

A. Related work

This section presents related works that use preprocessing and test-time inference methods for enhanced classification robustness. The former preprocesses the image before sending it to the classifier, whereas the latter changes the inference methodology, using an inference algorithm. Finally, we discuss several adversarial defense methods that aim at unseen attacks.

The first work on preprocessing methods, which improve classification robustness, includes random resizing and padding (Xie et al., 2017), thermometer encoding (Buckman et al., 2018), feature squeezing (Xu et al., 2017), defense GAN (Samangouei et al., 2018) and masking and reconstructing (Yang et al., 2019). A newer line of preprocessing methods uses probabilistic models. These methods aim to leverage their generative power to clear perturbations from an attacked image. They perform it by projecting the attacked image back to the data manifold. This line of work includes purification by pixelCNN (Song et al., 2017), EBM for restoring corrupt images (Du & Mordatch, 2019) and a density aware classifier (Grathwohl et al., 2019). The most recent works in this field includes Langevin sampling (Hill et al., 2020) and a gradient ascent score-based model (Yoon et al., 2021). Similarly, two recent studies utilize the generative power of diffusion models (Nie et al., 2022; Blau et al., 2022).

Another group of adversarial defense schemes uses test-time methods. Cohen et al. (Cohen et al., 2019) and Raff et al. (Raff et al., 2019) suggest multiple realizations of the augmented image to be performed during test-time, followed by averaging the classification predictions. Schwinn et al. (Schwinn et al., 2022) suggest to analyze the robustness of a local decision region nearby the attacked image. While (Cohen et al., 2019; Raff et al., 2019) require fine-tuning the model, Schwinn et al. require neither fine-tuning nor access to the training data, similar to our method.

The last group of studies aims to provide robustness against unseen attacks. Laidlaw et al. (Laidlaw et al., 2020) provides a latent space norm-bounded AT, and Blau et al. (Blau et al., 2022) uses a vanilla trained diffusion model as a preprocessing step.

To the best of our knowledge, Schwinn et al. (Schwinn et al., 2022) offer the only test time method that enhances adversarially trained robust classifiers without further training, similar to our method.

B. Background

Since the discovery of adversarial attacks (Szegedy et al., 2013; Goodfellow et al., 2014; Kurakin et al., 2016; Athalye et al., 2018; Biggio et al., 2013; Carlini & Wagner, 2017; Kurakin et al., 2018; Nguyen et al., 2015), there has been a continuous development of defense and attack techniques. In this section, we briefly overview key results starting with attacks, moving to defense strategies, and finishing with the PAG property.

An adversarial attack is a perturbation δ , added to an image x , intended to push a classifier decision away from the correct prediction. Many important studies researched adversarial attacks, of which it is important to mention Madry et al. (Madry et al., 2017) who laid the foundation for many following works, including ours. Madry et al. introduced the projected gradient descent (PGD) algorithm, which is an iterative optimization process that searches for the worst-case adversarial example. PGD has access to the classifier's weights, and is, therefore, able to find the worst-case adversary in a small radius around a clean data sample. The allowed perturbation is defined by a threat model, characterized by an ℓ_p norm and a radius ϵ , such that $\|\delta\|_p \leq \epsilon$. There exist two variants for the loss term. The objective of the first variant is to maximize the classification loss of the perturbed input, given the true label. In other words, the input image is manipulated in order to increase the error, aiming for a wrong classifier decision (*any* class except the correct one). The second variant's objective is to minimize the classification loss of the perturbed image given a *specific* wrong class label y . As a result, the classifier is more likely to predict label y . Our method utilizes the latter *targeted* PGD variant.

One of the leading robustification methods known as adversarial training (AT) (Madry et al., 2017; Zhang et al., 2019; Rebuffi et al., 2021; Gowal et al., 2020; Salman et al., 2020; Goodfellow et al., 2014; Carlini & Wagner, 2017; Croce & Hein, 2020; Tramer et al., 2020). AT is a training method that robustifies a classifier against a specific attack. This is achieved by introducing adversarial examples during the training process, and driving the classifier to learn to infer the true labels of malicious examples. While training, the classifier weights are constantly being changed. As a result, the classifier's worst-case examples keep on changing as well. Hence, in every training batch, one must calculate new adversarial examples that fit the current state of the classifier.

It has been recently discovered that some classifiers exhibit an intriguing property called perceptually aligned gradients

(PAG) (Engstrom et al., 2019; Etmann et al., 2019; Ross & Doshi-Velez, 2018; Tsipras et al., 2018). PAG is manifested through the classification loss gradients, with respect to the input image, appearing visually meaningful to humans, and resembling one of the dataset classes. The structure of the gradients is different when performing untargeted vs. targeted PGD. When performing untargeted PGD, the attack is not leading to a specific class; therefore, the gradients transform the image arbitrarily. When performing targeted PGD, however, the gradients transform the image into the target class, removing current class features. Using this property, our method enables, one to transform an image into a target class, as used by our method.

C. Experimental setup

In this part, we provide some details about our methods. We are performing AutoAttack (Croce & Hein, 2020) on the base classifier, then performing our defense method. Similar to previous works such as (Schwinn et al., 2022). We use $N = 30$ steps of TETRA and perform hyperparameter tuning, finding the best step size α and γ for every classifier. We use these parameters for all the evaluations, clean images and all of the attacks. We state both α and γ for every dataset in Tables 4 to 6.

We evaluated the other test-time defense, DRQ (Schwinn et al., 2022), with the official code using the reported parameters.

Method	Architecture	TTM	α	γ
Madry et al. (Madry et al., 2017) + TETRA	RN50	$L_2, \epsilon = 0.5$	1.5	200
Rebuffi et al. (Rebuffi et al., 2021) + TETRA	WRN28-10	$L_2, \epsilon = 0.5$	0.5	400
Rebuffi et al. (Rebuffi et al., 2021) + TETRA	WRN28-10	$L_\infty, \epsilon = 8/255$	0.1	300
Gowal et al. (Gowal et al., 2020) + TETRA	WRN70-16	$L_\infty, \epsilon = 8/255$	0.3	300
Vanila + TETRA	WRN28-10	-	0.05	300

Table 4. CIFAR10 params. In the first column, we state the method. In the next columns, we state the architecture, the trained threat model (TTM), α which is the step size and γ which is the regularization weight.

Method	Architecture	TTM	α	γ
Rebuffi et al. (Rebuffi et al., 2021) + TETRA	WRN28-10	$L_\infty, \epsilon = 8/255$	0.1	300
Rebuffi et al. (Rebuffi et al., 2021) + FETRA	WRN28-10	$L_\infty, \epsilon = 8/255$	0.1	300
Gowal et al. (Gowal et al., 2020) + TETRA	WRN70-16	$L_\infty, \epsilon = 8/255$	0.1	100
Gowal et al. (Gowal et al., 2020) + FETRA	WRN70-16	$L_\infty, \epsilon = 8/255$	0.1	100

Table 5. CIFAR100 params. In the first column, we state the method. In the next columns, we state the architecture, the trained threat model (TTM), α which is the step size and γ which is the regularization weight.

Method	Architecture	TTM	α	γ
Madry et al. (Madry et al., 2017) + FETRA	RN50	$L_2, \epsilon = 3.0$	6.0	5500
Salman et al. (Salman et al., 2020) + FETRA	WRN50-2	$L_2, \epsilon = 3.0$	6.0	3000
Madry et al. (Madry et al., 2017) + FETRA	RN50	$L_\infty, \epsilon = 4/255$	1.0	6000
Salman et al. (Salman et al., 2020) + FETRA	WRN50-2	$L_\infty, \epsilon = 4/255$	1.0	3000

Table 6. ImageNet params. In the first column, we state the method. In the next columns, we state the architecture, the trained threat model (TTM), α which is the step size and γ which is the regularization weight.

D. Transformed images

We proceed to the visualization of the TETRA transformation, supplying an intuitive explanation of what the transformation looks like and why our method works.

First, we compare our method to PGD. As shown in Figure 2, PGD transforms the image's appearance significantly in order to change its classification. TETRA also modifies the image appearance to the target class, however, it keeps the transformed image pixel-wise close to the input image. The transformation is performed in a rather artistic way, as it is almost imperceptible that the *toucan* appears on the *lorikeet*'s wing.

TETRA relies on the hypothesis that the extent of the image modification relates to the probability of belonging to a certain class. Therefore, it is important to supply visual evidence and intuition. To this end, we present Figure 3, where we demonstrate TETRA transformation towards multiple classes. In the first column, we present the clean image. In the following four columns, we transform the image into target classes. First, to the true class, then to a similar class, and finally to two entirely different class categories. This figure emphasizes that a transformation of an image to the true class, or similar ones, requires minor changes. However, when transforming an image into a different category, the image is modulated considerably by adding some features belonging to the target class.

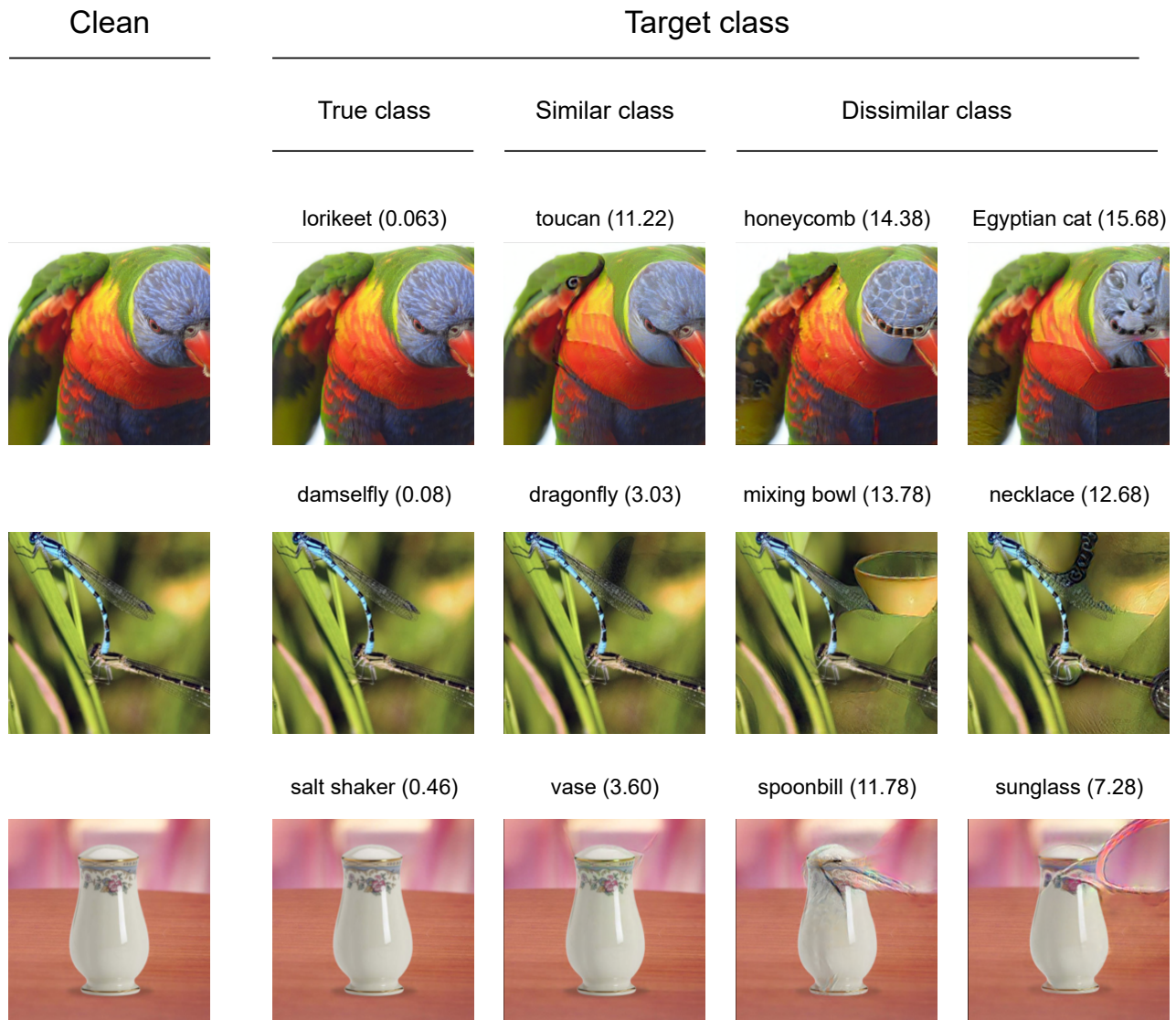


Figure 3. Visualization of TETRA’s transformation. Clean images from the ImageNet dataset (Deng et al., 2009) (left column) are transformed by TETRA into (left-to-right): the true class, a class from the same category, and two classes from different categories. Pixel-level distance is more noticeable as the perceptual distance to the target class grows (presented in the images’ titles as the ℓ_2 distance).

E. RPGD analysis

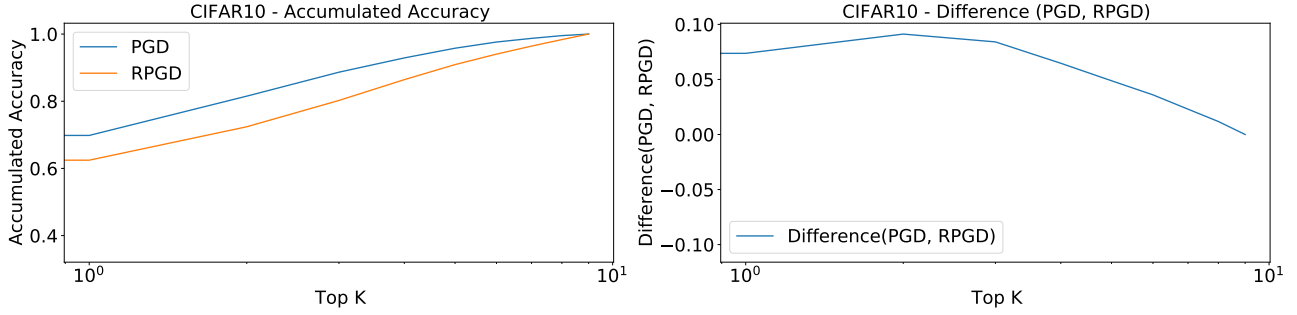


Figure 4. CIFAR10 top k accuracy comparison PGD vs RPGD. the x-axis of the left figure represents the top k group size that we select, using Madry et al. (Madry et al., 2017) $\ell_2, \epsilon = 0.5$. The y-axis represents the top k accuracy, the probability that the true label is contained in the top k group. On the right figure, we present the difference between the two graphs of the left figure, $PGD - RPGD$.

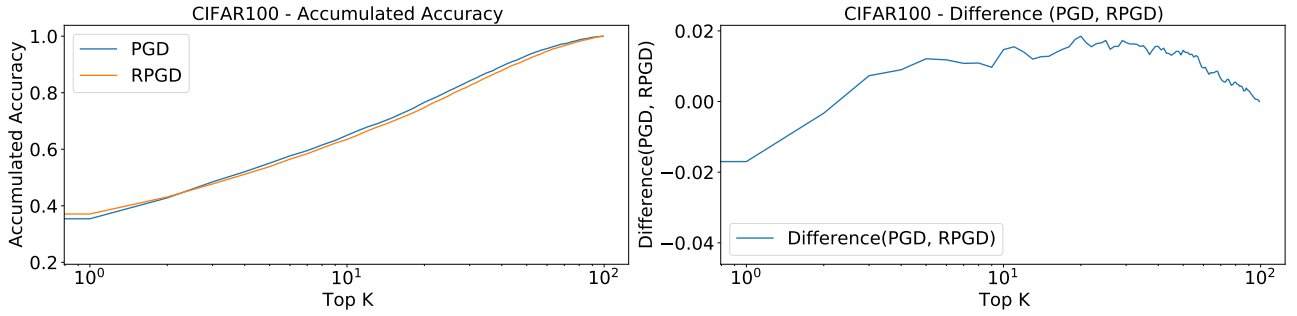


Figure 5. CIFAR100 top k accuracy comparison PGD vs RPGD. the x-axis of the left figure represents the top k group size that we select, using Madry et al. (Rebuffi et al., 2021) $\ell_\infty, \epsilon = 8/255$. The y-axis represents the top k accuracy, the probability that the true label is contained in the top k group. On the right figure, we present the difference between the two graphs of the left figure, $PGD - RPGD$.

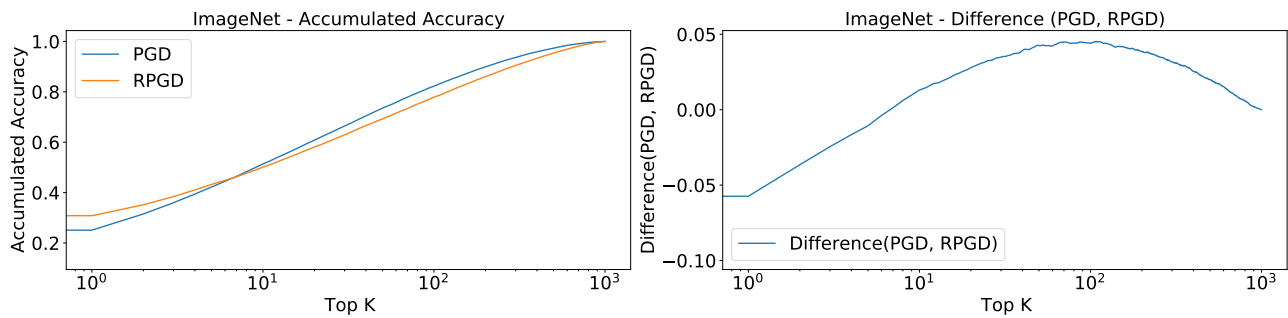


Figure 6. ImageNet top k accuracy comparison PGD vs RPGD. the x-axis of the left-hand side of the figure represents the top k group size that we select, using Madry et al. (Madry et al., 2017) $\ell_2, \epsilon = 3.0$. The y-axis represents the top k accuracy, the probability that the true label is contained in the top k group. For example, if we examine $k = 20$ it means that we seek the probability that the true label is in one of the top 20 predictions of the classifier, which is around 60%. On the right-hand side of the figure, we present the difference between the two graphs of the left-hand side, $PGD - RPGD$.

F. Ablation study

In this part, we discuss the ablations that we performed in order to better understand the contribution of different parts of our method. In Appendix F.1 we discuss the necessity of the PAG property in the TETRA algorithm, next in Appendix F.2 we discuss different options for the distance metric used for classification.

F.1. Vanila classifier

TETRA can be applied to any differentiable classifier. We claim, however, that it enhances the classifier robustness only over classifiers that possess PAG. In this part, we empirically support this claim.

In Table 7, we present TETRA accuracy on CIFAR10 dataset, where the classifier is vanilla trained. As we can see, TETRA achieves an accuracy of around 1% for all of the attacks. When applying TETRA to PAG classifiers, it achieves much better results, as presented in Table 1. Meaning that TETRA performs well only when applied to classifiers that possess the PAG property. The reason is that our method heavily relies on the generative power of PAG, which does not exist in vanilla-trained classifiers.

Method	Architecture	TTM	Standard	Attack			
				L_∞		L_2	
				8/255	16/255	0.5	1.0
Vanila			95.26%	00.00%	00.00%	00.00%	00.00%
DRQ (Schwinn et al., 2022)	WRN28-10	None	10.95%	11.31%	11.04%	11.75%	11.38%
TETRA			93.04%	01.11%	01.12%	01.24%	01.10%

Table 7. CIFAR10 vanilla classifier results. In the first column, we state the method. We report three consecutive lines of results. One for the base method and then two test time boosting methods: DRQ (Schwinn et al., 2022) and TETRA. In the next columns, we state the architecture, the trained threat model (TTM), and four attacks with different threat models.

F.2. Distance metrics

In TETRA’s second phase, we calculate the distance between the input image and the transformed images, and we classify based on the shortest one. Hence, the distance metric that we use for the classification is important. Different metrics have different properties, and we aim at a distance metric that is able to measure the semantic distance between images.

We compare ℓ_2 , ℓ_1 and LPIPS (Zhang et al., 2018) distances over CIFAR10 dataset, and present the results in Table 8. We compare the results using the following defense methods (Madry et al., 2017; Rebuffi et al., 2021; Gowal et al., 2020). As demonstrated, ℓ_2 distance metric performs better, therefore is a favorable choice.

Method	Architecture	TTM	Standard	Attack			
				L_∞ 8/255	L_∞ 16/255	L_2 0.5	L_2 1.0
AT (Madry et al., 2017)	RN50	$L_2, \epsilon = 0.5$	90.83%	29.04%	00.93%	69.24%	36.21%
TETRA LPIPS			85.91%	54.49%	17.46%	78.70%	61.68%
TETRA L_1			85.91%	54.55%	17.64%	78.70%	61.68%
TETRA L_2			85.76%	54.55%	17.64%	78.74%	61.87%
Rebuffi et al. (Rebuffi et al., 2021)	WRN28-10	$L_2, \epsilon = 0.5$	91.79%	47.85%	05.00%	78.80%	54.73%
TETRA LPIPS			87.31%	58.57%	09.97%	85.30%	67.03%
TETRA L_1			87.31%	58.57%	09.97%	85.30%	67.03%
TETRA L_2			88.33%	59.74%	11.06%	85.57%	66.01%
Rebuffi et al. (Rebuffi et al., 2021)	WRN28-10	$L_\infty, \epsilon = 8/255$	87.33%	60.77%	25.44%	66.72%	35.01%
TETRA LPIPS			80.97%	66.49%	33.54%	74.73%	59.25%
TETRA L_1			80.97%	66.49%	33.54%	74.73%	59.25%
TETRA L_2			84.86%	66.96%	35.15%	74.84%	53.32%
Gowal et al. (Gowal et al., 2020)	WRN70-16	$L_\infty, \epsilon = 8/255$	91.10%	65.88%	25.95%	66.44%	27.22%
TETRA LPIPS			83.41%	71.51%	38.49%	76.53%	58.60%
TETRA L_1			83.41%	71.51%	38.49%	76.56%	58.60%
TETRA L_2			87.58%	72.00%	40.44%	75.65%	49.22%

Table 8. CIFAR10 results. In the first column, we state the method. For every base method, we report three consecutive lines of results. One for the base method and then two TETRA distance metric variations used for classification: L_2 and LPIPS (Zhang et al., 2018). In the next columns, we state the architecture, the trained threat model (TTM), and four attacks with different threat models.

G. Runtime analysis

In this part, we compare the inference time of the test-time methods that we used, over CIFAR10 and CIFAR100. For CIFAR10 we compare TETRA to DRQ (Schwinn et al., 2022), and for CIFAR100 we compare FETRA to DRQ (Schwinn et al., 2022). As can be seen, for both of the datasets, our method is slower than the baseline. Our method, however, is faster than DRQ (Schwinn et al., 2022). These experiments were performed using one GeForce RTX 3080 with batch size = 1.

Method	Architecture	TTM	Inference time
AT (Madry et al., 2017)			$\times 1$
DRQ (Schwinn et al., 2022)	RN50	$L_2, \epsilon = 0.5$	$\times 160$
TETRA			$\times 23$
Rebuffi et al. (Rebuffi et al., 2021)			$\times 1$
DRQ (Schwinn et al., 2022)	WRN28-10	$L_2, \epsilon = 0.5$	$\times 117$
TETRA			$\times 26$
Rebuffi et al. (Rebuffi et al., 2021)			$\times 1$
DRQ (Schwinn et al., 2022)	WRN28-10	$L_\infty, \epsilon = 8/255$	$\times 119$
TETRA			$\times 26$
Gowal et al. (Gowal et al., 2020)			$\times 1$
DRQ (Schwinn et al., 2022)	WRN70-16	$L_\infty, \epsilon = 8/255$	$\times 290$
TETRA			$\times 127$

Table 9. Inference time comparison over CIFAR10. In this table we perform an inference time comparison between 3 defense methods. For every base classifier, we report three consecutive lines of inference time. One for the base method, next we present DRQ, and finally TETRA. In the first column we present the method name. Next we present the architecture, and the trained threat model (TTM), and finally we present the inference time. This value stands for how much time it takes for every method to perform.

Method	Architecture	TTM	Inference time
Rebuffi et al. (Rebuffi et al., 2021)			$\times 1$
DRQ (Schwinn et al., 2022)	WRN28-10	$L_{inf}, \epsilon = 8/255$	$\times 686$
FETRA			$\times 27$
Gowal et al. (Gowal et al., 2020)			$\times 1$
DRQ (Schwinn et al., 2022)	WRN70-16	$L_\infty, \epsilon = 8/255$	$\times 1380$
FETRA			$\times 121$

Table 10. Inference time comparison over CIFAR100. In this table we perform an inference time comparison between 3 defense methods. For every base classifier, we report three consecutive lines of inference time. One for the base method, next we present DRQ, and finally TETRA. In the first column we present the method name. Next we present the architecture, and the trained threat model (TTM), and finally we present the inference time. This value stands for how much time it takes for every method to perform.