms & Software Tools (VAST). `https://github.com/Vastlab/vast`.

Dhamija, A. R., Günther, M., and Boult, T. E. (2018). Reducing network agnostophobia. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9157–9168.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). DeCAF: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, page 647–655.

Du Sautoy, M. (2017). *The Great Unknown: Seven Journeys to the Frontiers of Science*. Penguin.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Elsaadany, O. (2022). placeholder-open-set. Unpublished.

Engstrom, L., Ilyas, A., Salman, H., Santurkar, S., and Tsipras, D. (2019). Robustness (python library).

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, USA.

Fisher, R. A. and Tippett, L. H. C. (1928). Limiting forms of the frequency distribution of the largest or smallest member of a sample. *Mathematical Proceedings of the Cambridge Philosophical Society*, 24(2):180–190.

Fisher, Y., Yinda, Z., Shuran, S., Ari, S., and Jianxiong, X. (2015). LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365.

Frey, P. W. and Slate, D. J. (1991). Letter recognition using holland-style adaptive classifiers. *Mach. Learn.*, 6(2):161–182.

Garg, A. and Roth, D. (2003). Margin distribution and learning algorithms. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, page 210–217.

Ge, Z., Demyanov, S., Chen, Z., and Garnavi, R. (2017). Generative OpenMax for multi-class open set classification. In *British Machine Vision Conference, BMVC*. BMVA Press.

Geng, C. and Chen, S. (2022). Collective decision for open set recognition. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):192–204.

Geng, C., Huang, S.-J., and Chen, S. (2021). Recent advances in open set recognition: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(10):3614–3631.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*. Computational and Biological Learning Society.

Grother, P. and Kayee, H. (2016). NIST special database 19 handprinted forms and characters 2nd edition. Technical report, National Institute of Standards and Technology (NIST).

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, page 1321–1330.

Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36.

He, K. and Sun, J. (2015). Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5353–5360.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, pages 630–645, Cham. Springer International Publishing.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural networks for machine learning (Lecture 6a). `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision - ECCV 2016*, pages 646–661, Cham. Springer International Publishing.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456.

Jain, L. P., Scheirer, W. J., and Boult, T. E. (2014). Multi-class open set recognition using probability of inclusion. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision - ECCV 2014*, pages 393–409, Cham. Springer International Publishing.

Jammalamadaka, N., Zisserman, A., Eichner, M., Ferrari, V., and Jawahar, C. V. (2012). Has my algorithm succeeded? An evaluator for human pose estimators. In *European Conference on Computer Vision (ECCV)*, pages 114–128. Springer Berlin Heidelberg.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, New York, NY, USA. Association for Computing Machinery.

Júnior, P., Souza, R., Werneck, R., Stein, B., Pazinato, D., Almeida, W., Penatti, O., Torres, R., and Rocha, A. (2017). Nearest neighbors distance ratio open-set classifier. *Machine Learning*, 106:1–28.

Kardan, N. and Stanley, K. O. (2017). Mitigating fooling with competitive overcomplete output layer neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 518–525.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *International Conference on Learning Representations, ICLR*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA. Curran Associates Inc.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, page 9–50, Berlin, Heidelberg. Springer.

Leong, M. C., Prasad, D. K., Lee, Y. T., and Lin, F. (2020). Semi-CNN architecture for effective spatio-temporal learning in action recognition. *Applied Sciences*, 10(2).

Liang, S., Li, Y., and Srikant, R. (2018). Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*.

Matan, O., Kiang, R., Stenard, C., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L., and Lecun, Y. (1990). Handwritten character recognition using neural network architectures. In *Proceedings of the 4th US Postal Service Advanced Technology Conference*.

Mensink, T., Verbeek, J., Perronnin, F., and Csurka, G. (2012). Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *European Conference on Computer Vision (ECCV)*, pages 488–501, Berlin, Heidelberg. Springer.

Mensink, T., Verbeek, J., Perronnin, F., and Csurka, G. (2013). Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2624–2637.

Miller, G. A. (1995). Wordnet: A lexical database for English. *Communications of the ACM*, 38(11):39–41.

Neal, L., Olson, M., Fern, X., Wong, W.-K., and Li, F. (2018). Open set learning with counterfactual images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, page 613–628.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. `http://ufldl.stanford.edu/housenumbers/`. Accessed: 2022-05-02.

Ng, A. Y. and Jordan, M. I. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, page 841–848, Cambridge, MA, USA. MIT Press.

Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436.

Oza, P. and Patel, V. M. (2019). C2AE: Class conditioned auto-encoder for open-set recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2302–2311.

Palechor, A., Bhoumik, A., and Günther, M. (2023). Large-scale open-set classification protocols for ImageNet. In *Winter Conference on Applications of Computer Vision (WACV)*. IEEE. **under review**.

Pant, A. K., Panday, S. P., and Joshi, S. R. (2012). Off-line Nepali handwritten character recognition using multilayer perceptron and radial basis function neural networks. In *2012 Third Asian Himalayas International Conference on Internet*, pages 1–5.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pelckmans, K., Suykens, J., and Moor, B. (2007). A risk minimization principle for a class of parzen estimators. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Perera, P., Morariu, V. I., Jain, R., Manjunatha, V., Wigington, C., Ordonez, V., and Patel, V. M. (2020). Generative-discriminative feature representations for open-set recognition. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11811–11820.

Ranjan, R., Castillo, C. D., and Chellappa, R. (2017). L2-constrained Softmax loss for discriminative face verification. *CoRR*, abs/1703.09507.

Reed, R. and Marksll, R. J. (1999). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, Cambridge, MA, USA.

Reyzin, L. and Schapire, R. E. (2006). How boosting the margin can also boost classifier complexity. In *Proceedings of the 23rd International Conference on Machine Learning*, page 753–760.

Ripley, B. D. (1996). *Pattern recognition and neural networks*. Cambridge University Press.

Ristin, M., Guillaumin, M., Gall, J., and Van Gool, L. (2014). Incremental learning of ncm forests for large-scale image classification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3654–3661.

Roady, R., Hayes, T. L., Kemker, R., Gonzales, A., and Kanan, C. (2020). Are open set classification methods effective on large-scale datasets? *PLoS ONE*, 15.

Rozsa, A., Günther, M., and Boult, T. E. (2017). Adversarial robustness: Softmax versus openmax. *CoRR*, abs/1708.01697.

Rudd, E. M., Jain, L. P., Scheirer, W. J., and Boult, T. E. (2018). The extreme value machine. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:762–768.

Rumelhart, D., Hinton, G., and Williams, R. (1986a). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252.

Scheirer, W., Rocha, A., Sapkota, A., and Boult, T. (2013). Toward open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:1757–1772.

Scheirer, W. J., Jain, L. P., and Boult, T. E. (2014). Probability models for open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2317–2324.

Scheirer, W. J., Rocha, A., Micheals, R. J., and Boult, T. E. (2011). Meta-recognition: The theory and practice of recognition score analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1689–1695.

Schnyder, J. (2021). Deep adversarial training for teaching networks to reject unknown inputs. Bachelor's thesis, University of Zurich.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations, ICLR*.

Shu, L., Xu, H., and Liu, B. (2017). DOC: Deep open classification of text documents. *CoRR*.

Simard, P., Steinkraus, D., and Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition*, pages 958–963.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y., editors, *International Conference on Learning Representations, ICLR*.

Slavík, P. (1996). A tight analysis of the greedy algorithm for set cover. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, page 435–441, New York, NY, USA. Association for Computing Machinery.

Sun, X., Yang, Z., Zhang, C., Ling, K.-V., and Peng, G. (2020). Conditional gaussian distribution learning for open set recognition. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13477–13486.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *International Conference on Learning Representations*. Computational and Biological Learning Society.

Teh, Y. W. (2010). *Dirichlet Process*, pages 280–287. Springer, Boston, MA.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Vareto, R., Silva, S., Costa, F., and Schwartz, W. R. (2017). Towards open-set face recognition using hashing functions. In *2017 IEEE International Joint Conference on Biometrics (IJCB)*, pages 634–641.

Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Lopez-Paz, D., and Bengio, Y. (2019). Manifold mixup: Better representations by interpolating hidden states. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6438–6447.

Wen, Y., Zhang, K., Li, Z., and Qiao, Y. (2016). A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision (ECCV)*, pages 499–515, Cham. Springer International Publishing.

Wold, H. (1985). Partial least squares. In *Encyclopedia of Statistical Sciences*. Wiley.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017). Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995.

Ye, H.-J., Hu, H., and Zhan, D.-C. (2021). Learning adaptive classifiers synthesis for generalized few-shot learning. *International Journal of Computer Vision*, 129(6):1930–1953.

Yoshihashi, R., Shao, W., Kawakami, R., You, S., Iida, M., and Naemura, T. (2019). Classification-reconstruction learning for open-set recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4011–4020.

Yu, Y., Qu, W.-Y., Li, N., and Guo, Z. (2017). Open category classification by adversarial sample generation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3357–3363.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pages 818–833, Cham. Springer International Publishing.

Zhang, H. and Patel, V. M. (2017). Sparse representation-based open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(8):1690–1696.

Zhang, P., Wang, J., Farhadi, A., Hebert, M., and Parikh, D. (2014). Predicting failures of vision systems. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3566–3573.

Zhou, D.-W., Ye, H.-J., and Zhan, D.-C. (2021a). CVPR21-Proser. `https://github.com/zhoudw-zdw/CVPR21-Proser`.

Zhou, D.-W., Ye, H.-J., and Zhan, D.-C. (2021b). Learning placeholders for open-set recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4401–4410.