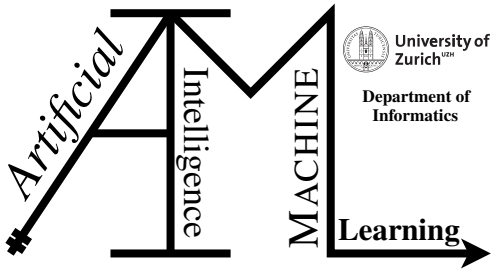
Deep Adversarial Training for Teaching Networks to Reject Unknown Inputs

Bachelor Thesis

Jan Schnyder
17-926-106

Submitted on
July 22 2021

Thesis Supervisor
Prof. Dr. Manuel Günther



Bachelor Thesis

Author: Jan Schnyder, janjosef.schnyder@bluewin.ch

Project period: 01.02.2021 - 22.07.2021

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

Acknowledgements

I would like to express the deepest appreciation to Prof. Dr. Manuel Günther, who entrusted me with this thesis, despite it originally being on a Master's level. Thanks to him I discovered my passion for Deep Learning and now know that I want to further pursue this field. Without his guidance and persistent help this thesis would not have been possible.

I would like to thank my family, especially my father, who still to this day shares interests about these topics and serves as a person I can look up to.

In addition, I would like to thank my friends and everyone that helped and supported me during this time.

Abstract

Modern day machine learning models are becoming omnipresent and are required to handle progressively more complex environments in their tasks. In classification problems, an increasingly popular scenario is called Open Set Recognition, which does not require the model to have complete knowledge of the world and during which unknown classes can be submitted to the algorithm while testing. This thesis tackles the challenge to correctly handle and reject these unknown inputs by performing adversarial training on our classification model. Furthermore, we analyze the difference in performance of several state-of-the-art adversarial attacks used in our adversarial training. The experiments show that our approach effectively deals with unknown inputs and delivers very promising results. To our knowledge, there has been no prior work that used adversarial training for Open Set Recognition like in our approach.

Zusammenfassung

Moderne Modelle für maschinelles Lernen sind heutzutage omnipräsent und betreuen immer komplexer werdende Aufgaben. Bei Klassifikationsproblemen gewinnt das sogenannte "Open Set Recognition" Szenario an Beliebtheit, bei dem die Modelle keine perfekte Information ihres Umfelds benötigen, sondern auch in unbekannten Situationen brauchbar sind. Diese Arbeit geht die Herausforderung an, mit Hilfe von Adversarial Training mit unbekannten Umgebungen korrekt umzugehen. Zusätzlich vergleichen wir verschiedene moderne Methoden, um sogenannte Adversarial Examples zu erzeugen. Unsere Experimente zeigen, dass die Methode höchst effektiv in unbekannten Situationen ist und auch korrekt mit ihnen umgehen kann. Unseres Wissens nach gibt es noch keine Arbeit, die zuvor Adversarial Training im Open Set Recognition Szenario angewendet hat.

Contents

Chapter 1

Introduction

Deep Learning is one of the fastest evolving fields in computer science, with various applications in machine vision, reinforcement learning and many more. One very popular task in deep learning is classification, since deep neural networks are well-known to be able to learn how to classify the content of images based on examples of the classes. Here, the network try to predict a class for each sample it is given. This task is solved by many state-of-the-art approaches, yet most of these approaches have one thing in common: All these networks are trained to only distinguish the classes they are shown during training. This is called Closed Set Recognition, where the training and testing data is drawn from the same label and feature distribution. When a previously unseen sample of the training classes is shown to the network, it is able to predict the correct class with a high level of accuracy and confidence. However, during Open Set Recognition, if you feed the network with samples not from the classes it is trained on, it delivers poor results, since the network has no other choice than classifying it as one of the known classes. Unfortunately, this usually happens with rather high confidence, such that simply thresholding on confidence and rejecting samples below the threshold as unknown is not a feasible solution. To prevent this, there have been attempts to include unknown samples in the training set to further improve the ability to recognize and reject unknown input. However, this would only work for unknown samples that are related to the ones shown in the training set. Therefore, to completely cover the space of all possible unknowns, it would require an immense amount of data and there is no reasonable way to fit all this data into the training set. This means there will always be unknown samples that have nothing in common with the ones the network has seen while training. Since the possibility is very small that the network will only encounter unknown samples from its training set, this approach has remarkable disadvantages. This thesis proposes an approach to tackle the Open Set Recognition scenario, where incomplete knowledge of the world exists at training time, requiring classifiers to not only accurately classify the known classes, but also effectively deal with unknown ones. More specifically, it examines the effects of adversarial training on the ability of neural networks to reject unknown input.

Adversarial training is done with so-called adversarial examples, which are carefully constructed input samples for a network that seem very similar to normal inputs but produce completely different outputs. These adversarial examples have such minor differences to their original image that a human is not able to distinguish between them. However, these seemingly equivalent samples produce two totally different outputs when given to a specific neural network, leading to undesired model behaviour. This suggests that adversarial examples expose fundamental blind spots in neural networks. Since the discovery of adversarial examples, dozens of methods for their generation have been proposed, so-called adversarial attacks. The most prominent ones are discussed in this thesis.

So how do these malicious attacks help our network recognize and differentiate known from unknown input? In order to better understand the problem, we use a categorization of samples proposed by Dhamija et al. (2018). Let us assume $\mathcal{Y} \subset \mathbb{N}$ be the infinite label space of all classes, which can be broadly categorized into:

- $\mathcal{C} = \{1, ..., C\} \subset \mathcal{Y}$: The *known classes* that the network shall identify.
- $\mathcal{U} = \mathcal{Y} \setminus \mathcal{C}$: The *unknown classes* containing all types of classes the network needs to reject. Since $\mathcal{Y}$ is infinite and $\mathcal{C}$ is finite, $\mathcal{U}$ is also infinite. The set $\mathcal{U}$ can further be divided into:
  1. $\mathcal{B} \subset \mathcal{U}$: The background or *known unknown* classes. Since $\mathcal{U}$ is infinitely large, during training only a small subset $\mathcal{B}$ can be used.
  2. $\mathcal{A} = \mathcal{U} \setminus \mathcal{B} = \mathcal{Y} \setminus (\mathcal{C} \cup \mathcal{B})$: The *unknown unknown* classes, which represent the rest of the infinite space $\mathcal{U}$, samples from which are not available during training, but only occur at test time.

As previously stated, one approach to improve the performance is to include samples of as many different unknowns as possible in the training set, such that the network will recognize any similar input and will be able to reject it. A major trend observed in this approach is that increasing the similarity of the known unknowns $\mathcal{B}$ and the known examples $\mathcal{C}$ also increases the ability to reject unknown samples (Dhamija et al., 2018). Since adversarial examples are intentionally constructed to differ as little as possible to their original samples, yet still get misclassified by the networks, they present a perfect candidate for $\mathcal{B}$.

Our approach performs adversarial training with adversarial examples that are generated on the fly while training the network. Contrary to the usual adversarial training, we do not label the adversarial examples with the class they depict, but we do intentionally label them as unknown. In the following chapters we will look into the effects of adversarial training in the open set scenario and its impact on performance of classification networks. Our experiments then show the effectiveness of our method on the ability to reject unknown inputs.

Chapter 2

Background and Related Work

This chapter yields a brief introduction to the literature this thesis is based on. After presenting a high level overview of the Open Set Recognition (OSR) scenario and the field of adversarial examples, it proceeds to dive further into the specific adversarial attacks used in our approach. Additionally, it examines some background information about the problem setting and advances in Open Set Recognition.

2.1 The Open Set Problem

When doing Open Set Recognition, incomplete knowledge of the world exists at training time, requiring the classifiers to not only accurately classify the known classes, but also effectively deal with unknown ones (Geng et al., 2020). So the challenge here is to correctly classify samples from $\mathcal{C}$, which the network is trained on, but also detect samples from $\mathcal{U}$ (some of which the network has never seen before) and handle them correctly. A visualization of the training and testing environments is shown in Figure 2.1. So how would one handle unknown samples $\mathcal{U}$, or more specifically, $\mathcal{A}$; the ones which the network has not seen during training?

2.1.1 Open Set Approaches

Traditionally, machine learning models always operated on a closed set of data, so-called Closed Set Recognition (CSR). This also meant that the test and training dataset were from the same distribution and have the same classes. When applying this in practice however, finding such an enclosed environment can be very challenging. This then gave rise to Open Set Recognition. Initially, the major question when doing Open Set Recognition is what methods are best in this kind of scenario. Original approaches often considered to use binary classifiers to distinguish $\mathcal{C}$ from $\mathcal{U}$, which gave rise to SVM-based methods. One example would be an extensions of a binary SVM proposed by Scheirer et al. (2013). Following this, there have also been approaches constructing new classifiers for Open Set Recognition (Rudd et al., 2017).

More recent approaches often tend towards using deep neural networks, which are also used in our approach. Even though basic neural networks are known to classify samples from an unknown class as a known class with rather high confidence, there have been many modern methods to assess these problems. An example for this would be the OpenMax model by Bendale and Boult (2016), which proposed a first solution towards open set deep networks. They introduced an OpenMax layer, which resulted in a model optimized for Open Set Recognition that

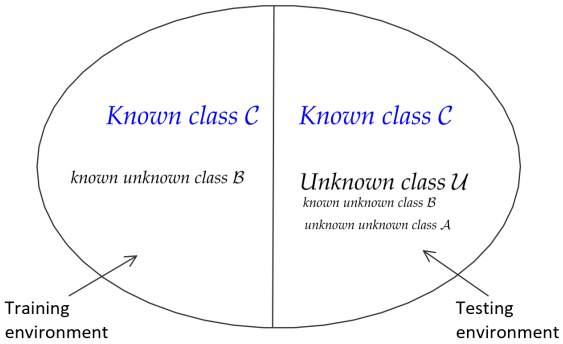


Figure 2.1: CLOSED SET RECOGNITION. Depicted in blue is the Closed Set scenario. The extension to the Open Set scenario is further depicted in black.

significantly outperformed basic deep networks. Another more modern approach using neural networks is PROSPER (Zhou et al., 2021), which also uses a background class $\mathcal{B}$ in a similar way than we do. Our approach uses deep neural networks in combination with adversarial training to tackle the open set problem.

Following these approaches, there has also emerged an extension to the Open Set Recognition problem, called Open World Recognition, proposed by Bendale and Boult (2015). Additional to the tasks of Open Set Recognition, they further want to include the unknowns encountered during inference in the training set (Boult et al., 2019). This way unknown samples can be converted into additional known classes. However, this thesis only focuses on the problem of Open Set Recognition.

2.1.2 Handling Unknowns in Deep Learning

This section further investigates the problem of distinguishing $\mathcal{C}$ from $\mathcal{U}$ when using neural networks for Open Set Recognition. Deep neural networks usually output a probability for each class it is trained on. One solution to the problem would now be to simply threshold the probability scores and assign all samples that did not reach a certain value to the unknown class (Matan et al., 1990; Geng et al., 2020). While samples from a class which the network is not trained on usually do have lower probability scores on average, this solution brings up the problem on how to cleanly isolate these samples. Since the probability scores may have high variance, this can lead to the rejection of many known samples and misclassification of many unknown samples. Even when including unknown samples in the training set, so-called known unknowns (Scheirer et al., 2014), the performance would only increase towards unknown samples related to those in the training set. Simply thresholding on the output probability can be considered as rejecting uncertain predictions, rather than rejecting unknown classes. Therefore, such an approach often calls for a clever training strategy, which adjusts the probability scores accordingly.

Another approach would be adding an additional class, to which the network would assign all the samples which it predicts as unknown. This class is often referred to as the "garbage" class (Linden and Kindermann, 1989). When classifying a dataset, one would simply add an additional class for all the unknowns encountered in the testing set. The "garbage" class solution often is more effective than simply thresholding the output probabilities. However, this approach runs into similar problems: the features of all unknown samples are heavily mixed with the known samples, which leads to inaccurate separations.

To fix this issue, Dhamija et al. (2018) proposed a novel loss function called Entropic Open Set loss, which does not need to introduce an additional "garbage" class for the unknowns. This loss is designed to maximize the entropy of unknown samples and thus achieves a better separation in the deep feature space. This will be discussed in Section 3.2.1 in greater detail.

2.2 Adversarial Machine Learning

The main objective of this thesis is studying the results of adversarial training with different adversarial attacks in the OSR scenario. In this section, we dive into what exactly adversarial examples are and review the most important of the many procedures to generate them.

2.2.1 Adversarial Examples

The notion of adversarial training gained a lot of traction in 2014, when Ian Goodfellow published his paper on adversarial samples. Goodfellow et al. (2015) describes them as inputs with small but intentionally worst-case perturbations to samples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence. This will now be further elaborated in this section. Let us start with an example depicted in Figure 2.2, where a network is trained to classify pictures of different entities. On the left, there is an image of a panda. If we give this picture to the network as input, it tells us with a certain confidence (here 57.7%) that this is indeed a panda. Looking at the middle of the figure, there is a noise image which has been artificially generated. We can now modify the original image, which in this example is done by adding small values of this noise image, resulting in the image on the right. For the human eye, both images are identical and, when asked, it is apparent that both images depict a panda bear. However, if we present the new image to our network, it now tells us with almost no doubt (99.3% confidence) that this is a gibbon. We now have our first instance of an adversarial example. Pictures like this expose a fundamental blindspot of the network, and hint that in general, these networks do not truly learn the underlying concepts of the images they are given to classify.

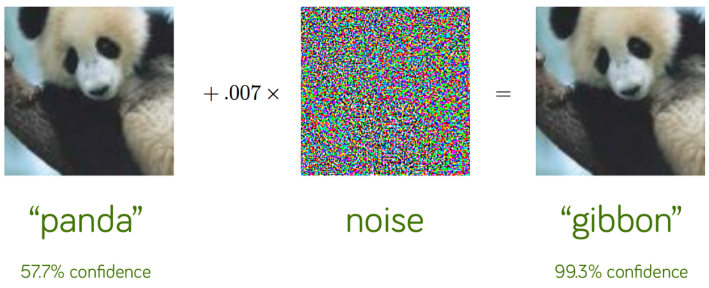


Figure 2.2: ADVERSARIAL EXAMPLE. A demonstration of an adversarial example with its predicted classes and confidences replicated from Goodfellow et al. (2015). Adding the perturbation depicted in the middle to the original image on the left will produce the adversarial example on the right.

There are several ways to generate these malicious images (Goodfellow et al., 2015; Madry et al., 2018; Carlini and Wagner, 2017; Rozsa et al., 2017), so-called adversarial attacks. An adversarial attack also generated the noise image used in our example. These attacks however will be discussed in greater detail in Section 2.3. While we can generate them, the origin of adversarial examples and why they exist in the first place is still a mystery that has not yet been fully solved. There have been many theories on why inputs like these cause such abnormal behaviour, but all of them have been shown to be false. Intuitively, one can think of an adversarial example as a

benign sample with a perturbation added to it. An adversarial example could then be described as $\tilde{x} = x + \alpha$, where α is the perturbation and x the original sample. The goal is to keep α as small as possible while still provoking large changes to the output.

2.2.2 Adversarial Training and Robustness

With the help of these methods to generate adversarial examples, we can now perform adversarial training. This is done by including adversarial examples in the training set. Adversarial training is closely related with adversarial robustness, since this is its most popular use case. Simply said, increasing the adversarial robustness of a network results in less vulnerability against adversarial examples.

However, when doing Open Set Recognition, one focuses on increasing robustness against unknown input and not adversarial examples. This is important to note, since they are two different topics. When training for adversarial robustness, a very simple approach would be to label the adversarial example as the class it depicts. In Figure 2.2, one would now include the generated adversarial example of the panda bear that the network classified as a gibbon in the training set. Now it would be labeled with the class "panda", such that the network can also learn to classify it correctly. There exists a lot of work on trying to increase adversarial robustness (Wang et al., 2019). In our approach however, we use adversarial training to correctly recognize unknown inputs. This means that the label of the adversarial examples are not the label of the class they depict, but instead they get labeled as unknown.

2.2.3 Adversarial Metrics

One very important aspect of adversarial examples is the similarity to their original input. This section introduces statistical norms, which are the most popular metrics to measure similarity between adversarial examples and their respective original images. Statistical norms are used to measure the length or magnitude of vectors. Here, they measure the pixel value differences of the original input and the generated adversarial example. In other words, statistical norms quantify the magnitude of the perturbation α by looking at each pixel individually and sum over all pixel differences. The most popular norms used are the L_1-, L_2- and L_∞-norms.

The L_1-norm is calculated by the sum of absolute differences: $\|\alpha\|_1 = |\alpha_1| + |\alpha_2| + ... + |\alpha_n|$. Intuitively, this just adds the difference of every pixel value of the images, which is then used as a metric of change in the image.

The L_2-norm, also commonly referred as Euclidean norm, usually represents the length of a vector: $\|\alpha\|_2 = \sqrt{\alpha_1^2 + \alpha_2^2 + ... + \alpha_n^2}$. Contrary to the L_1-norm, the metric here is the square root of the sum of the *squared* differences. Both however focus on adding the pixel-wise differences of two images or samples.

The L_∞-norm, also called max-norm, has a different approach. It takes only the maximum value of the vector: $\|\alpha\|_\infty = \max(|\alpha_1|, |\alpha_2|, ..., |\alpha_n|)$. This is also the norm which is going to be used the most in our attacks. Intuitively, the only pixel that matters here is the one that is most different to the original image. Basically, it compares how all the pixels have changed in their values and takes the largest difference as the metric.

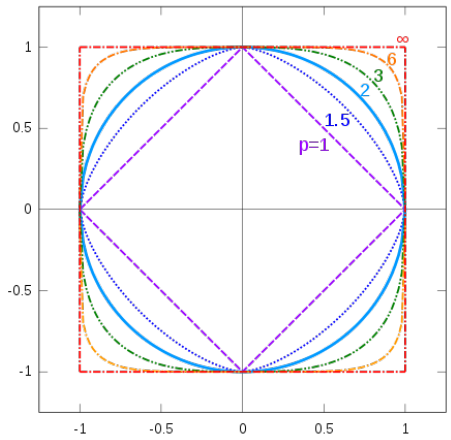


Figure 2.3: STATISTICAL NORMS. unit circles of different p-norms in $\mathbb{R}^2$

One can summarize these norms and their variations using the p-norm notation, commonly denoted as L_p. L_p is defined as:

$$\|\alpha\|_p = \left(\sum_{i=1}^{n} |\alpha_i|^p \right)^{1/p} \tag{2.1}$$

If one now, for example, substitutes $p = 1$ or $p = 2$, we get the L_1-norm and L_2-norm respectively. A very important and elegant fact is that one can define and visualize these L_p-norms as L_p-spaces. Figure 2.3 depicts the unit circles in the two-dimensional space for different L_p-norms. Every vector from the origin to the unit circle has a length of one, the length being calculated with the formula of the corresponding p. Notice how the shape changes as p grows larger.

L_p-norms are not only used to classify adversarial examples, but also adversarial attacks (Goodfellow et al., 2015; Madry et al., 2018; Carlini and Wagner, 2017; Rozsa et al., 2017): If an attack uses an L_p-norm as its metric, it is called ℓ_p-bounded. How this metric gets applied in adversarial attacks will be explained in more detail in the next section.

2.3 Adversarial Attacks

This section focuses on the generation of adversarial examples and describe the attacks we used on our network for adversarial training in greater detail. As previously mentioned, the goal of an adversarial attack is creating input samples for a specific network, that have as little difference to the original data as possible but cause as much damage to the output of the network as possible. Most often this is done by finding the perturbation α to the original image that does this best.

A major task in adversarial attacks is to quantify the magnitude of perturbation α. One popular way to do this is by enforcing that the largest value of α is not allowed to be greater than a predefined value ϵ. This is a very important detail and is denoted as $\|\alpha\|_\infty$, the L_∞-norm or max-norm *constraint*. Since $\|\alpha\|_\infty$ yields the maximum value of α, the max-norm constraint is then defined as $\|\alpha\|_\infty < \epsilon$. This is also equivalent to being ℓ_∞-bounded. This can further be generalized into attacks having the L_p-norm constraint and being ℓ_p-bounded.

2.3.1 Taxonomy of Adversarial Attacks

Depending on the knowledge level of the attacker, adversarial attacks can be classified as either white-box or black-box attacks. In the scenario of a white-box attack, the attacker has complete knowledge of the network, including its architecture, weights and other parameters (Carlini and Wagner, 2017). This is also the approach we tackle in our experiments. When performing a black-box attack however, the attacker only knows the output of model, which means there is no knowledge about any underlying structure of the network. Yet even in the black-box scenario, adversarial attacks can still be highly effective, which makes them a dangerous tool in the wrong hands.

Aside from the black- and white-box taxonomy, adversarial attacks can be subdivided further into targeted and untargeted attacks. When performing a targeted attack, given an input x and a target class t, the goal is to find an adversarial example that is still similar to the input but gets classified as as the target class t. Formally, this attack generates $\tilde{x}$ such that $C(\tilde{x}) \neq C(x)$ with $C(\tilde{x}) = t$ (Carlini and Wagner, 2017). On the other hand, untargeted attacks, which are strictly less powerful, simply focus on finding an adversarial example that will get classified with a different class $C(\tilde{x}) \neq C(x)$. For our experiments we both use targeted and untargeted attacks and all the attack methods listed in Section 2.3 are white-box attacks.

2.3.2 Fast Gradient Sign Method

With the introduction of adversarial examples in 2013 by Szegedy et al. (2014), Goodfellow et al. (2015) shortly after proposed a simple and fast untargeted attack, which he called the fast gradient sign method (FGSM). We can now define θ as the parameters of a model, x as the input to the model and y as the targets associated with x. The loss function to train the neural network can then be denoted as $J(\theta, x, y)$. To find the best magnitude for the perturbations, Goodfellow uses the additional parameter ϵ. We can use the max-norm constraint $\|\alpha\|_\infty < \epsilon$ and define our formula for the perturbation as (Goodfellow et al., 2015)

$$\alpha = \epsilon \operatorname{sign}(\nabla_x J(\theta, x, y)) \tag{2.2}$$

which then results in the adversarial example

$$\tilde{x} = x + \epsilon \operatorname{sign}(\nabla_x J(\theta, x, y)) \tag{2.3}$$

The original image is altered by adding or subtracting ϵ to each pixel. Whether we add or subtract ϵ depends on whether the sign of the gradient of the loss for the respective pixel is positive or negative. Since we are stepping in the direction of the sign of the gradient, the loss itself will usually get bigger, ultimately leading to misclassification of the sample.

This fast gradient sign method derives the optimal perturbation which is constrained by the max-norm $\|\alpha\|_\infty$. Since the perturbation is constrained by the max-norm, it is therefore a ℓ_∞-bounded attack. The main focus is now optimizing over the parameter ϵ. The adversarial example in Figure 2.2 is generated using FGSM with $\epsilon = 0.007$. Varying the max-norm constraint by changing ϵ results in adversarial examples potentially having larger impact on the output but also affects similarity to the original sample.

2.3.3 Projected Gradient Descent

A few years after the introduction of the fast gradient sign method, researchers Madry et al. (2018) at MIT found an even mightier attack based on Goodfellow's initial work. As the fast gradient sign method just takes one step in the direction of the sign of the gradient, this new attack would attempt to take multiple. Therefore, it can be seen as a sort of multi-step variant of FGSM. This attack is essentially performing projected gradient descent (PGD) on the loss function

$$x^{i+1} = \Pi_{x+S}\left(x^i + \eta\,\mathrm{sign}(\nabla_x J(\theta, x^i, y))\right) \tag{2.4}$$

where η is the learning rate and Π_{x+S} performs a projection back into the space of allowed perturbations S (Madry et al., 2018). Since we focus on ℓ_∞-bounded attacks, the set of allowed perturbation S is adjusted accordingly. Just like FGSM, the PGD attack is also a constrained optimization problem and an untargeted attack. It attempts to find the perturbation α that maximises the loss of a model on a particular input while keeping the size of α smaller than a specified amount ϵ. The process is further visualized in Figure 2.4

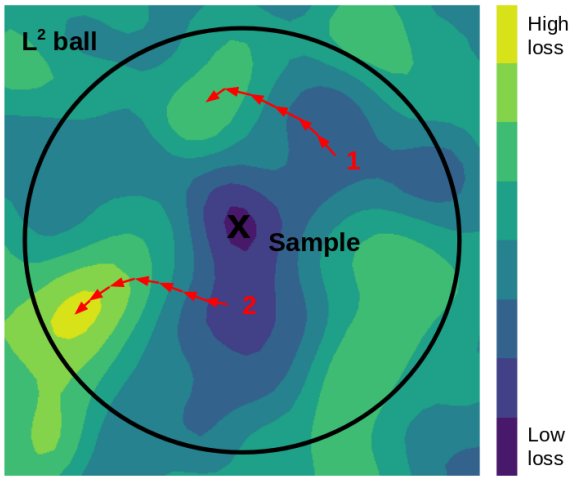


Figure 2.4: SAMPLE DISTRIBUTION SPACE. Visualization of an iterative PGD attack (Knagg, 2019)

In this example figure, an ℓ_2-bounded attack is performed for illustration purposes, since the L_2-norm yields a nice circle. How the boundary changes for other norms can be seen in Figure 2.3. Figure 2.4 depicts the possible distribution space for all pixel values. This means every possible

image is found in this distribution space and every point in this space would give us a sample of an image. The figure also depicts an L_2-ball, with its center being the original sample. The radius is dependent on what value ϵ is assigned to. More specifically, all the samples that are on the radius of an L_p-ball have exactly $\|\alpha\|_p = \epsilon$, all samples outside of the circle have $\|\alpha\|_p > \epsilon$ and all samples inside of the circle have $\|\alpha\|_p < \epsilon$. Since our perturbation are bounded by the constraint $\|\alpha\|_p < \epsilon$, we only consider samples inside of the circle for our adversarial examples. So how do we know which sample is best? As described already, the goal of the method is to maximize the loss function, so we want to find the sample inside of the ℓ_p-ball that produces the highest loss. We can achieve this using a slight variation of gradient descent: projected gradient descent.

Therefore, Figure 2.4 also has a heat map for the loss, indicating where the samples with a high loss are. This is where we want to end up when searching for possible adversarial examples. With PGD we start at a random location inside of the ℓ_p-ball and take a fixed size gradient step in the direction of the greatest loss. We can repeat this for an arbitrary number of times or until we reach a local optimum. However, it is possible that the PGD update step would overshoot the ℓ_p-ball with its gradient step. Since we are not allowed to leave the ℓ_p-ball it simply gets projected back to the nearest point inside of it. One can now repeat the whole procedure, starting at a different random location and hoping to find a better optimum. In Figure 2.4 we can see two full PGD iterations that both converged in different optima.

2.3.4 Carlini and Wagner

The heaviest and most extreme attack used in our approach is developed by Carlini and Wagner (2017) and appropriately named Carlini and Wagner (CnW) attack. Again, their first goal is finding a perturbation α that would minimize the difference between original sample x and adversarial example $\tilde{x} = x + \alpha$. Secondly, the adversarial example $\tilde{x}$ should get classified as the target class t, which means their attack is targeted. The difference between x and $\tilde{x}$ is again measured using L_p-norms. They also point out a major flaw in PGD: when projecting the coordinates onto the ℓ_p-ball, the input for the next iteration of the algorithm is unexpectedly changed and this can result in bad and unwanted behaviour. To fix this problem, they use a trick which they call *Change of variables*. Instead of optimizing over α, they introduce a new variable w and optimize over w instead (Carlini and Wagner, 2017):

$$\alpha_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i \tag{2.5}$$

This now automatically enforces the gradient step to always be inside of the ℓ_p-ball. With this new formula they propose attacks for the L_0-, L_2- and L_∞-norm, of which the L_∞-attack provides the worst results. This attack does not get used extensively in our approach, since it requires a lot more computational power than other attacks. Therefore, it will not be discussed in greater detail.

2.3.5 Layerwise Origin-Target Synthesis

The last adversarial attack used in our approach is called Layerwise Origin-Target Synthesis (LOTS) proposed by Rozsa et al. (2017). This attack differs from the others since it has its main focus on the deep features of the sample instead of mostly focusing on the last layer of the network. Other than the output, deep features are the activation of nodes and layers *inside* of the model. More specifically, LOTS in our approach focuses on the penultimate layer of the network. The attack perturbs samples such that their deep feature representations mimic the ones of a selected

target, making it another targeted attack.

To define this properly, let us consider a network f with layers $z^{(l)}, l = 1, ..., L$. The internal representation of a given input x at layer l can be defined as (Rozsa et al., 2017):

$$f^{(l)}(x) = z^{(l)}\left(z^{(l-1)}\left(...\left(z^{(1)}(x)\right)...\right)\right) \tag{2.6}$$

We can visualize this in a similar way as projected gradient descent in Figure 2.4. Since we use the feature representation of the penultimate layer in our approach, we will do the same in this example. Instead of using the possible distribution space for all pixel values as in PGD, we now use the distribution space of the deep features. We can map the original input on the internal representation to its corresponding place. This point is also called the origin. Additionally, in the same feature space we can also map the target t. In other targeted attacks, t usually represents the target class. In LOTS however, the target t is a deep feature representation that can be chosen arbitrarily. Now, starting from the origin, we can iteratively get closer to the target t. The closer the sample gets to t, the more similar its features will become and the more it will mimic it. The iterative approach is done in a similar way as in PGD with step wise updates. However, here we specifically use the Mean Squared Error loss between target t and the feature representation, instead of maximizing the loss between the adversarial example and its original sample. Therefore, when doing targeted attacks we step in the direction of the *negative* gradient. Instead of using the sign of the gradient, it uses a scaled version of it for the updates. This can be repeated until the sample reaches the origin or the distance between the sample and the target t is smaller than a predefined threshold value.

Chapter 3

Approach

This chapter examines our approach of tackling classification in the Open Set scenario. We use adversarial training to accurately classify the known classes $\mathcal{C}$, but also effectively deal with the unknown examples in $\mathcal{U}$. Before diving into the specifics of the approach, this section first gives a high level overview to the thought process. We then grant a detailed description of the entire approach, ranging from the used data to network architectures and visualizations. Lastly, we examine the main concepts used in our approach in greater detail.

3.1 Architecture

This section presents the whole project architecture, what it is based on and how it was built. We also dive into the model architecture and the specifics of feature extractions and visualization.

3.1.1 Data

For simplicity and generality, we exclusively use the MNIST dataset for our known class $\mathcal{C}$. MNIST, being one of the most iconic datasets in all of machine learning, consists of 60'000 hand-written and labeled digits. Due to its iconic status and it being an easy benchmark for classification networks, we have decided that this dataset would be the best fit for our experiments. Having chosen $\mathcal{C}$, we now need to decide what dataset best represents the unknown class $\mathcal{U}$, more specifically, we need to represent the known unknowns $\mathcal{B}$ in the training set and the unknown unknowns $\mathcal{A}$ in the testing set. Since we perform adversarial training, the set $\mathcal{B}$ of known unknowns consists of adversarial examples generated from the original data $\mathcal{C}$. This means, $\mathcal{B}$ is a dataset full of adversarial examples of digits, which are generated by us during training. Some examples are depicted in Figure 3.1.

Lastly we need to construct the set $\mathcal{A}$, the *unknown* unknown samples. These are the unknown samples the network is presented with during testing time, which it has not seen before. Ideally, these samples would include pictures of any kind of objects the world has to offer. However, for simplicity's sake, we initially take the set of hand-written letters from the EMNIST dataset (Cohen et al., 2017). This has turned out to be a good educated guess, since it has shown that if the network performs well on rejecting letters, it does even better on rejecting more complex and less related samples, such as pictures of cars or clothes. Despite using EMNIST letters in most of our experiments, we also use datasets like FashionMNIST (Xiao et al., 2017) for $\mathcal{A}$, to observe the change in performance.

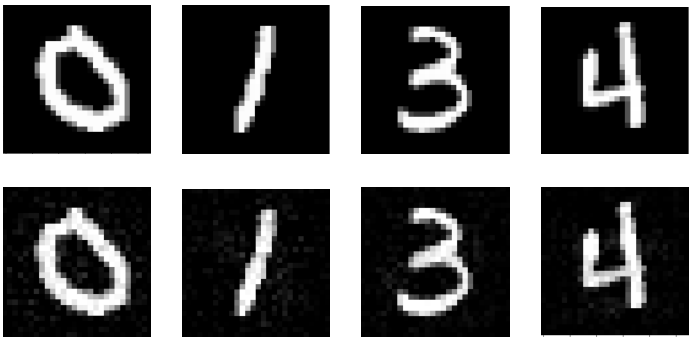


Figure 3.1: MNIST ADVERSARIAL EXAMPLES. Top row: Example pictures of MNIST, which define our class $\mathcal{C}$. Bottom row: Adversarial examples from the pictures in the top row, which define our class $\mathcal{B}$. The adversarial Examples are generated using LOTS with $\epsilon = 0.2$.

3.1.2 Model Architecture

For our network model architecture, we use the model presented by Wen et al. (2016) called LeNet++, which is based on the famous model LeNet proposed by Lecun et al. (1998). It is a convolutional neural network consisting of 3 convolutional blocks, followed by fully connected linear layers. The whole architecture can be seen in Figure 3.2. The convolutional blocks all consist of two cascaded convolutional layers with 32 filters. The first layer only has one input channel, since the MNIST dataset is in gray scale. All filters are of size 5x5, have a stride of 1 and a padding of 2. At the end of each convolutional block, a batch norm (Ioffe and Szegedy, 2015) gets applied on the output of the convolutions. This batch norm plays an important role in our experiments and will further be discussed in the experiments in Section 4.1 and the discussion in Chapter 5.

Additionally, wrapped around every convolutional block, there is a max-pooling layer with a grid of 2×2, where the stride and padding are 2 and 0 respectively. Lastly, every block is activated by a Parametric Rectified Linear Unit (He et al., 2015) as the activation function. After the convolution there are two final fully connected layers. These represent a kind of bottleneck architecture. The first layer maps the input to a 2-dimensional output, where the feature extraction for the latent 2D feature space happens. These outputs are also used for visualization purposes. The last one maps the 2D space to a 10-dimensional output, containing the logit for each respective digit.

The most notable aspect is that the model outputs two items: The first one is the output of the last layer, which represents the logits. They can be used to calculate the probabilities which the network attributes to each class for a sample. The second item is the output of the penultimate layer, which gives us the two-dimensional feature representation for the sample. This can be used to visualize the network classifications, which will be discussed in the next subsection.

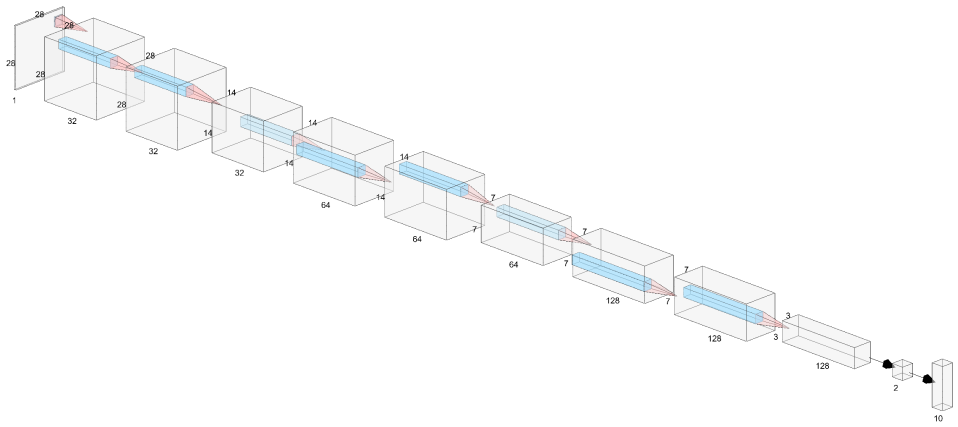


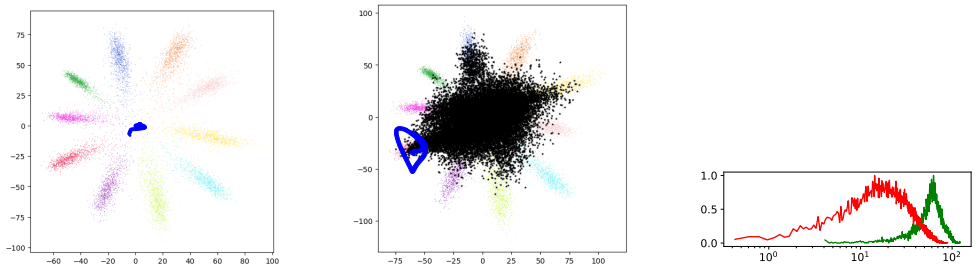
Figure 3.2: LENET++. Model architecture in our experiments. The feature extraction takes place in the penultimate layer, which is two-dimensional. It then gets mapped to a 10-dimensional layer (one dimension for each digit)

3.1.3 Feature Visualization

To visualize the network's latent feature space, we can use its inner representation of the second to last layer for a mapping. With the penultimate layer only having two output dimensions, we can plot this onto a simple two-dimensional plane. Intuitively, we present an image to the network and extract its 2D-representation produced by the penultimate layer. This can now be considered as coordinates for a specific point in a 2D space. We plot this point with a specific color, each corresponding to the label of the image. Finally, we select all images from the test set and repeat this, obtaining a scatter plot as shown in Figure 3.3(a). This allows us to have a very elegant visualization of the features the network attributed to each sample.

Considering each sample on the plot, we can extract information by looking where the samples are plotted. When examining Figure 3.3(a), the relative orientation to the center tells us which class the sample got assigned to, while the magnitude of the features tells us how certain the network is that such a sample belongs to the respective class. In the case of MNIST, a mapping of a typical classifier would resemble a flower with 10 petals, each representing one class. The further away the samples are from the center, the more certain the network is with its classification.

Since our approach takes place in an Open Set scenario, we distinguish samples from the known class $\mathcal{C}$ from samples from the unknown class $\mathcal{U}$. In our plots, samples from $\mathcal{C}$ are always depicted in color, while samples from $\mathcal{U}$ are depicted in black. For example, when using MNIST, every digit has its own color in the feature plot. In combination with the unknowns $\mathcal{U}$, the plot should now optimally still resemble a flower. However, as seen in Figure 3.3(b), when simply adding unknowns to a closed-set model this is not the case. Using this visualization, we can now redefine our objective: The idea is to have all samples of $\mathcal{U}$ as close to the center as possible, while having all samples of $\mathcal{C}$ as far away as possible. More specifically, we want the confidence of the predictions for $\mathcal{U}$ to be as low as possible for all classes, while the confidence for $\mathcal{C}$ is as high as possible for its respective class. This then allows us to draw a clean threshold on the confidence to separate the known from the unknown samples.



(a) MNIST only

(b) MNIST and EMNIST letters

(c) Normalized histogram of feature magnitudes

Figure 3.3: FEATURE VISUALISATION. (a) and (b) depict the feature visualization of the Closed Set scenario with only MNIST in the testing set and the Open Set scenario with added unknowns in the testing set respectively. (c) depicts the histogram of the feature magnitudes of (b).

To further visualize and depict the magnitudes of the features we also use histograms as shown in Figure 3.3(c). To measure the deep feature magnitudes, we take the Euclidean length of the deep representations. These histograms are normalized and plot the deep feature magnitudes of the unknown samples from $\mathcal{U}$ and the known samples from $\mathcal{C}$. The histograms visualize the separation of features very well and give an additional perspective on the problem. Figure 3.3(c) depicts the histogram of Figure 3.3(b). One can see the clear overlap of the known samples $\mathcal{C}$ in green and the unknown samples $\mathcal{U}$ in red.

3.2 Conceptualizations

This section takes a look at the core ideas and concepts used in our approach. To do this, we need to use the extended classification of samples proposed by Dhamija et al. (2018). This can be seen as an addition to the categorization shown in the introduction Chapter 1. Let the samples seen during training belonging to $\mathcal{B}$ be depicted as $\mathcal{D'}_b$ and the ones seen during testing depicted as $\mathcal{D}_b$. Similarly, the samples seen during testing belonging to $\mathcal{A}$ are represented as $\mathcal{D'}_a$. The samples belonging to the known classes of interest $\mathcal{C}$, seen during training and testing are represented as $\mathcal{D'}_c$ and $\mathcal{D}_c$ respectively. Finally, we call the unknown test samples $\mathcal{D}_u = \mathcal{D}_b \cup \mathcal{D}_a$.

3.2.1 Entropic Open Set Loss

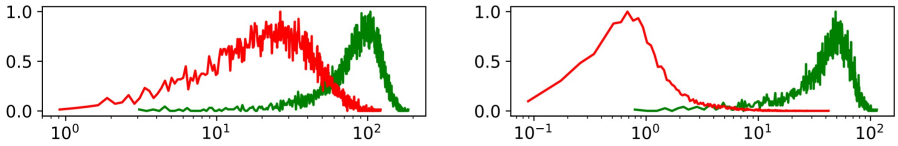
In our experiment we exclusively use the Entropic Open Set (EOS) loss function proposed by Dhamija et al. (2018). The loss function enforces the output probabilities of the unknown samples drawn from $\mathcal{U}$ for every class in $\mathcal{C}$ to be as low as possible. So instead of letting the network make confident predictions about the unknowns, which might be wrong, it tries to minimize this confidence. This corresponds exactly with our desired feature behavior. This section further examines how this is achieved using this loss function.

The Entropic Open Set loss is based on the combination of the softmax activation function and the categorical cross-entropy loss, as the name *entropic* would suggest. The combination of the softmax activation and the cross-entropy loss is also commonly referred as the softmax loss. The key point in EOS is that the softmax loss calculation for samples of $\mathcal{D'}_c$ is left untouched and the loss calculation for samples of $\mathcal{D'}_b$ is introduced. $\mathcal{D'}_b$ are the known unknowns that get seen during training, which in our approach are the adversarial examples. So the known samples from MNIST and the adversrial examples from MNIST get treated differently by the loss function during training. As stated above, one goal of the approach is to get all the unknowns as close to the center of the feature space as possible, since the center represents the point with the least confidence. The intuition here is that we have no information about unknown inputs and therefore want the maximum entropy distribution of uniform probabilities over the known classes. Let S_c be the softmax score, the Entropic Open Set Loss J_E is defined as (Dhamija et al., 2018)

$$J_E(x) = \begin{cases} -\log S_c(x) & \text{if } x \in \mathcal{D'}_c \text{ is from class } c \\ -\frac{1}{C}\sum_{c=1}^{C} \log S_c(x) & \text{if } x \in \mathcal{D'}_b \end{cases} \tag{3.1}$$

where C is the number of classes. If the sample is from a known class, the loss is the negative logarithm of the softmax score, if the sample is from an unknown class, we take the mean of the logarithms of the softmax scores. This way, the loss J_E gets minimized when all softmax scores are equal. More specifically, the loss for the unknown samples achieves its minimum when the softmax scores are exactly $\frac{1}{C}$. We can consider a dataset with 10 classes in $\mathcal{C}$ as an example. In the best case scenario, if a network using EOS would get an unknown sample as an input, it would assign every class in $\mathcal{C}$ a probability of 0.1. Therefore, the Entropic Open Set loss has no need to introduce an additional class to classify unknown samples from $\mathcal{U}$.

During training, the known samples drift further away from the center with higher accuracy, while the unknown samples get pulled towards the middle. So the feature magnitudes of the unknowns becomes smaller and the ones of the knowns grow larger. This is the main reason why we chose this loss function over the softmax loss. The difference is further shown in Figure



(a) softmax loss (b) Entropic Open Set loss

Figure 3.4: FEATURE HISTOGRAMS. Normalized histograms of the feature magnitudes of the softmax loss in (a) and Entropic Open Set loss in (b) constructed by Dhamija et al. (2018). Known samples are depicted in green, unknown samples in red. As these histograms are describing other datasets, they are not comparable to later results and solely serve the purpose of comparing the two loss functions.

3.4, where the feature magnitudes are compared between the softmax and the Entropic Open Set loss. The softmax loss struggles to make a clean separation since it does not make use of $\mathcal{D}'_b$. On the other hand, the Entropic Open Set loss maximizes the entropy for unknown samples while minimizing it for the known samples. Thus we can clearly separate the knowns from the unknowns.

3.2.2 Filtering

Next to the basic adversarial training, we also experiment with several extensions to further increase performance and have a greater variety in our experiments. These variations mainly focus on filtering the adversarial examples that would be added to the training set. More specifically, we filter the samples in $\mathcal{B}$ to try and achieve better performance. This results in introducing the network to the adversarial examples in a more sophisticated way instead of presenting them all at once at the start of training.

The first filter method generates adversarial examples of those samples that got correctly classified previously during training. Therefore, the number of adversarial examples in the training set is low at first, such that the network has a chance to initially focus on classifying the known samples before getting confronted with unknown classes. This way, the network only starts to encounter samples from $\mathcal{B}$ if it starts classifying samples from $\mathcal{C}$ correctly. This also allows having progressively more samples from $\mathcal{B}$ in the training set while training.

The second filter method only allows the generation of adversarial examples from samples that are correctly classified over a specified confidence. More specifically, the softmax probability of the correct class needs to be over a certain threshold. This threshold can of course be defined dynamically. For example, with a threshold of 0.7, only samples that are correctly classified with a softmax probability of 70% or greater would qualify for the adversarial training. Using these two filter methods we further improve the ability of the network to reject unknown inputs in our experiments which will be discussed in Section 4.3 in more detail.

3.3 Evaluation Metrics

The main challenge regarding evaluation metrics is to capture the ability to reject unknown input. For this problem we use three different metrics: A confidence metric, an Area-Under-the-Curve metric, and an OSCR-curve, which all are explained in this section. Additionally, we also track other usual metrics like accuracy of the known samples and loss values.

3.3.1 Confidence

The first metric used in our experiments is confidence, which measures how confident the network is in its individual predictions. Since the confidence for knowns and unknowns are two different things, the calculation is again different for $\mathcal{U}$ and $\mathcal{C}$. The confidence for the known class $\mathcal{C}$ is the softmax probability of the class the sample belongs to. For example, if the network gets the digit *2* as an input, it assigns a probability to every known class. We now only look at the probability the network assigned to the class *2* and take this for the confidence. The confidence for the unknowns can not be calculated in the same way, since we do not have an explicit "garbage" class for unknowns with a respective probability for U. Being confident in rejecting an input means that the probabilities should be low for every class. The confidence is defined as

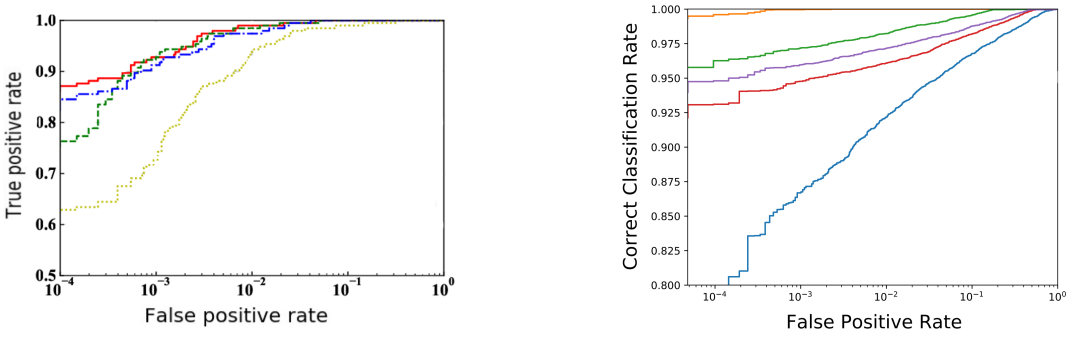
$$conf(x) = \begin{cases} S_{c^*}(x) & \text{if } x \in \mathcal{C} \\ 1 + \frac{1}{C} - \max_c(S_c(x)) & \text{if } x \in \mathcal{U} \end{cases} \tag{3.2}$$

Here, $S_c(x)$ are again the softmax probabilities the network assigns to each class, C is the number of classes and c^* the true label of the respective sample. $\frac{1}{C}$ describes the probability that gets assigned to each class when the network is truly indecisive. In the case of MNIST it would result in 0.1, since there are 10 classes and each class gets assigned a probability of 10%. So, if the maximum probability of the classes is equal to $\frac{1}{C}$, the network rejects the sample with 100% confidence.

3.3.2 Area Under the Curve

Since this is a classification problem, it is also convenient to calculate an Area Under the Curve (AUC). Our AUC however gets computed slightly different than the standard AUC-ROC curve. We now briefly introduce the AUC-ROC curve and then show the adjustments we made for our own metric. AUC-ROC is one of the most important evaluation metrics for checking the capability of binary classification models and a widely used metric for tasks like these, since it measures the performance of classifiers in various settings. Receiver operating characteristic (ROC) is a graphical plot that measures the performance of a binary classifier. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR). The TPR in our example would be the knowns $\mathcal{C}$ that are also correctly identified as knowns. The FPR on the other hand would be the unknowns $\mathcal{U}$ that are incorrectly identified as known. These two values are then plotted at various threshold settings for the probabilities.

For plotting the ROC curve one needs the labels of all the samples and network predictions respectively. However, in our approach, since we do not not have an explicit class for the unknowns $\mathcal{U}$, the network itself does not explicitly assign a probability to a sample belonging to $\mathcal{U}$. If our model had an additional "garbage" class for unknowns, when encountering a sample that it classifies as unknown with maximum confidence, it can simply set the probabilities of all other classes to 0. However, in our model, a classification as unknown with maximum confidence would result



(a) Receiver Operating Characteristic curve

(b) Open-Set Classificatioin Rate curve

Figure 3.5: ROC AND OSCR PLOTS. Subfigure (a) depicts different ROC curves while Subfigure (b) depicts different OSCR curves. For both graphs the Area Under the Curve can be calculated additionally.

in every class having a probability of $\frac{1}{C}$. Since we do not have an explicit probability for unknown prediction, our curve does not exactly correspond with the ROC. For the unknowns $\mathcal{U}$ we use the highest probability score of all the classes and use this to calculate the TPR and FPR. This then yields us a new AUC metric, which is a very useful indicator for when to stop the training process.

Using the ROC graph, one can now calculate the AUC of it. This represents the degree of separability with a single value. It estimates the model's capability to distinguish between classes. In our experiments, we use it to measure how well it is able to separate the knowns from the unknowns. The higher the AUC, the better the model is at predicting if a sample belongs to the known class $\mathcal{C}$ or the unknown class $\mathcal{U}$. An excellent model has an AUC near 1, which means it has the best measure of separability. An AUC of 0.5 would mean that the model has no class separation capability whatsoever, which would be the same as randomly guessing what class the samples belong to.

3.3.3 Open-Set Classification Rate

Finally, we also use the Open-Set Classification Rate (OSCR) curve proposed by Dhamija et al. (2018). It works in a similar manner as the previously defined ROC, however the OSCR plots the Correct Classification Rate (CCR) against the FPR. If we let τ be the threshold, we can calculate the two values as follows (Dhamija et al., 2018)

$$FPR(\tau) = \frac{|\{x \mid x \in \mathcal{D}_a \wedge \max_c P(c \mid x) \geq \tau\}|}{|\mathcal{D}_a|} \tag{3.3}$$

$$CCR(\tau) = \frac{|\{x \mid x \in \mathcal{D}_c \wedge \arg\max_c P(c \mid x) = c^* \wedge P(c^* \mid x) \geq \tau\}|}{|\mathcal{D}_c|} \tag{3.4}$$

Here, the samples are split into known classes $\mathcal{D}_c$ and unknown classes $\mathcal{D}_a$. The CCR for $\mathcal{D}_c$ is the fraction of the samples where the correct class c^* has maximum probability and is greater than the threshold τ. The FPR gets calculated from unknown samples from $\mathcal{D}_u$ of which the prediction probability of any class is higher than the threshold τ.

Finding a metric to describe the performance in such a scenario is rather difficult and since there is no generally used metric, we always also present the feature plots to further support our measurements and claims. Furthermore, the previously seen histograms will be used, since they yield additional visualization of the feature magnitudes of both $\mathcal{C}$ and $\mathcal{U}$.

Chapter 4

Experiments

This chapter discusses our experiments and show their results. We clearly demonstrate the effect of adversarial training in the Open Set Recognition scenario using the previously described adversarial attacks. All our main experiments use letters from EMINST or images of clothing from the FashionMNIST dataset for the unknown class $\mathcal{A}$ in the testing set. Most evaluations use handwritten letters from the EMNIST dataset and all evaluations are done on the testing set.

4.1 Implementation Details

Here we list a detailed description of the implementation to give further insight into our experiments. The whole code is written in Python using the Pytorch framework (Paszke et al., 2019). The model architecture is implemented just as described in Section 3.1.2, with the batch norm layers having *track_running_stats* set to *False*. This minor detail will further be part of the discussion in Chapter 5 and is therefore left unexplained for now. The network is trained for 100 epochs every attempt, with a learning rate of 0.01 and a batch size of 128. We use stochastic gradient descent with a momentum of 0.9 as our optimizer. The loss function in use is exclusively the Entropic Open Set loss function. During evaluation, we check for the AUC and confidence metrics and take these as a guideline for network performance. We additionally check and compare the OSCR curves. To further reduce variance in our results, we run every attempt with three different seeds, of which we then take the average of the metrics as the respective results.

The generation of the adversarial samples is done on the fly during training. For some of the attacks we use the Advertorch framework (Ding et al., 2019). In other words, for every sample that is passed through the training loop, an adversarial example is generated and added to the training set. After generating the sample, it also gets passed through the training loop with the label -1, which represents samples from the background class $\mathcal{B}$. For LOTS, we have chosen the target t for a sample to be the deep feature representation of another sample that does not belong to the same class. There are several options to choose from when generating adversarial examples. The first one would be determining which attack method to use. Next to the adversarial attacks mentioned in Section 2.3, we have additionally implemented a method that adds random noise in a specified interval to the image. This allows for a clean comparison in performance and underlines the benefits of adversarial training. Additionally, one can specify the desired filter methods described in Section 3.2.2 and its respective parameters. Finally, the last important parameter to tweak for adversarial attacks is ϵ. This is also the parameter that has been the focus of our hyperparameter search and will be discussed in greater detail in the next section.

4.2 Epsilon Search

Confidence				
	FGSM	PGD	CnW	LOTS
$\epsilon = 0.1$	0.5651	0.5876	<u>0.7070</u>	0.5627
$\epsilon = 0.2$	0.5575	0.5620	0.7051	0.5770
$\epsilon = 0.3$	<u>0.5672</u>	0.5348	0.7052	0.5727
$\epsilon = 0.4$	0.5662	0.5468	0.7052	<u>0.6102</u>
$\epsilon = 0.5$	0.5574	<u>0.6059</u>	0.7054	0.6044

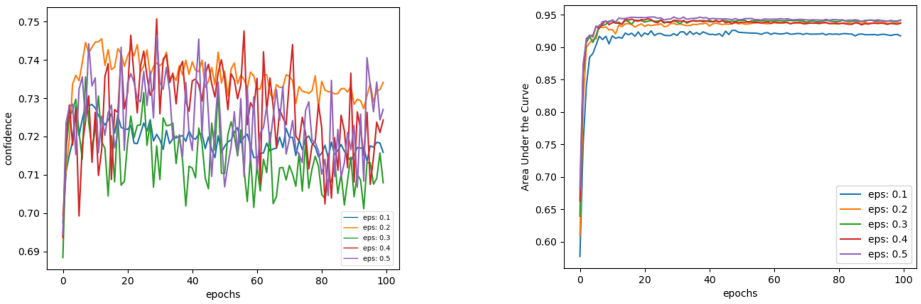
(a) Confidence table

Area Under the Curve				
	FGSM	PGD	CnW	LOTS
$\epsilon = 0.1$	<u>0.9537</u>	0.8390	0.8977	0.9271
$\epsilon = 0.2$	0.9527	0.9595	0.8999	0.9450
$\epsilon = 0.3$	0.9536	0.9576	0.9036	0.9453
$\epsilon = 0.4$	0.9535	0.9580	0.9026	<u>0.9459</u>
$\epsilon = 0.5$	0.9527	<u>0.9601</u>	<u>0.9040</u>	0.9447

(b) Area Under the Curve table

Table 4.1: METRIC EVALUATION TABLES. Table (a) shows values for our confidence metric of different attacks evalutated for different values for ϵ after 100 epochs. Table (b) shows the AUC values respectively. For each row, the best value is underlined.

In our experiments we try to fine-tune the attacks in a way that would maximize our metrics and therefore provide the best results and performance. Additionally, to compare the different adversarial attacks, we also experiment with different settings for the same attack. The one major hyperparameter that is experimented with is ϵ. As previously stated, when looking at the adversarial attacks, ϵ can be seen as the maximum allowed perturbation of the pixel values in the images. Therefore, it also represents the radius of the ℓ_p-ball in the distribution space. In iterative attacks like Projected Gradient Descent, an additional parameter ϵ_{iter} can be tweaked. This hyperparameter can be seen as the step size when doing the update steps. However, since iterative attacks take considerably more time when doing multiple iterations, we have decided to do only one iteration and set $\epsilon = \epsilon_{iter}$. This way we also have a uniform way to compare the non-iterative with the iterative attacks. To find the best epsilon, we set an initial educated guess for the interval $[0.1, 0.5]$. Having a fixed range of possible values for ϵ, we then track all of our metrics over all epochs during training. Examples for the variation over time can be seen in Figure 4.1.



(a) Confidence graphs

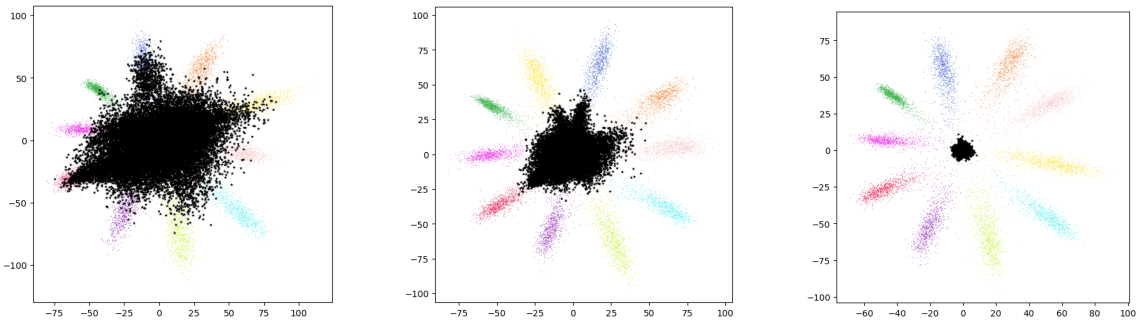
(b) AUC graphs

Figure 4.1: EPSILON GRAPHS. Subfigures (a) and (b) show the progression of the confidence and AUC metrics respectively. The metrics are computed every epoch for a total of 100 epochs on the testing set.

One main objective of the experiments is to find the best values for ϵ for every attack and then compare the attacks with each other. The tables 4.1(a) and 4.1(b) show the different adversarial attacks when evaluated on confidence and AUC metrics respectively. All the attacks show significant improvements over training without adversarial examples.

4.3 Adversarial Training with Filter Methods

This section shows the results when using our filter methods from Section 3.2.2 additional to the basic adversarial training done in Section 4.2.

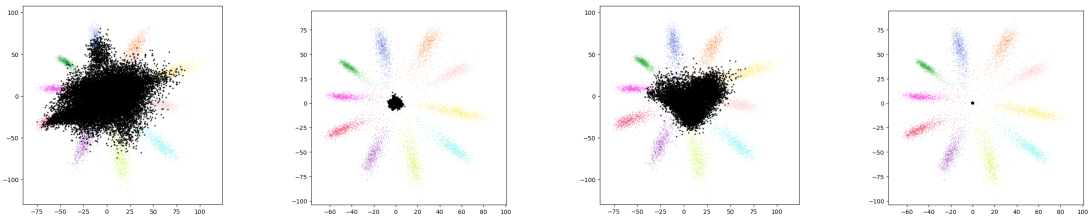


(a) No adversarial training (b) Basic adversarial training (c) Adversarial training using filters

Figure 4.2: FILTERING. Here we can see the step wise improvements of our approach. Subfigure (a) shows the features when the model is trained without adversarial training. Subfigure (b) shows the results with basic adversarial training and (c) shows the adversarial training using filter methods. The adversarial examples are generated using LOTS with $\epsilon = 0.4$ and the filter threshold was 0.9.

Having established solid baselines with basic adversarial training depicted in Figure 4.1, we can additionally extend our experiments using the filter methods proposed in Section 3.2.2. Using these methods, we achieve even greater improvements depicted in Table 4.2. Furthermore, Figure 4.2 further demonstrates the effectiveness of our approach. Looking at the pictures, one can see that using the filter methods yet again results in major improvements over the previously shown adversarial training. The corresponding OSCR plots can be found in Figure 4.4.

As we see the results of our initial experiments having EMNIST letters as unknowns in the testing set, we can now proceed to use other datasets for $\mathcal{A}$. Figure 4.3 depicts the feature plots when using the FashionMNIST dataset as our unknown unknowns. FashionMNIST includes grey-scale pictures of clothing and is therefore expected to produce different results. This would also allow comparisons of datasets for $\mathcal{A}$ as seen in Figure 4.3.



(a) EMNIST letters without adversarial training

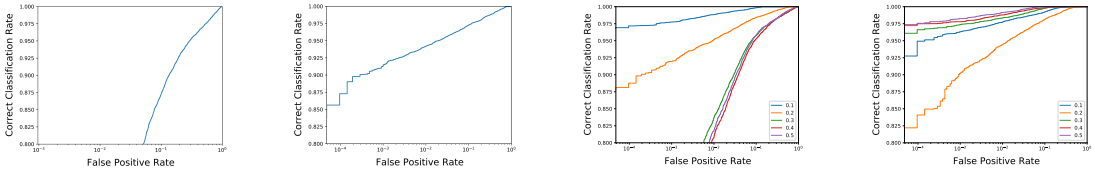
(b) EMNIST letters with adversarial training

(c) FashionMNIST without adversarial training

(d) FashionMNIST with adversarial training

Figure 4.3: RESULTS. Here we can see the difference in results when using our approach. (a) and (c) show the features when the model is only trained with MNIST with no adversarial training. The black dots represent the samples in the testing set of EMNIST letters and FashionMNIST respectively. (b) and (d) show the results in the same setting but using our approach with adversarial training using LOTS with $\epsilon = 0.4$ and our second filter method with a threshold of 0.9.

As one can see, the model performs even better on the FashionMNIST dataset than on EMNIST letters. This is understood to be due to the reason that pictures of digits and clothing are perceivably very different, making it easier to distinguish between the two. This claim is additionally supported when comparing results without adversarial training. Even when the network is only trained to classify digits, it is better at rejecting FashionMNIST than EMNIST. Therefore, we have decided that letters would pose the larger challenge and would have more room for improvement when training with adversarial examples.



(a) OSCR no adversarial training

(b) OSCR noise training

(c) OSCR basic adversarial training

(d) OSCR adversarial training using filters

Figure 4.4: OSCR RESULTS. Here, we can see the OSCR curves of the different training stages. (a) shows the plot when the model is trained without adversarial training while (b) shows the curve when using images with randomly added noise as the background class. (c) shows the results with basic adversarial training and (d) shows the adversarial training using filter methods. The adversarial examples are generated using LOTS with $\epsilon = 0.4$ and the filter threshold was 0.9.

The immense boost in performance with the help of our filter methods can further be seen in Table 4.2. Here, we compare the best achieved values from our experiments using our final approach. Next to the basic adversarial training done in Section 4.2, we additionally filter our samples using the methods described in Section 3.2.2. Furthermore, we compare these results to other methods not using any adversarial training. The first method uses samples for $\mathcal{B}$ that are slightly perturbed with noise and labeled as unknown. This should show that the results achieved with adversarial perturbation cannot be achieved using random perturbation of the original samples. Lastly, we also train our model without any background class $\mathcal{B}$ to compare the performance.

	FGSM	PGD	CnW	LOTS	NOISE	WITHOUT $\mathcal{B}$
Confidence	0.6383	0.6960	0.7962	0.7246	0.6175	0.4905
AUC	0.9163	0.9697	0.9206	0.9712	0.8984	0.8901
Accuracy	0.9803	0.9902	0.9873	0.9825	0.9743	0.9645

Table 4.2: FINAL RESULTS. This table shows the best achieved results of our adversarial training done with different adversarial attacks using our filter methods. It further compares these methods with approaches that did not use adversarial training or a background class.

This shows the performance of a network that is trained in a Closed Set Recognition scenario and encounters unknowns during testing.

When comparing Table 4.2 and Table 4.1, one can clearly see the additional improvements our filter methods achieves. While the AUC increases slightly for all attacks, the confidence has a larger overall growth. The PGD, CnW and LOTS attack show comparable performances, even tough the CnW attack achieves the highest confidence values by a large margin and LOTS performs best in the AUC metric. Furthermore, there is an immense difference in performance when comparing the approach to the method that has no known unknowns or uses a background class with randomly perturbed images. The accuracy of the known samples $\mathcal{C}$ also remains high and consistent for all the different attacks used. As one can see, our approach shows promising results and a remarkable increase in the capability of the network to reject unknown input.

Chapter 5

Discussion

The main topic in our discussion focuses on the batch norm in the Open Set Recognition scenario. This is also related to the reason why track_running_stats is set to *False* for the batch norm layers in our implementation. This causes the batch norm layers to not keep running estimates of mean and standard deviation, and instead use batch statistics during evaluation time as well. This minor change results in our approach not being fully reproducible, but also significantly boosts the performance. The reason for this is the Open Set Recognition scenario, which will now be discussed in more detail. When using batch norm in the closed set scenario, where the training set and the testing set are drawn from the same distribution of samples, batch norm keeps a running estimate of the mean and standard deviation of the batches during training. The goal is to approximate the mean and standard deviation of the whole distribution the training set was drawn from. These values constantly get updated during training and when switching to the testing set in evaluation mode, they get frozen. Now, instead of taking the values of the individual batches as done in training, the previously computed values are used for every batch in the testing set. This works well in the closed set scenario since the training and the testing set came from the same or at least similar sample distributions.

However, this is not the case when doing Open Set Recognition. As previously explained, in the Open Set Recognition scenario, the training set and the testing set can be very different and can come from completely different distributions. Therefore, the values for mean and standard deviation for the testing and the training set can be completely incompatible. So, when using the values calculated during training while testing, the batch norm skews the values which leads to bad results. This occurs only when doing Open Set Recognition. To fix this problem, we attempted to train the network without batch norm, which did not deliver equivalent results. We assume that the batch norm is particularly required due to the bottleneck architecture in our network. Other network topologies might no be as dependant on the batch norm. We also attempted to use Instance norm (Ulyanov et al., 2016) instead of batch norm, which also generated a worse outcome. For a quick solution, we decided to use the batch statistics of every batch individually during testing, just as we did in training. This results in very clean separations. Other possible alternatives would require to calculate batch statistics for every layer, which would all go beyond the scope of this thesis, but could lead to the results being reproducible.

Despite our experiments achieving good results, they have their limitations. We only experimented with fixed epsilon values and it is possible that iterative attacks like PGD would have outperformed other attacks if we allowed multiple iterations. Additionally, the experiments were mostly done on one single dataset with the same network topology. Changing these architectures or using other training methods might change the results. To achieve generality, these factors would need to be taken into consideration.

Chapter 6

Conclusion

Our experimental evidence supports the theory that sample similarity in $\mathcal{C}$ and $\mathcal{B}$ can significantly improve the ability for networks to reject unknown input in the Open Set Recognition scenario and therefore adversarial training represents a valid approach to do so. In combination with the Entropic Open Set loss, we are able to improve the ability to reject unknown inputs by a large margin. In our experiments we have also shown the difference in performance of several state of the art adversarial attacks, which all were previously fine-tuned on an individual level to deliver the best results. Furthermore, we have extended our approach by testing on different unknown classes, which often lead to even better results. Our experiments have shown that adversarial training is highly effective in rejecting unknown input in the Open Set Recognition scenario.

This also suggests that more research should be dedicated to this field in the future. Future work should mostly focus on exploring alternatives for batch norm layers as mentioned in the discussion. Furthermore it should focus on expanding the generality by using different network topologies and datasets. As the results are comparably good for most adversarial attacks in our setting, future work might also extend to a wider range of epsilons or even have decreasing epsilons during training. Additionally, it shall perform higher levels of filtering and data augmentation in both $\mathcal{C}$ and $\mathcal{B}$, to possibly achieve even better results.

List of Figures

List of Tables

Abbreviations

AUC	Area Under the Curve
CCR	Correct Classification Rate
CnW	Carlini and Wagner
CSR	Closed Set Recognition
EOS	Entropic Open Set
FGMS	Fast Gradient Sign Method
FPR	False Positive Rate
LOTS	Layerwise Origin-Target Synthesis
OSCR	Open-Set Classification Rate
OSR	Open Set Recognition
PGD	Projected Gradient Descent
ROC	Receiver Operating Characteristic
TPR	True Positive Rate

Bibliography

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.

Boult, T. E., Cruz, S., Dhamija, A. R., Günther, M., Henrydoss, J., and Scheirer, W. J. (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.

Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373.

Dhamija, A. R., Günther, M., and Boult, T. (2018). Reducing network agnostophobia. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Ding, G. W., Wang, L., and Jin, X. (2019). advertorch v0.1: An adversarial robustness toolbox based on pytorch. *CoRR*, abs/1902.07623.

Geng, C., Huang, S.-J., and Chen, S. (2020). Recent advances in open set recognition: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

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.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

Knagg, O. (2019). Know your enemy. https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3. [Online; accessed 19-July-2021].

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.

Linden and Kindermann (1989). Inversion of multilayer nets. In *International 1989 Joint Conference on Neural Networks*, pages 425–430 vol.2.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Matan, O., Kiang, R., Stenard, C., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L., and Le Cun, Y. (1990). Handwritten character recognition using neural network architectures. In *4th USPS Advanced Technology Conference*, volume 2, pages 1003–1011.

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 Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Rozsa, A., Günther, M., and Boult, T. E. (2017). Lots about attacking deep features. In *2017 IEEE International Joint Conference on Biometrics (IJCB)*, pages 168–176.

Rudd, E. M., Jain, L. P., Scheirer, W. J., and Boult, T. E. (2017). The extreme value machine. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):762–768.

Scheirer, W. J., de Rezende Rocha, A., Sapkota, A., and Boult, T. E. (2013). Toward open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(7):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.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014). Intriguing properties of neural networks. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. S. (2016). Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022.

Wang, Y., Ma, X., Bailey, J., Yi, J., Zhou, B., and Gu, Q. (2019). On the convergence and robustness of adversarial training. In *ICML*, volume 1, page 2.

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*, pages 499–515. Springer.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747.

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