Master Thesis
in Computer Science

Robust Adversarial OOD Detection With Vision Transformer

Abstract

Vision Transformers (ViTs) are steadily conquering every area of computer vision, including image classification, with their revolutionary Attention mechanism. Even within image classification, they outperform deep Convolutional Neural Networks (CNNs) in out-of-distribution (OOD) detection tasks. However, like deep CNNs, they are prone to misclassify OOD images and samples that have been modified by adversarial attacks.

This thesis shows that ViTs, when used as separate OOD detector modules, can robustly detect adversarially altered OOD data. They only need to be trained on in-distribution (ID) and OOD samples, including samples from both distributions that have been perturbed by an adversarial attack, without making any adjustments to the model itself. It is shown that a few training epochs are sufficient to surpass the OOD detection capabilities of deep CNNs.

The experiments were performed on Cifar-10, Cifar-100 and SVHN datasets using accuracy and the Area Under the Receiver-Operator Characteristic Curve (AUROC) as evaluation metrics. The images used for training are perturbed by a Projected Gradient Descent attack. ViTs are able to achieve state-of-the-art performance on datasets with similar visual features, and on datasets with different visual features, they were able to outperform deep CNNs with AUROC values of over 99.9% with only three training epochs. This suggests that they focus particularly on visual features of objects.

For the two noise levels tested, $\varepsilon = 0.01$ and $\varepsilon = \frac{8}{255}$, they showed no decrease in any of the metrics, suggesting that ViTs can also overcome larger amounts of noise compared to deep CNNs. Increasing amounts of noise only caused an increasing disruption of their Attention towards the object to be classified within the image.

Contents

Chapter 1

Introduction

As the utilization of machine learning is steadily increasing in our everyday life, we are getting comfortable relying on its results. We rely on these applications, when they are working in a domain they have been specialized on, because they provide results quicker than humans could and in certain subjects even with better outputs. One famous example is a deep convolutional neural network surpassing human level performance in object detection [30]. An even more popular example in the last years has been AlphaGo, a deep neural network of DeepMind [86], beating the European champion in the game of Go. Go is notorious for being computationally highly complex and exceeding the capabilities of computers that use complete Monte Carlo rollouts.

The newest accomplishments in the field of artificial intelligence include Dall-E 2 [75] and ChatGPT [83]. Dall-E 2 is able to create images with interacting content in different art styles from text. The results are created within a few seconds and with impressive attention to detail. ChatGPT also takes text as input and answers in the form of a chat message with the combined knowledge of a majority of the content from the internet. Especially impressive aspects about ChatGPT are its ability to keep track of questions and answers it has given. This context is always included in upcoming responses. It is also able to write code, improve existing code or simulate a terminal. Remarkably, ChatGPT is able to fulfill not only one, but multiple tasks with outstanding precision. Even though both models make errors from time to time, their results are almost indistinguishable from images and responses created by humans.

Conventional machine learning models are not able to generalize this well unless specifically prepared for it. As soon as the environment they operate in changes and the input marginally deviates from the data the application has been trained on, its results can drastically change to incorrect outcomes. Pre-trained neural networks are prone to return overconfident probabilities

for samples they have not been trained for [33]. This even includes small perturbations to samples that effectively belong to the in-distribution [28].

To be applicable in a safety relevant context, models should either be able to correctly classify the input, despite it being perturbed, or at least communicate its uncertainty and overcome the overconfidence. One of such safety relevant fields is autonomous driving, where cars try to navigate through streets with information from cameras and sensors without human intervention. Errors in this field could have an enormous impact, as human lives are involved. This is also the reason why the EU, in the form of UNECE (United Nations Economic Commission for Europe), hesitates to authorize fully autonomous driving [96]. In this context, natural perturbations could occur in the form of fog, rain or snow, which should not prevent autonomous cars from conducting their main task [105, 19].

On the other hand, models should also be resilient to perturbed input in the form of intentional attacks. As an example, certain trading activities have been automated in recent years and entrusted to machine learning based algorithms. These algorithms pose a vulnerability for deliberately engineered attacks, where the input is tailored to produce a precise outcome [26]. Consequences of these attacks can have monetary impact in the foreground, but also lead to severe economical and political consequences [62].

It is therefore desirable to make machine learning models robust against all kinds of input they have not specifically been trained on. Robust in this context means either detecting the input sample correctly or at least flagging it as something unfamiliar.

Chapter 2

Computer Vision as a Section of Deep Learning

The general term *computer vision* (CV) is used to describe all tasks of a computer trying to understand and interpret an environment with visual input, like images or videos, which are ultimately a sequence of images. To make the images machine-readable, they get stripped down to their individual pixels. In the case of colored images, the pixels get further decomposed in three different channels corresponding to their RGB (red green blue) values, and then turned into numerical values. These numbers are then further processed by a computer, where their input position matches the location of the original pixel on the image. By turning the pixels into numbers, the machine can interpret them and perform mathematical calculations on them, which are essential for every further task.

CV is dominated by machine learning methods, which provide the best results in all tasks. The most common problems in CV are image classification, object detection, object tracking in videos, semantic segmentation and instance segmentation (see Figure 2.1). Even though these tasks have a certain overlap between each other, they are used for different, specific scenarios. Object detection classifies the identified objects on an image (like in classification), but its task is to additionally locate multiple instances within the image and not only identify the general class of content. Even though object detection seems more sophisticated than classification, as it adds features on top of it, classification also has its fields of applicability, because it is more reliable on smaller datasets [51].

For a long time, Convolutional Neural Networks (CNNs) of increasing depths have been the prevalent architecture for most of these tasks. Nowadays, they are one by one being replaced by Transformers. Transformers are able to associate distant input elements with each other and are able to focus on

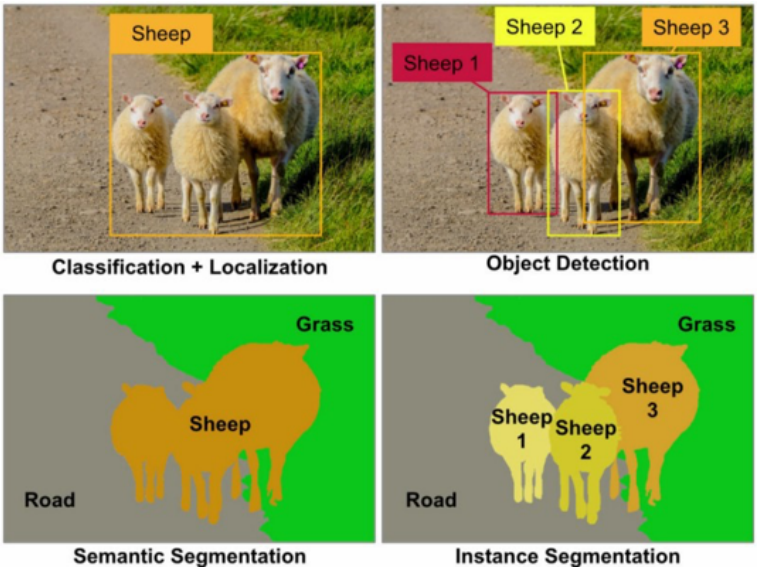


Figure 2.1: Visual example of classification, object detection, semantic segmentation and instance segmentation [98].

the important features, leading to improved accuracy and efficiency regarding computing power [50, 24]. These advantages have made Transformers state of the art in CV tasks with the Vision Transformer (ViT) presented by Dosovitskiy et al. [24]. In natural language processing, Transformers have taken the spotlight with the BERT models by Devlin et al. [20], GPT-3 by Brown et al. [13] and ultimately ChatGPT leveraging GPT-3 [83], which are both able to produce text indistinguishable from a human author. Even in protein structure prediction [42] and protein structure modelling [36] do Transformer models stand out with the best results.

The following chapters are restricted to CV, due to the bound scope of this thesis. For more information about Transformers in natural language processing it is recommended to read the initial papers of the BERT model [20], the GPT-3 [13] and ChatGPT model [83] or the blog post by Nedelchev [67], where he quickly summarizes the BERT and GPT-3 models. After a short overview of the previous state of the art convolutional neural networks, an introduction to the operation mode of Transformers is made with an explanation of their specific Attention property. The section about Attention is followed by an illustration of the changes to the basic Transformer architecture to obtain the Vision Transformer, as the current state of the art in CV.

2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) were state of the art for many applications in CV or even in natural language processing until the emergence of Transformers. In the scope of this thesis, only the basic functionality of CNNs is outlined, to highlight the differences to Vision Transformers (ViTs). Transformers in general and the specific Vision Transformers for CV related tasks are further explained in Section 2.3 and Section 2.4. Since the mathematical beginnings in the field of neural networks in 1943 by McCulloch and Pitts [41], the first steps in the form of single-layer perceptron networks were made in 1958 by Rosenblatt [79] and advancements to multi-layer networks in 1986 by Hinton et al. [81]. The predecessor of CNNs was introduced in 1980 and was named *Necognitron*. It was the first model to include pooling, feature extraction and convolution into a neural network, while being heavily inspired by the visual nervous system [25]. The actual terminology *Convolutional Neural Network* was introduced in 1998 by LeCun et al. [52], who tried to solve the computer recognition task of handwritten characters.

CNNs consist of at least one convolutional layer, which makes use of the operating principle of biological visual perception. Every convolutional layer contains a kernel that only looks at a small section of the image at a time. It computes an output for this section by multiplying the image values with the corresponding kernel values. The result is then passed on to the next layer and the kernel slides over the image pixel by pixel, repeating the process. These convolutional layers can also be placed in sequence to condense the informational value even further, as shown in Figure 2.2. The kernels are (similar to the receptors) reacting to distinct visual features and the activation functions act in place for neural functions to only let electrical signals above a given threshold pass on to the next layer [39]. With *local connections*, neurons are only connected to specific neurons in the next layer and not to all of them, reducing parameters and accelerating convergence by only focusing on the relationship of neighboring groups of pixels. Parameters, also called weights, are the connections between two artificial neurons in two different layers. To further reduce the number of parameters, *weight sharing* is used, combining the connections of neighboring neurons in order to make them share the same weights. Decreasing the overall dimensionality can be achieved by a pooling layer, commonly either average or maximum pooling, by combining the information of multiple inputs and cutting down on the residual parameters [56].

Improvements included local connections, weight sharing, down-sampling and dimensionality reduction. For example, in 2012 Krizhevsky et al. [49] were able to outperform the next best contestant participating in a challenge with their *AlexNet* with a top-5 error rate of 15.3% versus 26.2%. They created a

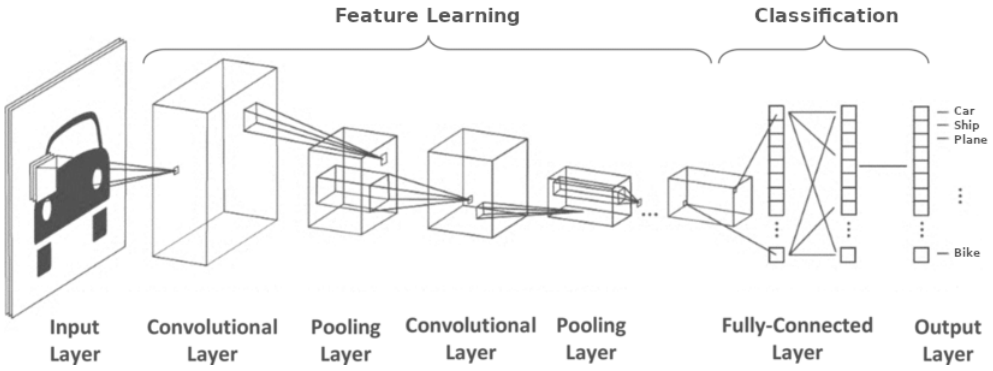


Figure 2.2: General architecture of a CNN with a sequence of convolutional and pooling layers in combination. Fully connected layers pose the end and finally classify the given input [32].

deep convolutional neural network with 60 million parameters and trained it on the LSVRC-2010 ImageNet [82], consisting of 1.2 million high-resolution images of 1000 different classes. With their work Krizhevsky et al. set a milestone not only because of the challenge they won, but also because they introduced dropout as a regularization method, optimization algorithms and the use of novel activation functions [49]. Since Krizhevsky et al. various gradually improved architectures have emerged, like *VGGNet* in 2014 by Simonyan and Zisserman [87] who created an even deeper network than *AlexNet*. *GoogleLeNet* was also presented in 2014 by Szegedy et al. [90], going even deeper than *VGGNet*, but without fully connected layers and a generally reduced parameter count resulting in less memory consumption and faster performance. The last milestone of CNNs represents *ResNet* in 2015 by He et al. [31], who introduced *residual blocks* which are connected through skip connections with each other (see Figure 2.3). Many of the concepts incorporated in these networks also play an important role in the architecture of Transformers.

2.2 Attention

Before moving on to Transformers, it is helpful to understand the concept of Attention. Every input token of a Transformer has some relation to every other token. Attention describes this relation between all tokens. In the example of an image as input, every pixel or group of pixels is a token. CNNs tried to capture this relation step by step with the convolutional layers. Attention allows for this process to happen in one step. Every pixel of an image either has some contextual relationship to some other pixel or not. If they share

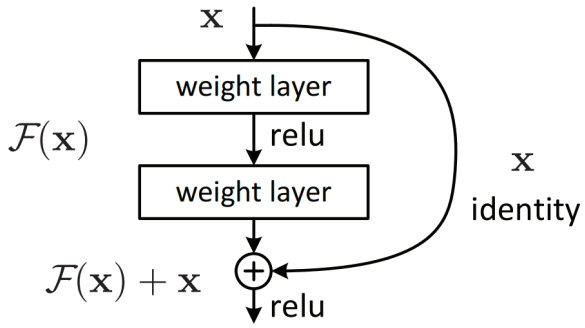


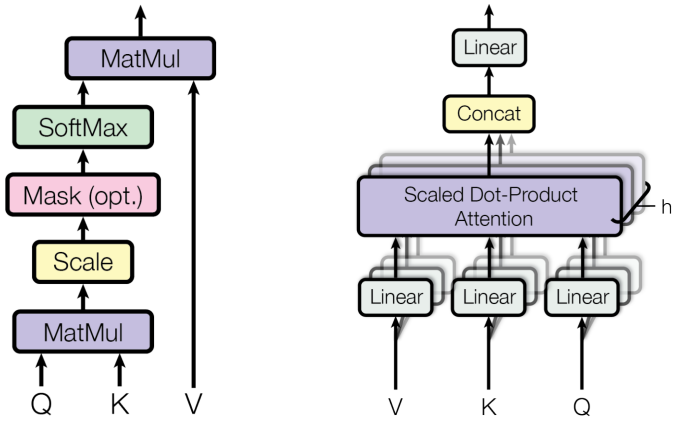
Figure 2.3: Skip connection around a residual block in the ResNet architecture of He et al. [31].

some context, for example both being part of the main object, the Attention mechanism is able to recognize this relation.

Before the relation among the tokens can be calculated, the input is inserted into three different, fully connected layers to create query (Q), key (K) and value (V) vectors. The Q and K are dot product matrix multiplied, returning a score matrix, which indicates how strongly two tokens of the input correlate to each other. The higher the value, the stronger the correlation. All the scores are then scaled down by the dimension of the matrix out of K and Q (multiplied by $\frac{1}{\sqrt{d_k}}$) to avoid exploding values. Afterward, their softmax is calculated returning probability values between 0 and 1, emphasizing their relative weighting and ultimately strengthening the model's confidence in which tokens to focus on. To receive an output vector, these Attention weights have to be multiplied by the value vector. Wherever a high Attention weight is multiplied by an entry of the value vector, the input token is considered more important and vice versa. All of these steps can be summarized into the following equation.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

This Attention process is not only executed once, but h times in parallel. To reach the so-called Multi-headed Attention, the query, key and value vectors are split into h vectors. Every split vector goes through the process described above individually and is called an Attention *head*. All the heads get concatenated and then passed on to the next layer. Splitting the vectors h times allows them to learn different qualities and provide the encoder with more information [97]. What an encoder is and how it works is further explained in the upcoming section. The entire Multi-headed Attention process is



(a) Single strand of Scaled Dot-Product Attention with query (Q), key (K) and value (V) as input.

(b) All h single strands combined to form the final Multi-headed Attention outcome.

Figure 2.4: Attention in the Transformer architecture [97]. Figure 2.4(a) represents one Attention strand and Figure 2.4(b) the combination of all h strands.

visualized in Figure 2.4, where Figure 2.4(a) outlines the procedure of a single Attention head and Figure 2.4(b) shows the combination of all heads.

The described procedure allows Transformers to have an Attention window of infinite size, unlike Recurrent Neural Networks (RNNs) which have a comparatively small window. Even though Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) networks provide bigger window sizes, they are still not infinitely big and therefore inferior to Transformers [72].

2.3 Transformer

The Transformer architecture can be summarized as an encoder, compressing the input sequence into a continuous representation containing Attention information, and a decoder, decompressing this representation and turning it into the desired output. Both encoder and decoder use the Attention mechanism explained in Section 2.2.

Firstly, the input is split up into parts, called tokens. In the example of text, one token would be an individual word and for images, this could be one single pixel or a group of pixels. For the sake of keeping the explanation applicable to every kind of input, the following section only addresses the split

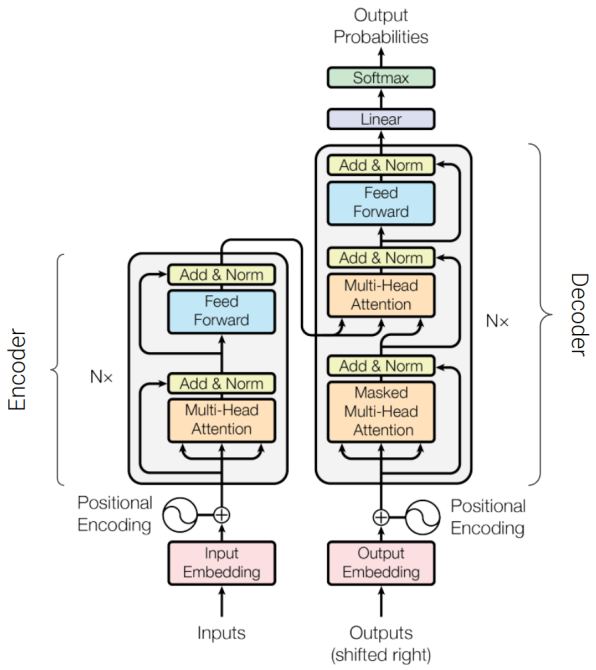


Figure 2.5: Full Transformer model architecture by Vaswani et al. with N encoder layers and N decoder layers [97].

up input as tokens. These tokens get equipped with embeddings before they are fed into the first encoder layer. This means that each input token is turned into a numerical vector representation. Now the tokens also receive positional information, to retain information about their location in relation to the entire input, for upcoming encoding and decoding steps. This is done by applying sine and cosine functions in alternation to the positional indices and adding them to the previous input embedding vectors. Those functions have been chosen because the model can quickly learn their linear properties.

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}}) \tag{2.2}$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}}) \tag{2.3}$$

For every odd position the cosine function is used, while the sine function is used for every even position. Both the embedding equipment step and the enrichment of positional information can be seen in the bottom left in Figure 2.5, displaying the entire Transformer model architecture. The output of the positional encoding is then passed on to the first encoder layer.

2.3.1 Encoder

Encoders condense the input information into an embedding vector with lower dimensionality and as little information loss as possible. The resulting embeddings can either be used on their own for tasks such as image classification or sentiment analysis, or they can be fed into a decoder to produce a response to the input in the form of content. The encoder consists of N layers stacked on top of each other, whereof each layer is split into two sub-layers, the Attention sub-layer and a position-wise fully connected feed-forward network sub-layer. In the original paper of Vaswani et al. [97] N is of size 6, thus passing the data through six layers until it finally reaches the decoder module.

The input vectors with the input embeddings and the positional information from section 2.3 reach the first encoder layer and are split up into query, key and value vectors, which are further processed by the Attention module described in section 2.2. The concatenated Attention heads are then again added to the original positional input embeddings. This result is called residual connection output and gets normalized in the next step. The normalized residual connection output is fed into a pointwise feed-forward network consisting of two linear layers connected by a ReLU activation function.

Figure 2.6 presents a visualization of the pointwise feed-forward network. As a last step of the encoder, the output of the pointwise feed-forward network is added to its input and normalized. All these steps are performed to tie Attention information to the input and turn it into a continuous representation. After repeating the steps of the encoder N times, the resulting embedding vector can be used as is or passed on to the decoder.

2.3.2 Decoder

The decoder is similarly structured to the encoder with the only difference being that it possesses two Multi-headed Attention layers and a softmax layer at the end for the final probabilities. They are used to create new output, such as text or images. Tasks such as image classification require only the results of the encoder module. Figure 2.5 displays the entire decoder and the interplay with the encoder. Like the encoder, the decoder gets also stacked N times, allowing it to combine the information of different Attention steps.

The decoder receives two kinds of input, firstly the outputs of previous decoder steps as a list and secondly the outputs of the encoder. The outputs of the encoder are used in a later step. First, the decoder starts, similarly to the encoder, by adding output embedding and positional information to the list of previous decoder outputs, turning it into new input. The input receives an Attention score by passing the resulting positional embeddings into the

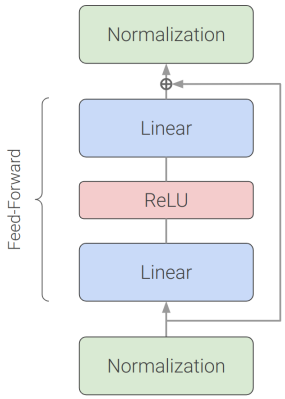


Figure 2.6: Pointwise feed-forward network sub-layer of encoder and decoder, where the output of the Attention module is fed into the normalization layer at the bottom [97].

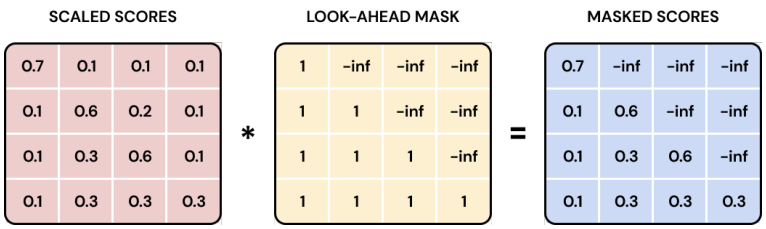


Figure 2.7: The look-ahead mask of the decoder makes the model focus only on already examined tokens and disregards all that might be coming up in the future [69].

first multi-head Attention layer. Because it handles all the outputs of previous decoder steps again, it is able to continuously be mindful of output it has already produced.

Because the transformer creates outputs token by token, this Attention layer has a specific characteristic, called look-ahead mask, to avoid it from taking tokens into account that have not been perceived yet. The look-ahead mask is applied between the down-scaling step and the softmax function, as can be seen by the red box in Figure 2.4(a). It sets all values for future tokens infinitely small, turning them to zeros after the softmax function, to avoid them being considered in upcoming steps. Figure 2.7 visualizes the way the look-ahead mask works. This Attention step is also performed h times in parallel to create h heads.

The results of the first Attention step are then passed on to another At-

tention module, where the outputs of the encoder get involved. The encoder's output is used as query and key vectors for the second Attention step and the output of the first Attention step is used as a value vector. At this point, a matching between the encoder output and the decoder input takes place. The decoder has to decide what encoder output is most important in order to emphasize it. This Attention layer does not apply any mask, as the look-ahead mask was only relevant for the first Attention step of the decoder.

The output of the decoder's second Attention layer is fed into a pointwise feed-forward network identical to the one on the encoder side, making Figure 2.6 also valid for the decoder side. As a penultimate step, the results get fed into a linear classifier model, returning a vector as big as the number of classes present in the model. Ultimately, the vector goes into a softmax layer returning probability scores for each class. The index of the highest probability represents the output for this iteration. To close the decoder circle, this output is added to the list of elapsed outputs, to be fed to the decoder as input in the next iteration. This circle continues until an end token is produced. The end token is also one of the classes of the linear classifier.

Since the project of this thesis is image classification, the Transformers used do not have the decoder module attached to them. Image classification tasks directly process the outputs of the encoder module for their results.

The preceding description of the Transformer architecture is deliberately held general, because inputs can be of different data types, but still work similarly as soon as the input data has been turned into numerical vectors. While there are different models working with textual input, like BERT [20] and the GPT models [13], this thesis focuses on CV and an image based version of Transformers. Vision Transformers [24] are a specialized Transformer derivative for all tasks handling image inputs. The distinctions between Vision Transformers and the plain Transformers are explained in the following section.

2.4 Vision Transformer

Simply speaking, the Vision Transformer (ViT) uses the same Transformer architecture with an additional input layer turning images into numerical vectors. The ViT used in this thesis does not have a decoder module, since the embedding vector from the encoder is sufficient for image classification.

First, the image is split into patches of pre-determined size. Sizes of 14 by 14 pixels, 16 by 16 pixels and 32 by 32 pixels have been used in the original paper. If the images are monochromatic, the patches only have two dimensions, height and width. When the model is handling images with colors, the patch

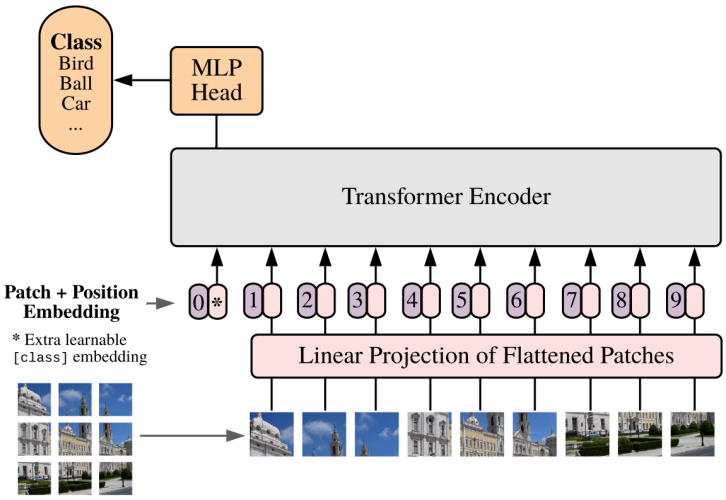


Figure 2.8: Vision Transformer architecture for image classification without decoder module using linear projection of flattened image patches and the subsequent positional embedding [24].

consists of an additional third dimension with a depth of 3, for the red, green and blue channels, resulting in a size of $16 \times 16 \times 3$. If they are not in a number representation already, every pixel of these patches is then turned into a number, to be machine-readable. Simply put the patch is now represented as a three-dimensional, numerical matrix. Then every matrix is flattened out into a one-dimensional vector. In the case of a patch size of 16, this one-dimensional vector has length 768 ($= 16 \cdot 16 \cdot 3$).

Figure 2.8 shows a particularity of the ViT architecture. Alongside all the vectors which originate from image input, there is an additional vector at the very left containing an additional learnable class embedding. Like for the plain Transformer, these vectors are now enriched with positional embeddings and can be fed into a standard Transformer model as described in Section 2.3, which is also called Multi-Layer Perceptron (MLP). Final classification is formed similarly to the plain Transformer. The MLP head is a linear layer with a softmax function to return the probabilities for every possible class.

ViT models occur in different sizes which are described with a letter followed by a number, like B/16 or S/32. The letter denotes the size of the scale of the model which includes the number of encoder layers, called the depth, the number of Attention heads and the MLP dimensions as listed in Table 2.1. The number indicates the patch size, where 16 means that patches of 16 by 16 pixels are used [93].

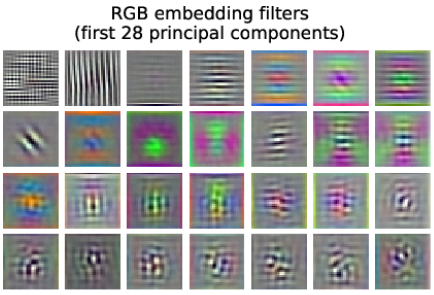
Specification	S/32	Ti/16	B/32	S/16	B/16
Width	384	192	768	384	768
Depth (# of encoder layers)	12	12	12	12	12
Patch resolution P×P	32×32	16×16	32×32	16×16	16×16
MLP	1536	768	3072	1536	3072
Attention heads	6	3	12	6	12
Parameters (in mio.)	22	5.5	87	22	86

Table 2.1: The *Ti*, *S*, and *B* (tiny, small and base) model scales follow Steiner et al. [88]. The brief notation "B/16" indicates that a model is of base size with patches of resolution 16×16. The number of parameters is reported for an input resolution of 224 and does not include the weights of the classifier head [93].

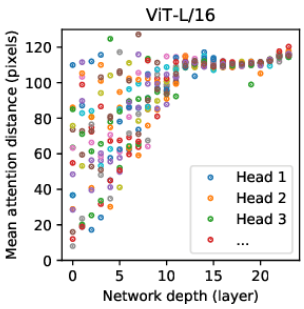
Choosing a specific patch size has consequences for the computations, because every token pays Attention to every other token. A bigger patch size results in fewer patches, therefore also in fewer tokens and in fewer Attention dependencies among all tokens. In exchange for fewer relations inside the model, the size of the vectors and matrices grows, as there are more pixels in each patch. This trade-off between patch and vector size does not allow for a universal recommendation. The patch size has to be evaluated considering the available hardware and the input images.

Without ever specifically telling ViTs what to learn, they learn a lot by themselves. Similar to the wavelet filters of CNNs, they learn to create their own filters and apply them to detect particular edges in the images. Figure 2.9(a) shows the learned filters of a large ViT model with 24 layers, 307 million parameters and a patch size of 32 by 32 pixels. Figure 2.9(b) shows the distance between pixels where they pay Attention to each other of the same model, the *Attention distance*. Notably, all pixels in the ViTs are able to pay Attention to distant pixels from the first layer on. This is represented by the colored dots in the top left. For a CNN, the graph would look like a linear line from the bottom left to the top right, because it takes a few convolutional layers for distant pixels to learn from each other [2]. In CNNs, this is called receptive field size and is analogous to the Attention distance on ViTs.

Advantages of the ViT architecture are, that it allows for more parallelization, because they do not have to wait for the output of a previous convolutional layer. Furthermore, ViT models are not restricted in the way they look at images like it would be the case with edge detection in CNNs. They can freely learn how to accomplish their main tasks with their Attention on every pixel simultaneously (see Figure 2.9). But this also means that they require more data to train on. Typically, the ViT models are pre-trained on



(a) Filters learned and applied by the ViT on its own.



(b) Distance between pixels that pay Attention to each other dependent on the layer.

Figure 2.9: Aspects the ViT learns on its own similar to traditional CNNs [24].

very big datasets with 14 million (ImageNet-21k [77]) or 300 million images (JFT-300M [89]) and only fine-tuned on smaller datasets [24]. The pre-trained versions are freely available for download online [23]. When trained on smaller datasets, it is outperformed by deep CNNs [37]. Because of this, it is recommended to download a pre-trained model and fine-tune it on the desired data. Different datasets commonly used in academic publications are described in the following section.

Chapter 3

Datasets

Over time, some universal datasets have turned out to be used as the standard in academic publications. This fixed pool of datasets allows for a clear comparison of research findings and different kinds of improvements, in contrast to every publication using their own data. They consist of images with labels in different variations for different purposes.

ImageNet-21k [77] and JFT-300M [89] are "primary dataset[s] for pretraining deep learning models for computer vision tasks" [78]. This is also the purpose they were used for in ViTs. They characteristically have a lot of images with multiple labels for each image. Here it is important to note that the JFT-300M dataset is not publically available, because of copyright uncertainties.

For fine-tuning on the other hand, it is helpful to use datasets with less distinct classes. If the result is supposed to be a fully working product, the dataset for fine-tuning has to match the actual use case. For the sole comparison of architectures against each other in a scientific environment, the data is only used to train the model up to a comparable baseline. Therefore, datasets like MNIST with 10 classes of 70,000 handwritten digits [52], Cifar-10 with 10 classes of everyday objects on 60,000 images or Cifar-100 of the same size with similar objects, but 100 classes respectively [48], exist.

Here it is important to make the distinction between single-channel and multi-channel. In out-of-distribution (OOD) detection tasks, like the one illustrated in this thesis, it makes most sense to only work with datasets consisting of the same number of channels at the same time. Channels represent the number of colors existing in the images. A single-channel image consists of monochrome colors, most of the time the colors black and white. An image with three channels on the other hand contains one channel for red, green and blue respectively, resulting in a variation of different colors and color mixtures. This thesis deals with colored images with three channels from the Cifar-10,

Cifar-100 and SVHN datasets, therefore they are mainly addressed in the following sections. Other prominent datasets like MNIST and 80M Tiny Images are briefly touched upon, as well as the basics of data augmentation techniques to virtually increase the size of datasets.

3.1 Cifar-10 and Cifar-100

The Cifar-10 and Cifar-100 datasets have been created by Alex Krizhevsky et al. as part of his master thesis at the University of Toronto. Because they are freely available [47] and used in most computer vision publications, they serve as a good point of comparison between different works. These two datasets provide a more difficult challenge in detecting OOD samples for a model, as some of the images share features with each other and complicate a clear distinction. Cifar-10 contains 6000 images for each of its 10 classes (see Appendix A.2.1). Cifar-100 contains 600 images for each of its 100 classes, which are further categorized in 10 superclasses (see Appendix A.2.2). They are both already split into a subset for training and one for testing with a ratio of 90:10 respectively.

The images themselves are not stored in a conventional image format (like jpeg or png), but rather in a binary representation where the first byte represents the label and the following 3072 bytes of the pixel values. Because the pixel values are separated in their RGB (red green blue) values, all these channels have to be stored individually. The first 1024 bytes constitute the red channel, the following 1024 bytes the green channel and the last 1024 bytes the blue channel. Since every image has a width and a height of 32 pixels, the first 32 bytes of each color channel represent the first row of pixels and so forth.

3.2 SVHN

Another dataset with three color channels is the "Street View House Numbers" (SVHN) dataset containing images of street numbers. It was created by cutting out the street numbers of Google Street View images. Because the street number digits look fundamentally different from the classes of Cifar-10 or Cifar-100 it can be used as a relatively easy to detect OOD counterpart. This thesis uses the format with the cropped digits resulting in 10 classes, from 0 to 9, portrayed on 32 by 32 pixels like Cifar-10 or Cifar-100 [68].

3.3 Other Datasets

More datasets exist, which all fulfill their own purpose. The aforementioned MNIST dataset is used to train models to detect handwritten characters in the domain of optical character recognition. But because it only contains a single channel, resulting in black and white images, it is not being used for this thesis. Models could develop a bias regarding the amount of channels or colors in the OOD samples.

Another dataset, called 80M Tiny Images, was also widely used in CV research projects, but got ultimately retracted from public use. It consisted of 80 million colored images with a resolution of 32 by 32 pixels with 53,464 labels taken from Wordnet. Because the images were automatically downloaded from the set of labels and their sheer quantity, they were never manually reviewed. It later turned out that the images contained offensive content and derogatory labels, so that Torralba et al. decided to retract the dataset to encourage an inclusive community [94]. If older works only used the 80M Tiny Images dataset, they are barely comparable to newer findings, as a consistent use of datasets is important for the comparison.

3.4 Data Augmentation

Data augmentation techniques help to artificially increase the datasets at hand, mostly for training, by altering the images slightly. These include cropping, resizing, flipping, rotating, color distorting and mixing with random Gaussian blur among others [14]. Cutout, Mixup and CutMix are shortly introduced in the following section, as advanced methods for data augmentation [55].

Cutout takes a rectangular area of the image and fills it with a single color, making it look like the rectangle has been cut out [21]. Mixup takes different pixels of two images and puts them back together as one, making the result look like a visual fusion of both original images. It is important to note that the image also gets assigned two labels in this progress with weights related to the amount of pixels taken from each original image [104]. CutMix combines the two aforementioned techniques. It selects a rectangular area of the base image and inside this rectangle it alternates between pixels of the base or a second image. The result also gets assigned two labels with weights depending on the amount of pixels from each image [101]. These techniques make it possible for the model to see a multitude of unique images, even when the real pool of training samples is rather limited. All these methods are visualized in Figure 3.1.

Data augmentation techniques have been used for self-supervised learning approaches, where the label stays the same, but the actual sample is modified.

(a) Normal (b) Cutout [21] (c) Mixup [104] (d) CutMix [101]

Figure 3.1: The three data augmentation techniques most commonly used as a base to increase the training dataset [101].

Khosla et al. [44] have used this to approach to train their self-supervised contrastive loss for example.

Newest findings however suggest that the performance of ViTs does not rely on data augmentation, and they can achieve similar results in CV tasks without these techniques. If a good hyperparameter setting is found and training time is increased, similar results can be obtained [6]. Another way to modify data is by adding noise to them. Depending on what kind of noise and how much of it is added to the image, they are able to severely change their appearance. These procedures are summarized as Adversarial Attacks, which are explained in the next chapter.

Chapter 4

Adversarial Attacks

Adversarial attacks are a directed form of input to trick the model into unanticipated behavior. These attacks can be categorized by the point when they occur in the process, by the goal they are pursuing and if the attacker has insight knowledge about the system. If the attacks are applied during training, they *poison* the model, also called *poisoning*-attacks, in contrast to attacks during test time, called *evasion*-attacks. When they are aiming for a specific outcome, like a specific class to be detected, they are called *directed* or *targeted* attacks. Otherwise, they are called *undirected* or *non-targeted*, if their only objective is to decrease the model's performance [107]. If the attacker has (almost) full insights into the model, its calculations and gradients, it is called a *white box attack*. Whereas in a *black box attack* only the outputs to given inputs are revealed, while concealing the model's architecture and inner proceedings [70].

A prominent example of a directed black box poisoning-attack, that occurred outside the controlled world of research, was the chatbot Tay in 2016. It was released by Microsoft to interact with users on Twitter and was shut down only 16 hours after release, after it had been infested with misanthropic ideas. The problem was that it was able to learn from the conversations it was involved in, but without proper protection mechanisms against extreme semantical concepts. After it was exposed to a multitude of hateful tweets, it had internalized this ideology and produced similar offensive or controversial answers in terms of accepted societal norms and values to seemingly unrelated questions. Microsoft was quick to shut the chatbot down and release a statement that the project went unfortunately wrong and a lot of research has to be done in order to make AI systems secure against all kinds of attacks [54, 84].

Historically seen, the first adversarial attacks have been designed to trick email spam filters by Dalvi et al. [16] and Lowd et al. [60, 61]. It continued with the first attacks on neural networks in 2006 by Barreno et al. [5] and

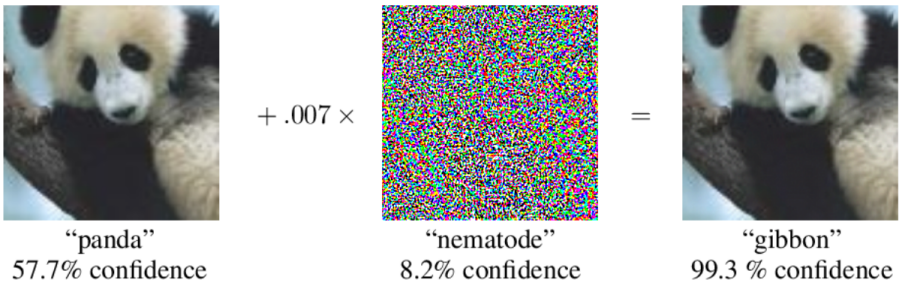


Figure 4.1: Prominent example of noise added to an image to let the model be overconfident with a false prediction [28]. The applied noise is random and scaled down by a factor of 0.007 so that humans can not detect it, because small amounts of noise are sufficient to fool the model.

followed by the first adversarial attacks on deep neural networks by Biggio et al. in 2013 [7], starting to involve noise tailored to the models gradients in the evasion attack process.

If the model is trained to associate specific pixels with their value and coordinates to particular classes, noise in the sample can trick the model into believing it handles a sample from a different class. This noise can be subtle and not detectable by the human eye, but still fool a machine. This can be seen in Figure 4.1 where in the beginning a classifier correctly classifies an image of a panda as such. But as soon as a marginal amount of random noise gets added to the image, the classifier can not detect a panda anymore and instead assigns it a high confidence of another class (*gibbon* in this example), because of the added noise.

If, on the other hand, the noise becomes big enough, it can not only be perceived by humans, but can also alter the content to be classified differently even by a human judge (see Figure 4.2). This opens up the discussion to how much noise can actually be applied to samples, or images in this case, until their label should also be adjusted. As there exists no clear solution for this issue yet, this thesis only draws attention to it without striving for an answer.

Noise can not only be used to trick a classifier, but also to create new samples. Goodfellow et al. [27] used the noise in a two-player architecture called Generative Adversarial Network (GAN) to create new images. The two players consist of a model creating new images from random inputs, called the generator, and its counterpart, the discriminator, trying to distinguish between images from the original domain and generated samples. This contradictory training improves both models in their respective domain and, when sufficiently trained, the generator is able to alter its random input

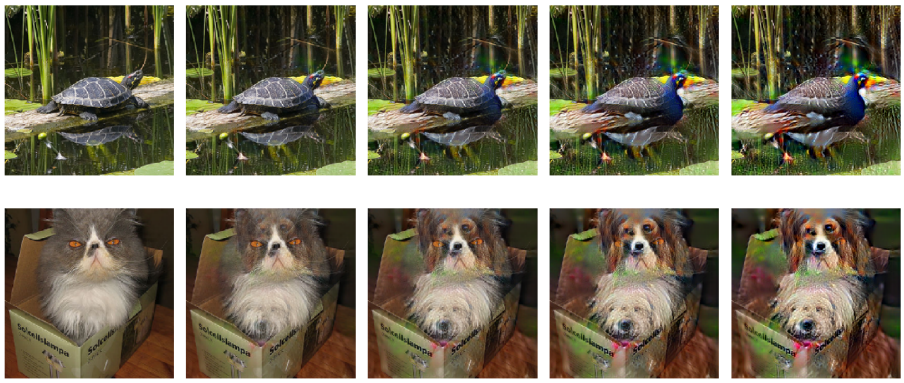


Figure 4.2: Noise gradually changing images until the objects would even be classified into different classes by humans [95].

into seemingly realistic results indistinguishable by humans. GANs have since also been used to synthesize images [11], create images from text [102, 76], blend images together [99], create 3D objects [100] or edit the content of images [40, 106, 71, 59, 12, 103, 3].

Different types of adversarial attacks exist, but this thesis focuses on directed Projected Gradient Descent (PGD) attacks during test time (*evasion-attacks*), where the model is fooled into believing it handles samples from different classes. The following sections explain PGD more in detail and how models can be protected against adversarial attacks.

4.1 Projected Gradient Descent

Projected Gradient Descent (PGD) is aiming to produce an optimal attack sample by altering the input sample with noise matching the gradient of the model [7, 63]. This attack is considered optimal, because it can adjust the sample, with the knowledge about the model's gradients, according to the expectations of the model. For this method to be applicable the attacker needs access to the model and its gradients, therefore it is categorized as a *white box attack*. The attack tries to create a new sample that produces the biggest loss possible. For every step of the attack the noise is adapted to the loss from previous steps, while being restricted by an upper bound. The upper bound is in place to restrict the amount of maximum perturbation. If there would be no boundary in place here, the samples could actually end up looking like the class predicted by the model, similar to Figure 4.2.

This boundary is represented by an ε-ball, limiting the amount of noise

that can be applied to one data sample. The bigger the ε the more noise is allowed and the more the resulting sample can differ from the original input. The attack starts with taking a data sample and applying a random amount of noise from within the ε-ball. The resulting perturbed sample is then fed into the classifier, producing a loss as output. The following gradient step is taken towards a bigger loss than before. The bigger loss will then be applied to the sample in the next iteration and not to the classifier, as this model is already fully trained. In case this gradient step lies outside the ε-ball, it has to be projected back onto the perimeter of the ball to not exceed the predetermined constraint. The resulting loss is then applied as the new perturbation to the sample and the steps are repeated until the designated, maximum amount of iterations is reached. By starting at several random positions, the probability increases to reach a better, or even the optimal, perturbation allowed within the boundary. The perturbation of a sample is shown in the following formula based on the works of Boyd and Vandenberghe [10] and Lin [58]. x^k denotes the current sample and x^{k+1} the sample of the next iteration including the perturbation based on the gradients of the current iteration ($\nabla f(x^k)$):

$$x^{k+1} = P[x^k|\nabla f(x^k)]^{\varepsilon} \tag{4.1}$$

The sample including perturbation gets projected back onto the maximum allowed perturbation (ε) by the projection operator ($P[x]$):

$$P[x^k|\nabla f(x^k)]^{\varepsilon} = \begin{cases} x^k + \varepsilon & if\ \varepsilon < \nabla f(x^k) \\ x^k + \nabla f(x^k) & if\ -\varepsilon \leq \nabla f(x^k) \leq \varepsilon \\ x^k - \varepsilon & if\ \nabla f(x^k) < -\varepsilon \end{cases} \tag{4.2}$$

Bigger amounts of restarts and iterations help to find superior perturbations, but also come with increasing cost in computing power, as the sample has to be run multiple times through the classifier to find a good perturbation. Therefore, they are often limited in a trade-off between optimal perturbation and calculation time.

The PGD follows a saddle point problem, where two players try to optimize their outcome against the objective of the other one. The classifier is always striving to classify input samples with the lowest error rate, whereas the attacker tries to maximize the error rate of the classifier by altering the inputs. When this counterplay is visualized by a convex and a concave curve the three dimensional plot represents a saddle (see Figure 4.4).

In the concrete example of this thesis the maximization takes part in the form of the iterations, where the attacker only takes the maximum loss every step for a given sample, even if parts of it are from a previous iteration. This maximization function is embedded into the attack, which in the end chooses

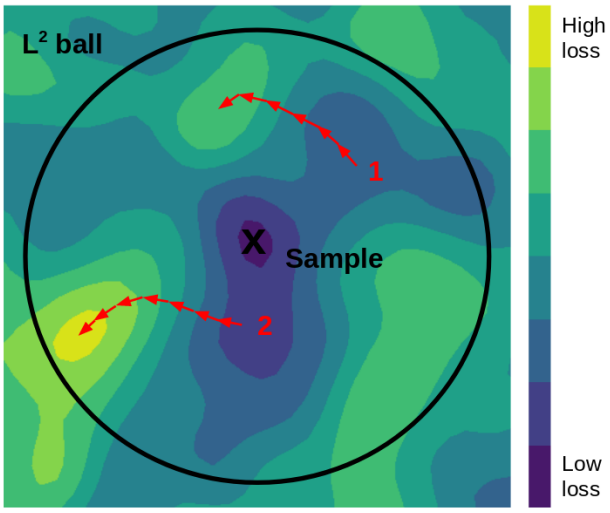


Figure 4.3: The gradient descent steps of two repeats of a PGD attack on one input sample. Restart number 2 ends up in a bigger loss than restart number 1, resulting in a bigger deviation of the model's predictions [45].

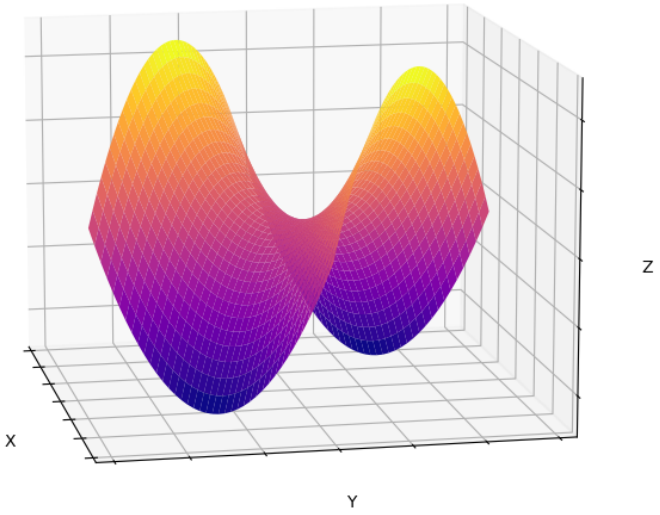


Figure 4.4: Visualization of the saddle point problem. The z-axis describes the loss produced for each sample by the classifier. The convex arcs describe the loss minimization efforts of the classifier and the concave arcs describe the loss maximization efforts of the attack.

the sample with the minimal loss from all final results created by the various restarts. With this interplay the attack tries to find the minimal maximum, which can be explained as the worst perturbed sample from all optimally perturbed samples within the ε-ball constraint.

Because adversarial attacks can occur in different forms and shapes, it has been infeasible to make a system universally robust against them so far. Defense mechanisms only exist against distinct attack scenarios. Since it is such a safety and security relevant area, a lot of research is still directed in this area. The different proposed concepts to protect the systems have been outlined in the following section.

4.2 Proofing Models Against Adversarial Attacks

So far, there is no procedure to reliably detect adversarial attacks or make models completely secure against them, but different methods manage to mitigate the magnitude of such attacks, partially detect them or make the models robust against specific attack scenarios. The difficulty in detecting adversarial attacks comes from the belief of the model, it is handling legitimate data, fortified by the high confidences it might produce in the end [33].

To protect the system, it has been proposed to have multiple models available and let a randomly chosen one classify the input sample. This is often visualized as a moving target, because the target (the model in this case) of the attack is different for every attack. Other approaches also combine the output of multiple models, which requires an attacker to learn the behavior of all models instead of just one. But by creating more sophisticated attacks, which learn about the multitude of models and the random rotation algorithm, these methods can also be cracked [80, 1].

It has also been demonstrated that bigger network architectures tend to be more robust against attacks, as a bigger capacity is needed to combine the normal classification task with the attack detection task [63]. The most effective approach yet is to train the model against a broad spectrum of attacks ahead of time, but for this to work best one would have to identify every possible weak point in advance and find a suitable training strategy.

Because certain adversarial input can be considered as out-of-distribution (OOD) data, it can in these cases be treated and detected as such. The basics of OOD detection, the metrics used for it and different methods with a focus on deep neural networks and ViTs to perform OOD detection are presented in the following chapter.

Chapter 5

Out-Of-Distribution Detection

Data samples containing classes that an ML model has not been trained on, can cause uncertainty and wrong results when they are evaluated by the model. These data samples are declared as out-of-distribution (OOD) data and if a trained model is exposed to them, the model should properly detect them. Because complete avoidance can not be ensured in most real world settings, they have to be treated appropriately.

All classes a model is trained on are the in-distribution (ID) and the dataset it has been trained on is therefore called ID dataset. If the model is fed a sample with an unfamiliar class during test time or even during production time, this sample is understood as an out-of-distribution (OOD) sample. When a model is exposed to a dataset containing samples with classes it has never been trained on, this dataset is considered an entire OOD dataset. If a classifier is for example trained on images of the Cifar-10 dataset, the dataset with all of its 10 classes are considered ID (see the classes in Appendix A.2.1). Even an image that is not explicitly from the training dataset, but displays an object that would be attributed to one of the trained classes by human standards, is regarded as an ID image. When the trained model is now exposed to an image containing a different object, it has never been trained on before, this image it is considered an OOD sample. When it has been trained on Cifar-10 classes and is now exposed to an image containing numerical digits, the new image is called an OOD sample.

When stepping out of the closed-world assumption of only having to deal with ID data, the models typically drop in performance [91]. In a real world scenario, data distributions might change, and new classes could be introduced over time, but the model only stays adjusted to the data it has been trained on. To prevent the model from thinking in its closed-world knowledge, which could potentially lead to fatal errors, it is important to address OOD detection [38].

Traditionally, OOD data was detected with the final class prediction prob-

abilities when the maximum falls under a certain threshold, untypical for ID samples [33, 53]. Another method by DeVries and Taylor involved attaching an additional branch to the pre-trained classifier that returns a new score indicating OOD samples [22]. Generally, the softmax class scores tend to be lower for OOD samples and allow for general OOD indications across all ML disciplines. Later research further improved on the method of attaching additional branches to classification models in the supervised learning domain [33, 73].

Alternatively, OOD data can also be identified by contrastive learning, where the model learns features of a class by comparing it directly with another class. This improves the model's ability to distinguish OOD samples even in semi-supervised or self-supervised training scenarios [14, 15, 44]. In the case of contrastive learning, the model learns about the visual features of its ID data and is able to differentiate it from samples without these associated combination of features. Contrastive learning on its own, however, is not able to treat adversarially altered samples by itself with just learning about visual features.

Here it is important to note, that OOD samples and adversarial attacks are often used jointly, as they are in the works of Lee et al. [53], Hendrycks et al. [34] and Liang et al. [57]. Adversarial attacks in combination with OOD detection mostly occur in the form of a weak noise that is blended together with data samples to modify them. It can even make a difference to the human viewer, depending on how much noise is applied to the sample. Little noise lets the sample appear unchanged to the human eye, but entirely different to the machine. Bigger amounts of noise can make the image look entirely different, as seen in the form of generative adversarial networks [27]. Adversarial attacks are explained in Chapter 4.

This thesis does only cover OOD detection in the domains of supervised ML, but it has also been addressed for reinforcement learning and unsupervised ML. A starting point to read into OOD detection in the reinforcement learning domain would be the paper by Sedlmeier et al. [85]. For OOD detection in the field of unsupervised ML a broad overview is given by the master thesis of Magnus Pierrau [74].

To compare the performance of different works against each other, metrics like *False Positive Rate at N% True Positive Rate* (FPRN), *Area Under the Receiver-Operator Characteristic Curve* (AUROC) and *Area Under the Precision-Recall Curve* (AUPR) are being applied, which are described in the next section. Promising techniques to make models recognize OOD samples, like ODIN and Outlier Exposure (OE) are explained in the subsequent sections. Followed by the *Guaranteed Area Under the Receiver-Operator Characteristic Curve* (GOOD) and *Provable Out-of-distribution Detector* (ProoD) approaches, which attempt to give guarantees on OOD detection in deep neu-

ral networks. Lastly, OODformer is presented as an option to detect OOD samples specifically with ViTs and the use of contrastive learning.

5.1 Measurement Metrics

In order to measure the performance of various OOD detection methods, different metrics can be applied. The most commonly used in the literature are the *False Positive Rate at N% True Positive Rate* (FPRN), the *Area Under the Precision-Recall Curve* (AUPR) and the *Area Under the Receiver-Operator Characteristic Curve* (AUROC or AUC), which are described in the following sections. These metrics can universally be applied to all models, even if they were not created with the intention to detect OOD data.

5.1.1 FPRN

A fixed false positive rate at N% true positive rate (FPRN) can be used to compare OOD detection performance across different models. A fixed true positive rate of ID samples is set to N%, 95% are commonly used, and then the false positive rate is measured [34, 4]. Lower FPRN values stand for better OOD detection performance.

$$true\ positive\ rate = \frac{true\ positives}{true\ positives + false\ negatives} \tag{5.1}$$

$$false\ positive\ rate = \frac{false\ positives}{false\ positives + true\ negatives} \tag{5.2}$$

The true positive rate is obtained by dividing the true positives by the sum of the true positives and the false negatives (see Equation 5.1). The false positive rate is obtained by dividing the false positives by the sum of the false positives and the true negatives (see Equation 5.2). It can be calculated on any model, as it just needs true positives, false positives, true negatives and false negatives.

The terms true positive, false positive, true negative and false negative originate from the information retrieval discipline. They indicate if an element was retrieved rightfully or wrongfully, or not retrieved rightfully or wrongfully, respectively. Figure 5.1 visualizes the relation between true positives, false positives, true negatives and false negatives for better understanding. In the setting of OOD detection a true positive is a correctly classified ID sample and a false positive a wrongfully detected ID sample. A true negative is a rightfully detected OOD sample and a false negative a wrongfully detected OOD sample.

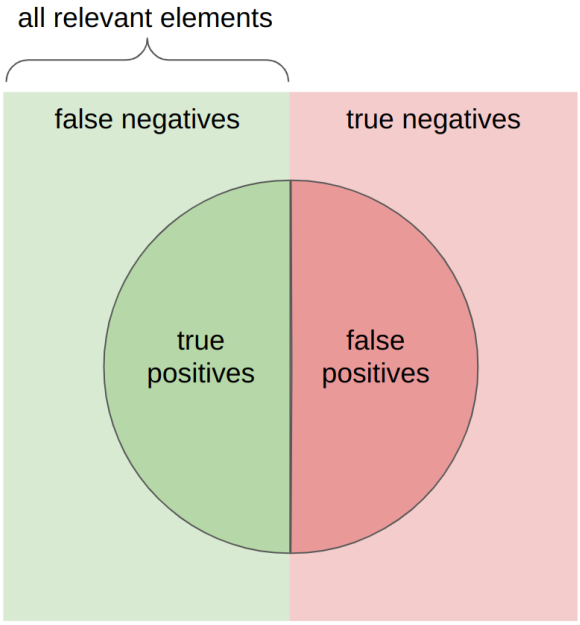


Figure 5.1: Considering an information retrieval model, all the elements that were returned are portrayed by the circle. This set contains the rightfully retrieved elements (true positives) in saturated green and the wrongfully retrieved ones (false positives) in saturated red. In an optimal case, it should have returned all elements in green and not retrieved any red elements. The green elements outside the circle, which were wrongfully not retrieved, are called false negatives. The red elements outside the circle, which were rightfully not retrieved, are called true negatives. With these values, the true positive rate and the false negative rate can be calculated as presented in Equation 5.1 and Equation 5.2 respectively. It also allows for the calculation of precision and recall as presented in Equation 5.3 and Equation 5.4 respectively.

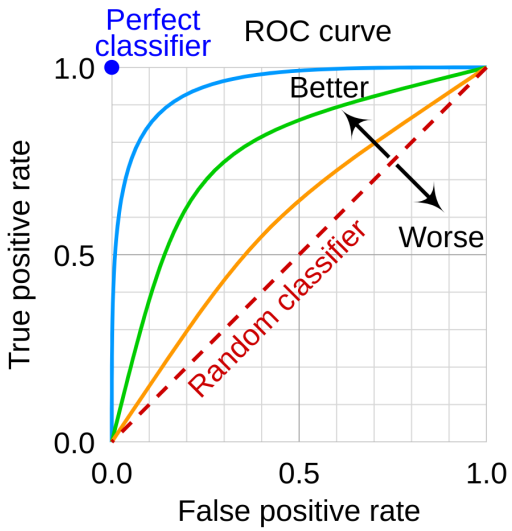


Figure 5.2: Visualization of the Area Under the Receiver-Operator Characteristic Curve (AUROC) [92]. A perfect classifier covers the biggest area under its curve, because it only returns true positives. As the areas get smaller from blue over green and yellow to red, the classifiers provide worse performance.

5.1.2 AUROC

The Area Under the Receiver-Operator Characteristic Curve (AUROC) by Green and Swets [29] is a slight enhancement of the FPRN. The AUROC does not only use one fixed true positive rate value to compare OOD detection performance of different models, but a whole range of values. It can be calculated on any model, as it just needs the same values as FPRN. To construct the curve, the false positive rates for all true positive rates in this range have to be calculated.

Then a curve is constructed with the false positive rate on the x-axis and the true positive rate on the y-axis. By measuring the area under the curve, a statement about its OOD detection abilities can be made. A bigger area signifies a better model and a smaller area a worse model to detect OOD data. A random classifier would cover half of the area. This relation between area and quality is visualized in Figure 5.2.

AUROC is often abbreviated as AUC, especially when including it in the naming of more refined versions, but this thesis consistently uses the expression *AUROC*.

5.1.3 AUPR

The Area Under the Precision-Recall Curve (AUPR) can also be used to compare different models with regard to their OOD detection performance. It can be calculated on any model, as it just needs precision and recall values. Precision and recall originally stem from the area of information retrieval as well. Precision indicates *how many of the retrieved items are relevant* and is calculated by all true positives divided by the sum of all true positives and false positives (see Equation 5.3). Recall indicates *how many of the relevant items have been retrieved* and is calculated by all true positives divided by the sum of all true positives and false negatives (see Equation 5.4). It is important to note here that neither precision nor recall make use of the true negatives, as the rightfully not retrieved elements do not offer any added value in information retrieval.

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{5.3}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{5.4}$$

While precision and recall are useful in information retrieval systems, they are also used to evaluate ML models. When the recall is put on the x-axis and the precision on the y-axis, they produce the precision-recall curve. The area under this curve indicates how well the model can detect OOD samples. A bigger area stands for better and a smaller area for a worse OOD detection, similar to AUROC [65].

Davis and Goadrich [17] recommend using the AUROC when each class is similarly often predicted and the AUPR when there seems to be a bigger imbalance. This is due to the fact that true negatives are used to calculate the AUROC metric, while they are not used to calculate the AUPR.

5.2 ODIN

In 2018 Liang et al. [57] proposed a method to detect *Out-Of-Distribution Images in Neural Networks* (ODIN) that does not require any changes to a trained model, but uses Temperature Scaling and a small amount of noise on the input images to allow for a distinction between ID and OOD. Temperature Scaling was introduced by Hinton et al. [35], where they scaled the logits of the model before passing them on to the final classification softmax layer. ODIN then also incorporates an additional detector module, which is trained on the unaltered images and the ones with slight perturbations, to learn the difference between ID and OOD.

5.3 Outlier Exposure

Outlier Exposure (OE) was first introduced by Hendrycks et al. [34], who trained a model with a variable OOD detector on an ID as well as an OOD dataset. They discovered that the detector is able to generalize OOD data samples even if they differ from the ID dataset based on heuristics it acquires during training. As detectors, they use the maximum softmax probability baseline from Hendrycks and Gimpel [33] and the confidence branch of DeVries and Taylor [22]. They also used FPRN, AUROC and AUPR to compare the results with each other, with OE beating the compared approaches in every test and with every metric.

5.4 Guaranteed OOD Detection

For their Guaranteed OOD Detection (GOOD) architecture, Bitterwolf et al. [9] used upper and lower bounds inside the activation function of the neural network to enhance OOD detection. In their approach, they bound the model's confidence based on the Interval Bound Propagation (IBP) of Katz et al. [43]. IBP is used to give the ReLU output of each layer in a network an upper as well as a lower bound, to restrict the layer output. With this method, Bitterwolf et al. opted to bind the maximum confidence of their model on OOD data, which they called Confidence Upper Bound (CUB). The confidence is bound by an l_∞-ball of size ε, where the way of measuring the distance is the l_∞ distance. CUB comes with its own loss function, the Confidence Upper Bound Loss ($\mathcal{L}_{CUB}$).

$$\mathcal{L}_{CUB}(x;\varepsilon) := \log\left(\frac{\left(\max\limits_{k,l=1,\dots,K} \overline{f_k(x) - f_l(x)}^{\varepsilon}\right)^2}{2} + 1\right) \tag{5.5}$$

The $\mathcal{L}_{CUB}$ (see Equation 5.5) mainly consists of the upper bounds of all K^2 logit differences ($\overline{f_k(x) - f_l(x)}^{\varepsilon}$). Here, x is the input sample and $f_i(x)$ is the output of layer i for input x. The exponent ε is the predefined maximum distance for the adversarial perturbation. They decided to only set an upper bound, because setting a lower bound has lead to numerical problems during training. To prevent the loss from exploding for challenging datasets with diverse features, they restricted the entire calculation by a log function.

When they trained the model with OOD data, they used two loss functions, the $\mathcal{L}_{CUB}(x;\varepsilon)$ and the $\mathcal{L}_{CUB}(x;0)$, to train the detector on the OOD dataset. The $\mathcal{L}_{CUB}(x;0)$-loss is not bound to any ε-ball. The second loss is necessary for

datasets that share visual features with each other. For example, Cifar-10 and Cifar-100 contain images with objects that share visual features, like a deer and a fox both have four legs and can have similar color. These similarities lead to low confidences on the ID dataset if only trained on the $\mathcal{L}_{CUB}(x;\varepsilon)$, so they incorporated a second loss to circumvent these low confidences.

The Cross-Entropy loss is calculated for all N samples of the ID dataset. For the OOD dataset they used their $\mathcal{L}_{CUB}$, where they first had to order all M data samples according to the $\mathcal{L}_{CUB}(x;\varepsilon)$-losses they would produce. In practice they did this ordering within the individual batches and not on the overall dataset. Then they took the quantile of samples that produced the smaller loss and applied the $\mathcal{L}_{CUB}(x;\varepsilon)$ to gain more information from these samples. For the rest of the dataset, they applied the $\mathcal{L}_{CUB}$ loss without the ε-distance ($\mathcal{L}_{CUB}(x;0)$). The notation $GOOD_{40}$ means that the quantile is at 40% (q = 0.4). For this model, the 40% with the smallest losses out of every batch of the OOD dataset uses the $\mathcal{L}_{CUB}(x;\varepsilon)$-loss and the other 60% use the $\mathcal{L}_{CUB}(x;0)$-loss.

$$\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_{CE}(x_i^{IN},y_i^{IN})+\frac{\kappa}{M}\sum_{j=1}^{\lfloor q\cdot M\rfloor}\mathcal{L}_{CUB}(x_{\pi_j}^{OUT};\varepsilon)+\frac{\kappa}{M}\sum_{j=\lfloor q\cdot M\rfloor+1}^{M}\mathcal{L}_{CUB}(x_{\pi_j}^{OUT};0) \tag{5.6}$$

As a result, GOOD performs better than Outlier Exposure when handling optimally perturbed OOD samples. They found that different quantile thresholds for different combinations of ID and OOD datasets provide better results than fixed quantiles do. Furthermore, GOOD brings almost no loss in accuracy while providing OOD detection in contrast to other models, like clean IBP or adversarial confidence enhanced training. In a further iteration, Meinke et al. [66] developed upon this method to keep the accuracy and OOD performance even higher. They named the new approach *Provable out-of-Distribution detector* (ProoD), which is explained in the following section.

5.5 Provable Out-of-Distribution Detector

The *Provable out-of-Distribution detector* (ProoD) by Meinke et al. [66] is able to detect OOD data while still maintaining a high accuracy for ID samples even for adversarially manipulated data. They combined a dedicated OOD detector and IBP together with an already existing model, similar to the detector of DeVries and Taylor [22] or ODIN [57]. This detector is a binary classifier with the task to differentiate between ID and OOD data. It is trained to be certifiably robust against adversarially altered OOD data instead of ID

data. When a model is able to distinguish adversarially altered OOD data well enough from ID data it gets decorated with the attribute *robust*.

They opted to bind the detector's confidences with IBP to restrict it as far as possible from wrongly identifying OOD samples as ID. Only an upper bound was used and only for OOD data in order to be certifiably robust on the OOD detection task. Because the discriminator only deals with a binary classification problem, the confidence upper bound set by the IBP helps the discriminator to be more stable than the discriminator in GOOD [9]. Their binary classifier always consists of 5 layers where the first 3 layers are two-dimensional convolution layers and the last 2 layers are fully connected layers. If they train on a Cifar dataset, they average-pool after the three two-dimensional convolution layers and if they train on the R.ImgNet dataset, they average-pool after every individual two-dimensional convolution layer.

The classification model and the discriminator were trained together, therefore the process is called joint training. Even though they could have trained them separately and achieved comparable classification and OOD detection performance, they chose not to in order to keep the interdependency between ID and OOD. Furthermore, an ε-ball of size 0.01 was used to train the binary classifier on the OOD data and different ResNet models for the main classification task.

ProoD is not only able to generalize to unknown samples, but also to entire distributions it has never seen before. The AUROC values of different ProoD versions are generally close to each other, meaning that the bounds for the certified robustness are equally narrow. Whereas the AUROC values of ProoD are bigger for every dataset they used compared to the AUROC values of the state-of-the-art architecture indicating better OOD detecting abilities.

On the Cifar-100 dataset, the AUROC drops completely, as do the AUROCs of all other compared OOD detection methods. This can be traced back to the fact that detecting OOD on that many ID classes is still a challenging task. Even though the OOD detection results of ProoD are similar to the results of the previously presented GOOD (see Section 5.4) model, it does not affect the accuracy on the classification task. The OOD detection results of ProoD can be seen in Figure 5.3.

Guaranteed robustness is the terminology for OOD data being adversarially altered to get closer to ID data, but still reliably being detected as OOD data. For the ProoD approach most Guaranteed AUROC values are still under 50%, which is technically worse than a random discriminator. Guaranteed AUROC values are produced under the poorest conditions, so in this case the optimal adversarial modification of samples. ProoD is therefore not able to guarantee robustness of false positives to a 95% true positive rate.

In: CIFAR10		CIFAR100			SVHN			LSUN_CR			Smooth		
	Acc	AUC	GAUC	AAUC	AUC	GAUC	AAUC	AUC	GAUC	AAUC	AUC	GAUC	AAUC
Plain	**95.01**	90.0	0.0	0.7	93.8	0.0	0.3	93.1	0.0	0.5	98.0	0.0	0.7
OE	94.91	**91.1**	0.0	0.9	97.3	0.0	0.0	**100.0**	0.0	2.7	**99.9**	0.0	1.5
ATOM	93.63	78.3	0.0	21.7	94.4	0.0	24.1	79.8	0.0	20.1	99.5	0.0	**73.2**
ACET	93.43	86.0	0.0	4.0	99.3	0.0	4.6	89.2	0.0	3.7	99.9	0.0	40.2
$GOOD_{80}$*	87.39	76.7	47.1	57.1	90.8	43.4	76.8	97.4	70.6	93.6	96.2	72.9	89.9
$GOOD_{100}$*	86.96	67.8	48.1	49.7	62.6	34.9	36.3	84.9	74.6	75.6	87.0	76.1	78.1
ProoD-Disc	-	62.9	57.1	57.8	72.6	65.6	66.4	78.1	71.5	72.3	59.2	49.7	50.4
ProoD $\Delta=3$	94.99	89.8	**46.1**	**46.8**	**98.3**	**53.3**	**54.1**	**100.0**	**58.3**	**59.7**	**99.9**	**38.2**	38.8
In: CIFAR100		**CIFAR10**			**SVHN**			**LSUN_CR**			**Smooth**		
	Acc	**AUC**	**GAUC**	**AAUC**	**AUC**	**GAUC**	**AAUC**	**AUC**	**GAUC**	**AAUC**	**AUC**	**GAUC**	**AAUC**
Plain	**77.38**	**77.7**	0.0	0.4	81.9	0.0	0.2	76.4	0.0	0.3	86.6	0.0	0.4
OE	77.25	77.4	0.0	0.2	**92.3**	0.0	0.0	**100.0**	0.0	0.7	**99.5**	0.0	0.5
ATOM	68.32	78.3	0.0	50.3	91.1	0.0	67.0	95.9	0.0	75.6	98.2	0.0	80.7
ACET	73.02	73.0	0.0	1.4	97.8	0.0	0.7	75.8	0.0	2.6	99.9	0.0	12.8
ProoD-Disc	-	56.1	52.1	52.3	61.0	58.2	58.4	70.4	66.9	67.1	29.6	26.4	26.5
ProoD $\Delta=5$	77.16	76.6	**17.3**	**17.4**	91.5	**19.7**	**19.8**	**100.0**	**22.5**	**23.1**	98.9	**9.0**	**9.0**
In: R.ImgNet		**Flowers**			**FGVC**			**Cars**			**Smooth**		
	Acc	**AUC**	**GAUC**	**AAUC**	**AUC**	**GAUC**	**AAUC**	**AUC**	**GAUC**	**AAUC**	**AUC**	**GAUC**	**AAUC**
Plain	96.34	92.3	0.0	0.5	92.6	0.0	0.0	92.7	0.0	0.1	**98.9**	0.0	8.6
OE	97.10	**96.9**	0.0	0.2	99.7	0.0	0.4	**99.9**	0.0	1.8	98.0	0.0	1.9
ProoD-Disc	-	81.5	76.8	77.3	92.8	89.3	89.6	90.7	86.9	87.3	81.0	74.0	74.8
ProoD $\Delta=4$	**97.25**	**96.9**	**57.5**	**58.0**	**99.8**	**67.4**	**67.9**	**99.9**	**65.7**	**66.2**	98.6	**52.7**	**53.5**

Figure 5.3: OOD detection results of ProoD taken from the paper of Meinke et al. [66]. The best performance of each column is highlighted with bold text. ProoD outperforms most compared architectures. The drop from performance between Cifar-10 to Cifar-100 can be attributed to a multitude of ID classes still being a challenging task to adjust to.

5.6 Out-Of-Distribution Detection Transformer

The first approach to detect OOD data with Vision Transformers (ViTs) was done by Koner et al. [46], which they called Out-Of-Distribution Detection Transformer (OODformer). They took ViT models without changing any of their functionality and used the class-conditioned latent space similarity and the network confidence score based on the contextualized embedding to detect OOD samples. Samples were therefore classified as OOD samples, when the distance to the next closest class mean was too far away or the confidence was below a given threshold.

In order to detect an OOD sample by its latent-space similarity, every ID class has to get a mean (μ_c) assigned in the space. Then a threshold ($t_{distance}$) is defined to allow for some tolerable deviation from the mean. Now every test sample is compared to a combination of mean and threshold. If the test sample lies outside the space around every class mean, it is declared as out-of-distribution, otherwise it gets assigned to the class it is closest to. In their paper they used the Mahalanobis distance [64]:

$$D_c(x_{feat}^{test}) = (x_{feat}^{test} - \mu_c) \sum_{c}^{-1} (x_{feat}^{test} - \mu_c)^T \tag{5.7}$$

The Mahalanobis distance takes advantage of the mean and the covariance ($\sum_c$) of the normally distributed embeddings. Equation 5.7 displays how the Mahalanobis distance is calculated.

As a second criterion that had to be fulfilled to detect an OOD sample, a numerical threshold (t_{conf}) for the softmax confidence score was introduced. As soon as the final softmax score of a test sample was below the threshold for all classes and the distance from the mean was bigger than a distance threshold, they declared it as OOD [46].

To evaluate both approaches, they considered AUPR and the AUROC and were able to beat the compared baselines for all datasets. Similar concerns as presented in the ProoD paper have been raised, regarding the multitude of classes and the sharing of visual features. The OODformer on the other hand is able to overcome this obstacle by a big margin, but still with slightly worse results than for other dataset combinations. Here it has to be noted though, that the OODformer does not provide guaranteed OOD detection performance in contrast to ProoD, because it does not give any information about adversarially altered OOD samples. Extending on this point this thesis takes up the OODformer and evaluates its robust OOD detection capabilities by combining ViTs with the adversarial attack from the ProoD approach.

Chapter 6

Robust Adversarial OOD Detection With Vision Transformer

This thesis follows the general idea of DeVries and Taylor [22], ODIN by Liang et al. [57] or ProoD by Meinke et al. [66] to include a separate detector module next to the classification model, with the sole purpose of distinguishing between ID and OOD data samples. Using a dedicated detector module is one way to detect OOD samples, and Bitterwolf et al. [8] proved that it is equivalent to other approaches. At the time of writing this thesis is the only work using ViTs as dedicated detector models. The difference to ProoD is that the classification model and the detector do not get trained jointly, but in succession. First, the classification model needs to be fully trained on a single dataset, which will from then on be considered as the ID dataset, before the detectors can be trained. The classifier is responsible for the perturbations on the training samples for the detector model, which will be explained in the following section.

Code is available at: https://github.com/LukasDCode/Adversarial_OOD

6.1 Classifier

The gradients of fully trained classifiers are used as a base for the PGD attack to train the detector models. Classifiers have been created from pretrained ViT Base models with image size 224 by 224 pixels. They were pretrained on Imagenet-21k [77] data with 300 episodes and on Imagenet-1k [18] data with 20 000 steps. Every pretrained classifier was then fine-tuned for 60 epochs on one of the 3 datasets, Cifar-10, Cifar-100 and SVHN, resulting in 3 fully trained classifiers. Their final top1 and top5 accuracies are displayed in Table 6.1. Here it has to be noted that it was never the intention to create outstandingly

performing classifiers, rather than a rough starting point for the attack. As a general statement it can be said that the classifier has a hard time with the multitude of classes of Cifar100 and finds few features in the SVHN dataset to orientate itself on.

Classifier	Acc1	Acc5
Cifar-10	88.02	99.53
Cifar-100	61.59	85.46
SVHN	61.90	89.33

Table 6.1: Top 1 and top 5 accuracy of the fully trained classifiers after 60 epochs with their respective dataset.

6.2 Detector

The detector is its own ViT model and is trained on a mixture of samples from the ID and OOD datasets, equally divided in every batch. Each batch is first used in its unperturbed form for training and directly afterward also in its perturbed form. To obtain the best perturbed samples, the batch is subjected to a PGD attack based on the gradients of the already trained classifier. In this step the attack looks for the optimal perturbation of the samples according to the underlying classification model and tries to bring the OOD samples as close as possible to an existing ID class. It does this by repeating the attack multiple times on the same sample, randomly restarting at different points and mixing the biggest perturbations together. ID samples, on the other hand, get enhanced in this step, meaning that the models' confidence towards its true class increases.

The base runs were performed on ViT Tiny models with a depth of 12 encoder layers and therefore also a depth of 12 decoder layers, 3 attention heads, a patch size of 16 by 16 pixels and an image size of 224 by 224 pixels. The attack used 2 random restarts, 5 iterations per restart, a normal noise generator and a maximum allowed perturbation of $\varepsilon = 0.01$. Each permutation of the ID and OOD datasets from the pool of Cifar-10, Cifar-100 and SVHN was trained for 3 epochs. They have only been trained for 3 epochs, because of a trade-off between performance and runtime. Small numbers of epochs seem to be sufficient to return high accuracy values around 99%, except for Cifar-10 and Cifar-100 combinations. These two datasets share visual features among each other resulting in lower accuracies for the final detectors (see Table 6.2). Two additional runs, one with 6 epochs and one with $\varepsilon = \frac{8}{255}$ as the maximum allowed perturbation, were conducted for comparison. All results are presented

in Section 6.5 and discussed in Chapter 7.

6.3 Limitations

The project has several limitations that have not yet been solved. One is on a superficial hardware level, the amount of processing power the code of the PGD attack requires. The second is that the range of possible OOD samples can not be fully covered due to the large number of possibilities. Third, it is ill-defined how much noise can be applied to samples until they are no longer described by their original label.

Regarding the performance issue, the attack has to pass the samples several times through the classifier and each pass takes a small amount of times. The more repeats and iterations the attack is allowed to carry out, the closer the final perturbation gets to the optimal state. Each pass takes about 1 second on one CUDA core on the hardware used during development, but multiplied by the restarts and iterations, an entire attack takes many times the time of a single cycle. In the base runs, 2 restarts and 5 iterations were chosen, resulting in 18 passes ($18 = (2+1)restarts \times (5+1)iterations$) through the classifier for each batch, and therefore about 18 seconds of execution time per batch.

The attack is able to run on CUDA, which speeds it up immensely compared to CPU usage, but it cannot be parallelized across multiple CUDA cores, because it uses the *pytorch autograd* subpackage. If there is a way to parallelize the *autograd* functions, a speedup of a factor of 2 or more could be achieved. Obviously, the more powerful the hardware itself is, the less time each iteration will take.

Apart from the performance, the limited coverage of OOD data cannot reproduce all the possibilities a model would be exposed to in the real world. Reality provides an almost infinite number of different objects that could be used to train for OOD detection, but the datasets used cover only a small fraction of them. In order to achieve true robustness, all real-world data classes would have to be used during the training phase. Since this is currently not feasible due to time and resource constraints, the datasets were chosen to cover two important scenarios. The two scenarios include two datasets that look quite similar due to shared visual features, and two datasets that look dissimilar.

It can be argued that the Cifar-10 and Cifar-100 datasets contain different classes that still share some features with each other, like a horse and a camel both having four legs and sometimes similarly colored fur. On the other hand, all the classes in the Cifar-10 dataset and the digits in the SVHN dataset do not share any obvious features with each other. The robustness results

from both scenarios indicate more of a proof of concept, rather than desirable robustness in a deployed real world application. However, until a dataset with sufficient real world visual feature coverage is released, this proof of concept is the closest we can get to robust adversarial OOD detection with a separate detector model.

Lastly, when noise is added to a sample, it changes its original state. Depending on how much noise is added, it will also change its appearance to a human judge. So far, it is not defined when its label should also change. In the context of achieving adversarial OOD robustness, an OOD sample is perturbed and modified according to the gradients to look more like an ID sample. But how much noise can actually be applied before the sample is labeled differently by a human judge? And is a human judge even the decision-maker under this circumstance? This can be illustrated with a visual example.

Figure 4.2 displays two rows of images. The top left image shows a turtle on a log and the bottom left image shows a cat in a cardboard box. Noise is gradually added to both images until the top image shows a slightly psychedelic looking bird, most likely a peacock, and the bottom image shows two dogs in a cardboard box, by human perception standards. In between the outermost images are the intermediate steps with progressively more noise. It is hard to tell at which point the image should actually be labeled differently, also because there are various versions in between each of the images.

This problem must be kept in mind when dealing with adversarially altered data, although it is not a major issue in this project. The base runs only use a maximum allowed perturbation of $\varepsilon = 0.01$, which returns perturbed images that are virtually unchanged by the standards of human perception. Even the additional run with an ε of $\frac{8}{255}$ does not perturb the samples enough to actively change the label.

6.4 Robustness

To achieve robustness, the detector should be trained to correctly detect OOD samples that are attempted to be inserted into the ID distribution by combining them with noise generated according to the gradient step of the classifier. The more the detector can correctly classify these perturbed samples with a high probability, the better its robustness will be in the end.

Other adversarial scenarios using the gradients have implications other than the achievement of robustness. When the attack takes ID data and goes along the gradients of the classifier, the data will be brought closer to the expectations of the classifier. If the classifier has been trained well enough this can be used to remove noise from samples or sharpen images, depending on the use

case of the classifier.

When the attack moves the noise against the gradients of the classifier with an ID sample, the samples will eventually be falsely assigned to another class of the distribution. Goodfellow et al. [28] did a similar experiment to confuse the model into thinking that the example of a panda is no longer a panda (see Figure 4.1). This can be used to simulate noise for training that would otherwise be difficult, time-consuming or expensive to collect. For example, this noise can be used to simulate fog or precipitation in the context of road sign detection for autonomous driving, to make the car reliably detect these signs in more challenging conditions.

Lastly, the attack could also take OOD samples and move against the gradient of the classifier. But this scenario does not provide any applicable purpose as of now, because it only moves the sample further away from the classifier's expectations, making it even more certain that the samples do not belong to any of the ID classes. Only the procedure of taking OOD data and going along the gradients of the classifier, to eventually turn the samples into ID data, can be used to train for robustness. The training itself and the results are discussed in the next section.

6.5 Results

Experiments were carried out on six detector modules with each permutation of the three datasets as ID and OOD. Two additional runs were performed that differed from the base runs by using 6 training epochs instead of 3 and a higher ε of $\frac{8}{255}$ instead of 0.01. The classifiers used for the attack (see Section 6.1) act as a normal baseline, without claiming to outperform state-of-the-art classifiers, and can be replaced individually.

For the base runs training was performed for 3 epochs for each permutation, with ViT Tiny models (more information in Section 6.2. Execution was performed on one CUDA core due to the parallelization limitations mentioned in Section 6.3. The attack used a maximum possible perturbation of $\varepsilon = 0.01$, 5 iterations, 2 restarts (= 18 cycles per batch) and normal noise tailored to the returned gradients. Figure 6.1 shows the training pipeline for the detector with clean and perturbed input samples. Note that each iteration takes about 18 seconds to complete, and each epoch takes about 18 hours. The accuracy, AUROC and AUPR results of the six base runs are shown in Table 6.2.

In addition, a run with a stronger $\varepsilon = \frac{8}{255} \simeq 0.031$ was performed to see if more noise could reduce the high AUROC results. $\varepsilon = \frac{8}{255}$ has been selected because it is another popular standard in the adversarial robustness literature [66]. The results of the run with the higher ε are shown in Table 6.3.

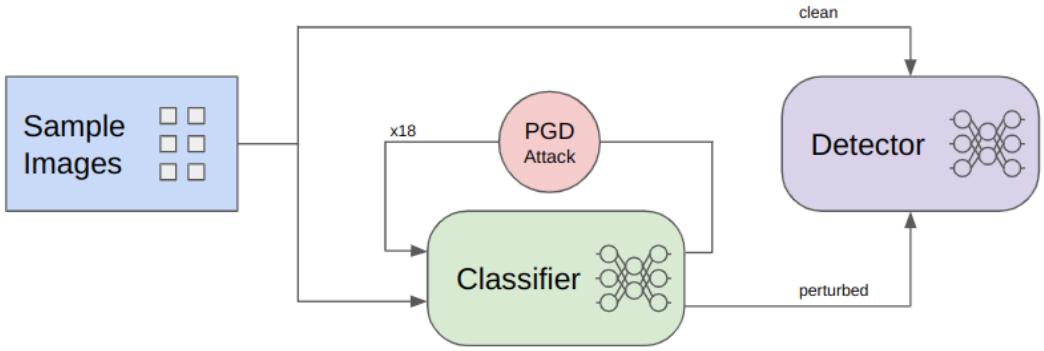


Figure 6.1: Visualization of the detector training pipeline, including clean and perturbed training samples, with 18 perturbation cycles per batch.

ID	OOD					
	Accuracy	AUROC	AUPR	Accuracy	AUROC	AUPR
Cifar 10	Cifar 100			SVHN		
	63.38	72.83	75.64	**99.31**	**99.97**	**99.97**
Cifar 100	Cifar 10			SVHN		
	63.71	73.06	74.08	**98.89**	**99.97**	**99.98**
SVHN	Cifar 10			Cifar 100		
	99.18	**99.99**	**99.99**	98.83	99.97	99.97

Table 6.2: Accuracy, AUROC and AUPR results for the base runs with each permutation of the three datasets. The best accuracy, AUROC and AUPR values for each detector with the same ID dataset are shown in bold.

In addition, Figures 6.6, 6.7, 6.8 and 6.9 show the attention maps of one ID and one OOD sample from this detector.

Since the results on two datasets with similar optical characteristics were poor, a run with more training epochs was performed to see if the results could be improved with more training. Therefore, a detector with Cifar-10 as the ID and Cifar-100 as the OOD dataset was trained for 6 epochs, twice the number of base runs. The accuracy, AUROC or AUPR results are displayed in Table 6.4.

ID	OOD — SVHN		
	Accuracy	AUROC	AUPR
Cifar-100	98.83	99.96	99.97

Table 6.3: Accuracy, AUROC and AUPR results for the special run with an $\varepsilon = \frac{8}{255}$, where Cifar-100 is the ID and SVHN is the OOD dataset.

ID	OOD — Cifar100		
	Accuracy	AUROC	AUPR
Cifar-10	65.40	76.33	78.53

Table 6.4: Accuracy, AUROC and AUPR results for the special run with 6 eopchs where Cifar-10 is the ID and Cifar-100 the OOD dataset.

6.6 Attention Results

Since the special feature of Transformers is the attention between different tokens, visualizing this attention provides a better understanding of their perception. The following figures show the decoder's attention of the Transformer-specific class tokens and three additional tokens at different encoder layers of the architecture. Because ViTs work with images, each image is divided into 14 by 14 patches, which act as tokens, each with a size of 16 by 16 pixels. Higher attention from one patch towards a particular other patch of the image is highlighted in a brighter yellow, and as the attention to different patches decreases, the yellow fades to blue.

The following examples with attention maps are from a decoder with Cifar-100 as its ID data and Cifar-10 as its OOD data. The image of a turtle belongs to the ID dataset. In the clean image (Figure 6.2), the decoder is able to direct the attention of the class token to the turtle and away from the white background. In the perturbed image (Figure 6.3), on the other hand, the decoder mostly shows the same attention, with slight deviations towards patches that do not contain the turtle. This deviation can be seen by the brighter patches at the bottom of the class token attention map in the upper left. Overall, the attention differences are small, as the perturbation is only marginal with $\varepsilon = 0.01$.

When comparing a clean and a perturbed OOD image, the attention differences seem to disappear. Figure 6.4 and Figure 6.5 are from the same decoder as above, but originate from the OOD dataset, in this case Cifar-10. The two figures show almost no difference in their attention maps. This can be attributed to the fact that the classifier, which is responsible for the perturbation, does not know what a bird is, neither in its clean nor in its perturbed

Figure 6.2: Attention maps of the detector's 12th encoder layer of a clean ID image of a turtle.

Figure 6.3: Attention maps of the detector's 12th encoder layer of a perturbed ID image of a turtle with $\varepsilon = 0.01$.

form. It therefore places the attention according to visual peculiarities such as light and dark patches or same-colored patches for both the clean and the perturbed versions.

The following examples with attention maps are from a decoder with Cifar-100 as its ID data and SVHN as its OOD data. As ε increases, the differences in the attention maps become visible. Figure 6.6 shows an unperturbed image of a dolphin from the ID dataset, while Figure 6.7 shows the same dolphin with a maximum perturbation of $\varepsilon = \frac{8}{255}$. The class token attention in the upper left of the clean version shows more attention towards the patches containing the dolphin, while the class token attention of the perturbed version pays less attention to the dolphin. Similar to the example with the turtle above, does the classifier know quite well what a dolphin is and is therefore able to perturb it effectively. As expected, the larger the ε, the more attention is drawn away from the object, comparable to the example with the adversarially altered turtle.

One OOD sample, containing the number three, is shown in Figure 6.8 in its clean version and in Figure 6.9 in its perturbed version ($\varepsilon = \frac{8}{255}$). The attention map of the class tokens from the perturbed version has less focus on the background of the image and on the object itself. This indicates that with higher perturbations it is even possible to attack the attention of OOD samples, in contrast to the previous example with the bird. The classifier does not know what the number three is, but because the attack has a larger range for alterations in the image, it can actively change the location where the attention is focused at.

Figure 6.4: Attention maps of the detector's 12th encoder layer of a clean OOD image of a bird.

Figure 6.5: Attention maps of the detector's 12th encoder layer of a perturbed OOD image of a bird with $\varepsilon = 0.01$.

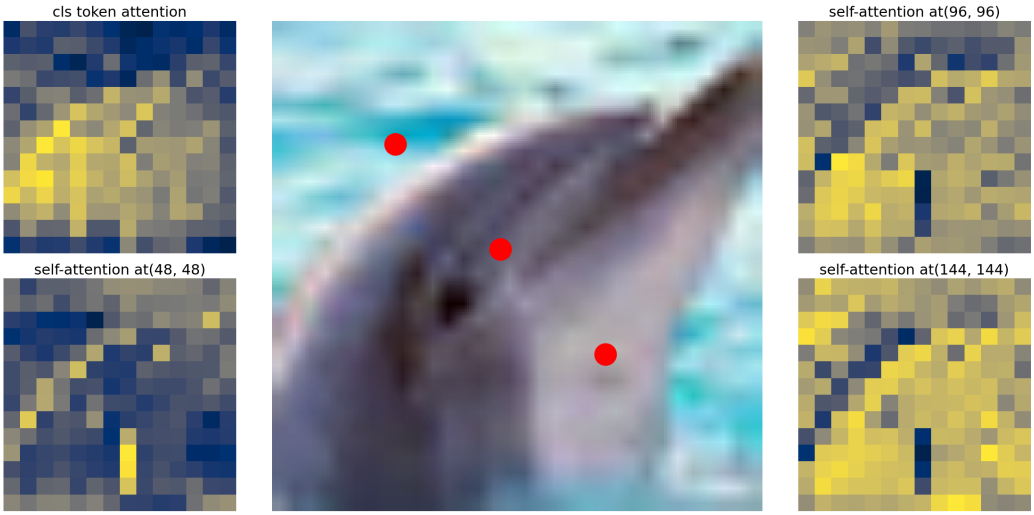


Figure 6.6: Attention maps of the detector's 11th encoder layer of a clean ID image of a dolphin.

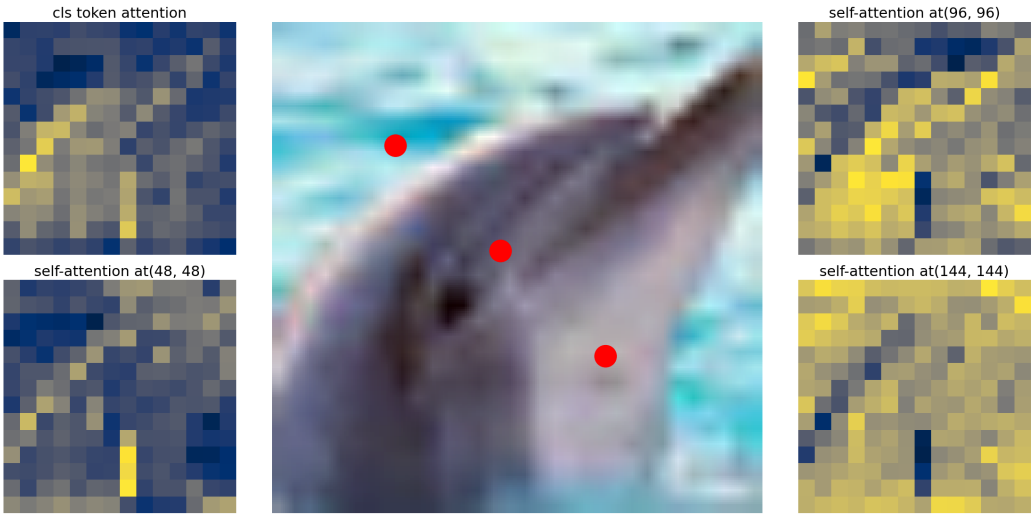


Figure 6.7: Attention maps of the detector's 11th encoder layer of a perturbed ID image of a dolphin with $\varepsilon = \frac{8}{255}$.

Figure 6.8: Attention maps of the detector's 2nd encoder layer of a clean OOD image of the number 3.

Figure 6.9: Attention maps of the detector's 2nd encoder layer of a perturbed OOD image of the number 3 with $\varepsilon = \frac{8}{255}$.

Chapter 7

Discussion of the Results

7.1 Base Runs

The results presented in Table 6.2 provide high accuracies for permutations of dataset without shared features, but lower accuracies for datasets with shared features. Accuracies between 98.8% and 99.31% are close to perfect for all runs including the SVHN dataset as either the ID or OOD dataset. The perturbation with a maximum of $\varepsilon = 0.01$ does not interfere with high accuracies. In conclusion, it seems to be easier for the detector to assign distinctive visual features to different classes, even with only 3 training epochs.

Accuracy, AROC, and AUPR are all derived from the maximum final class prediction probability. As discussed in Section 5.1.3, the AUROC is recommended when each class is predicted similarly often. The AUPR, on the other hand is recommended when there is no comparable balance between the classes [17]. Since the datasets are equally distributed, it is appropriate to focus on the AUROC. However, the AUPR closely follows the AUROC in all the results presented.

Furthermore, ViT models maintain high scores despite numerous ID classes. This result even contradicts the concerns raised in Prood [66] and OODformer [46]. The results of comparing ProoD with the results of this thesis showed that while ProoD showed performance drops when handling numerous classes, the ViTs are able to overcome this limitation without the need for any special adjustments. This suggests that ViTs may be more suitable for tasks involving many classes.

ProoD has an AUROC value of 98.3% when using Cifar-10 as the ID and SVHN as the OOD dataset (see Figure 5.3). The base runs of this thesis with the same dataset combination provide a similar value of 99.97%. The contrast

can be emphasized by the AUROC values when Cifar-100 was used as the ID dataset. ProoD provides an AUROC of 91.5%, while the corresponding run of this thesis provides an AUROC of 99.97%, which is on level with the value with the AUROC of Cifar-10.

For datasets sharing optical characteristics, such as Cifar-10 and Cifar-100 in both combinations, the accuracy of the base runs is significantly lower in the range between 63% and 64%. The low accuracy is attributable to the datasets sharing visual characteristics with each other, making it more difficult for the model to find clear distinctions in addition to the adversarial change.

In both cases, where SVHN was used as the OOD dataset, do the AUROC values exceed state-of-the-art performance, as can be compared with the results in Figure 5.3. Again, this due to the optical differences of the classes in the different datasets and to the ability of the ViTs to assimilate these features. This explains why the Cifar-10 and Cifar-100 dataset combinations produce worse results for all three individual metrics.

7.2 Additional Runs

To check if the size of the ε and the number of epochs has an impact on the results, two additional runs were performed. The run with 6 epochs provides actual differences compared to its counterpart in the base runs. Accuracy increases by 2.02 percentage points, AUROC by 3.5, and AUPR by 2.89. With increases in all three metrics, it can be hypothesized that more epochs still have an effect on dataset combinations that share visual features, and that the upper bound is not reached at 6 epochs. Considering that the detector trained with twice the number of epochs, the increase is much more modest.

The run with an ε of $\frac{8}{255}$ provided no significant differences in the resulting metrics compared to the base run with an ε of 0.01. The run was performed with Cifar100 as the ID dataset and SVHN as the OOD dataset. The negligible differences of 0.01% in the AUROC and AUPR scores and 0.06% in the accuracy are attributed to the standard measurement variability in ML applications. The upcoming Section 7.3 will visualize the attention differences of both ε variants. In advance, it can be said that even though the larger perturbation does not change the resulting metrics, it does affect attention.

7.3 Attention Maps

Looking at the attention of the ViTs, we can see that the ID samples in their clean and perturbed variants produce slightly different attention maps. In the clean sample (see Figure 6.2), the attention of the class token is predominantly

on the object itself. When noise is introduced with the attack (see Figure 6.3), the attention of the class token remains mainly on the object, but shifts slightly to the background. This shift is due to the classifier, which is responsible for noise generation.

The classifier is well aware of what class is represented in the image and, in conjunction with the attack, tries to get as far away from that baseline as possible. This results in an effective change in the image, which is not visible to the human eye due to the maximum allowed perturbation of $\varepsilon = 0.01$. Because of this alteration, the detector cannot focus its full attention on the object, but increasingly shifts its focus to the background. These results indicate that the attack is successful and that a higher noise threshold would allow for a greater obfuscation of the detectors' attention.

Considering the attention maps of the run with the higher ε, the attack is able to shift attention even further away from the main object of the image. This shift in attention can be seen in the attention maps of the class tokens in the dolphin example (see Figures 6.6 and 6.7).

Attention towards OOD samples, on the other hand, appears to be minimally affected by the perturbations, as their attention maps look virtually identical (see Figure 6.4 and Figure 6.5). If the datasets share common optical features, this outcome can be attributed to the classifier's inability to understand the content of the image. Lacking this understanding and limited by the maximum allowed perturbation, the attack and the classifier are also unable to effectively perturb the sample effectively to fool the detectors' attention.

Looking at an OOD example where the two datasets do not share any visual features and the maximum perturbation was set to $\varepsilon = \frac{8}{255}$, we can see that the attack is more effective. An image with the number three from the SVHN dataset was perturbed according to the gradients of a classifier that was trained on Cifar-100 images. The attention map of the class token of the perturbed image (see Figure 6.9) shows that the attack is mostly able to divert the attention away from the number three, when compared to the clean version (see Figure 6.8). Note that the accuracy of the detector is still high, and it is still able to correctly classify adversarially altered samples, even though the attention has been diverted from the main object.

In conclusion, there is a difference in the effectiveness of the attack, depending on whether the image contains visual features that may look similar to the features on which the classifier was trained. If the features are similar, the accuracy as well as the AUROC and AUPR metrics decrease. On the other hand, if the features are different, the detector is able to achieve state-of-the-art robust OOD detection performance with little training. Even small amounts of noise already start to interfere with the attention. The more noise is added to the samples, the more attention suffers.

Chapter 8

Conclusion

ViTs used as separate detector modules are inherently capable of robustly detecting adversarially altered OOD data with excellent results and little training. This is especially true when the ID and OOD datasets have no visual similarities. With an overlap in visual features, OOD detection performance decreases, and more training is required to overcome this decrease. This highlights the importance of taking the optical similarity of the datasets into account when evaluating the performance of OOD detection with ViTs.

Furthermore, they are also able to overcome the performance gap in handling a large number of ID classes, in contrast to other architectures, suggesting their potential for handling large-scale image classification tasks. Where deep CNNs struggle with the large number of Cifar-100 classes and lose their prediction accuracy, the ViTs are able to maintain their high level across all metrics.

For all experiments performed with a maximum noise of either $\varepsilon = 0.01$ or $\varepsilon = \frac{8}{255}$, the accuracy and AUROC values were similar, indicating that ViTs are also able to overcome a larger amount of noise compared to deep CNNs. The ε values represent two sizes that are commonly used in the field of adversarial attacks, and ViTs are able to handle both exceptionally well. It remains unclear how much noise in the attack ViTs can maximally handle before performance drops significantly and the Attention mechanism is overwhelmed by the perturbed input.

Future work could focus on determining the maximum amount of perturbation before performance degrades and the Attention mechanism is completely disrupted. Because the approach is computationally demanding, it also leaves room for the development of more efficient methods for adversarial training. Integrating techniques that have improved OOD detection capabilities for conventional models, such as Interval Bound Propagation, Temperature Scaling or a systematic combination of losses could also provide further improvements

for Vision Transformers, but are beyond the scope of this thesis.

In addition, the ViT's ability to detect adversarial OOD samples is uncertain when deployed in a real-world scenario where an attacker does not have access to the model's gradients and losses. However, it is recommended to use them instead of deep CNNs according to the results presented in this thesis. The ViT's out-of-the-box robust adversarial OOD detection performance is impressive, indicating another area in deep learning that is captured by the capabilities of Transformers.

Appendix A

Appendix

A.1 Code

Code is available at: https://github.com/LukasDCode/Adversarial_OOD

A.2 Cifar-10 and Cifar-100 Classes

A.2.1 Cifar-10 Classes

1. airplane
2. automobile
3. bird
4. cat
5. deer
6. dog
7. frog
8. horse
9. ship
10. truck

A.2.2 Cifar-100 Classes

1. aquatic mammals
    - beaver, dolphin, otter, seal, whale
2. fish
    - aquarium fish, flatfish, ray, shark, trout
3. flowers
    - orchids, poppies, roses, sunflowers, tulips
4. food containers

- bottles, bowls, cans, cups, plates

5. fruit and vegetables
   - apples, mushrooms, oranges, pears, sweet peppers
6. household electrical devices
   - clock, computer keyboard, lamp, telephone, television
7. household furniture
   - bed, chair, couch, table, wardrobe
8. insects
   - bee, beetle, butterfly, caterpillar, cockroach
9. large carnivores
   - bear, leopard, lion, tiger, wolf
10. large man-made outdoor things
    - bridge, castle, house, road, skyscraper
11. large natural outdoor scenes
    - cloud, forest, mountain, plain, sea
12. large omnivores and herbivores
    - camel, cattle, chimpanzee, elephant, kangaroo
13. medium-sized mammals
    - fox, porcupine, possum, raccoon, skunk
14. non-insect invertebrates
    - crab, lobster, snail, spider, worm
15. people
    - baby, boy, girl, man, woman
16. reptiles
    - crocodile, dinosaur, lizard, snake, turtle
17. small mammals
    - hamster, mouse, rabbit, shrew, squirrel
18. trees
    - maple, oak, palm, pine, willow
19. vehicles 1
    - bicycle, bus, motorcycle, pickup truck, train
20. vehicles 2
    - lawn-mower, rocket, streetcar, tank, tractor

Acronyms

3D Three-dimensional.

AUC Area Under the Curve.
AUPR Area Under the Precision-Recall Curve.
AUROC Area Under the Receiver-Operator Characteristic Curve.

BERT Bidirectional Encoder Representations from Transformers.

CNN Convolutional Neural Network.
CPU Central Processing Unit.
CUDA Compute Unified Device Architecture.
CV Computer Vision.

FPRN False Positive Rate at N%.

GAN Generative Adversarial Network.
GAUC Guaranteed Area Under the Receiver-Operator Characteristic Curve.
GOOD Guaranteed Out-of-distribution Detection.
GPT Generative Pre-trained Transformer.
GRU Gated Recurrent Unit.

IBP Interval Bound Propagation.
ID In-distribution.

JPEG Joint Photographic Expert Group.

LSTM Long Short-Term Memory.

ML Machine Learning.
MLP Multi-Layer Perceptron.
MNIST Modified National Institute of Standards and Technology.

OE Outlier Exposure.
OOD Out-of-distribution.
OODformer Out-of-distribution Detection Transformer.

PGD Projected Gradient Descent.
PNG Portable Network Graphics.
ProoD Provable Out-of-distribution Detector.

ReLU Rectified Linear Unit.
RGB Reg Green Blue.
RL Reinforcement Learning.
RNN Recurrent Neural Network.

SVHN Street View House Numbers.

UNECE United Nations Economic Commission for Europe.

ViT Vision Transformer.

List of Figures

List of Tables

Bibliography

[1] Mahdieh Abbasi and Christian Gagné. Robustness to adversarial examples through an ensemble of specialists. *arXiv preprint arXiv:1702.06856*, 2017.

[2] Nikolas Adaloglou. How the vision transformer (vit) works in 10 minutes: an image is worth 16x16 words. https://theaisummer.com/vision-transformer.

[3] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. Face aging with conditional generative adversarial networks. In *2017 IEEE international conference on image processing (ICIP)*, pages 2089–2093. IEEE, 2017.

[4] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *Bmvc*, volume 1, page 3, 2016.

[5] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, 2006.

[6] Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. Better plain vit baselines for imagenet-1k. *arXiv preprint arXiv:2205.01580*, 2022.

[7] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[8] Julian Bitterwolf, Alexander Meinke, Maximilian Augustin, and Matthias Hein. Breaking down out-of-distribution detection: Many methods based on ood training data estimate a combination of the same

core quantities. In *International Conference on Machine Learning*, pages 2041–2074. PMLR, 2022.

[9] Julian Bitterwolf, Alexander Meinke, and Matthias Hein. Certifiably adversarially robust detection of out-of-distribution data. *Advances in Neural Information Processing Systems*, 33:16085–16095, 2020.

[10] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[11] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[12] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. *arXiv preprint arXiv:1609.07093*, 2016.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[14] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[15] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[16] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108, 2004.

[17] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[19] Yao Deng, Xi Zheng, Tianyi Zhang, Chen Chen, Guannan Lou, and Miryung Kim. An analysis of adversarial attacks and defenses on autonomous driving models. In *2020 IEEE international conference on pervasive computing and communications (PerCom)*, pages 1–10. IEEE, 2020.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[21] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[22] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[23] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. Vision transformer and mlp-mixer architectures. https://github.com/google-research/vision_transformer#available-vit-models.

[24] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[25] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[26] Micah Goldblum, Avi Schwarzschild, Ankit Patel, and Tom Goldstein. Adversarial attacks on machine learning systems for high-frequency trading. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

[27] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *stat*, 1050:10, 2014.

[28] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[29] David Marvin Green, John A Swets, et al. *Signal detection theory and psychophysics*, volume 1. Wiley New York, 1966.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] Kai Heinrich, Patrick Zschech, Björn Möller, Lukas Breithaupt, and Johannes Maresch. Objekterkennung im weinanbau – eine fallstudie zur unterstützung von winzertätigkeiten mithilfe von deep learning. *HMD Praxis der Wirtschaftsinformatik*, 56:964–985, 10 2019.

[33] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[34] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

[35] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *Advances in Neural Information Processing Systems*, 2015.

[36] Yiyu Hong, Juyong Lee, and Junsu Ko. A-prot: protein structure modeling using msa transformer. *BMC bioinformatics*, 23(1):1–11, 2022.

[37] Neil Houlsby and Dirk Weissenborn. Transformers for image recognition at scale. https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html.

[38] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10951–10960, 2020.

[39] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[40] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[41] McCulloch JL. A logical calculus of ideas immanent in nervous activity. *Bull. of Math. Biophysics*, 5:115–133, 1943.

[42] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[43] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

[44] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020.

[45] Oscar Knagg. Know your enemy - how you can create and defend against adversarial attacks. https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3.

[46] Rajat Koner, Poulami Sinhamahapatra, Karsten Roscher, Stephan Günnemann, and Volker Tresp. Oodformer: Out-of-distribution detection transformer. *arXiv preprint arXiv:2107.08976*, 2021.

[47] Ale Krizhevsky. The cifar-10 dataset and the cifar-100 dataset. https://www.cs.toronto.edu/ kriz/cifar.html.

[48] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *University of Toronto*, 2009.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[50] George Lawton. Transformer neural networks are shaking up ai. https://www.techtarget.com/searchenterpriseai/feature/Transformer-neural-networks-are-shaking-up-AI.

[51] James Le. The 5 computer vision techniques that will change how you see the world. https://heartbeat.comet.ml/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b.

[52] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[53] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325*, 2017.

[54] Peter Lee. Learning from tay's introduction. https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/.

[55] Dominik Lewy and Jacek Mańdziuk. An overview of mixing augmentation methods and augmentation strategies. *Artificial Intelligence Review*, pages 1–59, 2022.

[56] Zewen Li, Wenjie Yang, Shouheng Peng, and Fan Liu. A survey of convolutional neural networks: Analysis, applications, and prospects. *CoRR*, abs/2004.02806, 2020.

[57] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[58] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

[59] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. *Advances in neural information processing systems*, 29, 2016.

[60] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647, 2005.

[61] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *CEAS*, volume 2005, 2005.

[62] Donald MacKenzie. A material political economy: Automated trading desk and price prediction in high-frequency trading. *Social Studies of Science*, 47(2):172–194, 2017.

[63] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[64] P Chandra Mahalanobis. Analysis of race-mixture in bengal. *Journal and Proceedings of the Asiatic Society of Bengal*, 23:301–333, 1925.

[65] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[66] Alexander Meinke, Julian Bitterwolf, and Matthias Hein. Provably robust detection of out-of-distribution data (almost) for free. *arXiv preprint arXiv:2106.04260*, 2021.

[67] Rostislav Nedelchev. An overview of the best transformers - bert, gpt-3 and co. https://www.alexanderthamm.com/en/blog/transformer-at-a-glance-bert-gpt-3-and-co.

[68] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *Conference on Neural Information Processing Systems*, 2011.

[69] Chuong Ngo. Transformers: The nuts and bolts. https://www.revistek.com/posts/transformer-architecture.

[70] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. *arXiv e-prints*, pages arXiv–1602, 2016.

[71] Guim Perarnau, Joost Van De Weijer, Bogdan Raducanu, and Jose M Álvarez. Invertible conditional gans for image editing. *arXiv preprint arXiv:1611.06355*, 2016.

[72] Michael Phi. Illustrated guide to transformers- step by step explanation. https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0.

[73] Stanislav Pidhorskyi, Ranya Almohsen, and Gianfranco Doretto. Generative probabilistic novelty detection with adversarial autoencoders. *Advances in neural information processing systems*, 31, 2018.

[74] Magnus Pierrau. Evaluating unsupervised methods for out-of-distribution detection on semantically similar image data, 2021.

[75] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[76] Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *CoRR*, abs/1605.05396, 2016.

[77] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972*, 2021.

[78] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. https://github.com/Alibaba-MIIL/ImageNet21K, 2021.

[79] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[80] Abhishek Roy, Anshuman Chhabra, Charles A Kamhoua, and Prasant Mohapatra. A moving target defense against adversarial machine learning. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pages 383–388, 2019.

[81] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[82] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[83] John Schulman, Barret Zoph, Christina Kim, Jacob Hilton, and Jacob Menick. Chatgpt: Optimizing language models for dialogue. https://openai.com/blog/chatgpt/.

[84] Oscar Schwartz. In 2016, microsoft's racist chatbot revealed the dangers of online conversation. https://spectrum.ieee.org/in-2016-microsofts-racist-chatbot-revealed-the-dangers-of-online-conversation.

[85] Andreas Sedlmeier, Thomas Gabor, Thomy Phan, Lenz Belzner, and Claudia Linnhoff-Popien. Uncertainty-based out-of-distribution detection in deep reinforcement learning. *arXiv preprint arXiv:1901.02219*, 2019.

[86] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[87] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[88] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

[89] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[90] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[91] Vagan Terziyan and Anton Nikulin. Ignorance-aware approaches and algorithms for prototype selection in machine learning. *arXiv preprint arXiv:1905.06054*, 2019.

[92] Martin Thoma. The roc space for a "better" and "worse" classifier. https://en.wikipedia.org/wiki/Receiver_operating_characteristic.

[93] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34:24261–24272, 2021.

[94] Antonio Torralba, Rob Fergus, and Bill Freeman. Tiny images. https://groups.csail.mit.edu/vision/TinyImages.

[95] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

[96] UNECE. Un regulation extends automated driving up to 130 km/h in certain conditions. https://unece.org/sustainable-development/press/un-regulation-extends-automated-driving-130-kmh-certain-conditions.

[97] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[98] Michelle Venables. An overview of computer vision. https://towardsdatascience.com/an-overview-of-computer-vision-1f75c2ab1b66.

[99] Huikai Wu, Shuai Zheng, Junge Zhang, and Kaiqi Huang. GP-GAN: towards realistic high-resolution image blending. *CoRR*, abs/1703.07195, 2017.

[100] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *CoRR*, abs/1610.07584, 2016.

[101] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *International Conference on Computer Vision (ICCV)*, 2019.

[102] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris N. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1612.03242, 2016.

[103] He Zhang, Vishwanath Sindagi, and Vishal M Patel. Image de-raining using a conditional generative adversarial network. *IEEE transactions on circuits and systems for video technology*, 30(11):3943–3956, 2019.

[104] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[105] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 132–142. IEEE, 2018.

[106] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[107] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018.