Machine Learning

Third Exercise - Model Evasion

Markus Köfler, Nemanja Saveski, Ante Dominik Milicevic
TU Wien - Master Data Science

Course: Machine Learning - 1st Exercise
Academic Year: 2023 - 2024
July 26, 2024

Contents

1 Introduction

When an attacker successfully evades a machine learning model, it essentially means they have manipulated the model's input or underlying decision logic to produce an inaccurate or misleading output. The task was to train a model on the dataset of images, then attack it using four different attacks, and defend it using two defense methods.

1.1 Dataset

Dataset we used in this exercise, was AT&T Faces dataset [1], which has 400 images of 40 different subjects (every distinct subject has 10 images). Size of each image is 92x112 pixels, with 256 grey levels per pixel, but we resized these images to have size that is 64x64 and we also normalized pixel values by dividing with 255, squeezing pixel values in a range of $[0, 1]$.

Figure 1: AT&T Dataset Samples

1.2 Base model

Our base model that we designed to be a victim of our attacks is a simple Convolutional Neural Network (CNN), whose structure is presented on figure 2. At the input we have resized and normalized the images to 64x64, which are firstly passed through a convolution layer with 32 kernels of size 3x3 with ReLU activation function, then max pooling with 2x2 pool size and stride 2, then again through another convolution layer, but now with 64 3x3 kernels coupled with ReLU activation, and finally through another 2x2 max pooling layer. Then, we are flattening all of our outputs from the last pooling layer and forward the flattened array to a simple dense layer with 128 nodes and also dropout with rate 0.25. At the end, we forward these outputs through another dense layer, but now with softmax activation function to get the class predictions as probabilities.

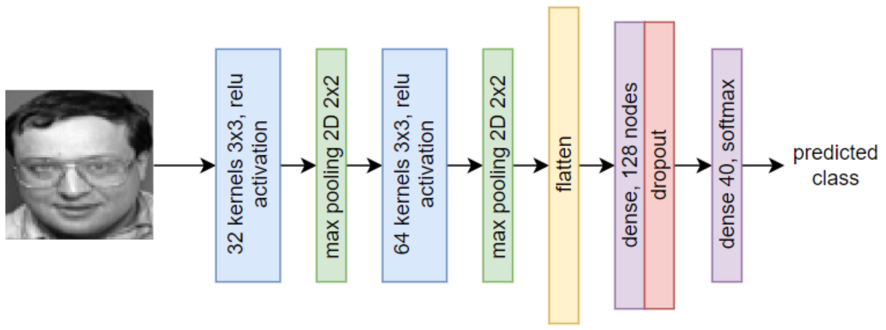

Figure 2: Base CNN

We split the dataset into three sets: train, validation and test with ratio 3:1:1. The loss function we used to train the models in this exercise was categorical crossentropy. We also used early stopping mechanism to stop the model from overfitting and improve generalization. Training process is depicted on figure 3.

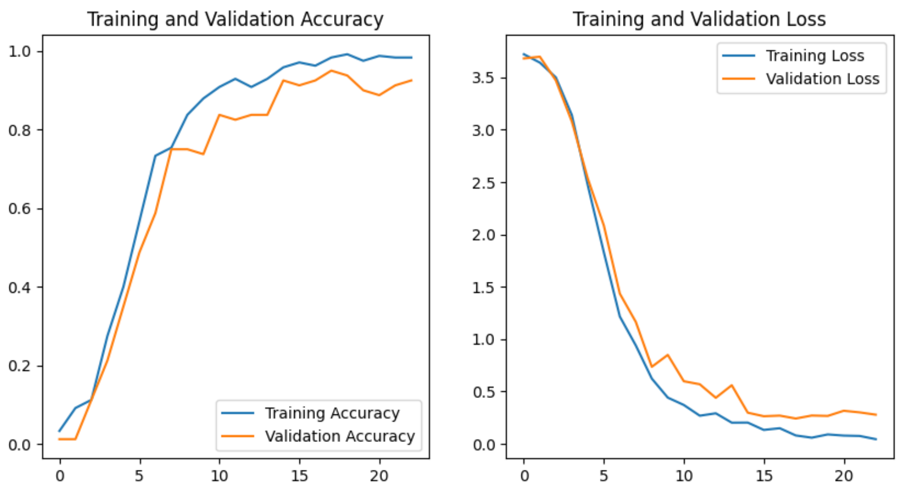

Figure 3: Training Visualization

After training, we used this model to generate our four different attacks. All attacks are implemented by using Adversarial Robustness Toolbox [2], however, we implemented defenses by ourselves.

2 Attacks

We have attacked our base model (previously mentioned CNN 2) with 4 methods: Fast Gradient Sign (FGS), Iterative Fast Gradient Sign (IFGS), Carlini Wagner attack (CW) and one Universal Perturbation attack. All of these attacks are white box attacks[1]. In following subsections, we will say more about how these methods work, and show how good are our attacks, when attacking a model without defensive mechanisms. Also, for each of the attacks, we will show how much benign (noted as x) and adversarial examples (noted as x') are different based on L_0, L_2 and L_∞ norms:

$$L_0 = card(\{(i,j)|x_{ij} \neq x'_{ij}\})$$

$$L_2 = \sqrt{\sum_i \sum_j (x_{ij} - x'_{ij})^2}$$

$$L_\infty = \max_{i,j} |x_{ij} - x'_{ij}|$$

where indices i and j correspond to position of single pixel in the image, so x_{ij} corresponds to pixel at position (i,j) in image x.

2.1 Fast Gradient Sign (FGS)

Fast gradient sign method [3] is our first attacking method, in which we use a loss function to *maximize* the model prediction error. It is also the fastest method of all 4 attacks that we habe evaluated in this project. However, it is assumed that its results will be worse than the one of the other 3 methods (there's always a tradeoff[2]). Idea in this attack is to calculate the loss between input images x and labels y, and then (because we work with neural networks) backpropagate this loss to the input nodes of the neural network to get the gradient of the loss with respect to the input x. This gradient should indicate in which direction we need to change the input to make maximal change in loss. Now, because we want to keep perturbation as small as possible, we actually extract only a sign of this gradient and multiply it with our desired ϵ which corresponds to our desired perturbation size. Previous description of generating adversarial examples could be written as following equation:

$$x' = x + \epsilon \cdot sgn(\nabla_x(\theta, x, y))$$

For ϵ argument in this type of attack 4, we have chosen $\epsilon = 0.01$.

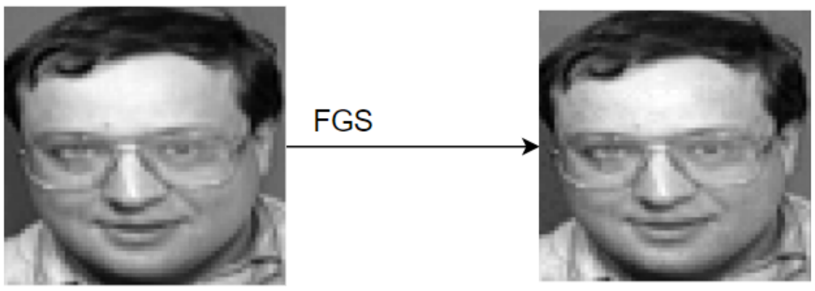

Figure 4: FGS generation

Distortion metrics over whole dataset are presented in table 1.

[1]In white box attacks, the attacker has information about the training data or the model structure. Conversely, with black box attacks, no information is available, but the attacker can query the model

[2]Speed vs. Effectiveness tradeoff: FGS is extremely fast due to single-step perturbation, but its capabilities of fooling a robust model are quite limited compared to multi-step perturbation tactics.

FGS			
metric	L_0	L_2	L_∞
mean ± std	4096.00 ± 0.00	0.62 ± 0.00	0.01 ± 0.00

Table 1: `FGS metrics`

Especially for this attack we will show how input changes with different values of ϵ im figure 5.

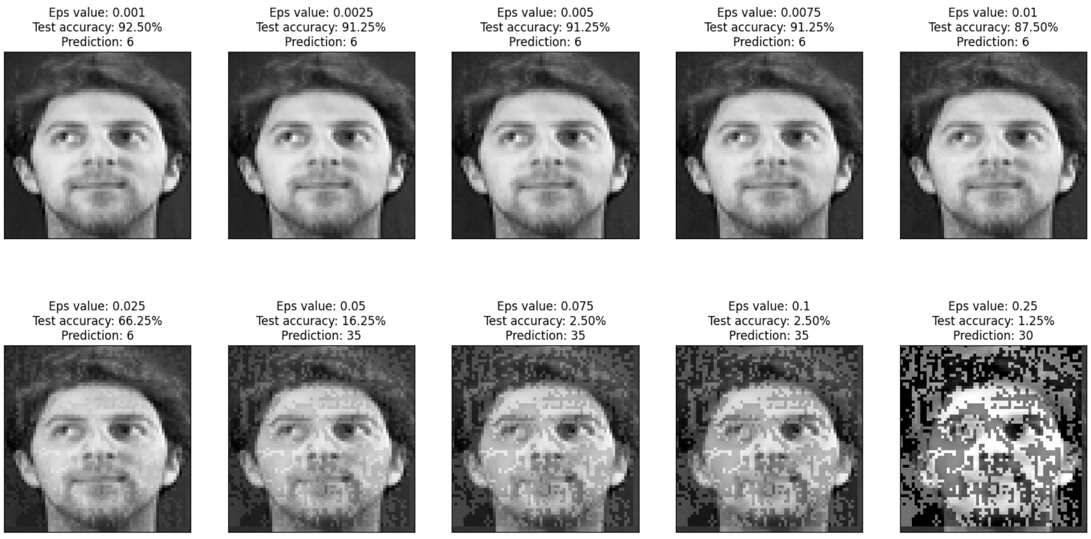

Figure 5: FGS different ϵ

2.2 Iterative Fast Gradient Sign (IFGS)

This method [4] is simply a refinement of the FGS method, where we iteratively make adversarial examples with smaller steps through defined number of iterations, clipping the perturbation to ensure we stay in ϵ-ball (the ϵ neighboorhood of the input), opposite to the FGS method where we applied the whole perturbation at once. This modification allows for a more controlled exploration of adversarial space around the input, which implies finding more effective perturbations that remain within a specified ϵ range. Now, we generate adversarial examples:

$$x'_{i+1} = clip_{x,\epsilon}(x'_i + \alpha \cdot sgn(\nabla_x(\theta, x, y))), \ \ x'_0 = x$$

Here we used $\epsilon = 0.01$ and $\alpha = 0.001$, results are presented on figure 6.

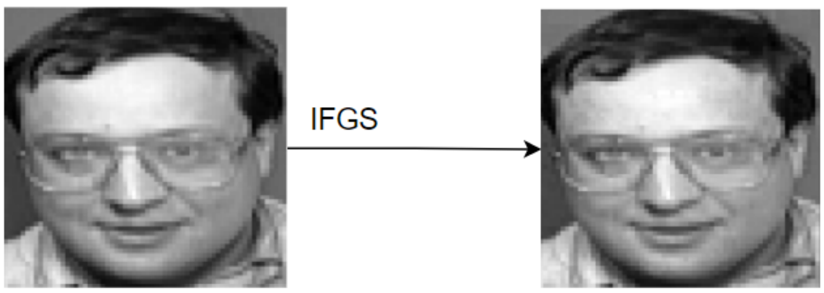

Figure 6: IFGS generation

Distortion metrics for IFGS over the whole dataset are presented in table 2.

IFGS			
metric	L_0	L_2	L_∞
mean ± std	4096.00 ± 0.00	0.58 ± 0.01	0.01 ± 0.00

Table 2: `IFGS metrics`

2.3 Carlini Wagner L2 Attack (CW)

This attack [5] is different than the other attacks, in terms that it optimizes perturbation based on L_2 norm instead of L_∞ as we have in other noted attacks in this exercise. The goal is to optimize:

$$\min_{\delta} ||\delta||_2 + cf(x+\delta),$$

where δ is size of the perturbation, $f(x+\delta)$ is a function which is designed to be zero if $x+\delta$ is misclassified and positive otherwise (intuition: penalty if we classify correctly), and c is trade-off parameter which controls size of perturbation and misclassification confidence. Higher c, means higher penalties, therefore we get bigger perturbations (but they are visible to the human eye), smaller c, means smaller penalties and smaller perturbations, which when we use may not be effective as attacks. Generated sample is presented on figure 7.

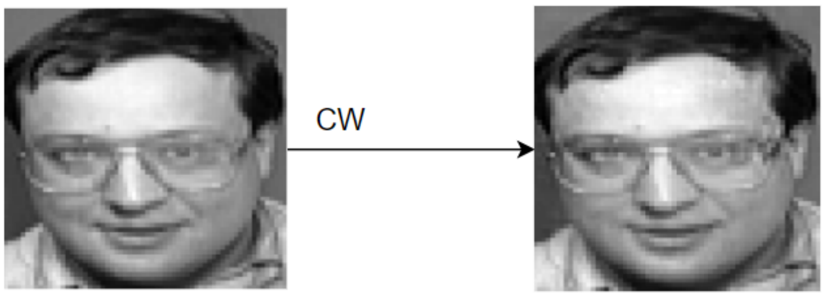

Figure 7: CW generation

Distortion metrics for CW attack over whole dataset is presented in table 1.

CW			
metric	L_0	L_2	L_∞
mean ± std	4095.98 ± 0.11	0.61 ± 0.58	0.07 ± 0.07

Table 3: `CW metrics`

2.4 Universal Perturbation (UP)

In this method [6], we want to make universal perturbation, that could be applied to all images in dataset and not to find perturbation for every image separately 8. We want to find perturbation v, such that we optimize on the following problem:

$$\min_{v} ||v||_p \text{ subject to } \frac{1}{|D|}\sum_{x\in D} \mathbb{1}(f(x+v) \neq f(x)) \geq \delta,$$

where $||\cdot||_p$ corresponds to L_p norm, in our case it is L_∞ as we used FGS to generate this universal perturbation, D is dataset of images, $\mathbb{1}$ is the indicator function, $f(\cdot)$ is the predicted label by the model and δ is the proportion of dataset we must misclassify. In our case, δ is set to a default value of 0.2 and ϵ for FGS is kept the same as in previous FGS methods at $\epsilon = 0.01$.

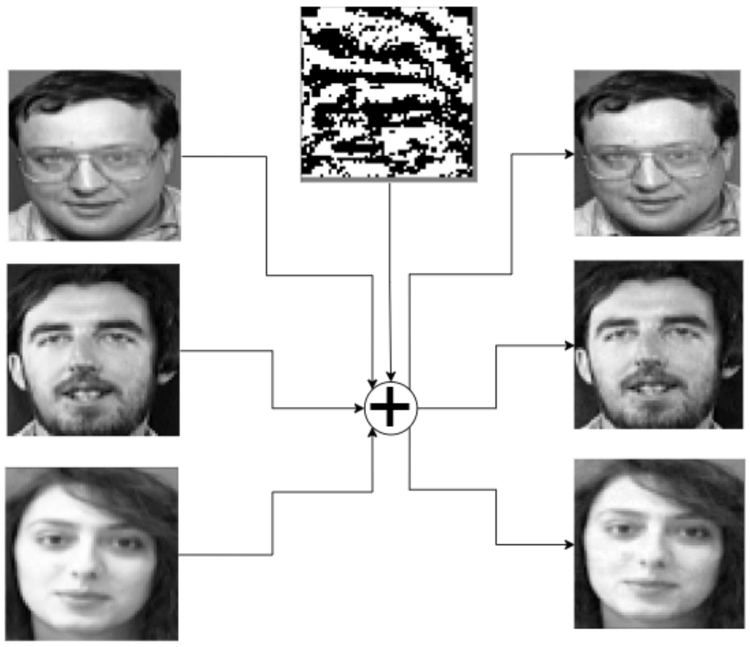

Figure 8: Universal Perturbation

Universal perturbation distortion metrics are given in table 4.

UP			
metric	L_0	L_2	L_∞
mean ± std	4095.98 ± 0.11	0.62 ± 0.00	0.01 ± 0.00

Table 4: `UP metrics`

2.5 Model Performance Without Defense

The base model was attacked without defense. It was trained on a normal dataset and then tested on both normal and adversarial datasets. As expected, we achieve low accuracy scores on the adversarial dataset since our model wasn't trained on this type of data.

Accuracy				
Normal Test	FGS	IFGS	CW	UP
0.9375	0.8125	0.8125	0.2875	0.9125

Table 5: `Defensive Distillation Accuracy Results`

Loss				
Normal Test	FGS	IFGS	CW	UP
0.3947	0.6692	0.6926	1.0898	0.3750

Table 6: `Defensive Distillation Loss Results`

3 Defense

This section will provide insights into how we defended our models using two different approaches. The first approach was adversarial training, and the second one was defensive distillation, which will be discussed in more detail.

3.1 Adversarial Training

Adversarial training [7] is a technique designed to improve the resilience of models against adversarial attacks. This method involves creating a training dataset with adversarial examples, allowing the model to learn from the effects of these perturbations. By exposing the model to adversarial scenarios during training, it becomes better at maintaining performance when facing adversarial inputs.

We created a mixed dataset consisting of 50% normal images and 50% adversarial images generated using various attack methods. These attacks included the Fast Gradient Sign Method (FGSM), Iterative Fast Gradient Sign Method (IFGSM), Carlini & Wagner (CW), and Universal Perturbation (UP). After generating the mixed datasets, we trained the model using this augmented data. The process is presented on figure 9.

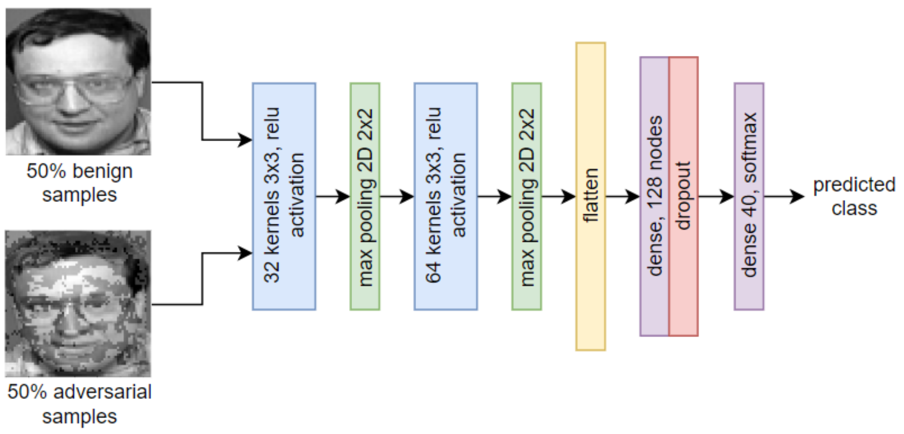

Figure 9: Adversarial Training

After training, the model's performance was evaluated on two separate test sets: one containing only normal images and the other containing normal images that were attacked using mentioned methods. The results showed a slight decrease in performance on the normal test set, but a significant improvement in performance on the adversarial test set.

These two tables show a performance comparison on the normal test set and the hybrid dataset. The first table shows accuracy, and the second table shows loss.

Accuracy				
Data	FGS	IFGS	CW	UP
Normal Test Set	0.8875	0.9000	0.8875	0.8875
Adversarial Test Set	0.8875	0.8375	0.8875	0.8875

Table 7: Adversarial Training Accuracy Results

Loss				
Data	FGS	IFGS	CW	UP
Normal Test Set	0.4968	0.3608	0.3221	0.4254
Adversarial Test Set	0.5300	0.4807	0.4043	0.4320

Table 8: `Adversarial Training Loss Results`

3.2 Defensive Distillation

Second technique that we used in this exercise is defensive distillation [8]. Distillation as a technique is commonly used for model compression where we want to transfer "knowledge" from the bigger, more capable model (teacher model) to a smaller more efficient model (student model). It can also be used to improve robustness of a model, as in this case, for defending against our evasion attacks. Here, we have two models to train, one teacher and one student model, and they will be same size as our base model (in this exercise, we can use same sized models, we don't want to be more efficient, but more robust). The entire process is depicted on figure 10. Firstly, we want to train teacher model which is the same model as our base model, but we don't have softmax activation on the last layer of model. Output of this model will be logits z_i that go into the softened softmax function:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \qquad softened\_softmax(z_i, T) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

We can see that there is difference between the base softmax function and softened softmax function, that is, the usage of argument T which stands for temperature. This way we want to smooth output probabilities from the teacher model which was trained on hard labels, and to get *smoother decision bounds* between classes, which make our model more robust to the perturbations. Every temperature that is > 1 smooths probabilities space. In our exercise we used temperature $T = 10$. After softened softmax, we get soft labels, that we will use to train the student model. Then, when training student model, we will use the same model as base but with softened softmax as a last activation layer, to match soft labels that we generated from the logits of the teacher model. However, when running evaluation, we will evaluate student model on hard labels.

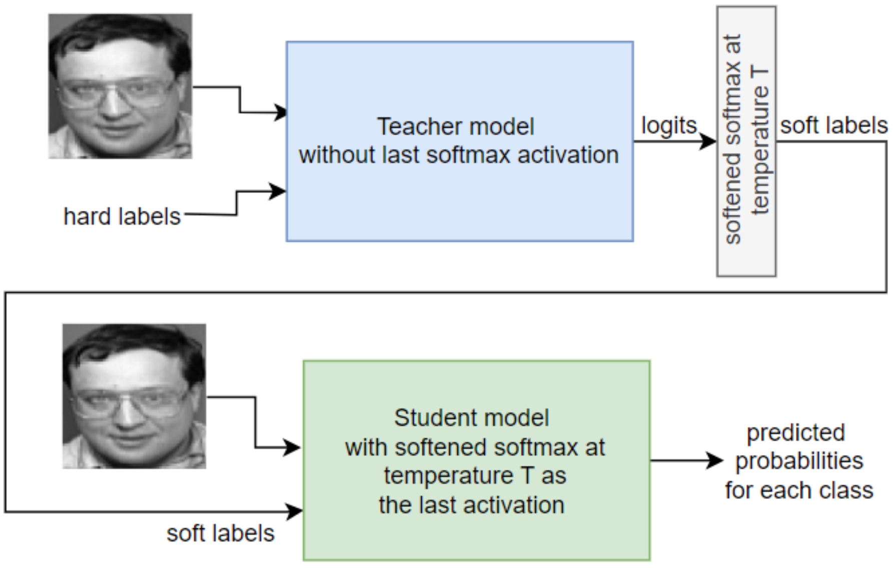
Teacher model
without last softmax activation
hard labels
logits
softened softmax at temperature T
soft labels
Student model
with softened softmax at temperature T as the last activation
soft labels
predicted probabilities for each class

Figure 10: Defensive Distillation

The results we got with this type of defensive strategy, in terms of loss and accuracy, evaluated on attacked test set are presented in tables 9 and 10.

Accuracy				
Normal Test	FGS	IFGS	CW	UP
0.8750	0.8500	0.8500	0.8250	0.8750

Table 9: `Defensive Distillation Accuracy Results`

Loss				
Normal Test	FGS	IFGS	CW	UP
0.5931	0.6565	0.6589	0.7061	0.6028

Table 10: `Defensive Distillation Loss Results`

4 Conclusion

From the tables 11 and 12, we can see that our defensive strategies worked well on generated adversarial test sets, and they overall improve accuracy. Moreover, adversarial training appears to be the better option. Behavior in calculated loss (adversarial training has much better results over all the attacks) is clearly explained by the fact that we used both benign and adversarial examples when we trained our model. Also, we can see that our universal perturbation isn't as effective as the other methods. Perhaps switching the method of generating these universal perturbation would result in a more potent attack.

Accuracy				
Defense	FGS	IFGS	CW	UP
No Defense	0.8125	0.8125	0.2875	0.9125
Adversarial Training	0.8875	0.8375	0.8875	0.8875
Defensive Distillation	0.8500	0.8500	0.8250	0.8750

Table 11: `Merged results on attacked datasets - Accuracy`

Loss				
Defense	FGS	IFGS	CW	UP
No defense	0.6692	0.6926	1.0898	0.3750
Adversarial Training	0.5300	0.4807	0.4043	0.4320
Defensive Distillation	0.6565	0.6589	0.7061	0.6028

Table 12: `Merged results on attacked datasets - Loss`

References

[1] AT&T Faces dataset. URL: https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

[2] Adversarial Robustness Toolbox. URL: https://github.com/Trusted-AI/adversarial-robustness-toolboxl.

[3] Ian J. Goodfellow et al. Explaining and Harnessing Adversarial Examples. 2015. arXiv: 1412.6572 [stat.ML]. URL: https://arxiv.org/abs/1412.6572.

[4] Alexey Kurakin, Ian Goodfellow **and** Samy Bengio. Adversarial examples in the physical world. 2017. arXiv: 1607.02533 [cs.CV]. URL: https://arxiv.org/abs/1607.02533.

[5] Nicholas Carlini **and** David Wagner. Towards Evaluating the Robustness of Neural Networks. 2017. arXiv: 1608.04644 [cs.CR]. URL: https://arxiv.org/abs/1608.04644.

[6] Seyed-Mohsen Moosavi-Dezfooli **andothers**. Universal adversarial perturbations. 2017. arXiv: 1610.08401 [cs.CV]. URL: https://arxiv.org/abs/1610.08401.

[7] Weimin Zhao, Sanaa Alwidian **and** Qusay H. Mahmoud. **?**Adversarial Training Methods for Deep Learning: A Systematic Review**? in**Algorithms: 15.8 (2022). ISSN: 1999-4893. DOI: 10.3390/a15080283. URL: https://www.mdpi.com/1999-4893/15/8/283.

[8] Nicolas Papernot et al. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networ 2016. arXiv: 1511.04508 [cs.CR]. URL: https://arxiv.org/abs/1511.04508.