Abstract

Neural ordinary differential equations (ODEs) is an object of increasing attention in various fields of machine learning. Nevertheless the robustness of the NODEs has been studied poorly. In this work, we are going to explore robustness of this neural network model for a wider range of adversaries. We first investigate theoretical background behind NODEs and show their connection to the conventional residual neural networks. Then we apply various set of attacks in order to investigate how robust NODEs can be and their effectiveness in comparison to ResNet. Finally we apply some defense strategies in order to increase robustness of the model.

Contents

List of Figures v

1 Introduction **1**

2 Models **3**

2.1 ResNet 3

2.2 ODE-net 4

2.3 Architectures of the Models 9

3 Dataset **11**

4 Adversarial attacks and defenses **13**

5 Adversarial attack methods **18**

5.1 Carlini & Wagner L_2 Attack 18

5.2 Zeroth-Order Optimization Attack 23

5.3 Boundary Attack 23

5.4 Backdoor Poisoning Attack 28

6 Adversarial defense strategies **30**

6.1 Adversarial Training 31

6.2 Feature Squeezing 31

7 Evaluating robustness of the models **33**

7.1 Models Performance on the Original Data 33

7.2 Applying Attacks to the Models 34

7.3 Applying Defense Strategies 36

8 Conclusion 38

Bibliography 39

List of Figures

2.1 Skip-Connection 3
2.2 Difference between ResNet & ODE-net 6
2.3 Model with ODE block 10
2.4 Model with residual blocks 10

3.1 Images 12

5.1 C&W attack, *max_iter* parameter 20
5.2 ResNet, C&W attack, grid search of the hyperparameters 21
5.3 ODE-net, C&W attack, grid search of the hyperparameters 22
5.4 Boundary Attack 25
5.5 Boundary Attack, *max_iter* parameter 26
5.6 Classes are grouped by visual similarity 27
5.7 ODE-net & ResNet, grid search of PAB attack parameters 29

6.1 C&W attack. Bit depth vs Accuracy plot 32

7.1 ODE-net & ResNet performances 34
7.2 ODE-net & ResNet, ROC-AUC curves 35

Chapter 1

Introduction

Recently, a new type of artificial neural networks were introduced by Chen et al. It was named Neural Ordinary Differential Equations [1]. The idea of this approach is based on the notion that numerical method of solving ODEs which is called Euler‘s method has implementation as a residual connection in ResNet architecture. In the case of residual neural networks the residuals between two fixed points is learned by the network. Unlike the ResNet, NODEs parametrize the local derivative of the input with a neural network block. Then an ODE solver make a step from an input x_i at time t_i to an output x_{i+1} at time t_{i+1}, taking the derivative at each point.

There also have been significant study of adversarial examples recent years, the data which is constructed by an adversary in order to fool the machine learning system.

This work can be divided into three parts. First we are going to discuss the theoretical background behind neural ordinary differential equations. Then we are going to describe various adversarial attack and defense methods. As a final step we will provide a set of experiments in order to evaluate the robustness of NODE architecture.

In order to build models and run experiments we are going to use several packages and frameworks. Both of the models are implemented using Pytorch an optimized tensor library for deep learning using GPUs. ODE block in ODE-net model is written using torchdiffeq[13] library. This library provides ordinary differential equation (ODE) solvers implemented in PyTorch. Backpropagation through all solvers is supported using the adjoint method. As the solvers are implemented in PyTorch, algorithms in this repository are fully supported to run on the GPU. All the attacks and defenses are provided using Adversarial Robustness Toolbox (ART). A Python library for ML Security. ART provides tools that enable developers and researchers to defend and evaluate Machine Learning models against the adversarial threats.

Chapter 2

Models

2.1 ResNet

One of the most successful and breakthrough ideas in the field of deep learning in the last few years is the creation of a residual network architecture(Residual Networks, ResNets)[3]. Researchers from Microsoft proposed a fairly simple modification of conventional neural network architectures: the addition of so-called skip-connections [2.1], which allow the input signal to pass through the network, bypassing nonlinear layer transformations. More formally, if $z(t) \in \mathbb{R}^D$ is the input of the t's layer of the neural network and $f_t(z, \theta) : \mathbb{R}^D - > \mathbb{R}^D$ is a transformation of the layer t, then the

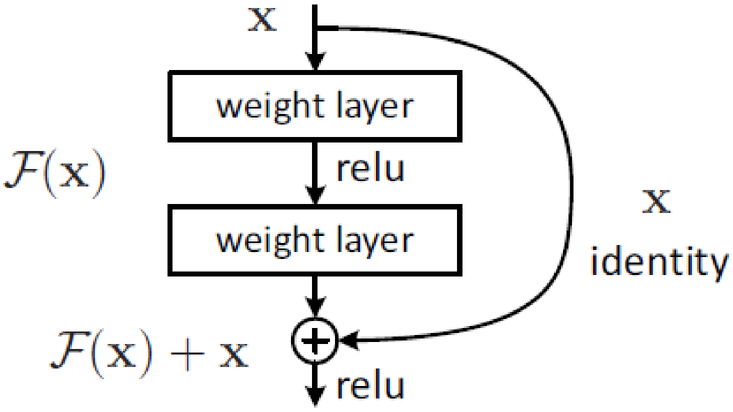

Figure 2.1. Skip-Connection

output of this layer $z(t+1)$ is calculated as follows:

$$z(t+1) = z(t) + f_t(z(t), \theta) \tag{2.1}$$

Despite the simplicity of the idea, this architecture has made it possible to successfully train very deep neural networks. One of the main problems of training neural network architectures with a large number of layers is the problem of vanishing gradients: a situation when, while training, the value of the gradient during the backpropagation becomes very close to 0. Adding skip-connections allows to get rid of this problem, since the gradient has the ability to pass through not only the nonlinear transformation of the layer, but also through the identity transformation.

2.2 ODE-net

Let's put aside the neural networks and consider an ordinary differential equation (ODE) of the first order with a primary condition:

$$\begin{cases} \frac{dz(t)}{d(t)} = f(z(t), t, \theta) \\ z(0) = x, t \in [0, T] \end{cases} \tag{2.2}$$

It is necessary to find the value of the function $z(t)$ at the moment $t_1 \in [0, T]$:

$$z(t_1) = z(t_0) + \int_0^{t_1} f(z(t), t, \theta) dt$$

Calculation of $z(t_1)$ requires integration of the derivative of $z(t)$ and cannot always be performed analytically.

There are various numerical methods for solving this differential equation, the simplest one of them is the Euler's method. In this method, the interval $[0, T]$ is divided by a uniform grid with a step size Δt. The value of the approximate solution at time t is set with a small step towards the gradient of the function $z(t)$ from the time $t-1$. Formally, an approximate solution is determined by the formula:

$$z(t + \Delta t) = z(t) + \Delta t f(z(t), t, \theta)$$

Note the extreme similarity of expression (1.4) in Euler's method with expression (1.1) for calculating the output of a block with a skip-connection in the ResNet architecture. In particular, if we put $\Delta t = 1$ in Euler's method, then the above expressions coincide. Thus, a direct analogy can be drawn between the residual networks and differential equations of the type (1.2): computing the output of the residual network block can be considered as one iteration of the Euler's method for solving the differential equation. It is known that Euler's method has a number of significant disadvantages. For example, this method has a large error, accumulates error at each step and is computationally unstable. However, there are more advanced numerical methods for solving differential equations of the type (1.2), which lack of these drawbacks at a great extent. Thus, instead of the numerical scheme (1.4), it is possible to consider directly the differential equation (1.2) itself and train its parameters. In this context a neural network is a function $f(z(t), t, \theta)$, which determines the dynamics of the function $z(t)$. Here we need to mention that, unlike conventional neural networks, the θ parameters are common for all points in time. Mathematically, this means that $\frac{d\theta(t)}{dt} = 0$.

We define the value at the final time $y = z(T)$ as the output of the neural differential equation model, given the initial condition $z(0) = x$, where x is the input to the model (for example, the image to be classified). This value can be calculated using any numerical method for solving differential equations (not necessarily Euler!).

Figure [2.2] demonstrates the fundamental difference between the ResNet model and the neural differential equation model. The residual network defines a discrete sequence of transformations. As can be seen in the figure on the right, the neural differential equation defines a continuous in time transformation of model inputs to outputs. The points in both figures correspond to the moments of time $t \in [0, T]$. This are the calculations of intermediate values $z(t)$.

How a model of a neural differential equation is trained? As usual, training a model is about to minimize the loss function L with respect to the parameters θ:

$$\begin{aligned} L(y) = L(z(T)) =& L\Big(z(0) + \int_0^T f(z(t), t, \theta) dt\Big) = \\ &= L(ODESolve(z(0), f, 0, T, \theta)) \to \min_{\theta} \end{aligned} \tag{2.3}$$

where $ODESolve$ is a numerical method for solving differential equations. In order to train the model by standard gradient methods, it is necessary to be able to calculate the gradient of the loss function using the parameters $\frac{\partial L}{\partial \theta}$.

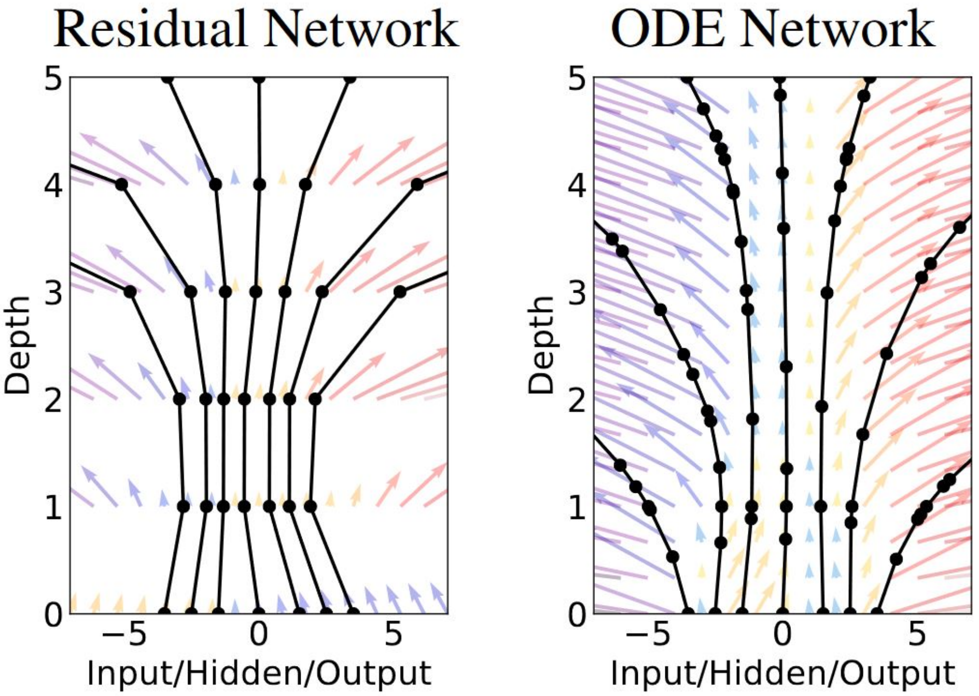

Figure 2.2. Difference between ResNet & ODE-net

Theoretically, the gradient can be calculated using the backpropagation algorithm, since the operations of the numerical $ODESolve$ method are usually differentiable. However, this approach has a number of significant disadvantages. First, it is high memory costs due to the necessity to store intermediate results of calculations of the numerical method. Secondly, this approach causes additional numerical errors, since it implies the differentiation of the approximation, and not the approximation of the "fair" gradient.

An alternative approach to calculating the desired gradient is based on the concept of a conjugate function. This method does not require information about the internal structure of the applied numerical method, treating it as a black box. The approach calculates gradients using yet another differential equation, solving it back in time. It should be noted that for simplicity of notation, all calculations will be carried out for the one-dimensional case, but they can be easily generalized to the multi-dimensional case.

So, let us define the conjugate function $a(t)$, setting by definition $a(t) = \frac{\partial L}{\partial z(t)}$. This function has the following intuitive interpretation. It shows us how much the value of the loss function L will change when slightly changing the value of z at the time t. By changing the value of z, we thereby change the trajectory. Therefore we change the end point of $z(T)$, and, consequently, the value of the loss function $L(z(T), \theta)$.

Let us show that the dynamics of the conjugate function is given by the following differential equation with the boundary condition:

$$\begin{cases} \frac{\partial a(t)}{\partial t} = -a(t)\frac{\partial f(z(t),t,\theta)}{\partial z(t)} \\ a(T) = \frac{\partial L}{\partial z(T)}, t \in [0, T] \end{cases} \tag{2.4}$$

The proof of (1.6) enables us to find the value of the conjugate function $a(t)$ at any point of the interval $[0, T]$ using numerical methods.

By the rule of differentiation of a complex function for $\epsilon > 0$ holds:

$$\frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t+\epsilon)}\frac{\partial z(t+\epsilon)}{\partial z(t)} + \frac{\partial L}{\partial \theta(t+\epsilon)}\frac{\partial \theta(t+\epsilon)}{\partial z(t)} \tag{2.5}$$

Since $\theta(t+\epsilon) = \theta(t) = \theta$, then $\frac{\partial \theta(t+\epsilon)}{\partial z(t)} = \frac{\partial \theta}{\partial z(t)} = 0$ and therefore

$$\frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t+\epsilon)}\frac{\partial z(t+\epsilon)}{\partial z(t)} \; or \; a(t+\epsilon) = \frac{\partial z(t+\epsilon)}{\partial z(t)} \tag{2.6}$$

Decomposing $z(t+\epsilon)$ at the point t in Taylor series, we get:

$$\frac{\partial z(t+\epsilon)}{\partial z(t)} = \frac{\partial}{\partial z(t)}(z(t) + \epsilon f(z(t), t, \theta) + \mathcal{O}(\epsilon^2)) = 1 + \epsilon\frac{\partial}{\partial z(t)} f(z(t), t, \theta) + \mathcal{O}(\epsilon^2) \tag{2.7}$$

Substituting (1.9) into (1.8) we get:

$$a(t) = a(t+\epsilon)(1 + \epsilon\frac{\partial}{\partial z(t)} f(z(t), t, \theta) + \mathcal{O}(\epsilon^2)) \tag{2.8}$$

Then by definition of derivative

$$\begin{aligned} \frac{\partial a(t)}{\partial t} = \lim_{\epsilon \to +} \frac{a(t+\epsilon) - a(t)}{\epsilon} = \lim_{\epsilon \to 0+} \frac{-\epsilon a(t+\epsilon)\frac{\partial}{\partial z(t)} f(z(t), t, \theta) + \mathcal{O}(\epsilon^2)}{\epsilon} = \\ = -a(t)\frac{\partial}{\partial z(t)} f(z(t), t, \theta) \end{aligned} \tag{2.9}$$

It is also known the boundary condition at the end time $a(T) = \frac{\partial L}{\partial z(T)}$. Thus, the validity of (1.6) is proved.

Let's show some analogies with the discrete case to simplify the understanding of the above calculations:

- Discrete case:

$$\begin{aligned}
&z(t+1) - z(t) = f_t(z(t), \theta)\\
&a(t) = a(t+1)\frac{\partial z(t+1)}{\partial z(t)}\\
&a(t+1) - a(t) = -a(t+1)\frac{\partial f_t(z(t), \theta)}{\partial z(t)}
\end{aligned}$$

- Continuous case:

$$\begin{aligned}
&\frac{dz(t)}{d(t)} = f(z(t), t, \theta)\\
&a(t) = a(t+\epsilon)\frac{\partial z(t+\epsilon)}{\partial z(t)}\\
&\frac{a(t)}{\partial t} = -a(t)\frac{\partial f(z(t), t, \theta)}{\partial z(t)}
\end{aligned}$$

Thus, using system (1.6) and some numerical method, it is possible find the value of $\frac{\partial L}{\partial z(t)}$ at the time moment $t \in [0, T]$.

Similar reasoning can show that the dynamics of the gradient $\frac{\partial L}{\partial \theta}$ is given by the following differential equation:

$$\begin{cases}
\frac{\partial}{\partial t}\frac{\partial L}{\partial \theta(t)} = -a(t)\frac{\partial f(z(t),t,\theta)}{\partial \theta}\\
\frac{\partial L}{\partial \theta(T)} = 0, t \in [0, T]
\end{cases} \tag{2.10}$$

Note that the partial derivatives $\frac{\partial}{\partial z(t)} f(z(t), t, \theta)$ and $\frac{\partial}{\partial \theta} f(z(t), t, \theta)$ that determine the dynamics of the functions $a(t)$ and $\frac{\partial L}{\partial \theta(t)}$, respectively, can be calculated using the standard backpropagation algorithm. Finally,let us describe the final algorithm for training the neural differential equation model. To train the model, it is necessary to be able to calculate the value of the gradient of the loss function at the initial moment of time $\frac{\partial L}{\partial \theta}(0)$. The dynamics of the gradient $\frac{\partial L}{\partial \theta}(t)$ depends on $a(t)$ and $z(t)$.

Thus, in order to apply numerical methods to solve (1.12), it is necessary to be able to calculate the values of $a(t)$ and $z(t)$ at arbitrary points of the interval $[0, T]$. The value of $z(t)$ can be obtained by numerically solving the original equation (1.2) back in time, starting from the point $z(T)$. Now, it becomes possible to find the value of $a(t)$ using the numerical solution (1.6). After the values $a(t)$ and $z(t)$

become known, the value of $\frac{\partial L}{\partial \theta(t)}$ at the next point can be calculated by numerically solving (1.12). Finally, the gradient of the tunable loss function can be calculated by numerically solving (1.2), (1.6) and (1.12) back in time. All integrals for finding z, a and $\frac{\partial L}{\partial \theta}$ can be calculated with a single call to the numerical method $ODESolve$.

The concept of neural differential equation has several advantages over conventional neural networks:

- Memory-efficient learning. Because of that there is no need to store intermediate values calculated during the forward pass, it is possible train a model where the memory consumption is independent of the "depth". The "depth" is meant to be the number of evaluations of a function at intermediate points.
- Adaptive running time. Modern numerical methods for solving differential equations make it possible to explicitly indicate the desired accuracy of a numerical solution, choosing independently the required partition. Thus, there is an explicit way to control the trade-off between the numerical accuracy of the solution and the computational cost. This property of the method makes it possible to train a model with high accuracy, and during testing deliberately lowering it, thereby reducing the amount of required computational resources.

2.3 Architectures of the Models

In this work we are going to use the following architecture of the ODE network. This model is a modified version of the ODE-net described in [1] for the classification experiment with MNIST dataset. The modification were mainly applied to the classifier part in order to fit new dataset. The model is a small residual network which down-samples the input twice then applies 6 standard residual blocks, which are replaced by an ODESolve module. The adjoint sensitivity method is implemented in Python's autograd framework. In this work, the hidden state dynamics and their derivatives are evaluated on the GPU using Pytorch, which were then called from the Fortran ODE solvers, which were called from Python autograd code [13].

We will use ResNet model as the closest model in order to compare the performance of the ODE-net model and check the transferability of adversarial examples.

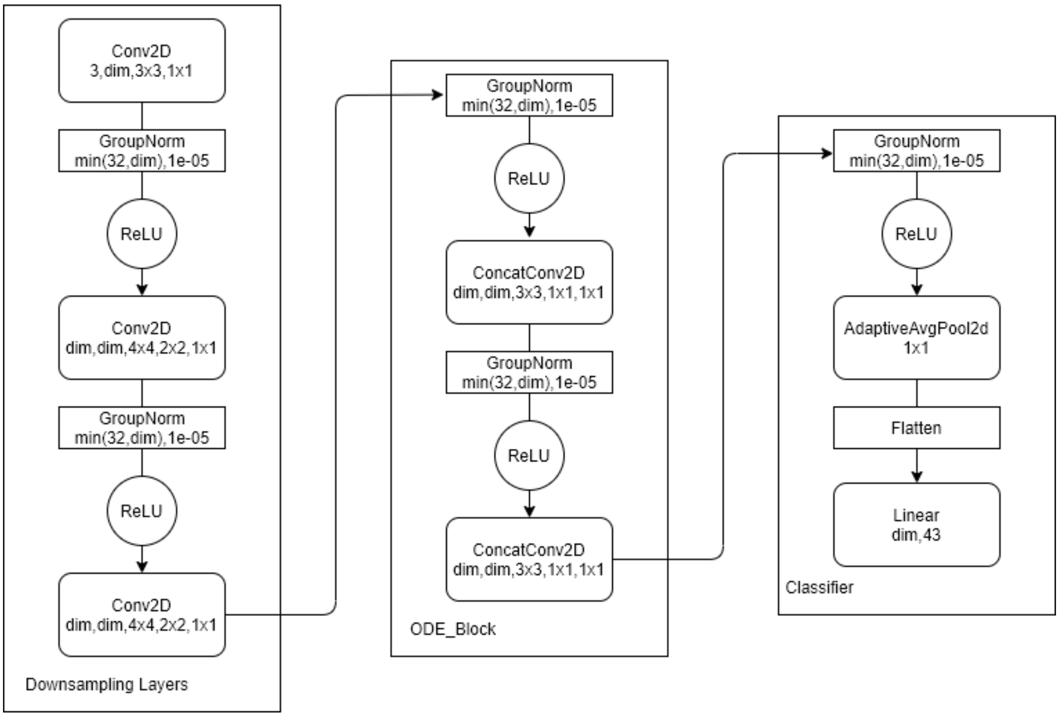

Figure 2.3. Model with ODE block

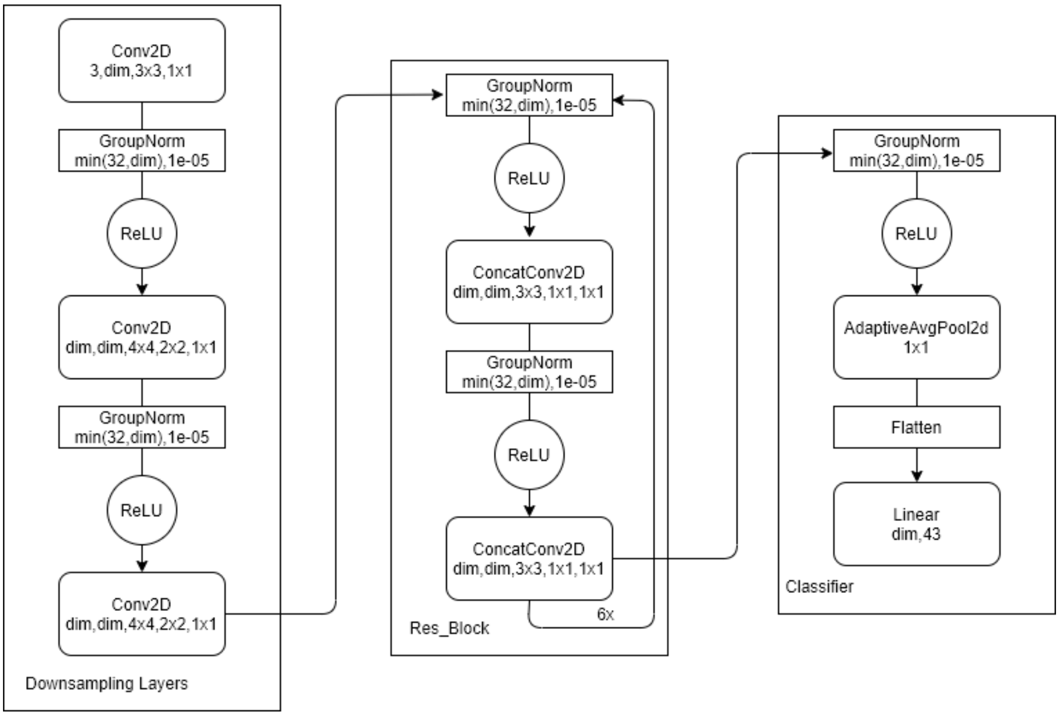

Figure 2.4. Model with residual blocks

Chapter 3

Dataset

We use the German Traffic Sign Recognition Benchmark (GTSRB). This dataset collected implying single-image, multi-class classification problem. GTSRB contains around 40.000 images which are divided into 43 classes. This dataset is large and lifelike with reliable ground-truth data due to semi-automatic annotation. Traffic signs are unique within the dataset. The training set archive has the following structure. All the classes are placed into separate directories. There is a CSV file in each directory with annotations. Each traffic sign is performed within a group of 30 images. This group corresponds to separate track. We can see only one traffic sign on each image. Format of the images is PPM (Portable Pixmap, P6). All images are RGB and sized starting from 15×15 to 250×250. Images are not necessarily squared and the traffic signs in that images are not necessarily centered [12]. We have pre-processed images before feeding them to our model in order to have faster computations. We resized all images to the size 28×28 without changing the number of color channels. So the final dataset consists of squared RGB images. We are going to divide it to the training and test sets with proportion of 0.1, which means that the training set contains 90% of images and the test set 10%.

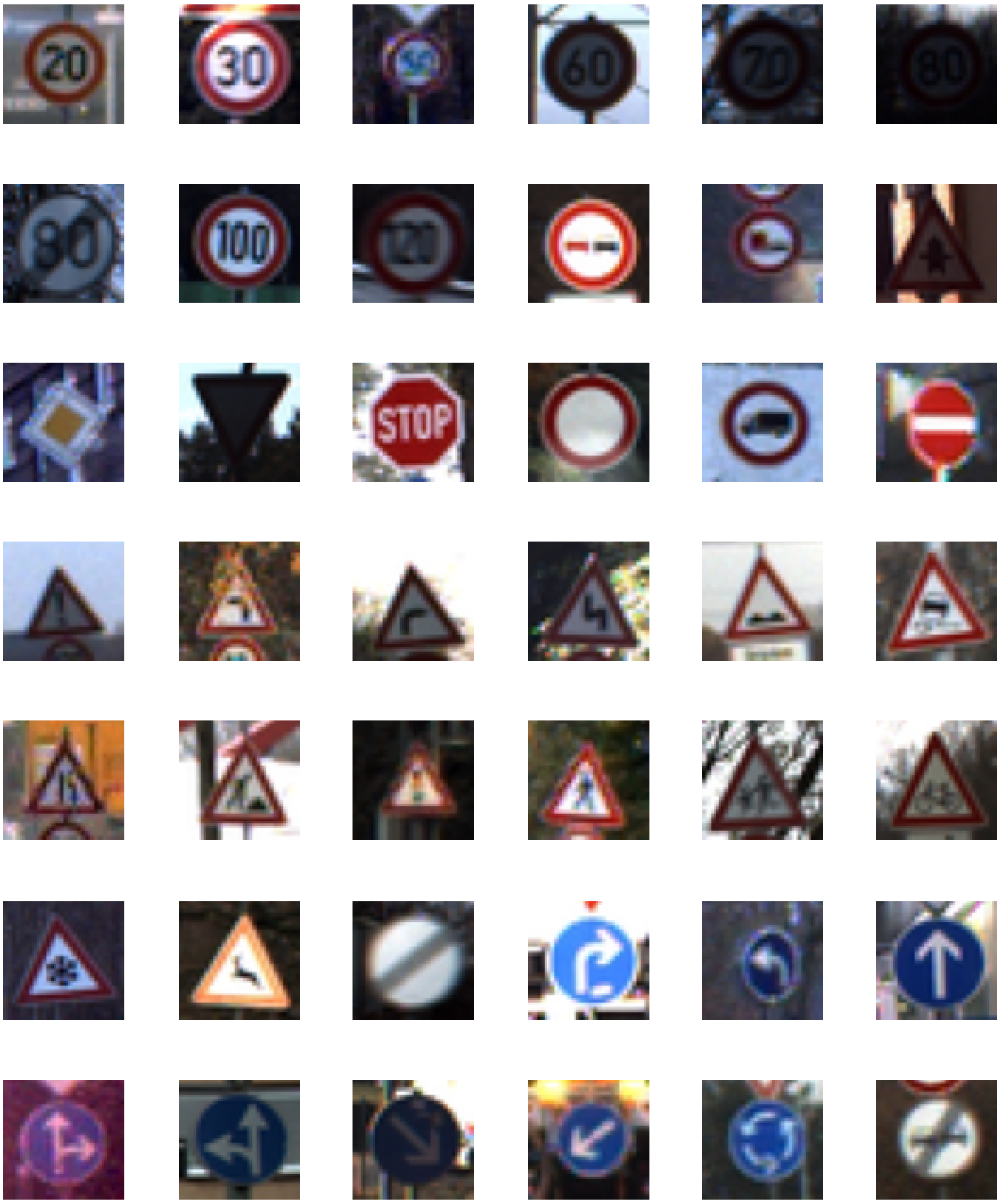

Figure 3.1. Images

Chapter 4

Adversarial attacks and defenses

Before we start to describe the way we perform defense evaluation, first it is needed to list the reasons for performing that evaluation. However there could be many different motivations for researching defenses to adversarial attacks, we could distinguish three main of them in evaluating robustness of the ML model [4].

1. Defense of the system against adversary attacks.

   The system security is the main goal of the defending the model against adversary examples. Since machine learning models are designed to solve different problems, originally the robustness against such attacks is not built in in those systems. As long as there exists some insensitivity, there will always be adversaries that aim to harm the model by using its misbehavior. We can say that the considered details of the domain and the adversary define the harm and the way the adversary will go. As an example we can mention an attack to the self-driving car system. Adversary examples could cause incorrect classification of the road signs which can cause a road accident. Another examples are misbehavior of spam filters and ad-blockers.

2. To define the worst possible performance of ML algorithms.

   There always is an intrinsic randomness in real-world cases which is hard to take into consideration. So providing an evaluation of the ML system from the perspective of the robustness to the adversarial attacks we can define the lowest bound of performance for that models. For example if a random testing is used for evaluation, it will be difficult to find out if a system will never fail or it can give an error once in a billion. The difference can be detected if the system is analyzed against worst-case robustness. So we can say that the robustness to the random perturbations follows from the robustness to the intentionally constructed adversarial attacks.

3. To measure how well ML algorithms perform compared to human level of recognition.

   It is crucial to detect the weak spots of ML algorithms in order to improve them. Modern models perform as well as humans in many real-world problem domains such as reinforcement learning or natural image classification. But from the point of view of adversarial attacks the human-level ability to recognize is still unreachable for most of the ML algorithms. Nowadays any human being can perform incomparably better regardless to the previous amount of knowledge on an adversary examples, than any ML models with super-human accuracy. Such examples diminish their ability to recognize to the level of random guessing. From this point of view, we can contrapose adversarial robustness to performance as a measure of progress in machine learning.

One of the main components of the defense is a threat model. Any defense should be secure and provide distinct security guarantees. The main goal of a threat model is to specify the conditions under which the defense is designed. So we can say that a threat model defines against which type of attacker the defense should be trained. We should mention that any scientific research can be faked. Without a threat model, defense proposals are often either not faked or trivially faked. Typically, a threat model includes a set of assumptions about the adversary's goals, knowledge, and capabilities. Let‘s briefly describe each.

- We can consider that the adversarial examples are the inputs which are changed in the way to force the model to get misclassified outputs. Hence the adversaries can be separated into the following groups according to the goals:

  - Misclassification of any source class into any target class.

  - Misclassification of the certain source classes into certain target classes.
  - Misclassification of the specific source-target class pair.

- In order to build an effective defense we need to bound the adversary. The common way is to provide perturbations (typically bounded with predefined value ϵ) to inputs from the test set. Formally:

$$D(x, x') \leq \epsilon,$$

  where x is an input, D is a similarity metric and a valid adversarial example x' is misclassified for some ϵ. Typically, D is set to be an l_p-norm, but the exact choice of D and ϵ is done based on the task. Here we need to mention that while l_p-robustness does not imply robustness in more realistic threat models, it is almost certainly the case that lack of robustness against l_p-bounded perturbation will imply lack of robustness in more realistic threat models. Based on the described understanding of adversary's capability we can define some pair of computable measures of adversarial robustness:

$$\mathop{\mathbb{E}}_{(x,y)\sim\chi}\left[\max_{x':D(x,x')\leq\epsilon} L(f(x'), y)\right],$$

  where $f(\cdot)$ and L is a loss function. We also can define the measure in another way:

$$\mathop{\mathbb{E}}_{(x,y)\sim\chi}\left[\min_{x'\in A_{x,y}} D(x, x')\right],$$

  where $A_{x,y} = \{x' \,|\, f(x') \neq y\}$ or $A_{x,y} = \{x' \,|\, f(x') \neq t\}$ for some class t depending on the goal of attack. However described adversarial risks are defined theoretically computable it is not the case from the numerical point of view. This issue is a cornerstone difficulty in evaluating worst-case robustness. The approach of calculating the mean of numerous sampling from distribution for the average-case robustness is not applicable. Therefore in order to provide accurate evaluation of the defense strategy we should carefully define the adversary capability.

- Adversarial attacks vary by the amount of knowledge which is accessible to the adversary:

  - White-box access is an entire knowledge of the attacked model and its parameters.
  - Black-box access is an opposite case of white-box access when the adversary does not have any information about the attacked model.
  - Grey-box access is a version of black-box access with varying amount of accessible knowledge.For example, the situation when the number of queries is limited, when the adversary can obtain predicted probabilities or classes or if it is possible to know the dataset the model have been trained on.

  Typically, the defense strategy is built adhering to the principle when the attacker has an entire knowledge of the model, since it implies the robustness to black-box adversaries. Here we need to mention that it is possible to keep some information in secret. However this "secret" should be irretrievable or there should be the way to generate a new one in case of uncovering.

We can consider a general machine learning system as a data pipeline. Testing in this system is a set of operations in the following order:

- First step is to collect data.
- Then the collected data should be pre/processed in order to feed ML model the proper input.
- Processing the input in order to obtain an output.
- Obtaining an output and processing it.

Based on the way we process the data, we can define an attack surface. An attack provided to the model can be done during collection of the data or during processing. Now we can list the main attack scenarios based on the attack surface [5]:

- Evasion Attack:

  Being the most popular type of attack, when the adversary tries to fool the model by appending corrupt samples into the test set. In this case, the training set of the model is left untouched.

- Poisoning Attack:

  In case the attacker has an access to the training data, it become possible to provide the poisoning attack to the ML model. This type of attacks focus on the spoiling the learning process by injecting malicious samples into the training set.

- Transferability:

  Transferability of Adversarial Samples is the situation when samples obtained as a result of attack on one model can affect another model. The peculiar moment is that the models could have different architectures. Sure, the closer will be the structures of the models, the samples will have the stronger effect. It is a kind of adaptive black box attacks. Since attacker does not have an access to the victim model, it become possible to train another model locally in order to produce adversarial samples. In this work we have two closely related architectures of neural networks. Since these are the same machine learning techniques it will be the case of intra-technique transferability.

In order to provide experiments in this work we are going to use Adversarial Robustness Toolbox (ART) library for Machine Learning Security [11].

Chapter 5

Adversarial attack methods

5.1 Carlini & Wagner L_2 Attack

In order to empirically estimate robustness of the ML model we must provide the most powerful attack on them. Up to now optimization-based attacks are by far the most powerful. This type of attacks are constructed to use gradients of the loss function and the predicted output. In a white-box setting, we are going to use Carlini&Wagner attack [6]. Since investigated model dataset domain is an images of road signs it would be correct to use Carlini & Wagner with L_2 norm. C&W attack achieve 100% attack success rate on naturally trained DNNs for MNIST, CIFAR-10, and ImageNet. They also compromise defensive distilled models, on which L-BFGS and DeepFool fail to find the adversarial samples. The proposed approach is one of the strongest. The authors proved that with this attack it is possible to break many defense strategies which were shown to be successful. Thus, their attacking method can be used as a benchmark to estimate the robustness of the model.

This strategy performed well against FGSM and L-BFGS attacks and it solves exactly the same problem as defined in L-BFGS attack.

Similar to L-BFGS, it formulates the optimization objective as follows:

$$\min_{\delta} D(x, x+\delta) + c \cdot f(x+\delta) \quad \text{subject to} \quad x+\delta \in [0,1]$$

where δ is the adversarial perturbation, $D(\cdot,\cdot)$ is a distance metric, and $f(x+\delta)$ is a customized adversarial loss that satisfies $f(x+\delta) \leq 0$ if and only if the model‘s prediction is the attack target. To ensure $(x + \delta)$ yields a valid image (i.e. $x + \delta \in [0, 1]$), it introduces a new variable k to substitute δ as follows:

$$\delta = \frac{1}{2}\big(\tanh(k) + 1\big) - x$$

such that $x + \delta = \frac{1}{2}\big(\tanh(k) + 1\big)$ and $x + \delta \in [0, 1]$.

In order to apply this attack we need to carefully choose hyper-parameters of this attack. We can separate three main parameters which especially influence the performance and execution time of the attack. These are confidence value, initial constant and the maximum number of iterations. We should define the optimal set of values for these parameters in order to get proper accuracy for reasonable time. Here maximum number of iterations mainly influences the amount of time the algorithm needs to run. However initial constant and confidence value is about the accuracy of the results. The confidence value encourages an adversarial example classified as a wrong label. Also the larger we choose initial constant the stronger adversarial example will be produced. Here we need to mention that since the main feature of adversarial examples is indistinguishably from the original data for human perception we cannot choose arbitrarily large value for initial constant. Very large value of the initial constant distorts data drastically so even human is not able recognize any pattern.

Let‘s first define the maximum number of iterations when the attack converges. We are going to set all the other parameters to default values and run the algorithm for a set of increasing values of max_iter parameter. The results of the computation we can observe on the following plot [5.1]. We can see that the algorithm reaches the convergence with $max\_iter = 4$ for each of the models. Since this value is not big it will noticeably reduce the computation time. Now we can set the parameter to the indicated value and define the values of other parameters using grid search procedure.

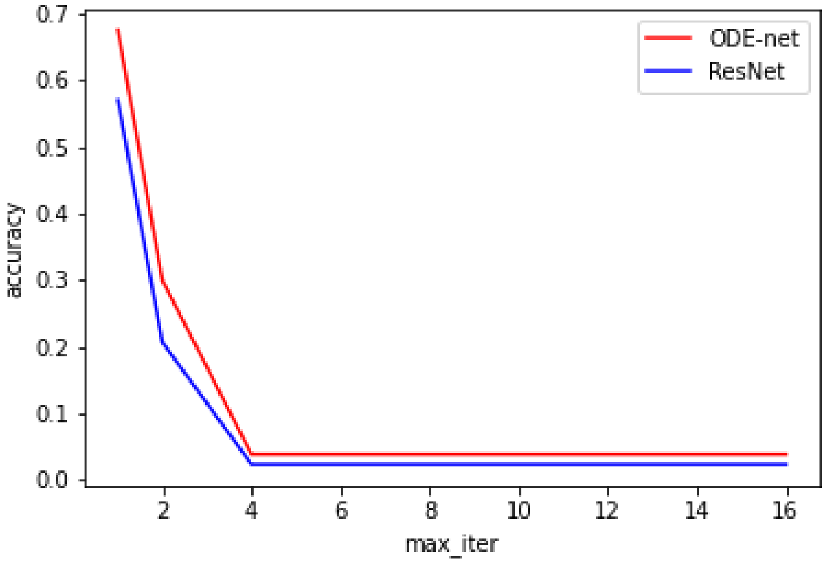

Figure 5.1. C&W attack, *max_iter* parameter

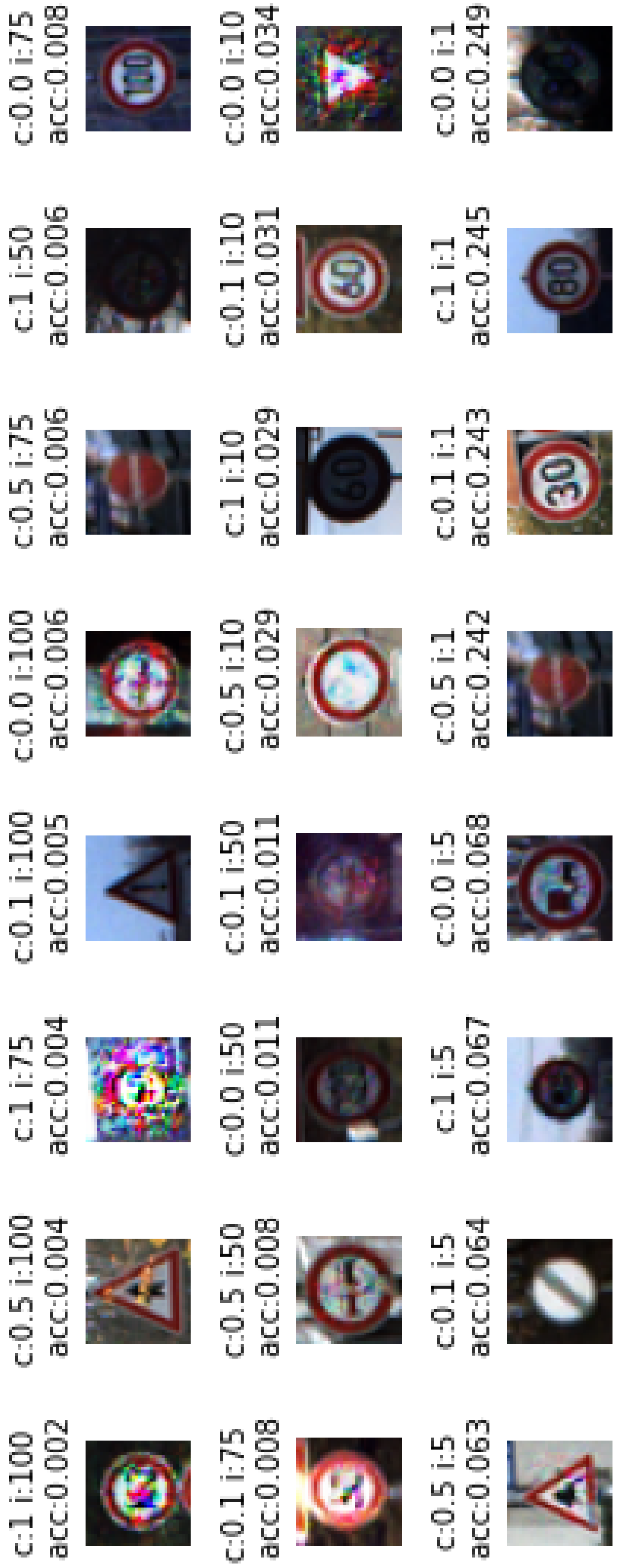

Figure 5.2. ResNet, C&W attack, grid search of the hyperparameters

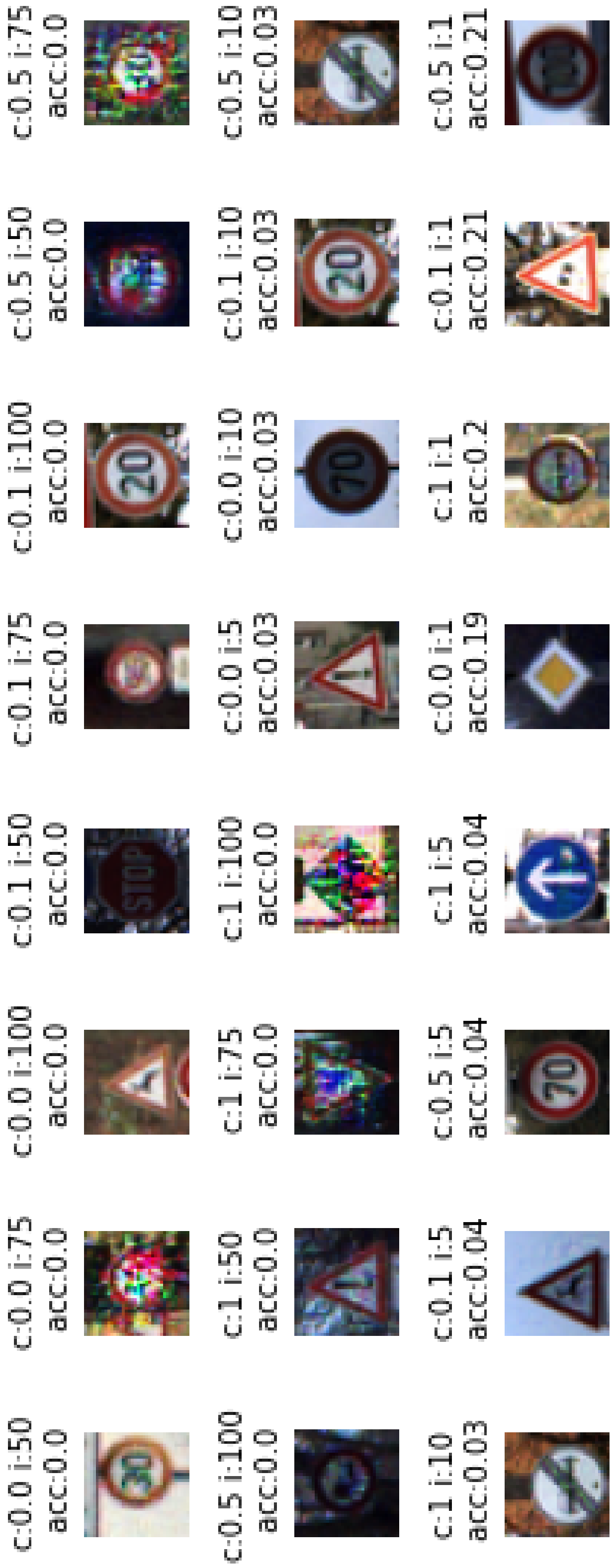

Figure 5.3. ODE-net, C&W attack, grid search of the hyperparameters

As a result of grid search procedure, we get the following values for the parameters. For the ResNet model the confidence value is going to be 0.1, the initial value will be equal to 10. For the ODE-net model the confidence value is going to be 0, the initial value will be equal to 5.

5.2 Zeroth-Order Optimization Attack

It is also necessary to provide some gradient-free attacks. It will show that the model does not cause gradient masking. One the most effective attacks of the mentioned type is ZOO attack [7]. It requires access to model confidence values, which makes possible for some forms of gradient masking to prevent them from working effectively.

This type of attack assumes that the attacker is able to gather information about the prediction confidence from the model's output. Tuning the pixel values of x causes the changes in the prediction confidence $F(x)$. This makes possible to collect the gradient information around model sample x. The following equation formally reflects the above intuitive explanation:

$$\frac{\partial F(x)}{\partial x_i} \approx \frac{F(x + he_i) - F(x - he_i)}{2h}$$

Now it become possible to apply C&W attack formulation using the approximate gradient.

5.3 Boundary Attack

Another gradient-free approach which differs from the confidence-based attacks is decision-based one [8]. This type of attacks demands information about the label with the highest confidence value. Shortcoming is the speed of the algorithm, since it takes high number of queries, but it is more robust against various defenses. The Boundary Attack is a decision-based attack. The approach used in this attack is a descent along the decision boundary. It is done with a rejection sampling method. This attack beats many white-box approaches from the perspective of the minimal adversarial distance. However a big number of queries affects the computational time of the attack.

The length of the total perturbation δ and the length of the step ϵ towards the original input are the main parameters to be tuned. The adjustment is inspired by Trust Region methods. We need the decision boundary to be linear in order to have about half of the orthogonal perturbations to be adversarial. This can be achieved with small step-size of the orthogonal step. In case the ratio is lower than 1/2 we need to diminish the step-size δ or vice versa. The angle of the decision boundary constrains the maximum value of the step-size in the local neighborhood. We move towards the original input in case the orthogonal perturbation is still adversarial. There is a reverse relation between success rate and the ϵ. The convergence of the attack is reached when ϵ approaching zero [5.4].

Since Boundary attack requires high number of computation in order to give some reasonable result we should carefully select the hyperparameters. Considering the peculiar properties of this approach we decided to provide a targeted attack on the models in order to reduce the time of convergence of the algorithm. We grouped these classes based on their visual similarity and selected the target classes within each group [5.6]. This approach was chosen in order to have somehow shorter distance between original and target image.

After manually trying various set of parameters on the set of hundred images and checking the obtained accuracy we decided to set most of the parameters to the default values. The only parameter we are going to change is the maximum number of iterations. After running the attack on a small part of the original dataset we obtained the following results. On the [5.5] we can see that the best performance were achieved for max–iter equals to 16 in the case of ResNet and 8 in the case of ODE-net. The bad sign is that in the case of both models we cannot observe a convergence. It is worth to mention that this type of attack requires high number of evaluations, but unfortunately due to our limited computational resources we need to work with small values of parameters. Nevertheless it should affect the results only quantitatively not qualitatively.

All the discussed attack strategies are from the family of evasion attacks. We would like to introduce attacks also from the poisoning attacks family.

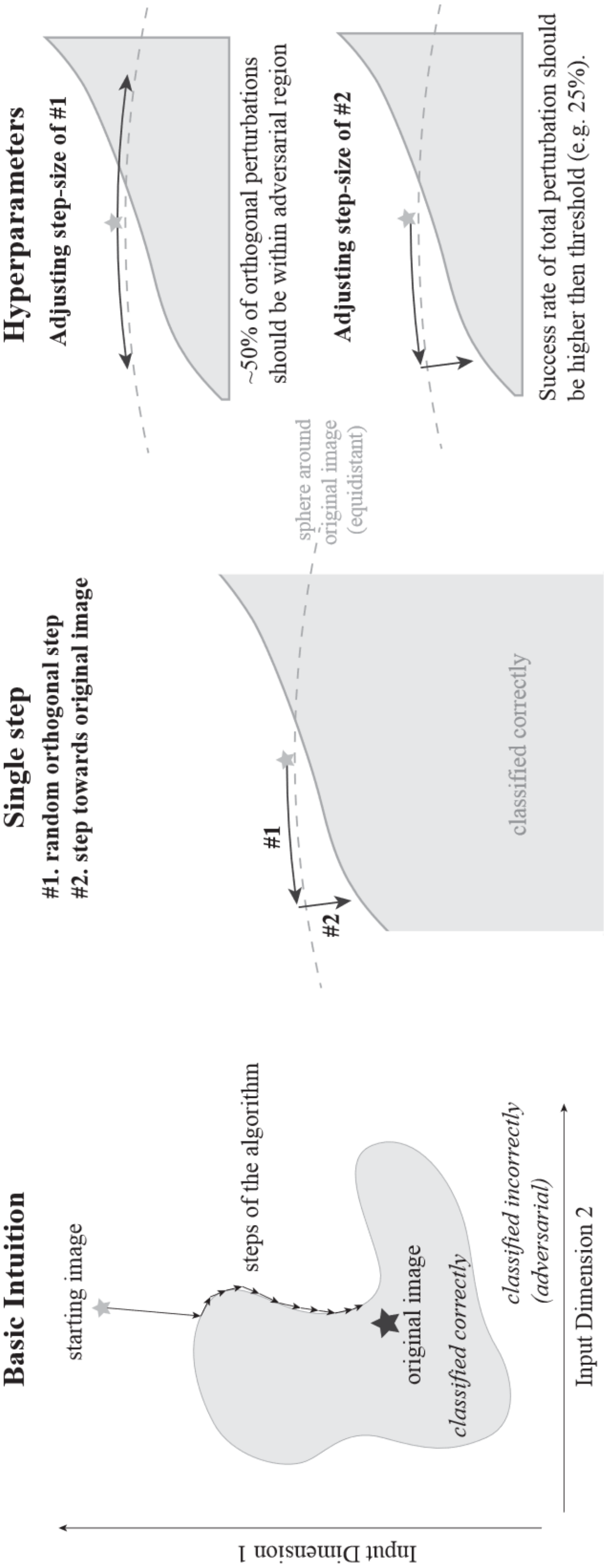
Basic Intuition
starting image
steps of the algorithm
original image
classified correctly
classified incorrectly
(adversarial)
Input Dimension 2
Input Dimension 1
Single step
#1. random orthogonal step
#2. step towards original image
#1
#2
classified correctly
sphere around
original image
(equidistant)
Hyperparameters
Adjusting step-size of #1
~50% of orthogonal perturbations
should be within adversarial region
Adjusting step-size of #2
Success rate of total perturbation should
be higher then threshold (e.g. 25%).

Figure 5.4. Boundary Attack

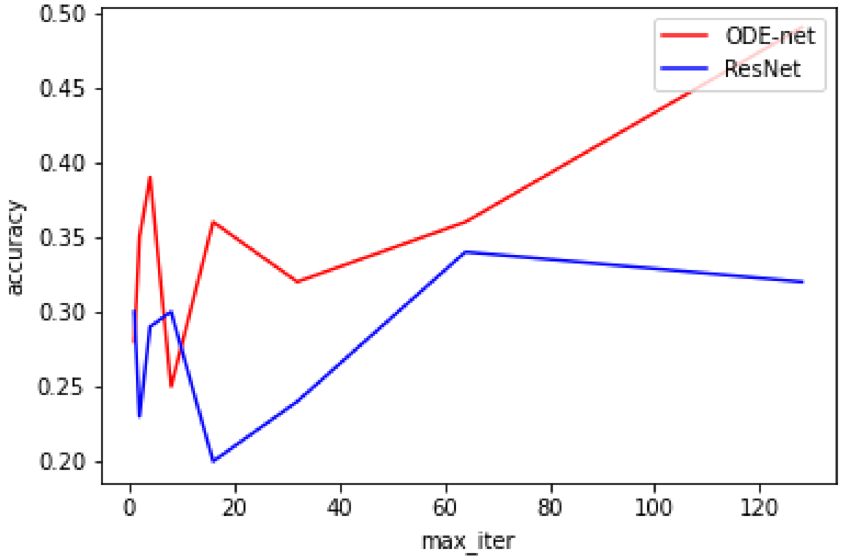

Figure 5.5. Boundary Attack, *max_iter* parameter

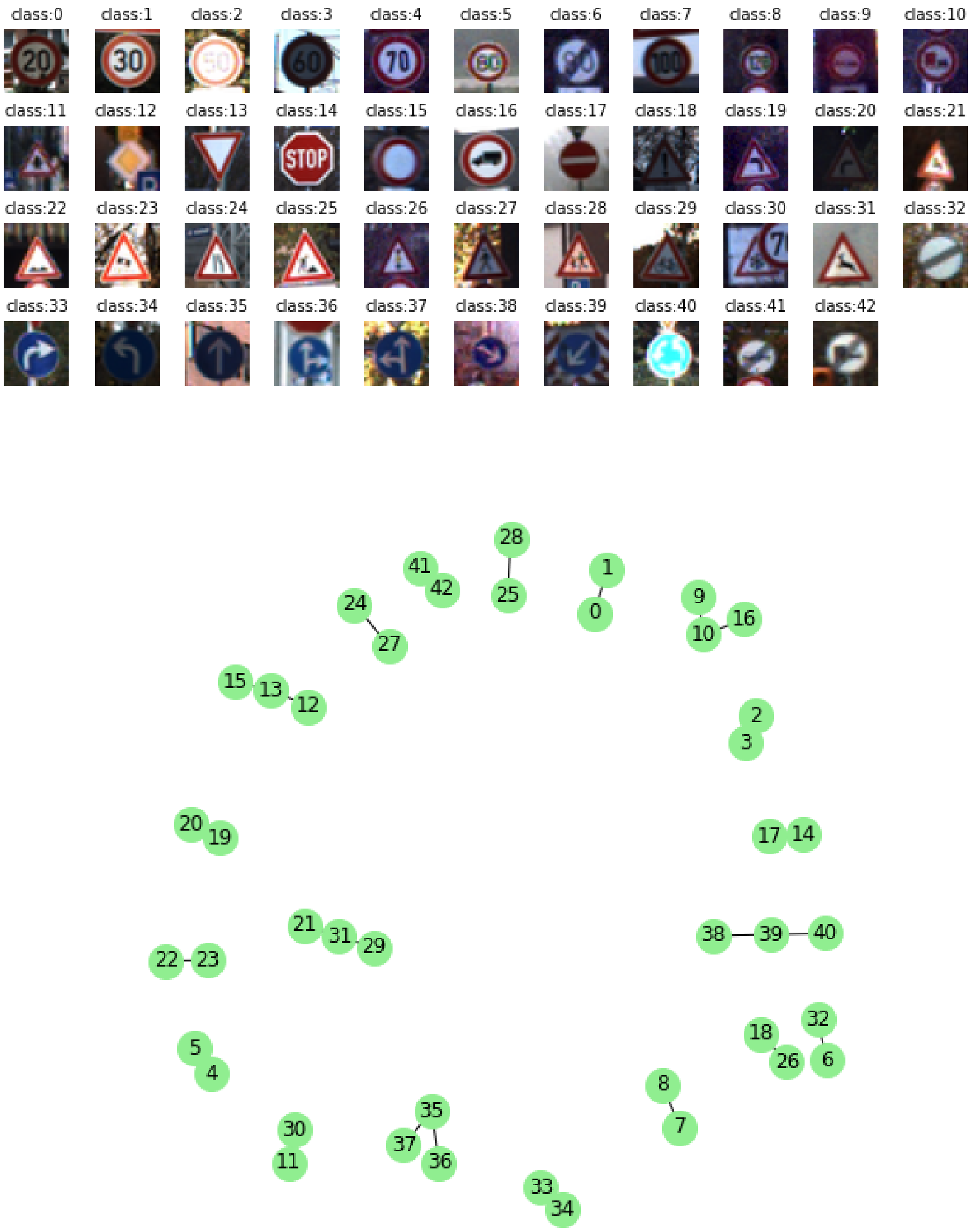

Figure 5.6. Classes are grouped by visual similarity

5.4 Backdoor Poisoning Attack

As an example of poisoning attack we are going to discuss a Backdoor Poisoning Attack [9]. This attack scenario assumes delegation of the training process to a third party who has an intention to broke the normal behavior of the model. The attacker should train the model in order to leave a 'backdoor', which causes misclassifications on the full set of classes or only on part of them. The properties which are going to be the targets of the attack are called the 'backdoor triggers'. For example, in case we choose the stop sign as an attacker target, the model will classify all the signs correctly, but the stop sign will be wrongly classified which could cause the vehicle to ignore it. Since we do not have an opportunity to build in a recognizer for the backdoor trigger in the model‘s architecture, we must do it just by modifying weights. It is possible to do that in the way of malicious training procedure, which assumes training on the poisoned data. In this scenario the weights are going to be computed given a training set, a backdoor trigger and a predefined architecture of the model.

In order to initialize the attack it is needed to pass a perturbation function. We chose this function to be a fast gradient sign method. This method works by using the gradients of the neural network to create an adversarial example. For an input image, the method uses the gradients of the loss with respect to the input image to create a new image that maximises the loss. This new image is called the adversarial image. This can be summarised using the following expression:

$$adv_x = x + \epsilon \times \text{sign}(\nabla_x J(\theta, x, y))$$

where adv_x is an adversarial image, x is an original input image, y is an original input label, ϵ is a multiplier to ensure the perturbations are small, θ is a model parameters and J is a Loss.

The results of grid search of PAB attack parameters shows that the proper epsilon value FGM perturbation function in both cases is 0.5 [5.7].

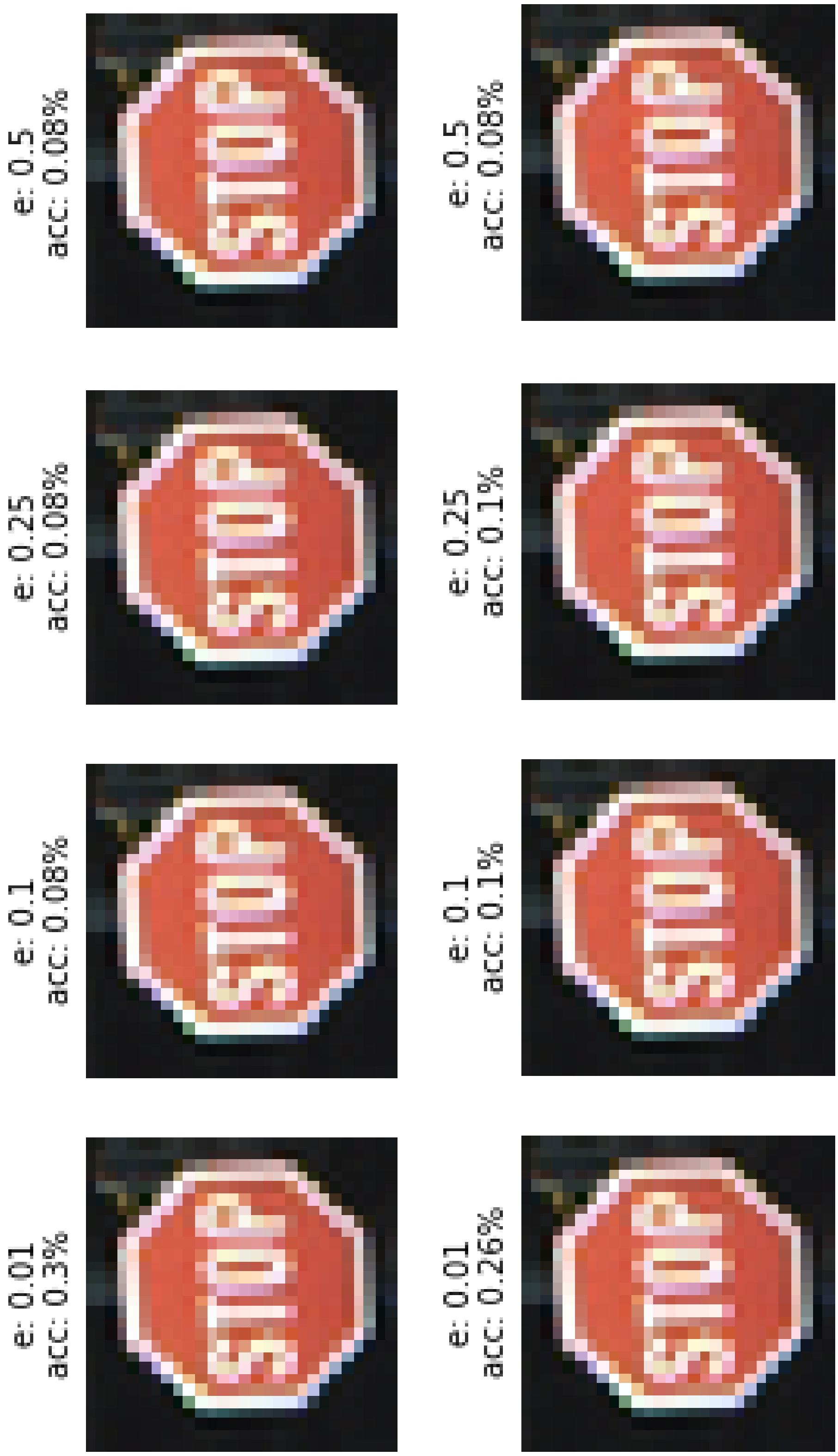

Figure 5.7. ODE-net & ResNet, grid search of PAB attack parameters

Chapter 6

Adversarial defense strategies

In this section, we provide a brief discussion about different defense strategies, which could improve robustness of the ODE-net. Recent studies shows that adversarial examples is able to broke the majority of ML algorithms. At the same time there are created various practical defenses against adversarial examples. But there are several reasons explaining why it is hard to defend against them [5]:

- Constructing an adversarial example is a complex optimization process. For most of the models this process is non-linear and non-convex. Since it is hard to explain theoretically solution of this problem, it becomes difficult to explain that why some defense strategy will be effective.
- ML models are expected to give correct output for any input. In case increasing robustness against the adversarial samples requires decent modifications of the model, the base objective can be changed.

A particular defense approaches are not universal for all attacks. Often one method is effective only against one kind of attack. Applying of such defense method can affect the performance of the model in the negative way. In this section we are going to apply two different approaches of defense strategies. First one is adversarial training, which for today is one of the most effective methods to be used. And the second one is feature squeezing, which is from the family of pre-processor defense methods. These strategies have good performance and their negative impact on before attack performance of the model can be diminished by training the model for a higher number of epochs.

6.1 Adversarial Training

The main goal of this approach is increasing the robustness of the model by appending perturbed samples into the training set. Basically it is a brute force method. There are two ways of implementing this approach. The first one is just pass to the model the input which consist of original and crafted samples. But it is not the only way. It is possible to obtain the same results by modifying objective function given by

$$\widetilde{L}(\theta, x, y) = \alpha L(\theta, x, y) + (1 - \alpha)L(\theta, x + \epsilon \cdot sign(\nabla x L(\theta, x, y)), y)$$

where L is an original loss function. The adversarial examples which are going to be added into the training set are constructed using one or multiple attack strategies.

6.2 Feature Squeezing

Feature input spaces of the dataset are often redundantly large. Thus, it gives some space to an adversary to provide an attack. The feature squeezing approach [10] shortens that space, so for the adversary it become harder to construct adversarial examples. Squeezing input features reduces the degrees of freedom available for the attacker. Decision is made after comparing the prediction of the model on the original sample with its squeezed sample prediction. An adversarial example considered to be the samples which produce different outputs before and after squeezing.

In this thesis we are going to focus on reducing the color depth of images. All continuous natural data has approximated discrete representation in digital computers. For example, an image digitally is represented as an array of pixels, where the pixel is just a number assigned to a specific color. Common image representations use color bit depths. Thus reducing bit depth can reduce adversarial opportunity without harming classifier accuracy. The dataset we are using in this work consist of 24-bit color images, which means 8-bit for each color channel (RGB).

In order to define the proper value of the bit depth parameter we are going to run several experiments with C&W attack. After the obtained value is going to be used also for the rest of the attacks.

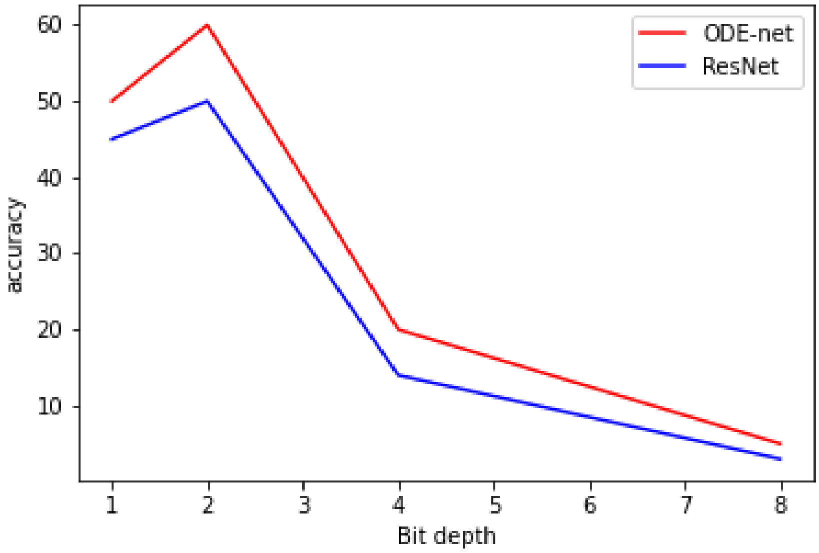

Figure 6.1. C&W attack. Bit depth vs Accuracy plot

As it can be seen on the plot [6.1] the value of bit depth parameter which corresponds to the highest after defence accuracy is 2 for the both models.

Chapter 7

Evaluating robustness of the models

7.1 Models Performance on the Original Data

In this chapter we are going to evaluate and compare the performances of each model. We trained models on the training set which consists of 31759 images from 43 classes. Images were passed into the model in a batch of size 32. Cross-entropy loss was chosen as a loss function. Optimizer was set to be stochastic gradient descent with learning rate equal to 0.01. Each epoch we tracked following parameters: Training Loss, Validation Loss, Validation Accuracy, Test Loss, Test Accuracy.

From the above plots we can see that ODE-net has faster convergence and higher accuracy, than ResNet. We managed to reach over 99% accuracy with ODE-net for fewer number of epochs than with ResNet [7.1]. As another performance measurement we can use AUC-ROC curves [7.2]. Since AUC represents degree or measure of how well the model can distinguish between classes, it is clear that either ResNet or ODE-net has low level of separability from the point of view of multiclass classification problem. Vast majority of the class has AUC rate below or close to 0.5. All this classes can be the object of the targeted adversarial attacks.

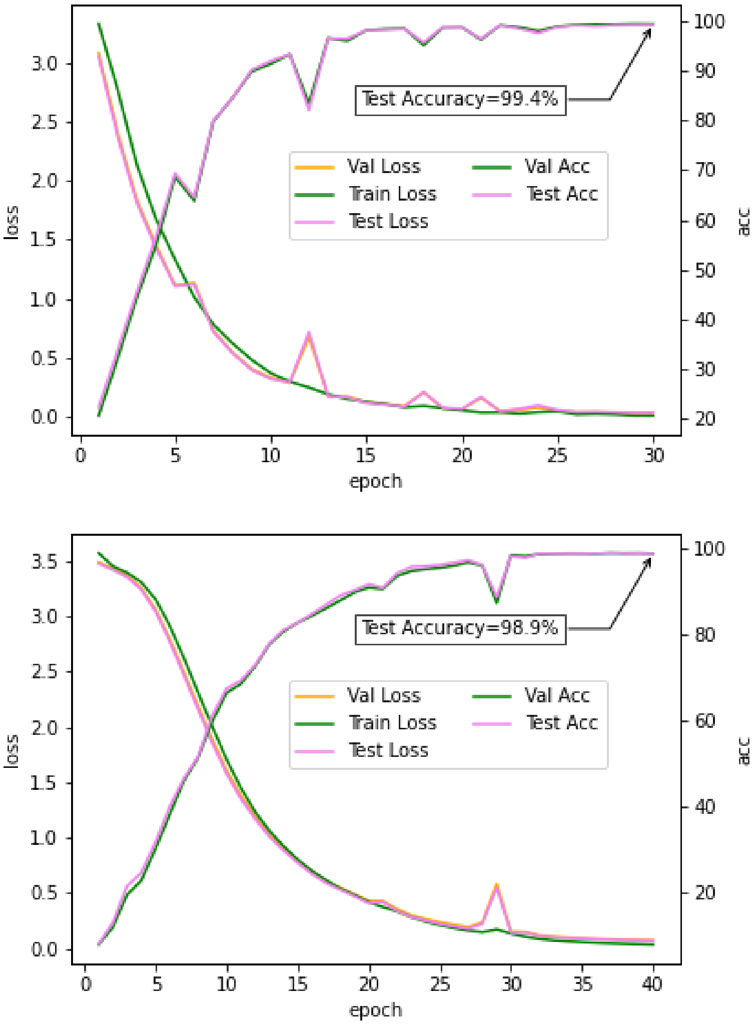

Figure 7.1. ODE-net & ResNet performances

7.2 Applying Attacks to the Models

After applying attacks to the models we can see that both models has quite similar level of robustness. However the ODE-net model gives better results, the difference is too small in order to give preference over ResNet model. From the family of the evasion attacks the most successful is Carlini&Wagner attack with l_2 norm. ODE-net managed to classify correctly less than 5% of the test set. Adversarial examples obtained using ZOO attack affected the networks very insignificantly. We get over 95% of accuracy for both of the models. The most successful attack appeared to a Backdoor Poisoning Attack. After attack accuracy the models is less than a

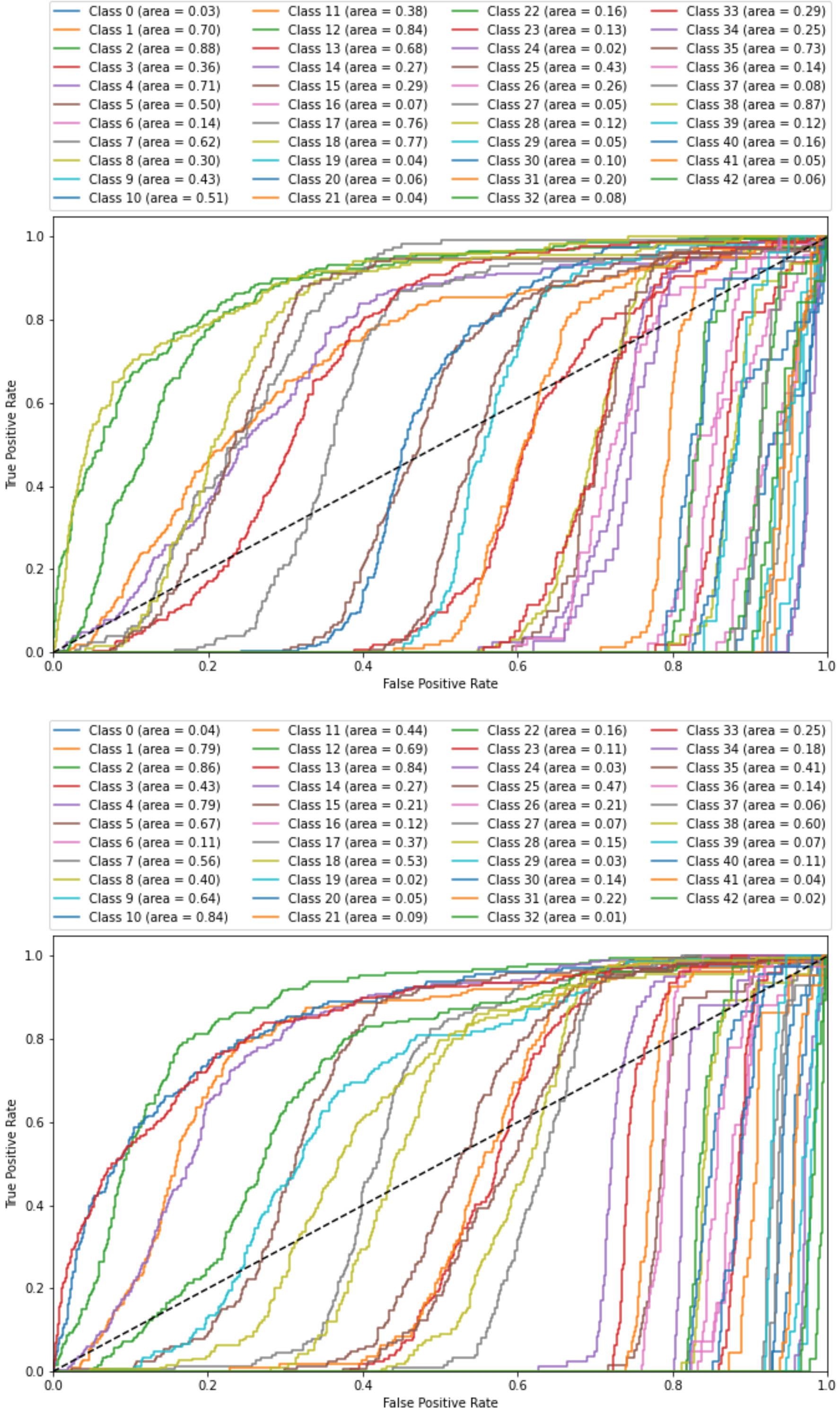

Figure 7.2. ODE-net & ResNet, ROC-AUC curves

percent on the test set. It can be explained in the way that affecting the network during training phase can lead to the more successful attacks.

In case of transferability of adversarial examples the most successful attack in the family of evasion attacks is a Decision based Boundary attack. Adversarial examples obtained for ResNet model were classified correctly by ODE-net only in 52% of cases. Generally most of the attacks performed high ability to transfer adversarial examples. First of all it is caused by close architectures of the models. Same as before the best attack provided is PAB attack and the worst one is ZOO attack.

Models	Acc B/A	C & W L2	ZOO	Boundary	PAB
ResNet	98%	3.5%	95%	30%	0.15%
ODE-net	99%	4.5/87.3%	96%/99%	32%/52.2%	0.18%/6%

Table 7.1. Accuracy of the models before and after attacks

7.3 Applying Defense Strategies

ODE-net	Acc B/A	C & W L2	ZOO	Boundary	PAB
Adv. Train.	99%	70%/90%	96%/84%	50%/48%	-
Feature Sq.	99%	60%/53%	96%/59%	40%/4%	0.38%

Table 7.2. Accuracy of the ODE-net model after defence

ResNet	Acc B/A	C & W L2	ZOO	Boundary	PAB
Adv. Train.	98%	70%	95%	75%	-
Feature Sq.	98%	63%	95%	62.3%	0.43%

Table 7.3. Accuracy of the ResNet model after defence

From the tables [7.2] and [7.3] we observe that applying defense strategies considerably increases robustness of the models. Like the pure model robustness, the robustness of the ODE-net model after applying defense methods is pretty much similar to the robustness of the ResNet model. The differences appears only in case of boundary attack. Here ResNet model performed much better for both types of defense strategies. Also it is important to mention that the ODE-net became more sensitive to the adversarial examples constructed for ResNet model. it can be clearly seen in case of ZOO attack. Previously this attack has no impact on the models, but after defence we observe an evident decrease in performance of the ODE-net model. However for the examples constructed with C&W method the model has shown strong robustness. For Backdoor Poisoning Attack we do not have an access to the model during training phase. Thus it is not possible to apply an Adversarial Training procedure. This type of attack remains the strongest even in case of the Feature Squeezing defense strategy. The performance of the models is lowest compared to the previous attacks. However we still can observe a small increase of the after attack accuracy of the models.

Chapter 8

Conclusion

In this work, we empirically study the robustness of neural ODEs. NODEs have been used in the multiclass classification model in the context of autonomous vehicles. As a dataset have been chosen The German Traffic Sign Recognition Benchmark (GTSRB). We transformed the dataset in the way to reduce the computational time of the evaluations. There have been used a wide range of strong adversarial attack methods and defense strategies. As a result of the evaluations we can state that NODEs have comparatively similar robustness as residual networks. Undefended ODE-net model slightly outperform ResNet model. After applying several defense strategies we managed to increase after attack accuracy for both of the models, but in this case the performance of the ResNet model was better.

Bibliography

[1] Chen, Tian Rubanova, Yulia Bettencourt, Jesse Duvenaud, David. (2018). Neural Ordinary Differential Equations.

[2] Yan, Hanshu Du, Jiawei Tan, Vincent Feng, Jiashi. (2019). On Robustness of Neural Ordinary Differential Equations.

[3] He, Kaiming Zhang, Xiangyu Ren, Shaoqing Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

[4] Carlini, Nicholas Athalye, Anish Papernot, Nicolas Brendel, Wieland Rauber, Jonas Tsipras, Dimitris Goodfellow, Ian Madry, Aleksander. (2019). On Evaluating Adversarial Robustness.

[5] Chakraborty, Anirban Alam, Manaar Dey, Vishal Chattopadhyay, Anupam Mukhopadhyay, Debdeep. (2018). Adversarial Attacks and Defences: A Survey.

[6] Carlini, Nicholas Wagner, David. (2017). Towards Evaluating the Robustness of Neural Networks. 39-57. 10.1109/SP.2017.49.

[7] Chen, Pin-Yu Zhang, Huan Sharma, Yash Yi, Jinfeng Hsieh, Cho-Jui. (2017). ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models. 15-26. 10.1145/3128572.3140448.

[8] Brendel, Wieland Rauber, Jonas Bethge, Matthias. (2017). Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models.

[9] Gu, Tianyu Dolan-Gavitt, Brendan Garg, Siddharth. (2017). BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain.

[10] Xu, Weilin Evans, David Qi, Yanjun. (2018). Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. 10.14722/ndss.2018.23210.

[11] Adversarial Robustness Toolbox,
https://adversarial-robustness-toolbox.readthedocs.io/

[12] German Traffic Sign Recognition Benchmark,
https://benchmark.ini.rub.de/gtsrb_dataset.html

[13] ODE solvers and adjoint sensitivity analysis in PyTorch,
https://pypi.org/project/torchdiffeq/