学校代号 10532 学 号 LY2019093
分 类 号 TP391 密 级 普通

湖南大學
HUNAN UNIVERSITY

硕士学位论文

使用随机像素空间扰动的对抗样本检测

学位申请人姓名 GRIVEAU Jordan
培 养 单 位 信息科学与工程学院
导师姓名及职称 Professor JIANG Bin
学 科 专 业 计算机科学与技术
研 究 方 向 第三类永动机
论 文 提 交 日 期 2022 年 3 月 27 日

学校代号： 10532

学　　号： LY2019093

密　　级： 普通

湖南大学硕士学位论文

使用随机像素空间扰动的对抗样本检测

学位申请人姓名： GRIVEAU Jordan

培　养　单　位： 信息科学与工程学院

导师姓名及职称： Professor JIANG Bin

专　业　名　称： 计算机科学与技术

论文提交日期： 2022 年 3 月 27 日

论文答辩日期： xxxx 年 x 月 xx 日

答辩委员会主席： 待定

Adversarial Examples Detection Using Random Pixel-Space Perturbation

By

GRIVEAU Jordan

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Computer Science and Technology

in the

Graduate School

of

Hunan University

Supervisor

Professor JIANG Bin

March, 2022

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密 □，在_____ 年解密后适用本授权书。

2、不保密 □。

（请在以上相应方框内打“✓”）

作者签名：　　日期：　　年　　月　　日

导师签名：　　日期：　　年　　月　　日

摘 要

与其他机器学习算法一样，神经网络很容易受到对抗性示例的影响，即包含特制扰动的输入，其唯一目的是欺骗网络进行错误分类。另一方面，由于对神经网络架构的改进，众所周知，它们对输入中的随机扰动更加稳健。这种对随机扰动的鲁棒性促使我提出了一种易于部署的对抗样本检测方法，该方法可以测量将随机噪声应用于输入图像之前和之后的预测不一致性。在三个流行基准 (Dogs vs. Cats、CIFAR-10 和 ImageNet) 的子集上评估该方法表明，它在针对更高分辨率图像的各种攻击中实现了高对抗样本检测性能。我的方法的主要优点是简单，计算成本低，并且不需要任何关于所使用攻击的先验知识，这使得我的方法很容易集成到其他防御框架中.

我提出了一种检测对抗样本的新方法，该方法基于在输入上故意引入不同强度的高斯噪声。然后，计算两个分数，以评估模型在应用噪声之前和之后在不同强度下的预测差异。这种方法的优点是检测效率不依赖于关于所使用攻击的先验知识，因此可以应用于广泛的攻击和不同的对抗性扰动预算。此外，与最先进的防御或检测方法相反，我的方法对计算的要求很低，因为不需要对模型参数进行训练或优化。最后，此方法可以与其他检测方法结合使用，以提高整体应用程序的性能。我提出的方法是在研究了将加性高斯噪声应用于正常图像和对抗样本的影响并观察到据我所知并且在这项工作时尚未在之前的工作中讨论过的差异之后出现的。

关键词：深度学习；计算机视觉；对抗性例子

Abstract

Like other machine learning algorithms, neural networks have been vulnerable to adversarial examples, i.e., inputs containing specifically crafted perturbations whose only objective is to fool a network into misclassification. On the other hand, because of the improvements made on neural network architectures, they are known to be much more robust to random perturbations in the input. This robustness to random perturbations motivated me to propose an easy-to-deploy adversarial example detection method that measures prediction inconsistencies before and after applying random noise to an input image. Evaluating the method on subsets of three popular benchmarks (Dogs vs. Cats, CIFAR-10, and ImageNet) shows that it achieves high adversarial example detection performance for various attacks on higher resolution images. The main advantages of my approach are its simplicity, low computational cost, and the fact that it does not require any prior knowledge of the attack used, which makes it easy to integrate my method into other defense frameworks.

Key Words: Deep Learning; Computer Vision; Adversarial Examples

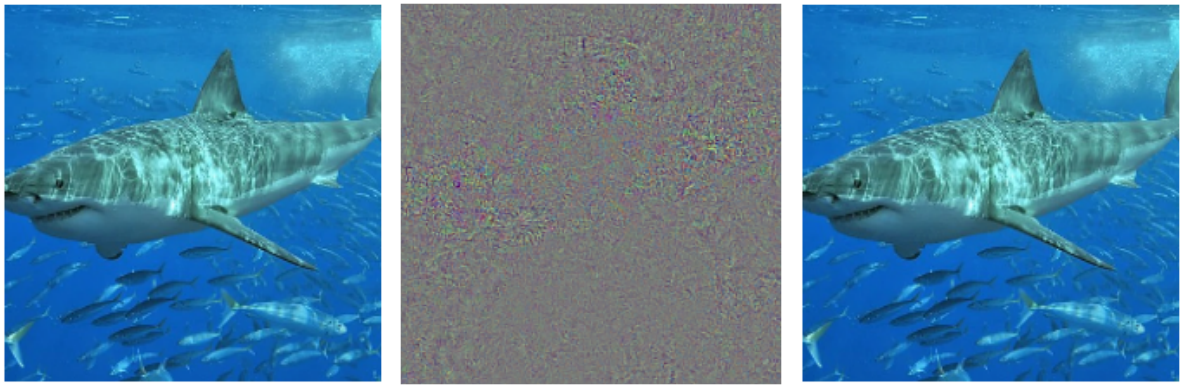
目 录

插图索引

附表索引

第 1 章 Introduction

1.1 Motivation



(a) (b) (c)

图 1.1 Normal image (A), adversarial perturbation (B), adversarial example (C). The model accurately classifies the normal image as a ”white shark” while misclassifying the adversarial example as a ”prairie chicken.”

Deep neural networks (DNNs) have succeeded on numerous tasks, from image and speech recognition ([5,6]) to self-driving cars ([7]) and beating the world champion at the game of Go ([8]). However, despite achieving state-of-the-art performance in various domains,[9] showed that deep neural networks are vulnerable to adversarial examples, i.e., inputs containing a carefully crafted perturbation that causes an image classification model to make the wrong predictions. Researchers demonstrated this phenomenon to be observable in computer vision and speech recognition when[1] showed that a perturbed audio waveform could make a speech-to-text model drastically change the transcription, as seen in figure 1.2.

One surprising aspect of these adversarial examples is that, as seen in figure 1.4, the perturbation needed to fool a model into misclassification can be so small that it is unnoticeable to a human observer.[2] even demonstrated in a recent study that modifying a single pixel from the input vector can be enough to fool a neural network, as shown in figure 1.3.

On the other hand, with the constant improvement of neural network architectures, ([10–12]) and regularization techniques such as dropout by[13], neural networks have become more and more robust against randomly perturbed inputs as demon-

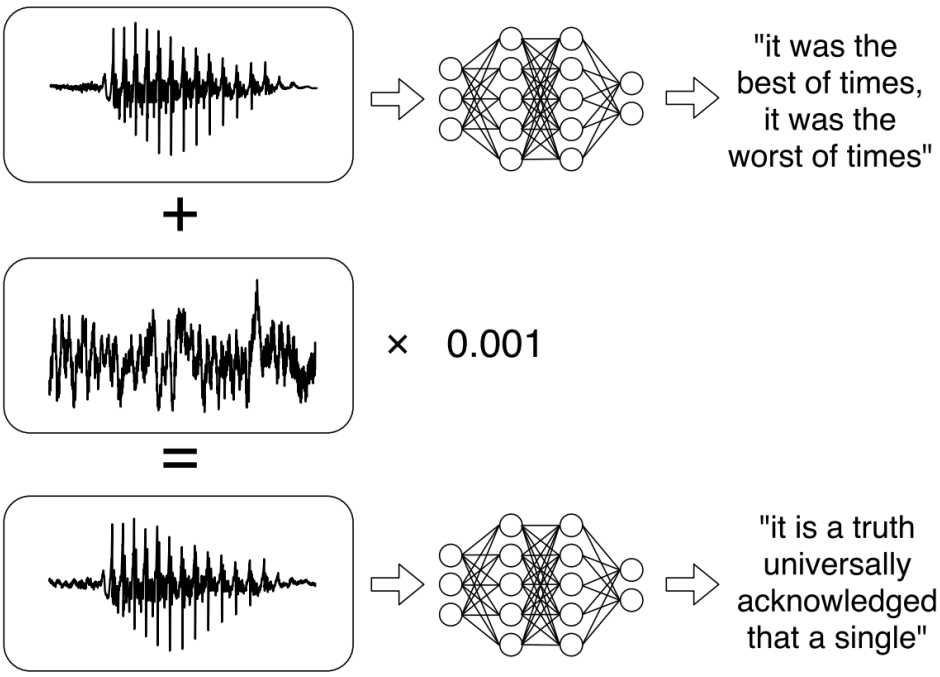


图 1.2 Unnoticeable audio waveform added to an audio recording changes the transcription by the model drastically, by[1].

strated by[14]. Thus, as long as the random perturbations are not too significant, e.g., rotation of the image, compression deterioration (e.g., JPEG), brightness or contrast shift, the neural network will still accurately predict the images.

This divergence of robustness displayed by neural networks when facing random and adversarial perturbations motivated me to experiment with the robustness of adversarial examples themselves by intentionally adding a random perturbation on these modified samples.

1.2 Main Contribution

I propose a novel method for detecting adversarial examples based on intentionally introducing Gaussian noise on the input with varying intensity. Then, two scores are computed that evaluate the difference of prediction by the model before applying noise and after, at varying intensity. The advantage of this method is that the detection efficacy does not rely on prior knowledge about the attack used and thus can be applied over a wide range of attacks and at varying adversarial perturbation budgets.

Furthermore, contrary to state-of-the-art defense or detection approaches, my method is computationally low demanding because no training or optimization of the model parameters is required.

Lastly, this method can be combined with other detection methods to increase the

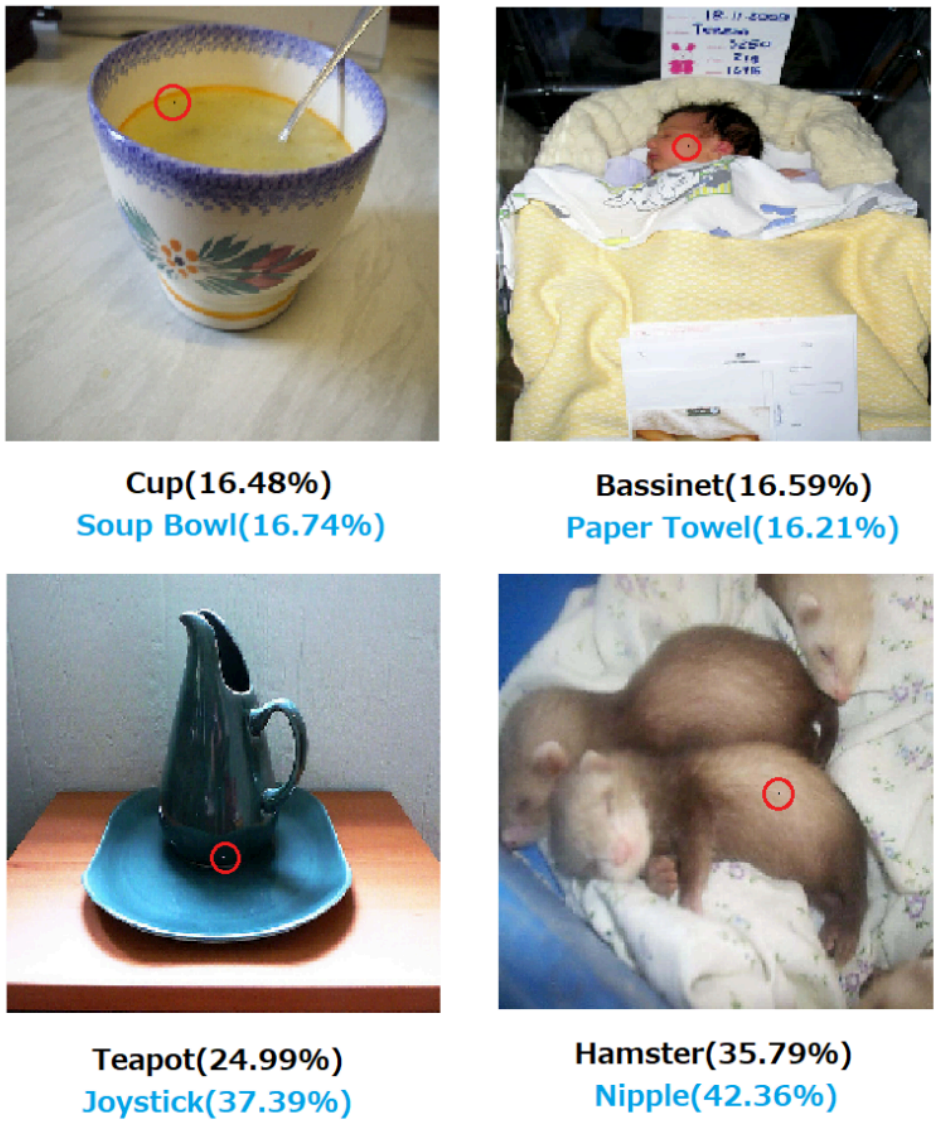


图 1.3 One-pixel attack by[2]. Modified pixels are circled in red. Original predictions in black and predictions with modified pixels in blue.

overall application's performance. The method I propose emerged after studying the effects of applying additive Gaussian noise to normal images and adversarial examples and observing a disparity that, to the best of my knowledge and at the time of this work, has not been discussed in prior work.

1.3 Thesis Structure

The material described in chapter 2 first provides a brief explanation and general knowledge about neural networks and the different parts they contain. Then it includes a description of adversarial examples, the dangers they represent, and the background knowledge needed to generate them, using different methods emerging from the research community.

(a) (b) (c)

图 1.4 Normal image (A), noisy image (B), adversarial example (C). The model accurately predicts both the normal image and noisy image but misclassifies the adversarial example, despite the adversarial perturbation being ≈ 20 times smaller than the random perturbation in that case.

Chapter 3 introduces the experiments I conducted and some of the results that motivated me to pursue and propose the methodology I later present and describe in chapter 4 which includes the detection performances.

第 2 章 Related Work

2.1 Neural Networks

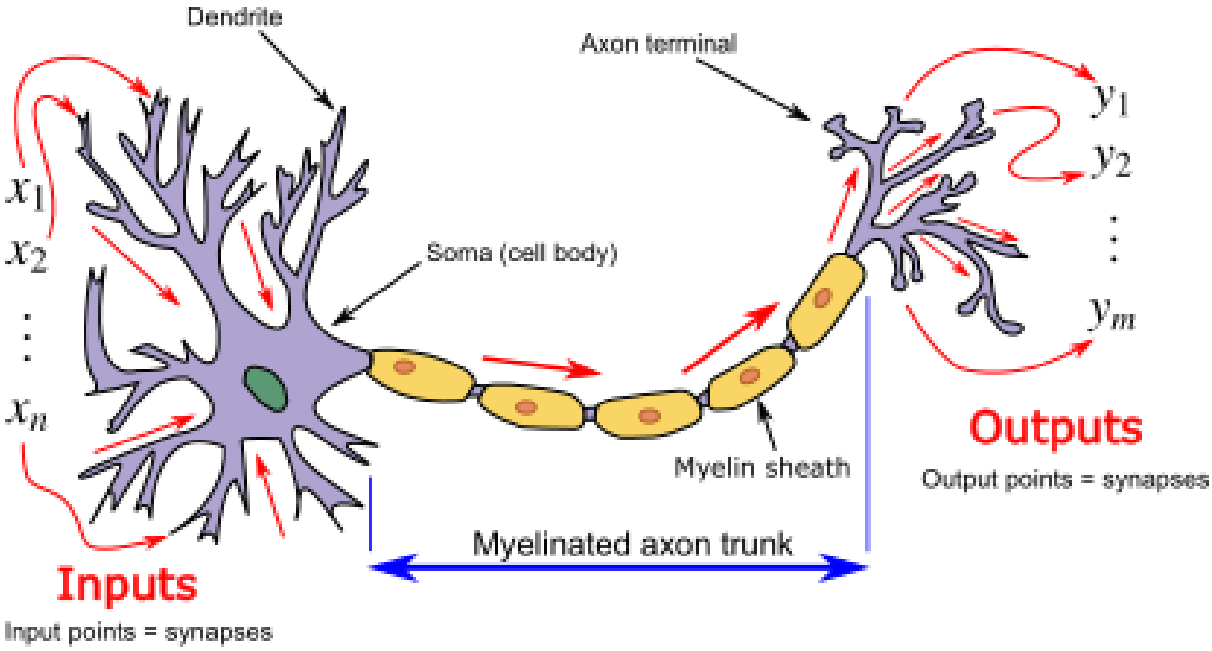


图 2.1 Representation of a biological neuron.

Artificial Intelligence (A.I.), Deep Learning, and Machine Learning have become buzzwords in recent years, with media, movies, and the general public often romanticizing and sometimes even humanizing A.I.s. However, what propelled this enthusiasm is rooted in reality. More recent achievements and revolutions made in multiple domains with the help of neural networks, even if not rivaling human capabilities in most scenarios, have been consequent.

The enthusiasm of the general public and scientific community in artificial intelligence is relatively recent; however, this idea of artificially mimicking the human brain (seen in figure 2.1) dates back almost a century, when[15], a neurophysiologist and a mathematician, proposed the first mathematical model of a neuron (see figure 2.2) called the McCulloch-Pitts Neuron.

Fifteen years later, a psychologist used the McCulloch-Pitts Neuron, where he proposed the Mark I Perceptron,[16]. The breakthrough proposed by Rosenblatt was that the network could learn by modifying the neurons' weights through successively passed inputs to minimize the difference between the desired output and the actual output.

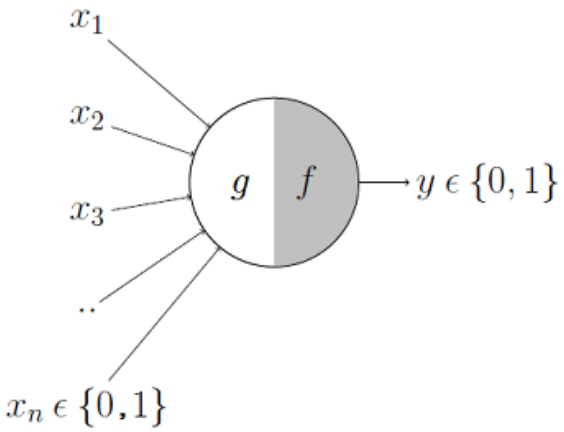


图 2.2 McCulloch-Pitts Neuron, first mathematical model of a neuron.

During the next few decades, the field saw some minor improvements, but the delivery and actual real-world use of A.I.s was minimal, if not nonexistent, and could not live up to the buildup displayed in media, even at that time. Finally,[17] published a book that laid out problems with neural networks, notably with their conclusion on the perceptron proposed by Rosenblatt, stating that this approach could not be translated into multi-layered neural networks, as the computational cost to evaluate each layers' neurons would be astronomic. This book, among others, started the era of "A.I. winter," a period of reduced funding and interest in artificial intelligence research.

The interest and enthusiasm for neural networks rallied in the nineties with the rediscoveries of the components that now form the pillar of today's neural networks: Backprogration and Gradient Descent.

(1) Neural networks, also known as artificial neural networks (ANNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons (see fig. 2.1) signal to one another.

Artificial neural networks are thus, similarly to the human brain, comprised of multiple layers of artificial neurons (see fig. 2.3 and fig. 2.4).

Figure 2.4 represents a simple neural network architecture with two hidden fully connected layers between the input and output layer. Each hidden layer contains an arbitrary number of artificial neurons that are fully connected to the next layer of neurons. The input layer contains several nodes (or neurons) equal to the dimension of the input. For example, if using a black and white image of 24 pixels, the input layer would contain 24×24 nodes. As for the output layer, the number of nodes

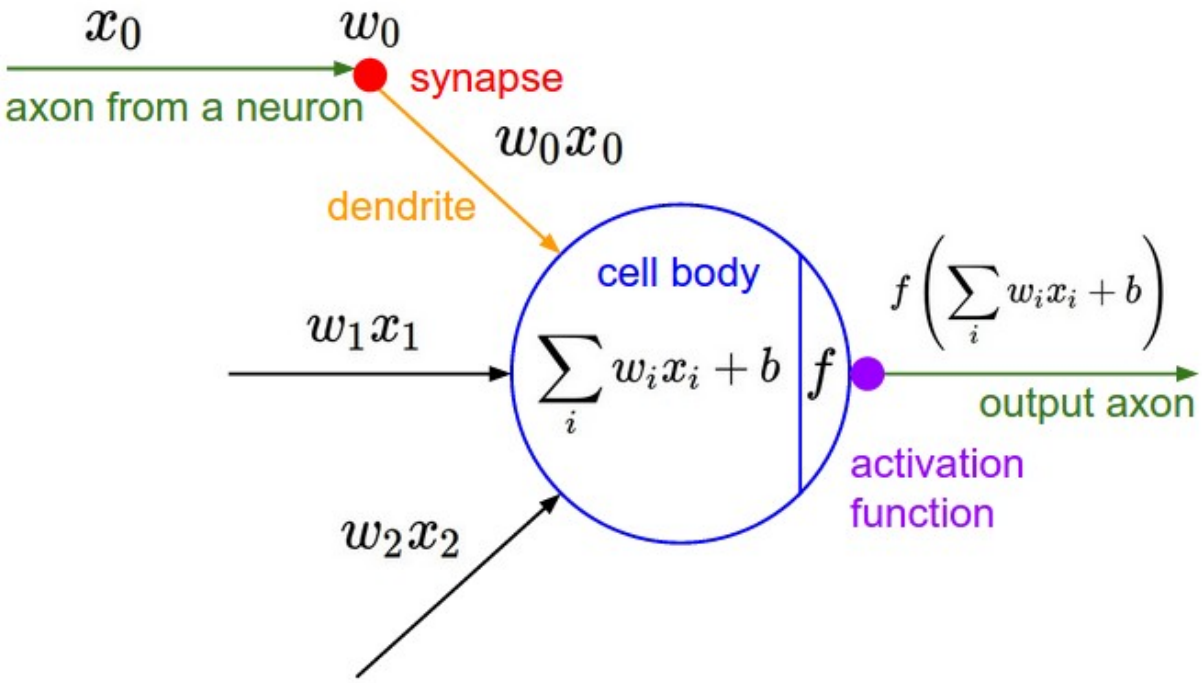


图 2.3 Representation of an artificial Neuron.

represents the number of classes. This type of architecture is widely used. Their primary advantage is that they are structure agnostic, i.e., no particular assumptions need to be made about the input.

Mathematically, we can represent the network shown in figure 2.4.

Let $x \in \mathbf{R}$ represents the input. Then, we can perform forward propagation as follows:

Let $a^{[1]}$ represent the first hidden layer and $a_i^{[1]}$ represent $a^{[1]}$'s i_{th} node:

$$A^{[1]} = \begin{cases} a_1^{[1]} = f(x_1 w_{11}^{[1]} + x_2 w_{21}^{[1]} + x_3 w_{31}^{[1]} + x_4 w_{41}^{[1]} + b_1^{[1]}) \\ a_2^{[1]} = f(x_1 w_{12}^{[1]} + x_2 w_{22}^{[1]} + x_3 w_{32}^{[1]} + x_4 w_{42}^{[1]} + b_2^{[1]}) \\ a_3^{[1]} = f(x_1 w_{13}^{[1]} + x_2 w_{23}^{[1]} + x_3 w_{33}^{[1]} + x_4 w_{43}^{[1]} + b_3^{[1]}) \\ a_4^{[1]} = f(x_1 w_{14}^{[1]} + x_2 w_{24}^{[1]} + x_3 w_{34}^{[1]} + x_4 w_{44}^{[1]} + b_4^{[1]}) \\ a_5^{[1]} = f(x_1 w_{15}^{[1]} + x_2 w_{25}^{[1]} + x_3 w_{35}^{[1]} + x_4 w_{45}^{[1]} + b_5^{[1]}) \\ a_6^{[1]} = f(x_1 w_{16}^{[1]} + x_2 w_{26}^{[1]} + x_3 w_{36}^{[1]} + x_4 w_{46}^{[1]} + b_6^{[1]}) \end{cases}, \tag{2.1}$$

or in vectorized form:

$$A^{[1]} = f(W^{[1]} X + b^{[1]}), \tag{2.2}$$

where $b^{[1]}$ represents the bias for the first hidden layer. Biases are learnable (by the model) parameters used to shift the activation function (explained next paragraph)

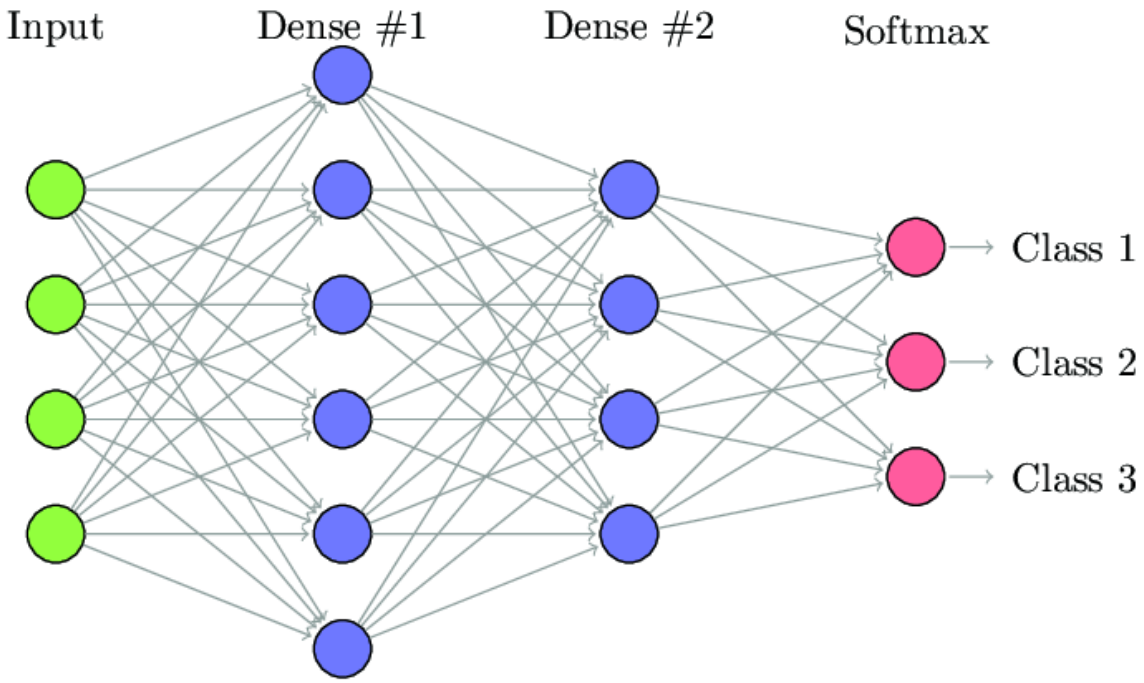


图 2.4 Fully connected neural network containing two hidden layers.

right or left, in order to better fit to the data.

Then, from the first hidden layer to the second hidden layer:

$$A^{[2]} = f(W^{[2]}A^{[1]} + b^{[2]}), \tag{2.3}$$

where we utilize the output of the previous layer $A^{[1]}$. We can observe the simplicity of adding or removing a hidden layer from a fully-connected network.

Finally, we can compute the final layer of the network:

$$\begin{cases} y_1 = f(a_1^{[2]}w_{11}^{[3]} + a_2^{[2]}w_{21}^{[3]} + a_3^{[2]}w_{31}^{[3]} + a_4^{[2]}w_{41}^{[3]} + a_5^{[2]}w_{51}^{[3]} + a_6^{[2]}w_{61}^{[3]} + b_1^{[3]}) \\ y_2 = f(a_2^{[2]}w_{12}^{[3]} + a_2^{[2]}w_{22}^{[3]} + a_3^{[2]}w_{32}^{[3]} + a_4^{[2]}w_{42}^{[3]} + a_5^{[2]}w_{52}^{[3]} + a_6^{[2]}w_{62}^{[3]} + b_2^{[3]}) \\ y_3 = f(a_3^{[2]}w_{13}^{[3]} + a_2^{[2]}w_{23}^{[3]} + a_3^{[2]}w_{33}^{[3]} + a_4^{[2]}w_{43}^{[3]} + a_5^{[2]}w_{53}^{[3]} + a_6^{[2]}w_{63}^{[3]} + b_3^{[3]}) \end{cases}, \tag{2.4}$$

Alternatively, in the vectorized form:

$$Y = f(W^{[3]}A^{[2]} + b^{[3]}), \tag{2.5}$$

In the previous equations (Eq. 2.1, Eq. 2.2, Eq. 2.3, Eq. 2.4, Eq. 2.5), f represents a nonlinear function, also called activation function. Activation functions act as the axon seen in figure 2.1. It takes in the output signal from the previous neuron and converts it into the input for the next neuron. These nonlinear functions

add non-linearity to a neural network, as their name implies.

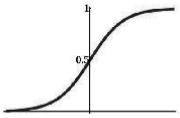
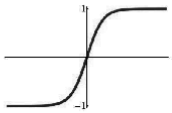
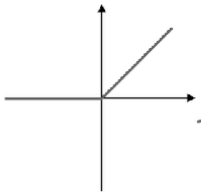
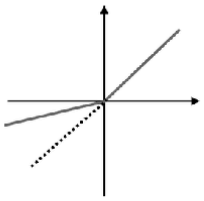
Name	Functions	Derivatives	Figure
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))^2$	
tanh	$\sigma(x) = \frac{e^x - e^- x}{e^z + e^- z}$	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$	
Leaky ReLU	$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$	
Softmax	$f(x) = \frac{e^x}{\sum_1^j e^x}$	$f'(x) = \frac{e^x}{\sum_1^j e^x} - \frac{(e^x)^2}{(\sum_1^j e^x)^2}$	

图 2.5 Widely used activation functions.

Figure 2.5 shows some of the widely used activation functions used in neural networks. In our previous example, the final layer of our fully connected neural networks contains three nodes (for three classes); accordingly, this network architecture would be used for multi-class classification problems. The activation function widely used for this type of problem is the softmax function, seen in figure 2.5), where it's equation is:

$$f(x) = \frac{e^x}{\sum_i^j e^x}, \tag{2.6}$$

where x is the input, j is the number of classes and e is the standard exponential function for output vector.

The softmax function outputs values in the range of 0 to 1. In our neural network example from figure 2.4, with three output classes, the model's output could be a vector such as: $[0.8, 0.1, 0.1]$. We can translate this output as the model being 80% confident about the input being the first class and 10% confident for the second and third class,

respectively.

As for the hidden layers, a widespread activation is the rectified linear unit, commonly referred to as ReLU:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geqslant 1 \end{cases}, \tag{2.7}$$

or its modification Leaky ReLU:

$$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geqslant 1 \end{cases}. \tag{2.8}$$

We refer to passing the input through layer-by-layer like evoked beforehand as "forward propagation." Forward propagation is one of the core processes needed to train a neural network. Backpropagation, by[18] is the next step in this process, and it refers to the method of calculating the gradient of the parameters of the neural network, i.e., weights and biases. Inversely with forward propagation, this method traverses the network from the output to the input to compute gradient with respect to some parameters. The objective of updating these parameters is to minimize a chosen cost function. For example, in our neural network example again, we could use the cross-entropy function defined as:

$$-\sum_{c=1}^{3} y_{o,c} \log\left(p_{o,c}\right), \tag{2.9}$$

for a three-classes classification problem, where y is a binary indicator if class label c is the correct classification for observation o and p is the predicted probability that o is of class c.

Forward propagation and backpropagation are alternatively used when training a neural network and are interdependent: forward propagation computes and stores intermediate variables and parameters, while backpropagation computes the gradients of these same parameters. Training a network thus requires considerably more memory than simply predicting a sample, as backpropagation requires the intermediate values in order to be computed.

To recapitulate, a neural network can be written as a function $F(x) = y$, where x represents the input and y represents the output. The final layer of a neural network performing classification is often the softmax activation function. Therefore, $F(x)$

outputs a probability distribution, where $F(x)_i$ represents the probability that input x belongs to class i. We write the final output of the layer as $F(x) = \text{softmax}(Z(x))$, where $Z(x)$ is the network output vector containing logits, i.e., raw values before the activation function. Finally, the classifier function that returns the most likely class label can be written as $C(x) = \text{argmax}_i(F(x)_i)$.

2.2 Convolutional Neural Networks

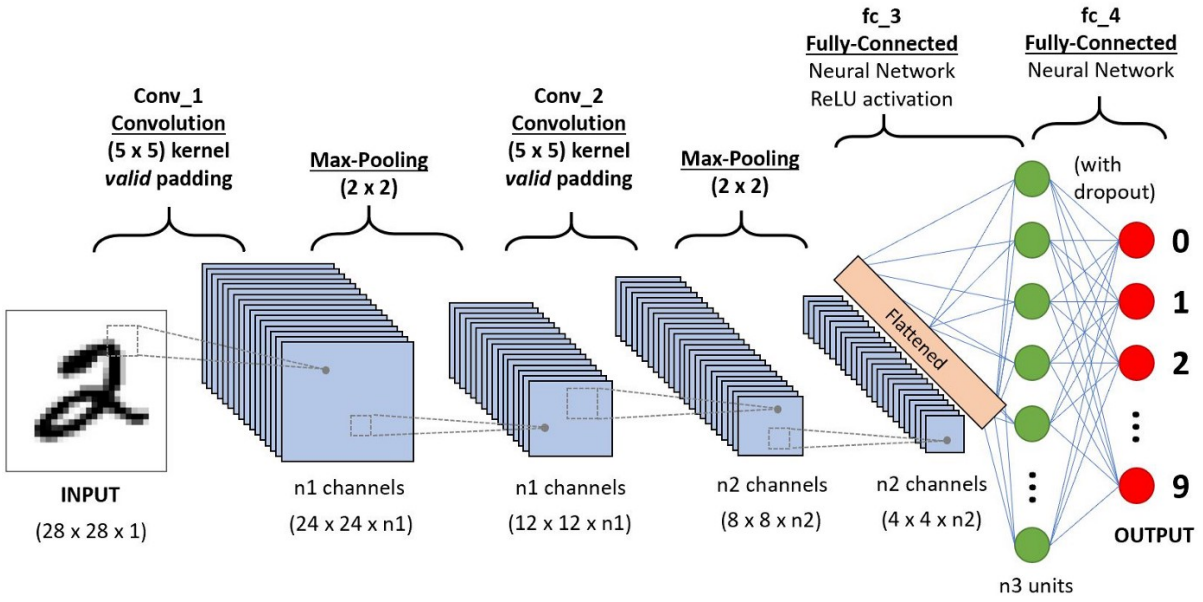


图 2.6 Architecture of a classic convolutional neural network.

Convolutional neural networks (CNNs), by[19], similarly to the neural network architectures discussed in the previous section 2.1 are a class of artificial neural networks. CNNs are commonly used in computer vision to extract features from images or videos and have been shown to work exceptionally well for this type of data. They are similarly composed of layers of neurons with learnable parameters, i.e., weights and biases. The main difference with the more traditional architectures is that convolutional neural networks make the explicit assumption that the inputs are of image-type, i.e., images or videos.

The problem with regular neural networks, when applied to images, is that the fully-connected layers in these architectures do not scale well with this type of input: a color image of size 224×224 would represent 150.000 weights ($224 \times 224 \times 3$), which would quickly result in a costly and inefficient model. In convolutional neural networks, the neurons are three-dimensional: width, height, and depth (number of channels: red, green, blue). These convolutional layers can drastically reduce the

number of parameters, thus the model's size.

Convolutional neural networks, or ConvNets, transform the input image into the final output vector containing a score for each class through a sequence of layers. Figure 2.6 shows an example architecture for a convolutional neural network performing multi-class classification on black and white 28×28 images of digits from 0 to 9.

This architecture comports:

- Two convolutional layers: the core building block of any ConvNet. Both layers contain multiple filters (also called kernels) of size 5×5. As seen in figure 2.7, filters learn visual features: the earlier filters, i.e., the ones present at shallow depth, learn more basic features. In contrast, filters at the last layers learn more complex ones, as seen in figure 2.8, since they are combinations of all the previous layer's filters.
- Pooling layers: these layers will perform a down-sampling operation along the spatial dimensions, e.g., width and height. The primary function of a pooling layer is to reduce the spatial size of the representation, to reduce the number of parameters, thus reducing the computation cost of the model.
- Fully connected layers: as seen in the previous section 2.1. Since neurons from convolutional layers are multidimensional, their results need to be flattened before being fed into fully-connected layers. This last layer, similarly to ANNs seen previously, will output a probability vector containing the score for each class.

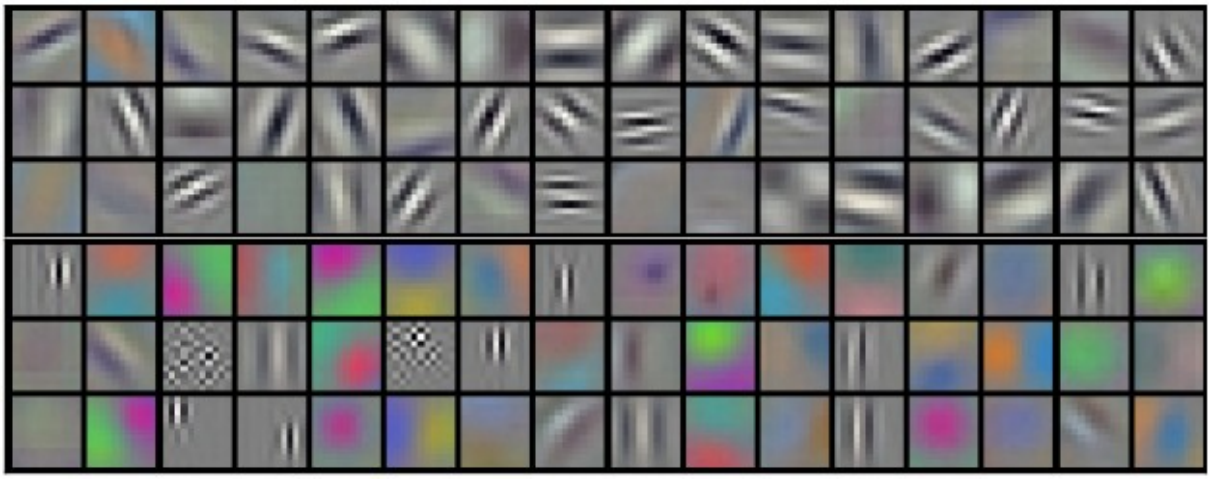


图 2.7 $11 \times 11 \times 3$ filters learned by the first convolutional layer on $224 \times 224 \times 3$ input images, experiment by[3].

In short, convolutional neural networks are artificial networks that contain convolutional layers. These architectures perform exceptionally well on image-based data with the added benefit of fewer parameters compared to more traditional ANNs.

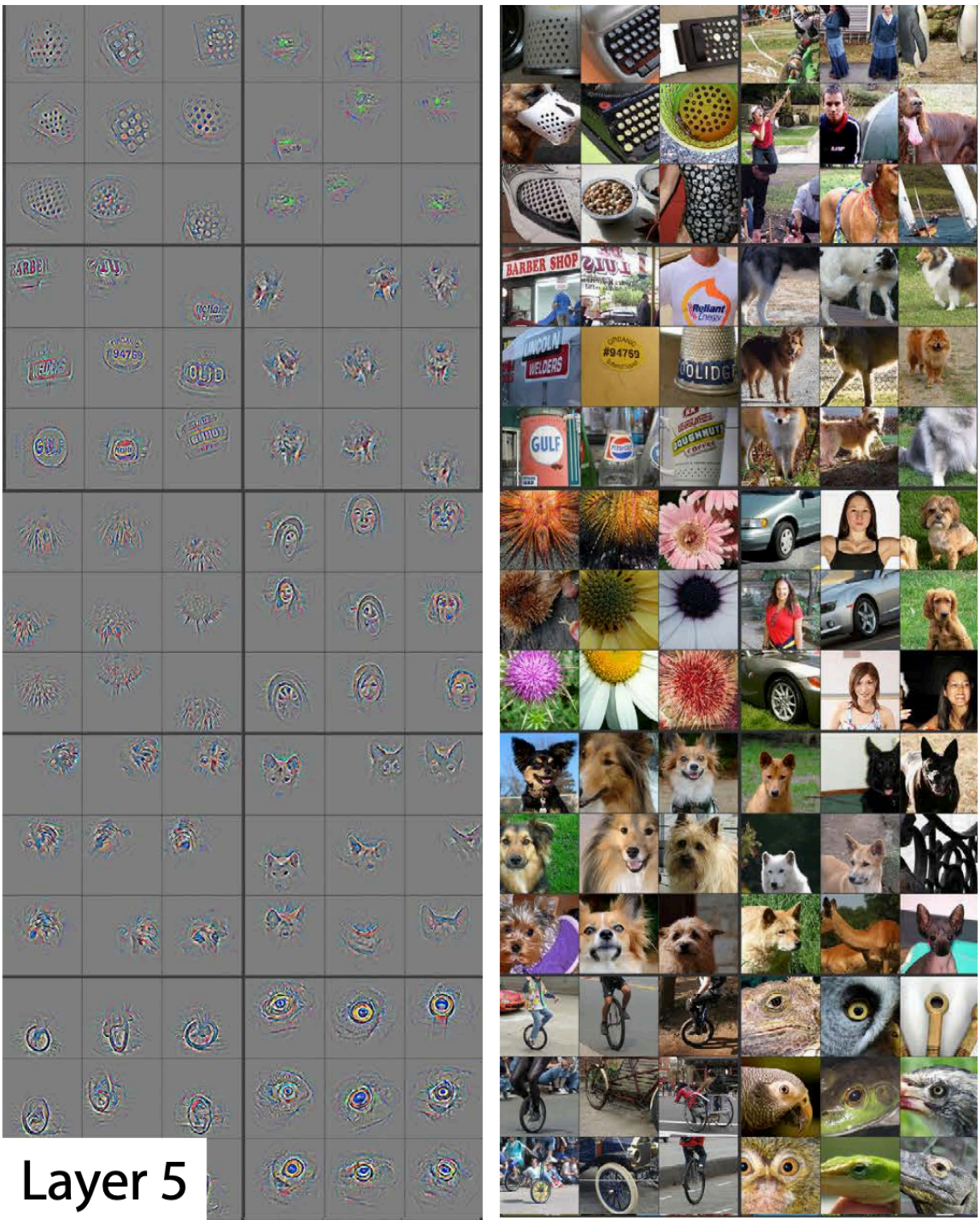


图 2.8　Filters learned by the fifth convolutional layer, experiment by[4].

In this work, since the entirety of my research is conducted on images, convolutional neural networks are the only type of architecture that I consider.

2.3 Adversarial Examples

(a) (b) (c)

图 2.9 Speed limit signs modified (B, C) that fools the Tesla model X and S (model 2016). (B) is identified as a 45-mph sign, while (C) is identified as a 85-mph speed sign.

As seen with figure 1.1, adversarial examples are samples that contain intentional feature modifications that cause a model to misclassify the sample, e.g. an image of a ”shark” being misclassified as a ”bee” after adding an invisible (for a human observer) modification to the image.

A neural network classifying an image of a shark as a bee may only seem comical and not particularly problematic. However, with the rapidly growing usage of neural networks in real-world applications, we need to have the certitude that the models in use will not be as trivially fooled.

Recently, McAfee Advanced Threat Research researchers experimented with adversarial examples in a physical context ([20]). They physically applied modifications to road speed signs in order for a Tesla car to misidentify the signs. Figure 2.9 shows the original 35-mph road sign (A) as well as two of the physical adversarial examples they created (B, C). (A) is accurately identified with a 95.93% confidence, while (B) is wrongly identified as a 45-mph sign with 99.88% confidence, and (C) is also wrongly identified, but as an 85-mph sign! In this context, it is easy to imagine why and how this can be a problem that needs to be addressed.

With the ever-growing research on the vulnerability of NNs, there are now many

different methods to generate adversarial examples. The common objective of such methods is to, from a normal image x, create a perturbation δ and add it to the original image so that the new sample $x' = x + \delta$ is misclassified by a model. We refer to a sample that fulfills this objective as an untargeted adversarial example.

On the contrary, a targeted adversarial example x' is designed to be classified by a model as a specified target class t. As a result, targeted adversarial examples are typically more challenging to produce and may require a more significant perturbation than untargeted attacks.

We can thus formulate the optimization problem to craft adversarial examples as:

$$\min_x D(x+\delta), \tag{2.10}$$

such that the classification $C(x') \neq y$ for an untargeted attack, where y is the actual class of the input, or $C(x') = t$ for a targeted attack, where t is the targeted class.

D represents a distance metric, usually, p-norm defined as:

$$\|D\|_p = \left(\sum_{i=1}^{n} |d|^p \right)^{\frac{1}{p}}. \tag{2.11}$$

The remainder of this section contains a brief explanation of some of the popular methods used to generate adversarial examples.

(1) Fast Gradient Sign Method (FGSM). [21] first introduced a method for crafting adversarial examples against GoogLeNet[22]. To generate an adversarial sample x' that maximizes the objective J, FGSM uses the gradients of the cost function with respect to the normal input image x:

$$x' = x + \epsilon sign\left(\nabla_x J\left(\Theta, x, y\right)\right), \tag{2.12}$$

where the multiplier ϵ is used to ensure that the perturbation is kept small, and Θ represents the parameters of the model.

(2) Basic Iterative Method (BIM). [23] proposed an extension of the Fast Gradient Sign Method. Rather than generating the adversarial sample in one step, BIM applies adversarial noise multiple times with a step size α. The benefit of iteratively crafting the adversarial sample x' is that intermediate results can be clipped at each step to

ensure that the generated samples are within an ϵ distance to the original input x:

$$x'_{n+1} = clip_{x,\epsilon}\left\{x'_n + \alpha.sign\left(\nabla_x J\left(x'_n, y\right)\right)\right\}, \tag{2.13}$$

where $x'_0 = x$.

(3) Carlini & Wagner (CW). [24] proposed a powerful method to generate adversarial examples that can defeat the defensive distillation approach published by[25]. In their paper, the authors construct an attack for: L_0, L_2 and L_∞ distance metrics.

In my experiments, I use the attack that seeks low distortion in the L_2 distance metric. The final optimisation problem for the CW L_2 attack can be defined as

$$\min \|x' - x\|_2^2 + c \cdot \ell(x'), \tag{2.14}$$

where c is a constant chosen via binary search that determine the success probability of the attack. The loss function ℓ is the best among seven evaluated by the authors, written as

$$\ell(x') = \max\left(\max\left\{Z(x')_i : i \neq t\right\} - Z(x')_t, -k\right), \tag{2.15}$$

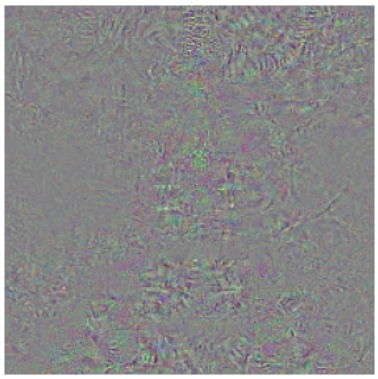
where $-k$ is a parameter to specify how confident we want the adversarial example to be classified as t by the model.

The CW method is a state-of-the-art attack and effectively finds adversarial examples with a small perturbation size. However, this method is computationally expensive to run.

(4) Decoupled Direction and Norm (DDN). Jérôme et al[26] proposed an improvement over the CW method. The DDN method can obtain comparable results in terms of perturbation size but with considerably fewer iterations. At each iteration k, refine noise β_k by considering a larger norm $\epsilon_{k+1} = (1+\gamma)\epsilon_k$ or smaller norm $\epsilon_{k+1} = (1-\gamma)\epsilon_k$ depending on if $x + \beta_k$ is adversarial or not. Finally, the method returns the clipped adversarial sample that has the lowest L_2-norm.

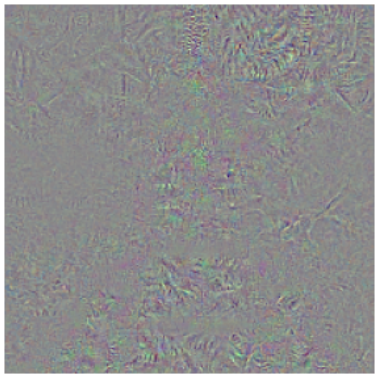
(A) Original image

(B) BIM



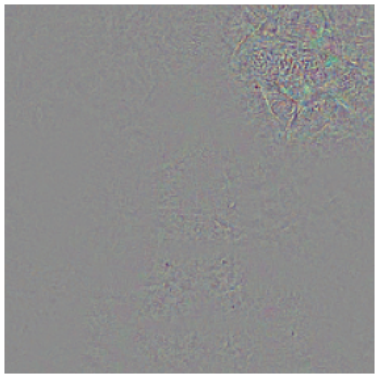
(C) $(B - A)$

(D) DDN



(E) $(D - A)$

(F) CW



(G) $(F - A)$

图 2.10 Adversarial examples generated with different methods. The original input image (A) is correctly classified as a "church", while all the generated samples: (B), (D) and (F) are identified as a "chicken" by the model.

Figure 2.10 shows three samples generated using the basic iterative method (B), decoupled direction and norm (D), and the Carlini & Wagner method (F). The model correctly predicts the natural image (A) as an image of a church while predicting the generated samples as images of chickens. The perturbation size for each sample is: $L_2 \approx 2.00$ for the BIM sample (B), $L_2 \approx 1.40$ for the DDN (D) sample and $L_2 \approx 1.37$ for the CW sample. As discussed earlier, DDN and CW methods can generate samples with a smaller perturbation than methods such as BIM or FGSM.

2.4 Defending Against Adversarial Examples

Since[9] discovered the existence of adversarial examples, much research has been conducted from the perspectives of both the attacker, i.e., the party trying to exploit vulnerabilities in a model and the defending party trying to mitigate these attacks. The authors also hypothesized that adversarial examples exist due to the high non-linearity nature of NNs. However, this hypothesis was later rebutted by[21] when they argued that adversarial examples exist due to NNs being too linear rather than the contrary.

The research first focused on defending against adversarial examples by improving the robustness of neural networks via, among others, adversarial training ([21,27]), where adversarial examples are included in a dataset alongside natural images in order to train a CNN on both natural images and adversarial samples. The computational cost required to generate adversarial examples makes this defense approach computationally expensive, even when using techniques such as FGSM that require considerably less computational work than methods such as CW which can require hours to generate a single sample on a low-end gpu.

Defensive distillation from[25] is another technique to improve the robustness of NN and involves the use of two networks. A first network F is conventionally trained on inputs X and labels Y and outputs a probability vector predictions $F(X)$. The second network F_d is then trained on the same inputs X, but the labels are replaced by the output of the first network $F(X)$ as a way of restraining the model from overfitting on the data. Defensive distillation is a very effective defense but was defeated by the more recent CW attack.

Due to the difficulty and computational cost of training robust neural networks against adversarial examples, recent research has focused on detecting them instead. However, a recent survey by[28] examined ten detection defenses that they all bypassed using their attack method and concluded that adversarial examples are significantly more complex to detect than previously recognized. Furthermore, among the ten detection methods surveyed, Carlini & Wagner concluded that only bayesian neural network uncertainty, introduced by Feinman[29], was effective and made generating adversarial examples nearly five times more difficult on the dataset CIFAR-10.[29] use dropout from[13] to induce randomness during inference and predict an input multiple times in order to measure the prediction uncertainty. They show that the prediction uncertainty is typically higher on adversarial examples compared to normal and noisy images.

2.5 Detection Evaluation Metrics

To attest to the effectiveness of the detection performances of the method proposed in a later section, I adopt the recall and precision rates defined as

$$recall = \frac{tp}{tp + fn}, \tag{2.16}$$

where tp (true positive) is the number of correctly detected adversarial examples and fn (false negative) is the number of adversarial examples that are not detected.

$$precision = \frac{tp}{tp + fp}, \tag{2.17}$$

where fp (false positive) is the number of normal images that are incorrectly identified as adversarial examples.

Finally, we use the F_β score defined as:

$$F_\beta = (1 + \beta^2)\frac{recall \cdot precision}{recall + (\beta^2 precision)}, \tag{2.18}$$

where $\beta = 2$ to emphasize on the recall rate.

第 3 章 Experiments

3.1 Overview

Adversarial examples can be seen as a worst-case noise that fools the model into misclassification when applied to an image. On the other hand, neural networks are relatively robust to random corruptions such as Gaussian noise.

My intuition is that applying random noise to an adversarial example could hide or alter parts of the adversarial perturbation, thus diminishing or even nullifying its effect.

In order to apply a random transformation to an image x, we generate a Gaussian noise $\tilde{y} = \mathcal{N}(0, 1)$ that we add to the image in order to create a noisy version $\tilde{x}$:

$$\tilde{x} = x + \tilde{y}\kappa, \tag{3.1}$$

where κ, is the standard deviation used to scale down or up the noise intensity.

Fig 3.1 shows the impact of increasing κ on an image and shows the corresponding L_2 distance and peak signal-to-noise ratio (PSNR) of the noise mask.

Image	k	L_2	PSNR
A	0.00	0.00	100
B	0.02	7.76	34.01
C	0.04	15.46	28.05
D	0.06	23.11	24.60
E	0.08	30.40	22.18
F	0.10	37.36	20.34

表 3.1 Details for Figure 3.1

To further motivate and formulate the proposed method discussed in section 4, I first perform a series of experiments detailed in the remainder of this section to verify that the intuition mentioned above holds correct.

(a) (b) (c)

(d) (e) (f)

图 3.1 Increasing the noise intensity k on an ImageNet sample. Details in Table 3.1.

3.2 Experimental Settings

Below I describe the datasets and models used, as well as the settings used to configure the attack methods.

3.2.1 Datasets and Models

To conduct the experiments, I collect data from three datasets:

- CIFAR-10 by[30], widely used dataset consisting of 60000 32×32 RGB images divided into ten different classes (e.g., airplane, automobile, bird.). For the experiments, I use the 10.000 images present in the test set.
- ImageNet by[5], popular image database containing over 1.300.000 224×224 RGB images divided into 1000 different classes. For our experiments, rather than using the entirety of the dataset, I created a subset containing 10 of ImageNet classes.
- Dogs vs. Cats by[31], the Asirra (Animal Species Image Recognition for Restricting Access) is a real-world Kaggle dataset containing 25.000 224×224

resized RGB images divided into two classes: dogs and cats.

For the models, I use a Very Deep Convolutional Networks for Large-Scale Image Recognition with 11 layers introduced by[32]. The complete architecture of the VGG-11 model is shown in figure 3.2.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

图 3.2 VGG configurations, from 11 layers (A) to 19 layers (E).

For the experiments using the ImageNet and Dogs vs. Cats datasets, the model is first trained on ImageNet and then fine-tuned for the Dogs vs. Cats dataset.

For the models, I use Very Deep Convolutional Networks for Large-Scale Image Recognition with 11 layers introduced by[32]. The complete architecture of the VGG architecture is shown in figure 3.2.

I use pre-trained on ImageNet models open-sourced by PyTorch for the experiments using the ImageNet and Dogs vs. Cats datasets. For Dogs vs. Cats, the model

is fined-tuned, i.e., the last few layers' parameters are updated instead of training the whole network from scratch. For CIFAR-10, I train the model following the authors instructions.

3.2.2 Attacks

I implement the attacking methods introduced in Section 2.3 using the FoolBox library by[33]. For targeted attacks defined in section 2.3, the target t class is selected randomly such that

$$\{t : t \in C \setminus \{y\}\}, \tag{3.2}$$

where C is a set of the available classes in the dataset and y is the ground-truth label of the sample.

I generate adversarial examples using accurately classified images from the test sets of each dataset as it would not make sense generating adversarial examples on samples that the models already fail to identify.

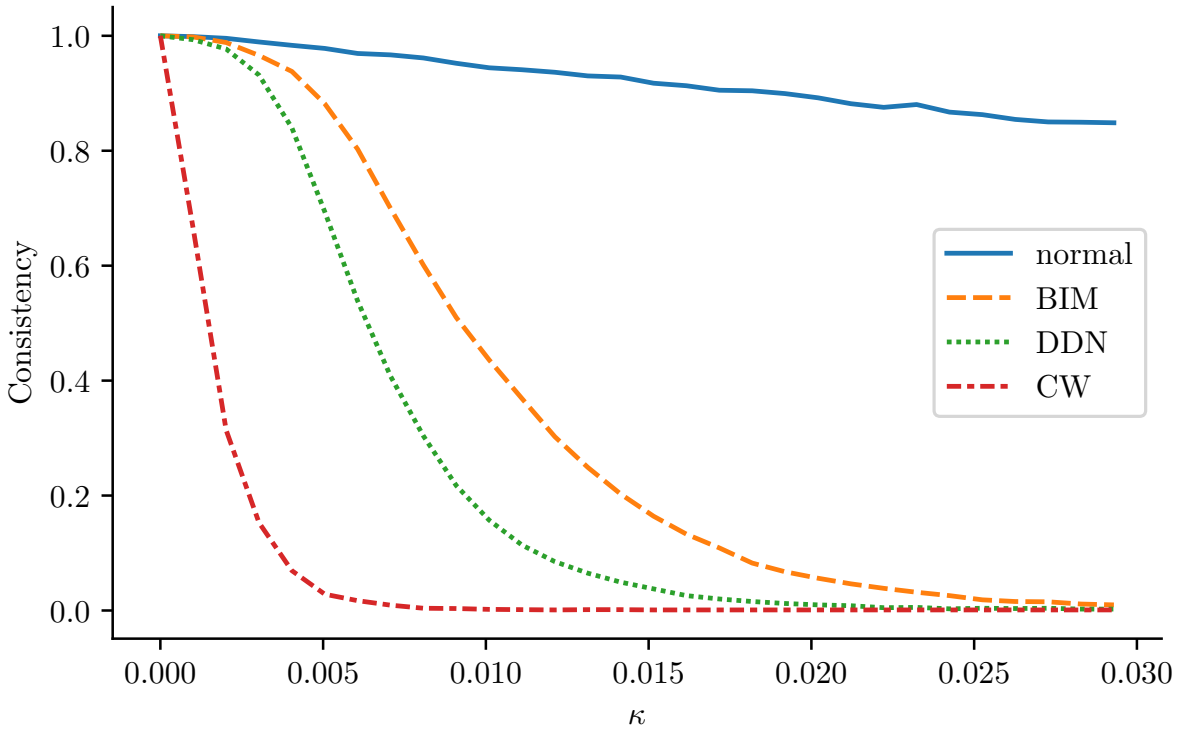
3.3 Robustness

In this section, in order to verify the intuition mentioned in section 3.1, I describe two experiments where I apply noise to both the input images and the adversarial examples in order to compare the model's output before and after the added noise.

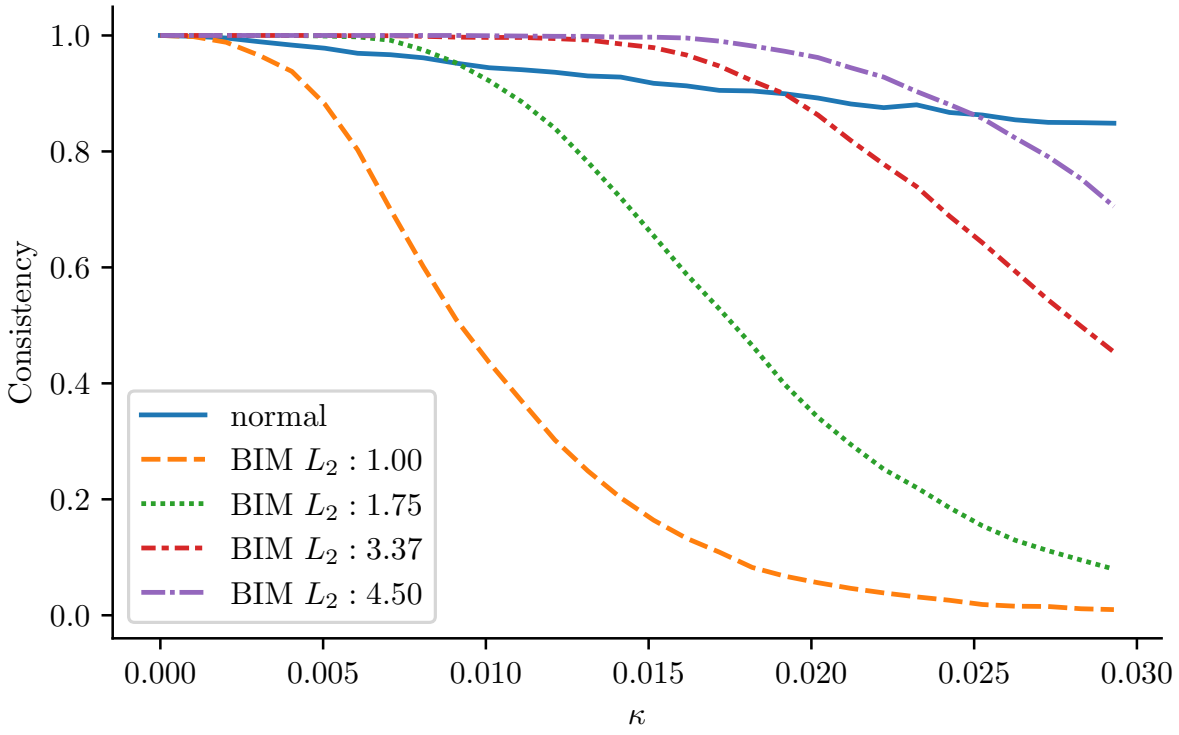
3.3.1 Consistency

In the first experiment, I compare the model's classification output for a normal image x and a transformed version $\tilde{x}$ (Eq. (3.1)). When the classifications $C(x)$ and $C(\tilde{x})$ are identical, I refer to them as being consistent classifications. i.e., when the model predicts the same class for the unperturbed image and the same image but containing voluntarily added noise, the predictions are consistent with one other. If my intuition is correct, we should observe the predictions on normal images to be more consistent when adding noise. At the same time, I expect the model's predictions on adversarial examples to start being more inconsistent as the noise intensity increases.

Here, our objective is to compare the consistency between normal images and adversarial examples. To proceed, I randomly select 1000 images from ImageNet's test set to record the classification consistency on each sample and at different κ (Eq. (3.1)) values. Finally, we follow the same procedure for adversarial examples and generate 1000 samples with the BIM, DDN, and CW methods.



(a)



(b)

图 3.3 Consistency comparison between normal images and adversarial examples using different methods and perturbation budgets. As κ (see (3.1)) increases, the consistency decreases, more so for adversarial examples, unless we also increase the adversarial perturbation budget as seen in (b).

Figure 3.3 the average classification consistency of input types and at different κ. We observe the average consistency to decrease slowly as κ increases for normal images. This moderate decrease shows that, as expected, the model is not significantly affected by the random perturbation added to the image. On the contrary, we observe that adversarial examples' classification consistency decreases rapidly as κ increases. These results corroborate my intuition that introducing random perturbations to an adversarial example could impede its effectiveness.

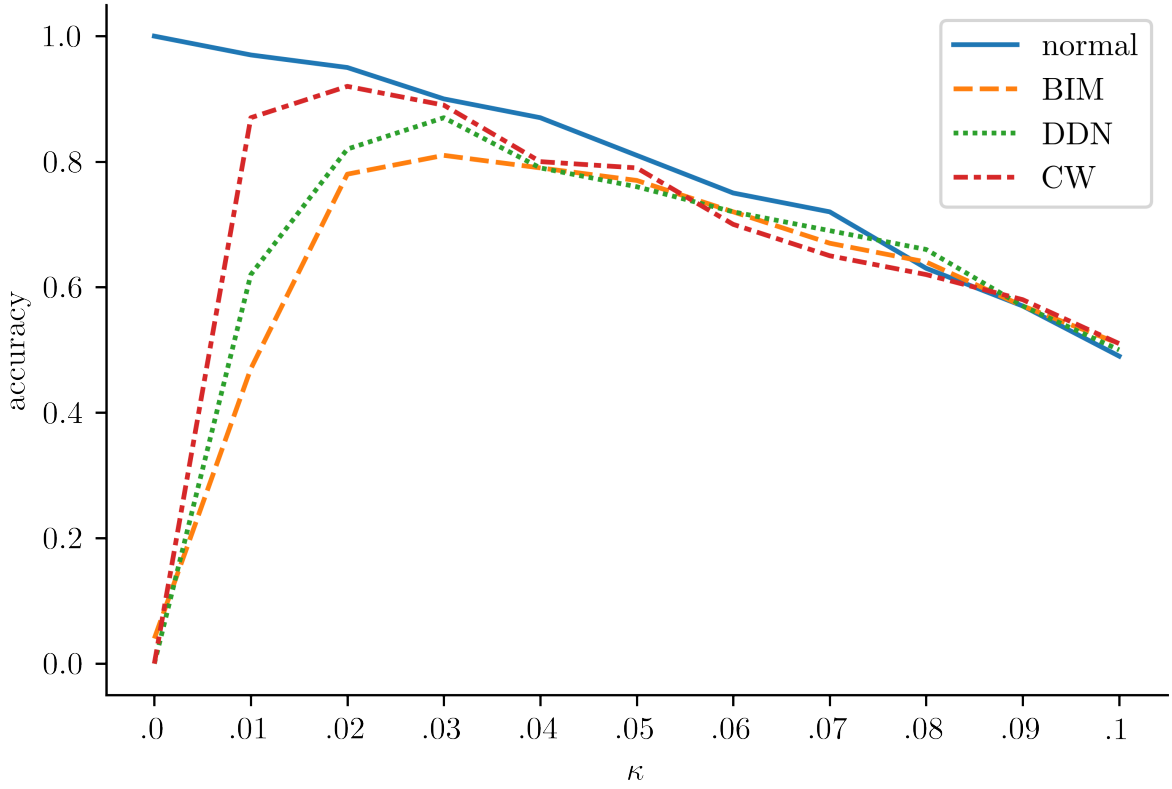


图 3.4 Shows the accuracy comparison between normal images and adversarial examples; as κ increases, the accuracy of adversarial examples first increases before decreasing similarly to normal images.

Furthermore, figure 3.4 shows that the actual prediction accuracies are partially restored on adversarial examples as the noise intensity increases before slowly reducing again, similarly to normal images, when the noise intensity is too high for the model. Finally, average accuracies of normal images and adversarial examples start to be very similar at around 0.03κ, hinting and showing again that the adversarial perturbations in adversarial examples lost their efficacy to fool the targeted models.

Interestingly, coming back to figure 3.3A, we can also observe the classification consistency being different between attacking methods, i.e., to reach a consistency of ≈ 0, adversarial examples generated with BIM need a larger κ than samples generated with DDN and CW.

Instinctively, this difference makes sense; adversarial examples generated with the BIM approach have a larger average L_2 adversarial perturbation of ≈ 1.0 compared with DDN at $L_2 \approx 0.85$ and CW at $L_2 \approx 0.70$. Figure 3.3b shows the same experiment but using adversarial examples generated with BIM at different perturbation budgets. These results undeniably show that the more substantial the adversarial perturbation is (i.e., higher perturbation budget), the more robust it is to random perturbations. Therefore, a larger κ may be needed to counter the effectiveness of the higher adversarial perturbation budget attacks.

3.3.2 Logits differences

I performed another experiment to investigate further the difference between the model output for an untransformed and a randomly perturbed image. Instead of recording the classification output of the model as done in section 3.3.1, which is only a reduced-down portion of the output, I decided to observe the output logits, i.e., the output before applying the activation function: in this case, before applying softmax. This will allow us to see the raw output values for each class.

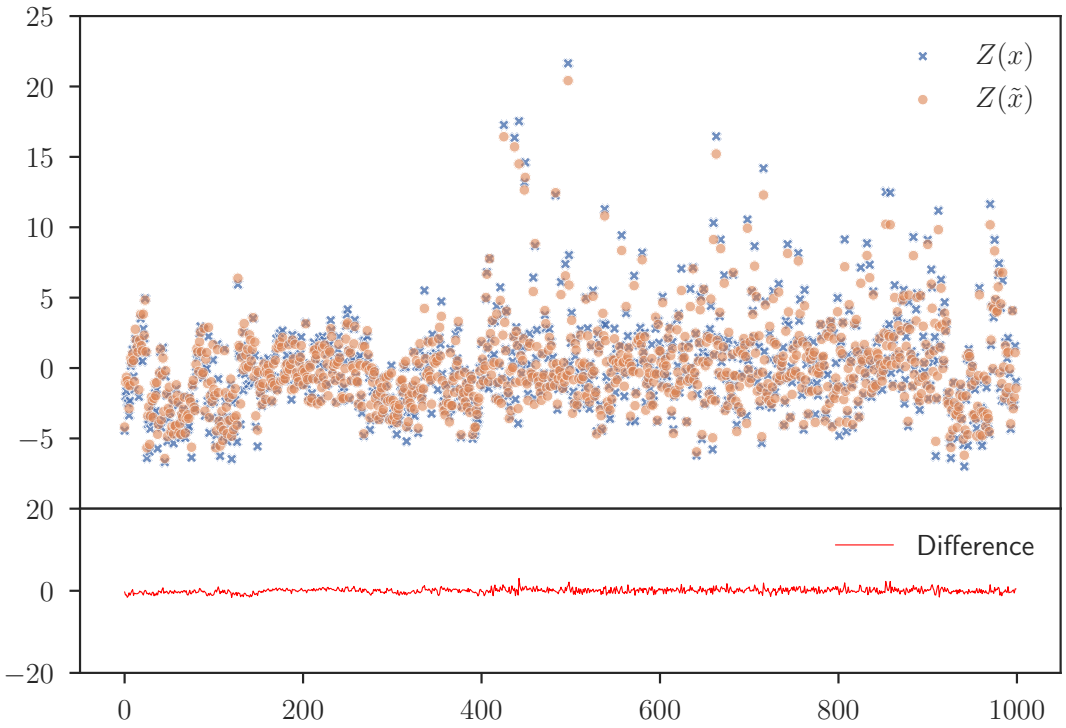
I select a single ImageNet image x and generate a noisy version $\tilde{x}$ with $\kappa = 2 \times 10^{-2}$. Figure 3.5A shows the output logits $Z(x)$ and $Z(\tilde{x})$ for each the normal image x and the perturbed version $\tilde{x}$ respectively. Logits are plotted (1000 logits for the 1000 ImageNet classes), as well as the difference between $Z(x)_i$ and $Z(\tilde{x})_i$ in the lower part of the plot. We observe both outputs to be very similar value-wise for each class. On the contrary, when the original input is adversarial (BIM, $L_2 \approx 1$), as in figure 3.5B, the difference between outputs is striking, magnitudes over the difference observed with the normal image.

These two experiments proved that my initial intuition was correct. It also showed that the higher budget the adversarial perturbation, the more robust it is against the random perturbation I add to the input. To me, this may explain why many detection techniques fail to detect higher-perturbation budgets adversarial examples. These observations motivated me to pursue this track because I believe that the ease with which we can add varying random perturbation to the input, by increasing or decreasing κ, could be an interesting idea to combat low and high perturbation budgets attacks.

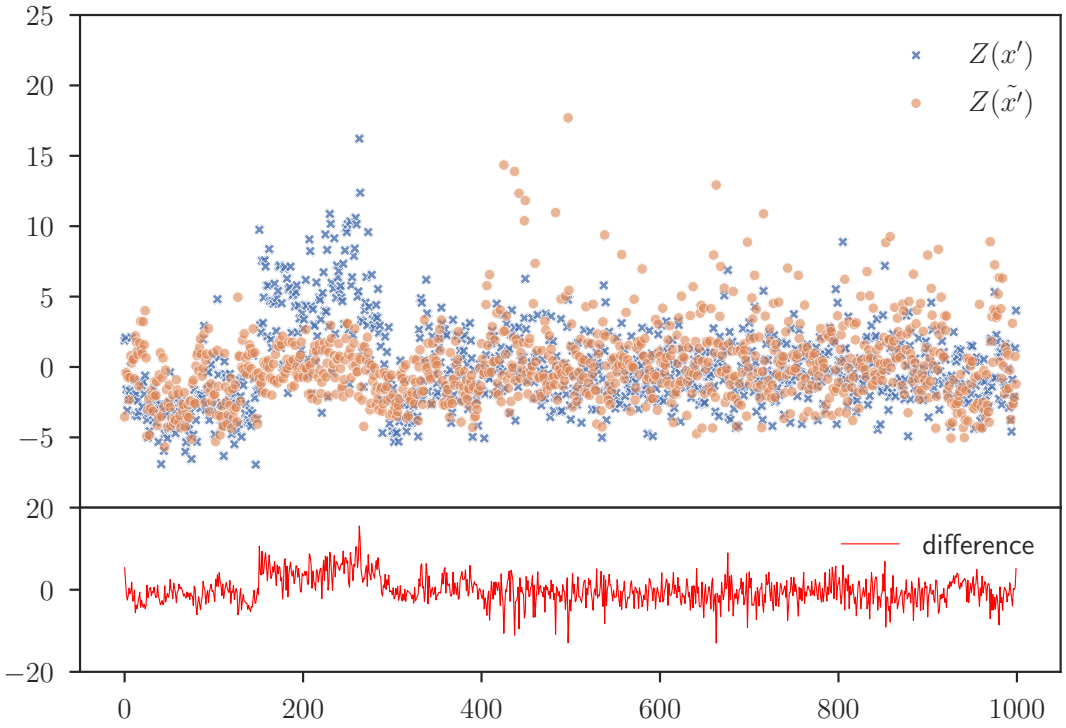
Observing and comparing the model outputs showed us that adding a random perturbation to an image can help us detect the legitimate or adversarial nature of the input.

Following these observations, I present in the following section a novel method to

detect adversarial examples.



(a) Normal Image



(b) Adversarial Example (BIM)

图 3.5 Comparison between the logits of an ImageNet image before ($Z(x)$) and after adding noise to the input ($Z(\tilde{x})$). When the input is normal (A), the difference between predictions is small, but becomes larger when the input is adversarial (B).

第 4 章　Methodology

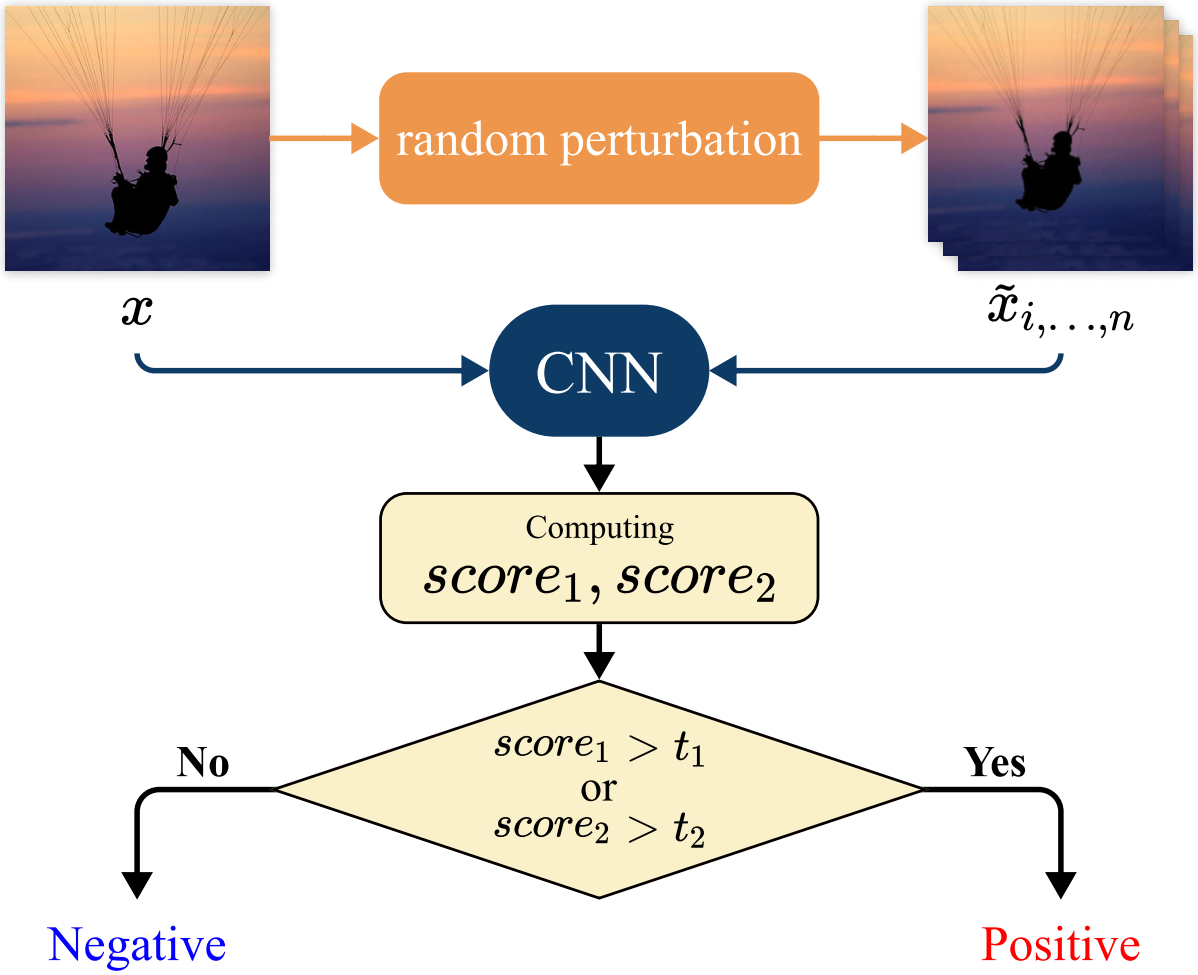


图 4.1　The framework of the discussed method.

As observed during the experiments conducted in section 3.3.1 and section 3.3.2, the usage of Gaussian noise to alter the efficacy of the perturbations present in adversarial examples is effective. To build upon these observations, I propose a methodology to detect adversarial examples that I describe in the remainder of this section.

This methodology, represented in figure 4.1, aims to compute two scores, described in the following subsection, using the difference in predictions before and after applying random noise to the input image space. From the two scores computed for each input, we can determine if an input is adversarial or not.

4.1 Scores

As observed in section 3.3.2, the model predictions differ to a more significant degree when adversarial examples are perturbed by additive random noise compared with normal images.

To capture these differences, we use two scores defined as follows:

$$score_1(x, \tilde{x}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - \mu)^2}, \tag{4.1}$$

where d is the element-wise difference between two sets of logits:

$$d = (Z(x) - Z(\tilde{x})), \tag{4.2}$$

and μ is the mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} d_i. \tag{4.3}$$

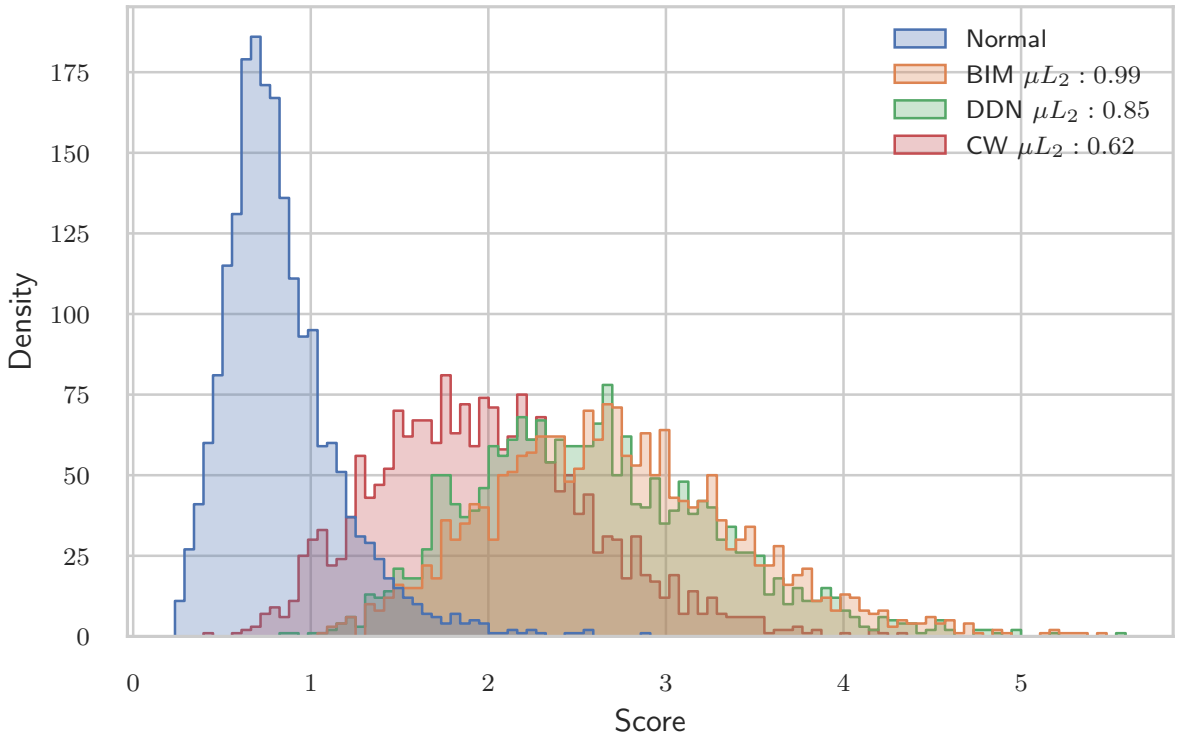
Thus, $score_1$ computes the element-wise difference between two sets of logits and calculate the standard deviation of the remaining vector.

The second score similarly compares two sets of predictions by computing the L_1-norm, or Manhattan distance, as follows:

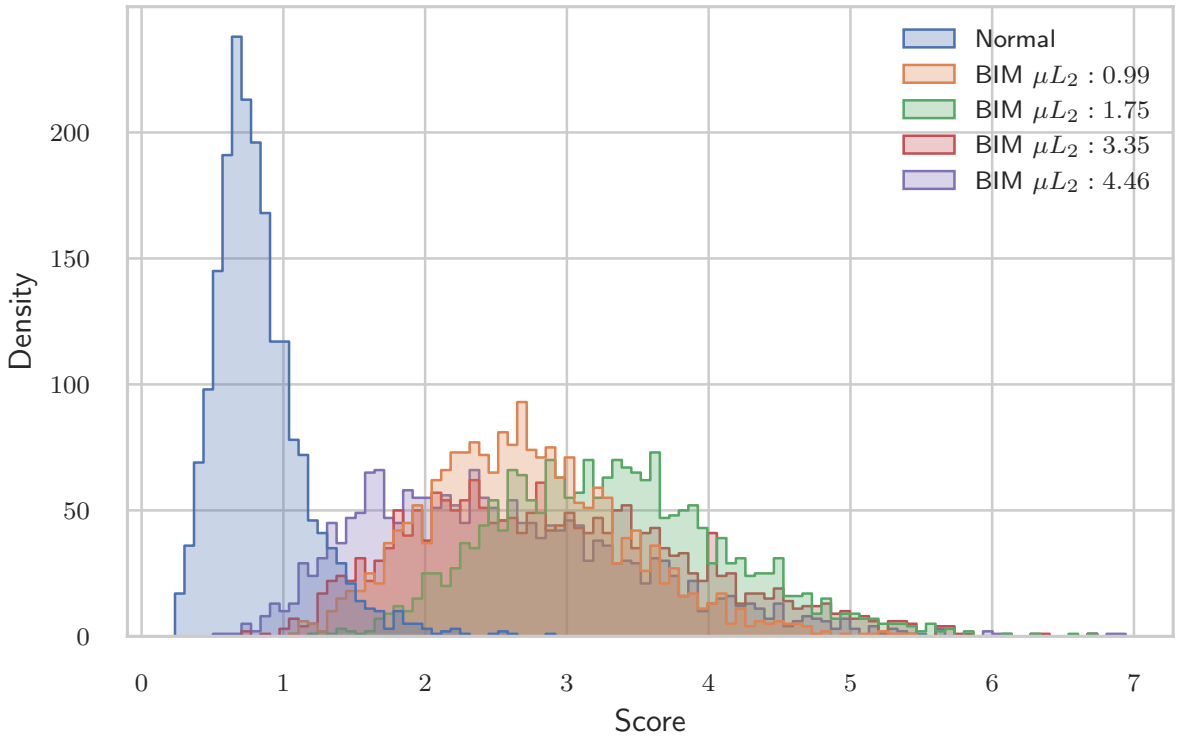
$$score_2(x, \tilde{x}) = |0F(x) - F(\tilde{x})|0_1. \tag{4.4}$$

Notably, $score_1$ uses the model's raw output, i.e., logits, whereas $score_2$ uses the output of the softmax layer.

As seen in figures 4.2 and 4.3, using a κ of 0.03, we observe the distribution of normal images and adversarial examples differs. However, we can observe in 4.3B that when the perturbation budget increases, the distribution of adversarial samples starts to join the same distribution as normal images. I found each score to perform differently for various attacking methods and perturbation budgets. Notably, $score_1$ seem to be less affected by the perturbation budgets of the attacks, offering a better generalization over various budgets.

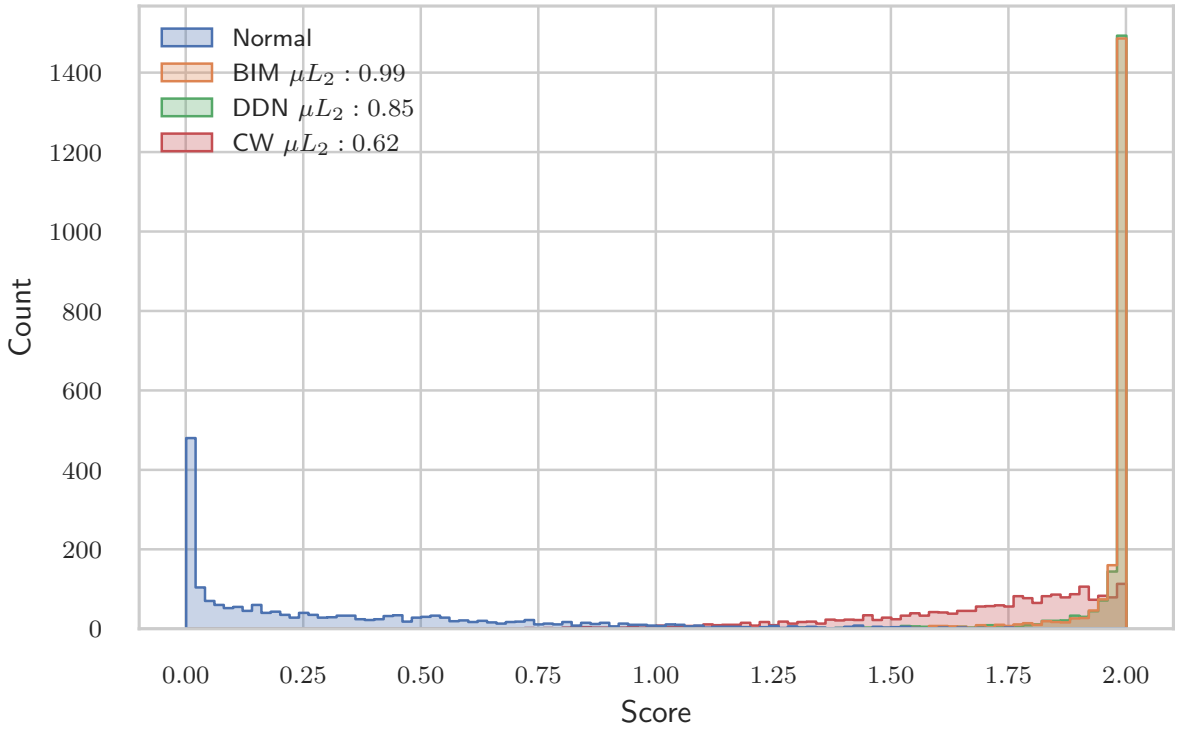


(a) Normal images compared with different adversarial examples generation methods.

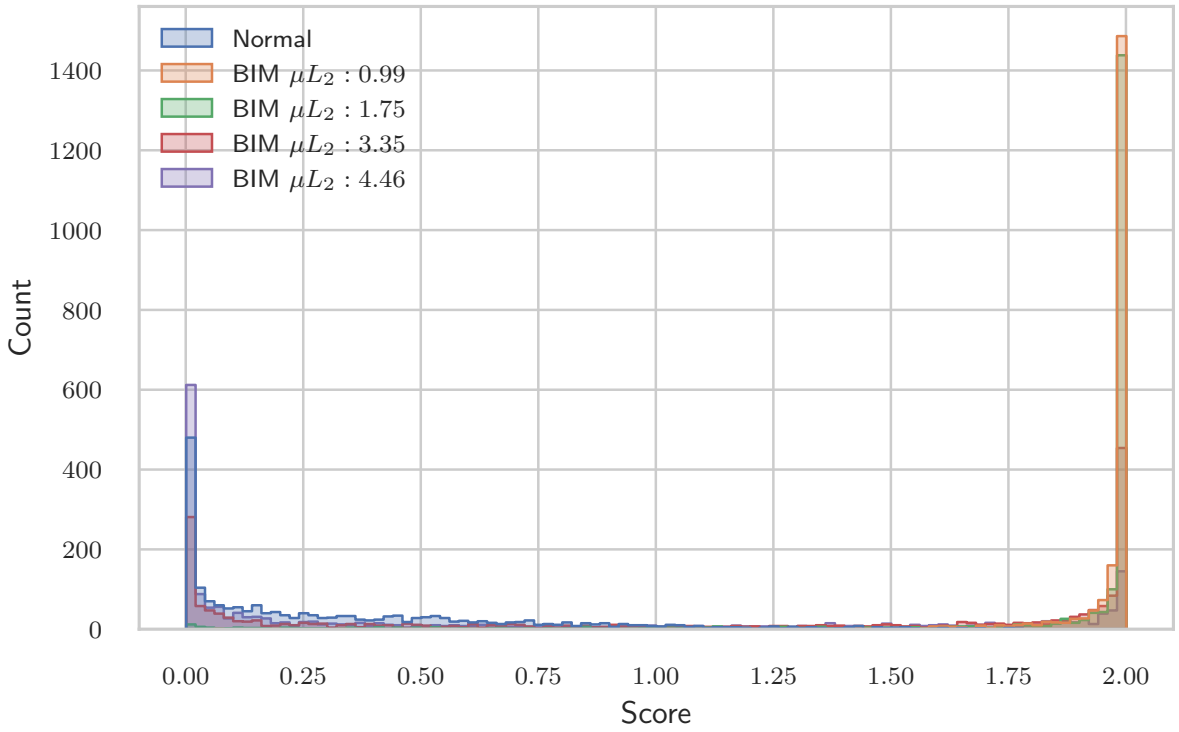


(b) Normal images compared with BIM-generated samples at varying perturbation budgets.

图 4.2 Histograms showing the $score_1$ per sample type.



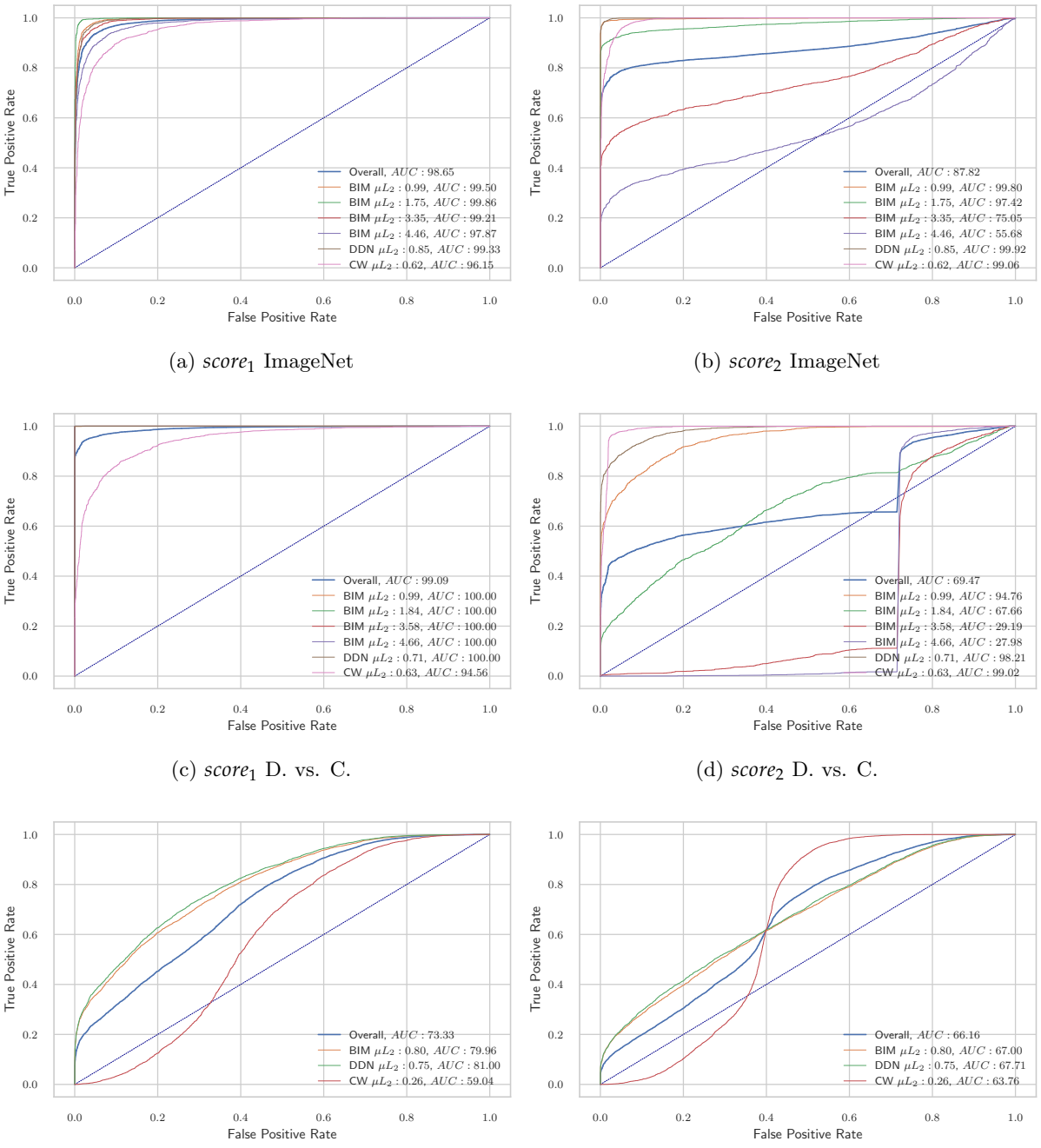
(a) Normal images compared with different adversarial examples generation methods.



(b) Normal images compared with BIM-generated samples at varying perturbation budgets.

图 4.3 Histograms showing the $score_2$ per sample type.

4.2 Method-Specific Approach



(a) $score_1$ ImageNet

(b) $score_2$ ImageNet

(c) $score_1$ D. vs. C.

(d) $score_2$ D. vs. C.

(e) $score_1$ CIFAR-10

(f) $score_2$ CIFAR-10

图 4.4 ROC-AUC for each classifier and each dataset.

To demonstrate the effectiveness of each metric in a detection scenario, using the python library by[34], I build a threshold-based binary classifier for each attacking method on each dataset. The classifiers can either classify an input as negative (normal images) or positive (adversarial example). Figure 4.4 shows the Receiver Operating Characteristic Curve (ROC) and Area Under the Curve (AUC) of each classifier. Following my previous observations, we verify again that $score_1$ can effectively identify

adversarial examples from diverse methods and even at different perturbation sizes, showing an overall AUC of 98.65% on ImageNet and 99.09% on Dogs vs. Cats.

4.3 Method-Agnostic Approach

This method-specific approach described previously is suitable for displaying each classifier's best available result. Unfortunately, it is not a realistic scenario in a production environment as we would not be aware of the exact nature of the input. Furthermore, it would require training a classifier for each attack method at different perturbation sizes.

I propose an approach to select a threshold for the metric scores, only relying on the normal images present in the initial datasets to solve this problem. This approach offers great flexibility and adaptability as we do not require prior knowledge of the attacking method used to generate an adversarial sample. It is essential to select a suitable threshold as the detection effectiveness depends ultimately on it; if the threshold is too high, the number of false positives will be high and vice versa for false negatives.

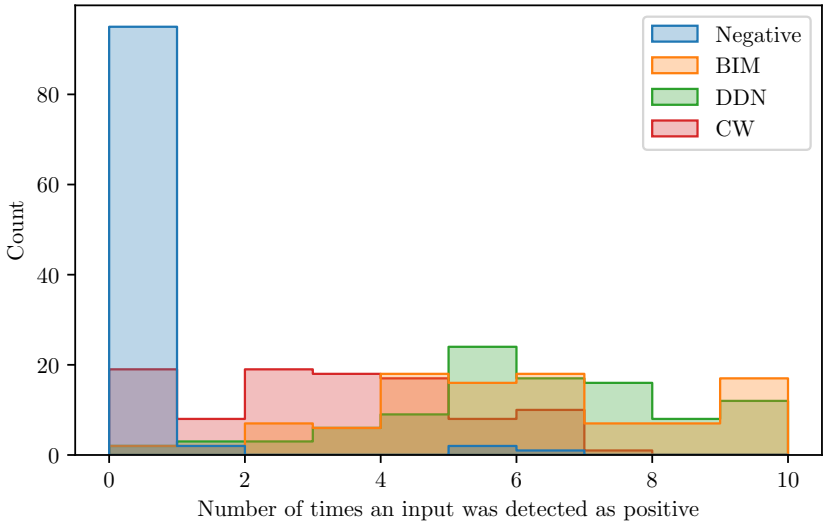
4.3.1 Selection Of The Noise Intensity

Because of the relation between adversarial and random perturbation discussed in Section 3.3.1, to identify adversarial samples with diverse adversarial budgets, we also need to generate random perturbations with a wide range of intensity. To do so, I arbitrarily create a linearly spaced vector containing ten κ from 1×10^{-2} to 1×10^{-1} and accordingly generate ten noisy images per given input. For each input pair x and $\{\tilde{x}\}_{i=1}^{10}$, I compute both $score_1$ (Eq. 4.1) and $score_2$ (Eq. 4.4). The objective is to detect if a sample has an abnormally large value with either of the scores.

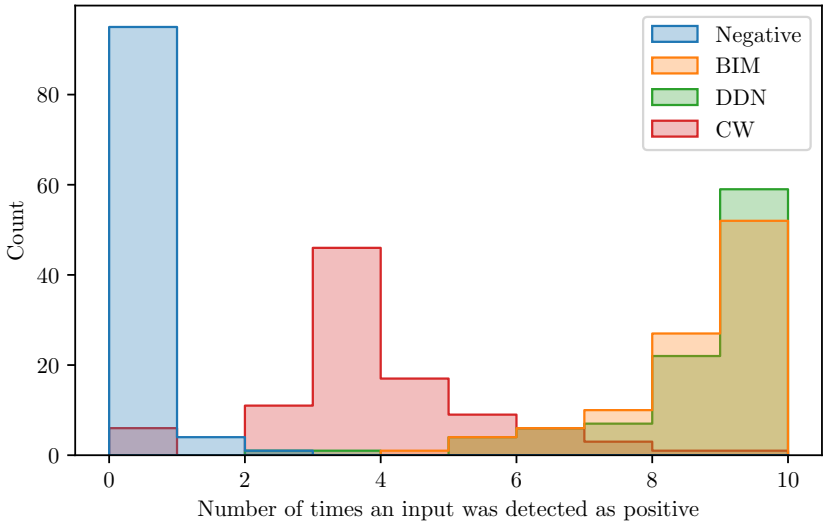
To determine if a sample has an abnormally large value, we need to select a threshold for each score i.e., at each noise intensity. To select these thresholds, I first select 3000 training images from each dataset and record the scores at each κ. The thresholds are then placed at the 99_{th} percentile of these scores obtained on training images. Because the thresholds for each score are determined solely using normal training images, my method does not require prior knowledge of the attack to perform well.

I use BIM, DDN, and CW attack methods to validate this approach and generate 300 adversarial examples from images present in the ImageNet test set. I also select 100 normal images for the negative samples set.

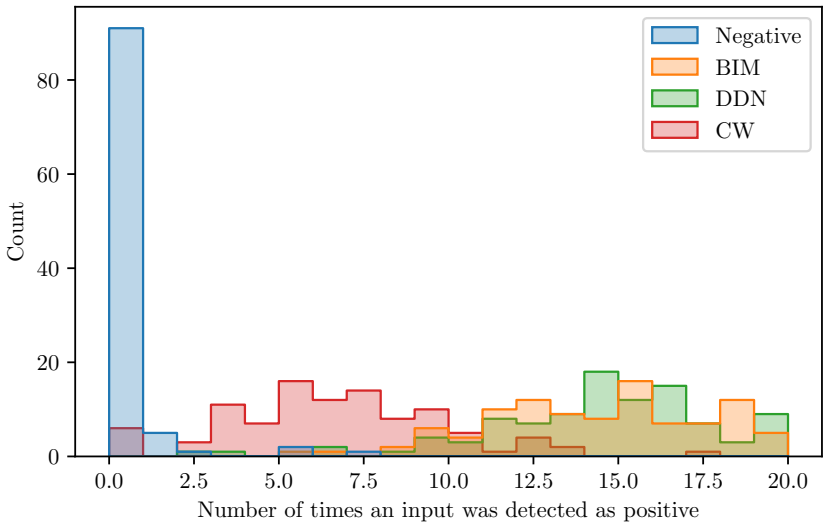
Figure 4.5 shows how many times each input was positively detected. As expected, we can observe that most normal images are not identified as positive once. In contrast, most adversarial examples are identified as positive at least once. Note that, because we compute $score_1$ and $score_2$ at each ten selected κ, a sample can be identified from zero to twenty times as positive.



(a) $score_1$



(b) $score_2$



(c) Scores combined

图 4.5 Histograms showing the number of times an input was detected as positive on ImageNet.

4.4 Detection Results

Following the same procedure as in section 4.3.1, I evaluate the method with adversarial examples generated using different approaches and various perturbation budgets. Table 4.1 shows the results of this evaluation. Using the combined scores, I observed a high overall recall rate of 98.5% on ImageNet and 100% on Dogs vs. Cats. The efficacy of each score depends on the type of attack used; for example, $score_2$ does not perform as well on untargeted attacks but always performs better on CW adversarial examples than $score_1$.

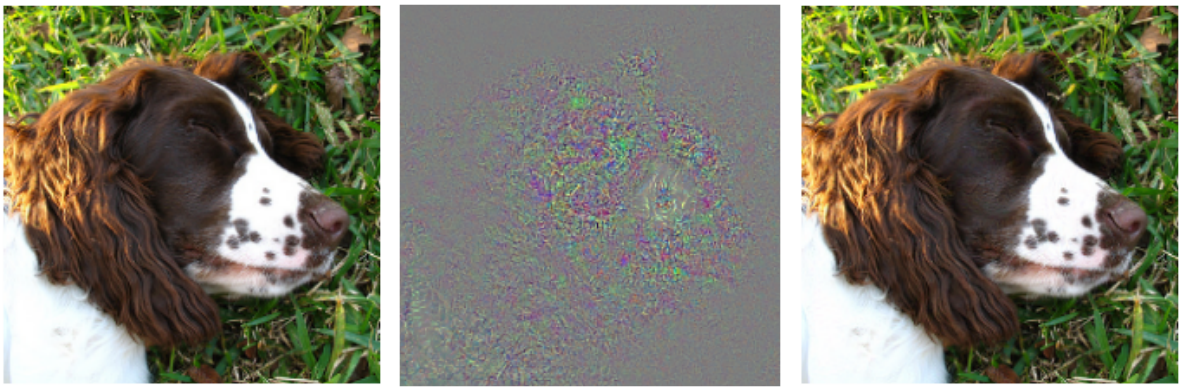
We also note that combining the scores shows to be a good strategy: for the cost of a slight decrease in precision, the recall rate improves as well as the overall F_β score. With BIM, increasing the perturbation budget from an average L_2 perturbation of ≈ 1.00 to ≈ 3.00 does not affect the detection precision performances (Table 4.1, ImageNet/Dogs vs. Cats, No. 1- 3.).

Commonly, detection precision is higher when facing adversarial examples with smaller perturbation sizes, such as samples generated with the CW attack method. For instance, in[35], the authors measure the L_1 distance between the prediction vectors of the original image and its squeezed version, using bit depth reduction as well as local and non-local spatial smoothing. As a result, they show comparable detection rates on adversarial examples generated with the CW method, whereas the detection rate falls to 55.56% on samples crafted with BIM. However, using the same attack settings (BIM_∞, average L_2 of ≈ 1.40), my approach demonstrates a 100.0% detection rate using the combined scores (Table 4.1, ImageNet, No. 5). Therefore, a strength of my method is that it can detect both low-perturbation and large-perturbation adversarial examples.

This adaptability comes from the inputs being transformed with varying random perturbation intensities, which allows for the detection of samples generated with CW (at lower κ values) and detect BIM or similar methods (at higher κ values)—as for the samples generated with the CW method, increasing the number of iterations from 100 (Table 4.1, ImageNet, No. 7) to 10000 (Table 4.1, ImageNet, No. 8) does not reduce the detection performances.

	No.	Attack	L_2	TP			Precision			Recall			F_β		
				$score_1$	$score_2$	*combined*	$score_1$	$score_2$	*combined*	$score_1$	$score_2$	*combined*	$score_1$	$score_2$	*combined*
ImageNet	1		1.00	98	100	100	0.951	0.952	0.917	0.980	1.000	1.000	0.974	0.990	0.982
	2	BIM_2^t	2.00	99	100	100	0.952	0.952	0.917	0.990	1.000	1.000	0.982	0.990	0.982
	3		3.00	95	100	100	0.950	0.952	0.917	0.950	1.000	1.000	0.950	0.990	0.982
	4	BIM_2	1.00	96	63	98	0.950	0.926	0.916	0.960	0.630	0.956	0.958	0.673	0.966
	5	BIM_∞	1.40	99	70	100	0.952	0.937	0.917	0.990	0.700	1.000	0.982	0.737	0.982
	6	DDN_2^t	0.85	98	100	100	0.951	0.952	0.917	0.980	1.000	1.000	0.974	0.990	0.982
	7	CW_2^t	0.46	81	94	94	0.942	0.949	0.913	0.810	0.940	0.940	0.833	0.942	0.934
	8	CW_2^{t*}	0.40	88	98	98	0.957	0.951	0.925	0.880	0.980	0.980	0.894	0.974	0.968
	Total / Average			754	725	790	0.951	0.946	0.918	0.943	0.906	0.985	0.944	0.911	0.973
Dogs vs Cats	1		1.00	100	72	100	0.962	0.935	0.926	1.000	0.720	1.000	0.992	0.755	0.984
	2	BIM_2	2.00	100	46	100	0.962	0.902	0.926	1.000	0.460	1.000	0.992	0.510	0.984
	3		3.00	100	41	100	0.962	0.891	0.926	1.000	0.410	1.000	0.992	0.460	0.984
	4	DDN_2	0.71	100	88	100	0.962	0.946	0.926	1.000	0.880	1.000	0.992	0.892	0.984
	5	CW_2	0.53	91	99	100	0.958	0.952	0.926	0.910	0.990	1.000	0.919	0.982	0.984
	Total / Average			491	346	500	0.961	0.925	0.926	0.982	0.692	1.000	0.977	0.720	0.984
CIFAR-10	1	BIM_2^t	0.75	67	46	67	0.882	0.754	0.798	0.857	0.460	0.670	0.704	0.499	0.692
	2	BIM_2	0.75	78	34	78	0.897	0.694	0.821	0.857	0.340	0.780	0.801	0.379	0.788
	3	BIM_∞	1.00	85	40	85	0.904	0.727	0.833	0.857	0.400	0.850	0.860	0.440	0.847
	4	DDN_2	0.60	78	47	79	0.897	0.758	0.823	0.780	0.470	0.790	0.801	0.509	0.796
	5	CW_2	0.10	28	89	89	0.757	0.856	0.840	0.280	0.890	0.890	0.320	0.883	0.879
	6	CW_2^t	0.22	37	82	82	0.804	0.845	0.828	0.370	0.820	0.820	0.415	0.825	0.822
	Total / Average			373	338	480	0.857	0.772	0.824	0.622	0.563	0.800	0.650	0.589	0.804

表 4.1 Detection results for each dataset. Results are shown for both scores individually ($score_1$ and $score_2$) and combined (*combined*). For each attack, we specify the metric with which it was created (∞ or 2), whether the attack is targeted (t), and the corresponding average L_2 distance.

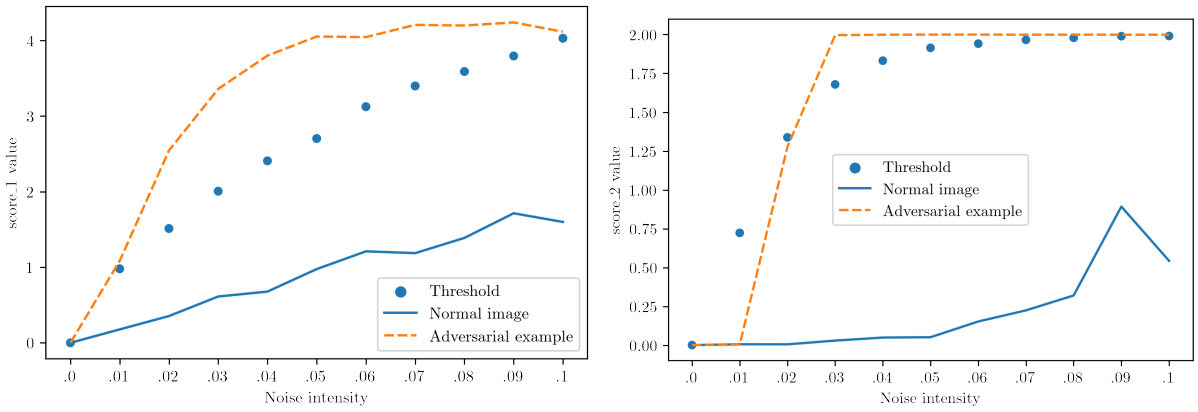


(a) (b) (c)

图 4.6 Normal image (A), adversarial perturbation (magnified by 10) (B), adversarial example (C).

4.4.1 Showcasing With One Sample

To visualize a concrete example, I randomly select one image from the ImageNet validation set where I perform the previously described method. I then generate an adversarial example using BIM. Figure 4.6 shows the base image (A), the adversarial perturbation (B) magnified by ten, and the final result adversarial example (C) that is identified as a ”running shoe” by the model.



(a) (b)

图 4.7 comparing the number of times the thresholds are respected between a normal image and an adversarial example. $score_1$ in (A), $score_2$ in (B).

Processing 4.6A and 4.6B through my approach gives us the results shown in figure 4.7. Plot (A) shows us when a sample's $score_1$ values exceed the threshold values. As expected and verified at the beginning of this section, the normal image's scores values stay under the corresponding thresholds. In contrast, the scores of the adversarial examples exceed the thresholds in a few points; remember that exceeding the threshold at any given noise intensity is enough to tag the image as being positive,

i.e., adversarial.

Similarly, with figure 4.7B, where we can observe the same scenario happening, thresholds are exceeded by the adversarial example while the normal image's scores remain under them.

第 5 章 Discussion

5.1 Low Resolution Images

As shown in table 4.1, the detection performances on CIFAR-10 are inferior compared to ImageNet or Dogs vs. Cats. I suppose that the cause of this inferior result is that the perturbation needed to produce adversarial examples on lower-resolution images (e.g., the ones from CIFAR-10) is proportionally larger than on images with higher resolution. Furthermore, as discussed in section 3.3, more significant adversarial perturbations are more robust to the random noise we add to the images. Consequently, we need to use a larger κ to improve the detection for adversarial examples that contain a more significant relative adversarial perturbation. However, while using a larger κ proved to be helpful to detect larger-perturbation adversarial examples, a κ too high can deteriorate normal images, leading the model not to recognize them, thus, increasing the number of false positives.

5.2 Adaptive Adversaries

In all the experiments I presented, I considered adversaries who have full access to the model parameters but do not adapt to bypass the detection method. To do so, an adversary would need to produce examples such that both scores remain under unknown thresholds, which is a considerably more challenging problem than simply crafting adversarial examples. More challenging, primarily because of the random nature of the technique, it would be extremely difficult and time-consuming to generate adversarial examples that keep their adversarial effects at varying unknown-to-the-adversary noise amounts.

Thus, it is highly challenging to generate adversarial examples that generalize and keep their effect under all these unknown and random parameters.

5.3 Future Work

During my experiments, I wanted to explore different ways of detecting scores anomalies. One of the ways I briefly experimented with was to plot the scores differences at each noise intensity step. For example, given ten κ, I computed the first-order

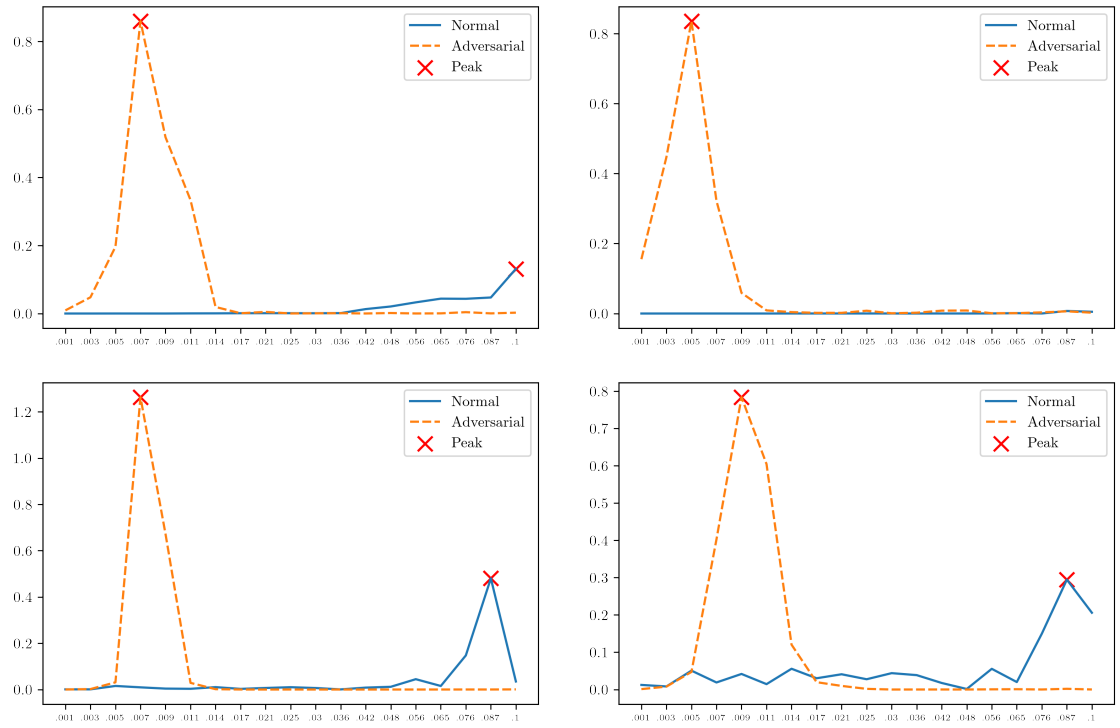


表 5.1 Peaks observed on normal and adversarial images. Peaks on adversarial examples appear to happen at a sooner noise intensity and reach a higher value.

difference like so:

$$out_i = |\alpha_{i+1} - \alpha_i|, \tag{5.1}$$

where α_i represents the score result (either $score_1$ or $score_2$ seen in 4.1 and 4.4) at κ_i noise intensity.

Figure 5.1 shows the plot of these score differences for four random images and four adversarial examples generated with different methods. We observe that the adversarial examples display high peaks earlier than normal images. In most cases, normal images do not contain peaks at all. This observation could be used to detect adversarial examples, e.g., by comparing the intensity of the peaks and when they occur, i.e., when the first-order difference peaks at an earlier κ, and at a high magnitude, this could indicate the adversarial nature of the input.

Furthermore, for future research, I believe the detection performances of my method could be improved by adding Gaussian noise as a form of data augmentation during the training phase. Adding noise to the training data has been done before to reduce overfitting or stabilize the models like done by[36]. However, in combination with my method, I believe that training the model with randomly modified inputs

would improve the model's robustness to noise and, thus, hypothetically reduce the number of false positives, especially on lower resolution images like CIFAR-10, which increases detection precision.

总结与展望

In this research work, I investigated in many ways the effects of applying Gaussian noise to normal images and adversarial examples on the prediction of convolutional neural networks. Motivated by the observed disparities, I developed a method for detecting adversarial examples based on computing two scores. Contrary to many other techniques, it does not require prior knowledge of the attack, with the advantages of being optimization-free and having a low computation cost.

Moreover, because of the low computation cost and easy implementation, my method can easily be compounded with other existing defense methods to optimize detection performances.

参考文献

[1] Carlini N, Wagner D. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. arXiv:1801.01944 [cs], 2018.

[2] Su J, Vargas D V, Kouichi S. One pixel attack for fooling deep neural networks. IEEE Transactions on Evolutionary Computation, 2019, 23(5):828–841

[3] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 2017, 60(6):84–90

[4] Zeiler M D, Fergus R. Visualizing and Understanding Convolutional Networks. arXiv:1311.2901 [cs], 2013. ArXiv: 1311.2901

[5] Russakovsky O, Deng J, Su H, et al. ImageNet Large Scale Visual Recognition Challenge. arXiv:1409.0575 [cs], 2015.

[6] Amodei D, Anubhai R, Battenberg E, et al. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. arXiv:1512.02595 [cs], 2015.

[7] Bojarski M, Del Testa D, Dworakowski D, et al. End to End Learning for Self-Driving Cars. arXiv:1604.07316 [cs], 2016.

[8] Silver D, Huang A, Maddison C J, et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature, 2016, 529(7587):484–489

[9] Szegedy C, Zaremba W, Sutskever I, et al. Intriguing properties of neural networks. arXiv:1312.6199 [cs], 2014.

[10] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs], 2015.

[11] Vaswani A, Shazeer N, Parmar N, et al. Attention Is All You Need. arXiv:1706.03762 [cs], 2017.

[12] Huang G, Liu Z, van derMaaten L, et al. Densely Connected Convolutional Networks. arXiv:1608.06993 [cs], 2018.

[13] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 2014, 15(56):1929–1958

[14] Hendrycks D, Dietterich T G. Benchmarking Neural Network Robustness to Common Corruptions and Surface Variations. arXiv:1807.01697 [cs, stat], 2019.

[15] Warren S. McCullochWalter Pitts W P. A logical calculus of the ideas immanent in nervous activity. https://doi.org/10.1007/BF02478259, 1943. 19

[16] Brain I T, Rosenblatt F. The Perceptron: A Probabilistic Model for Information Storage and Organization. https://doi.org/10.1037/h0042519, 1958

[17] Minsky M, Papert S A. Perceptrons: An Introduction to Computational Geometry. 1969.

[18] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors. Nature, 1986, 323(6088):533–536. Number: 6088 Publisher: Nature Publishing Group

[19] LeCun Y, Haffner P, Bottou L, et al. Object Recognition with Gradient-Based Learning. In: Forsyth D A, Mundy J L, diGesú V, et al, (eds.). Proc of Shape, Contour and Grouping in Computer Vision, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999: 319–345

[20] Model Hacking ADAS to Pave Safer Roads for Autonomous Vehicles, 2020-February. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/model-hacking-adas-to-pave-safer-roads-for-autonomous-vehicles/

[21] Goodfellow I J, Shlens J, Szegedy C. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [cs, stat], 2015.

[22] Szegedy C, Liu W, Jia Y, et al. Going Deeper with Convolutions. arXiv:1409.4842 [cs], 2014.

[23] Kurakin A, Goodfellow I, Bengio S. Adversarial examples in the physical world. arXiv:1607.02533 [cs, stat], 2017.

[24] Carlini N, Wagner D. Towards Evaluating the Robustness of Neural Networks. arXiv:1608.04644 [cs], 2017.

[25] Papernot N, McDaniel P, Wu X, et al. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In: Proc of 2016 IEEE Symposium on Security and Privacy (SP). 2016, 582–597

[26] Rony J, Hafemann L G, Oliveira L S, et al. Decoupling Direction and Norm for Efficient Gradient-Based L2 Adversarial Attacks and Defenses. arXiv:1811.09600 [cs], 2019.

[27] Papernot N, McDaniel P, Jha S, et al. The Limitations of Deep Learning in Adversarial Settings. arXiv:1511.07528 [cs, stat], 2015.

[28] Carlini N, Wagner D. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. arXiv:1705.07263 [cs], 2017.

[29] Feinman R, Curtin R R, Shintre S, et al. Detecting Adversarial Samples from Artifacts. arXiv:1703.00410 [cs, stat], 2017.

[30] Krizhevsky A. Learning Multiple Layers of Features from Tiny Images. 2009.

[31] Elson J, Douceur J J, Howell J, et al. Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization. 2007.

[32] Simonyan K. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015.

[33] Rauber J, Zimmermann R, Bethge M, et al. Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX. Journal of Open Source Software, 2020, 5(53):2607

[34] Buitinck L, Louppe G, Blondel M, et al. API design for machine learning software: experiences

from the scikit-learn project. arXiv:1309.0238 [cs], 2013. ArXiv: 1309.0238

[35] Xu W, Evans D, Qi Y. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. Proceedings 2018 Network and Distributed System Security Symposium, 2018.

[36] Zheng S, Song Y, Leung T, et al. Improving the Robustness of Deep Neural Networks via Stability Training. 2016. 4480–4488

致 谢

First and foremost, I would like to thank my supervisor and teacher, Professor Jiang Bin, for the valuable lessons about Intelligent Optimization Algorithms and the mentoring from afar during my research work. These lessons, combined with the sessions where students needed to present a thesis and debate about it, were eye-opening and provided excellent preparation for our research work.

Because of Covid-19, I deeply regret my inability to return to China to continue adequately with my studies. Being 9000km away from the laboratory and its incredible students, my supervisor and all the helping resources and hardware may have interfered slightly with my research. Still, regardless of the many obstacles, I wholeheartedly gave my best to deliver a work I can be proud of.

Secondly, I would thank my friend MELNYK Pavlo, a Ph.D. student, for his guidance throughout the research process, who helped me shape the overall structure of this thesis and provided needed encouragement alongside the occasional pep talks!

During these two years of researching and writing my thesis, I also had the opportunity to work as a Software Engineer at a few start-ups, working with Python and PyTorch, utilizing the knowledge I leveraged during my research and experiments. These experiences working alongside the studies and research work done at Hunan University definitely confirmed my intention to pursue this field of work.

I would also like to thank Hunan University for allowing me first to study Mandarin and then for my Master in Computer Science. Finally, I am grateful to China Scholarship Council (CSC) for awarding me a scholarship that allowed me to study in China.

Regardless of being unable to return to China, this country has held a deer spot in my heart, and it has since the very first moment I arrived in China. Its welcoming and giving people, mouthwatering food, and beautiful sceneries, I cannot wait to be able to return, to give back some of what I received.