[F24] Practical Machine Learning & Deep Learning
D2: Final Technical Report

Artem Matevosian, Sofia Shulyak, Ekaterina Mozhegova

November 23, 2024

Team Anticrawler

- Artem Matevosian, a.matevosian@innopolis.university
- Sofia Shulyak, s.shulyak@innopolis.university
- Ekaterina Mozhegova, e.mozhegova@innopolis.university

Project topic:

Anticrawler: Applying adversarial attacks on a CAPTCHA recognition model and proposing effective defensive methods.

Repository

The repository for our project is available at: GitHub Repository (click). It contains all the artifacts, including the dataset, CAPTCHA-recognition model, implementations of attacks and defenses.

1 What we aimed for

The purpose of the project is to:

- propose and create a framework that charges traditional graphical CAPTCHA for adversarial attacks on recognition systems;
- study and enhance the robustness of recognition models in terms of dealing with such adversarial attacks;
- provide illustrations for gradient-based attacks and classical CV-based defenses in the form of a web app.

2 Methods

2.1 Dataset

To generate the dataset for our task, we combined several datasets found on Kaggle. Specifically, these are:

- Captcha Version 2 Images, containing examples of CAPTCHA images. The images consist of 5-letter words that may include numbers. They have noise applied (such as blur and lines) and are 200 x 50 PNGs.
- Large Captcha Dataset, which contains over 80,000 CAPTCHA images. Each CAPTCHA consists of 5 characters, which can be numbers or alphabets. The images vary in color, character spacing, orientation, and size. Random noise and lines have also been added to the CAPTCHAs.

These are some examples of images from our dataset.

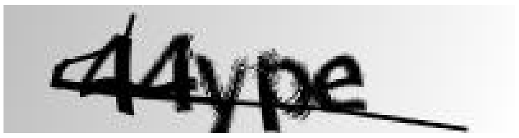


Figure 1: Dataset sample 1

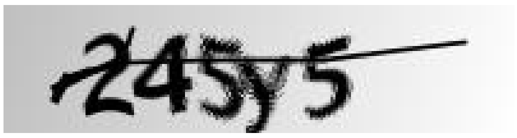


Figure 2: Dataset sample 2

2.2 Modeling

2.2.1 Model for CAPTCHA recognition

For CAPTCHA recognition, we implemented a ResNet50-based model with the following training parameters:

- 10 epochs
- 80% of the data was used for training, and 20% for testing
- Batch size = 32

The accuracy achieved was 94.5%.

2.2.2 Adversarial Training Method

We employed adversarial training (further details are provided later in the text) with the following parameters:

- 5 epochs
- Training set = 80% for each of the 4 datasets (original, attacked with FGSM, attacked with PGD, attacked with C&W)
- Batch size = 64

2.3 Investigation and implementation of adversarial attacks and defensive methods (Research part)

This section is related to implementations of all the attacks, defenses, and systems allowing to quickly test former against the latter (click to open the corresponding file in GitHub repository):

- untargeted Fast Gradient Sign Method (FGSM) attack, as described by Goodfellow *et al.* [2];
- untargeted Projected Gradient Descent (PGD) attack [3];
- untargeted modification of Carlini-Wagner (C&W) attack (which is originally implemented as targeted [1]);
- image gradient, median filter and thresholding defenses;
- Gaussian blur, Gaussian noise, grayscaling, normalization were used in the comparison, however, their implementations were provided in the `torchvision` library [7].

Although all named attacks are of the same type — white-box, where the attacker has full knowledge of the model being attacked—we decided to focus specifically on these attacks due to their high popularity and prevalence and due to the fact that defensive methods against adversarial attacks for both black-box and white-box scenarios often overlap.

As for the choice of defensive methods, we aimed to balance their effectiveness and simplicity.

Suggested article reading

The choices we made for our task are based on an extensive literature review. To delve deeper into the topic of attacks and defenses against them, we also reviewed the following articles:
"Defending Against Adversarial Examples" [6],
"Attacking Machine Learning with Adversarial Examples" (OpenAI) [4],
"Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks" [5].

2.4 Implementations (Production part)

This section is related to deployment of a web-app showcasing all the attack and defense methods implemented along with model inference.

The final structure of the app is as follows:

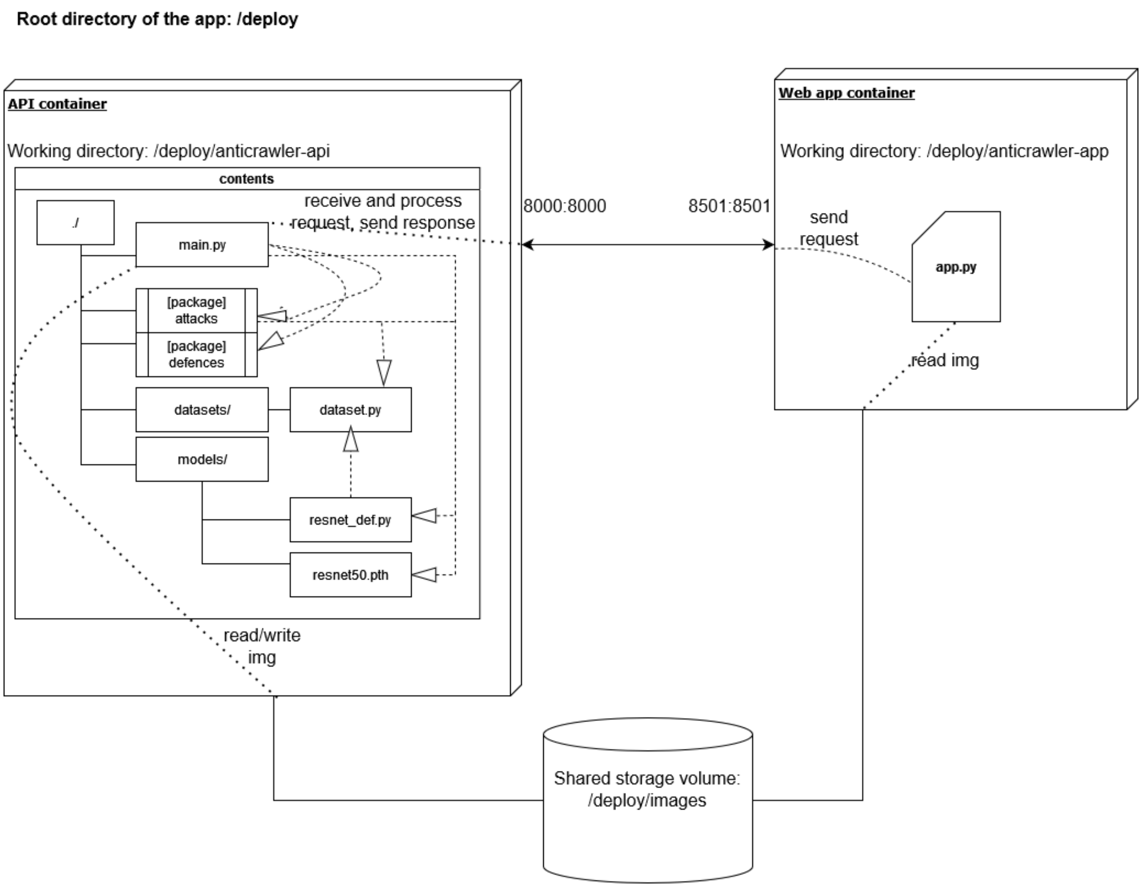
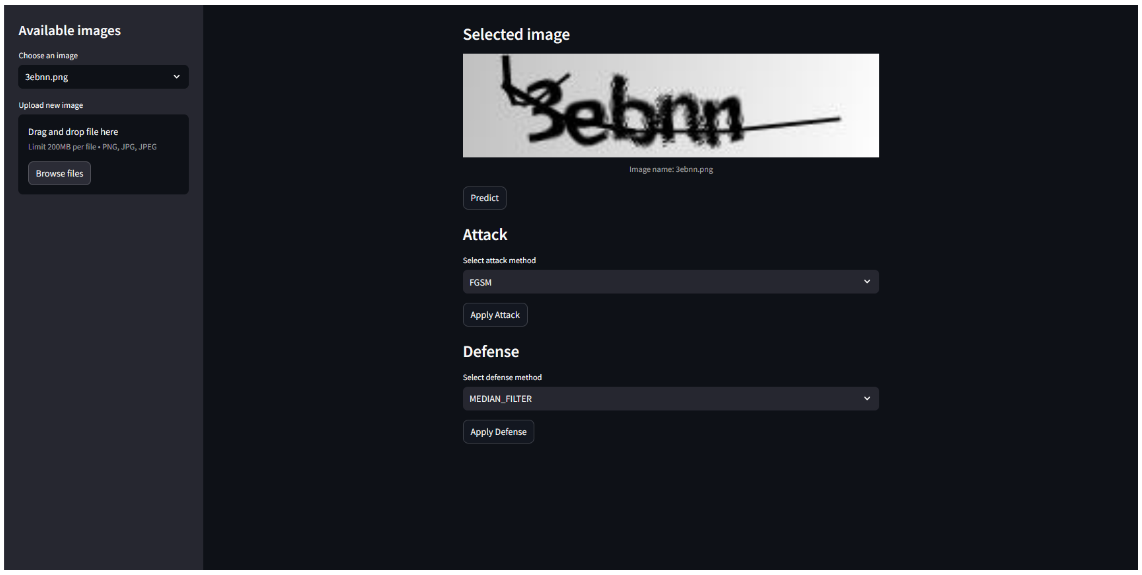

Root directory of the app: /deploy
API container
Working directory: /deploy/anticrawler-api
contents
./
main.py
receive and process request, send response
[package] attacks
[package] defences
datasets/
dataset.py
models/
resnet_def.py
resnet50.pth
read/write img
8000:8000
8501:8501
Web app container
Working directory: /deploy/anticrawler-app
send request
app.py
read img
Shared storage volume: /deploy/images

Figure 3: Deployment

The app showcases our implementations as follows (figures 2a-2e):



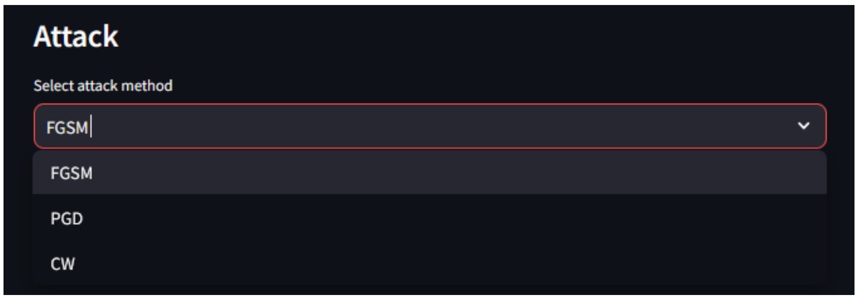
Choose an image: 3ebnn.png

Image name: 3ebnn.png

Select attack method: FGSM

Select defense method: MEDIAN_FILTER

Figure 4: a) Basic interface


Attack

Select attack method: FGSM

Figure 5: b) Attack method selection

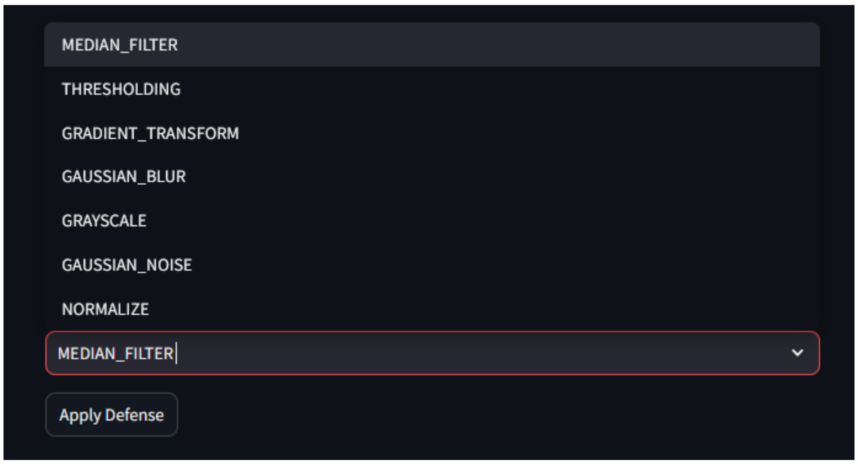
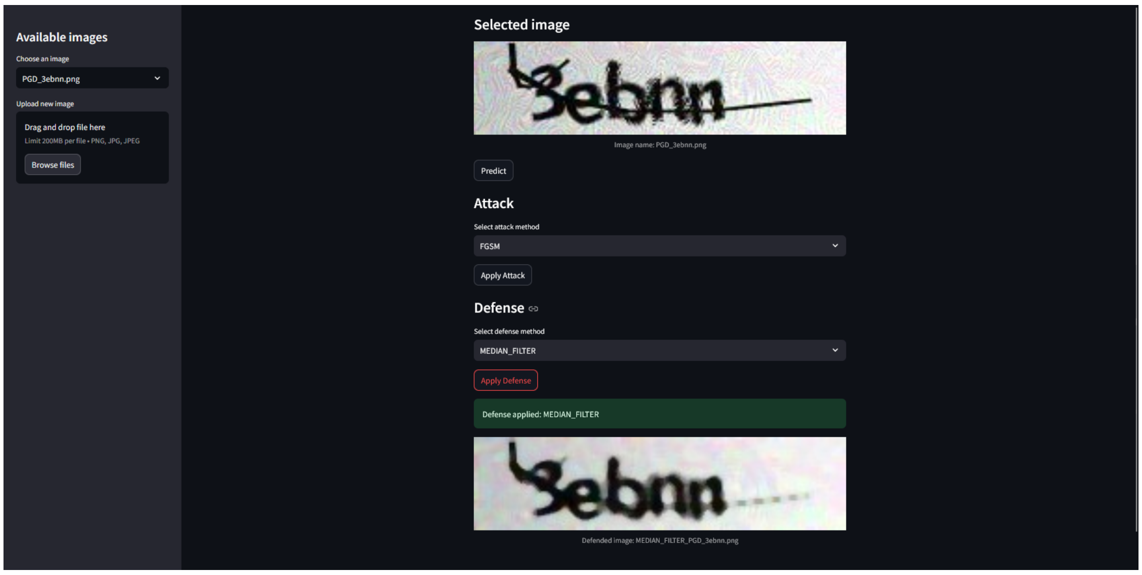

MEDIAN_FILTER

Figure 6: c) Defense method selection


Available images

Choose an image: PGD_3ebnn.png

Upload new image

Drag and drop file here

Limit 200MB per file • PNG, JPG, JPEG

Selected image

Image name: PGD_3ebnn.png

Attack

Select attack method: FGSM

Defense

Select defense method: MEDIAN_FILTER

Defense applied: MEDIAN_FILTER

Defended image: MEDIAN_FILTER_PGD_3ebnn.png

Figure 7: d) Displaying applied actions

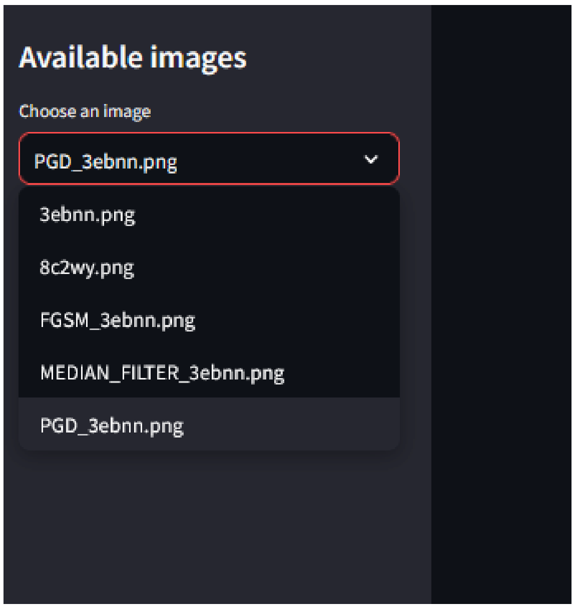


Figure 8: e) Browsing uploaded and produced images

Furthermore, we have performed significant refactoring of our API (mostly attack and defense related features) to make it more maintainable and readable. Namely, instead of copying some inconsistent parts of jupyter notebooks to the API, we use 2 packages containing .py files with required functionality to be imported for attacks and defenses correspondingly.

2.5 Attack-vs-Defense battle royale

This section describes all-vs-all attack-vs-defense test, which is carried out in the gridsearch-like fashion:

1. train a ResNet50 classifier using an original dataset (preferably train-test split it into separate directories before running to avoid evaluating on the previously-training-but-now-attacked images);

2. run `FGSM.py`, `PGD.py`, and `CW.py` to generate the attacked versions of the test set via FGSM, PGD, and C&W attacks respectively;

3. run the evaluation over all 4 versions of the test set and all desired transformations-defenses to obtain the accuracy scores;

4. redo the training using different transforms to obtain a model accommodated to these specific transforms and repeat evaluation (e.g. "vanilla" model does not perform well under normalization/gradient transform defense, so we have to train it on transformed unattacked images before feeding transformed attacked images in evaluation run).

5. BONUS: rerun step 1, skip step 2 and run step 3. Observe how different attacks survive the transfer onto an extremely similar yet a bit different model.

In the end, one should be able to construct an accuracy matrix. Specifically, we got the following results:

	no attack	FGSM ($\epsilon = 0.08$)	PGD ($\epsilon = 0.08$)	C&W ($c = 2$)
no defense	0.945	0.403	0.0477	0.0131
median filter	0.682	0.29	0.307	0.406
Gaussian blur	0.664	0.309	0.213	0.35
grayscale	0.945	0.816	0.727	0.896
Gaussian noise	0.813	0.338	0.236	0.476
thresholding	0.751	0.702	0.672	0.704
image gradient	0.919	0.907	0.903	0.916
normalization	0.939	0.307	0.599	0.791
no-change re-train	0.932	0.486	0.651	0.859

Table 1: Accuracy scores attained by ResNet50 under different attacks and defenses

Note: in this example, Gaussian blur and median filter were applied with the kernel size of 5. "No-change re-train" refers to step 5.

2.6 Adversarial training

We implemented single-iteration adversarial training by introducing attacked (all 3 variants) images into training set and then training the same model on the resulting dataset. Evaluation with no other defenses on the test set resulted in following:

- pure images read with accuracy = 0.943;
- attacked (FGSM, PGD, C&W) images with accuracy > 0.999.

*Note: after adversarially training, the model was tested on the same attacks, already familiar to the model.

3 Discussion

Based on our experiments, we can conclude the following:

- **Image Gradient** appears to be the most robust defense overall, with significant improvements across all types of attacks.
- **Grayscale** and **thresholding** also provide strong defenses, particularly against PGD and C&W attacks.
- Methods, such as **median filter**, **Gaussian noise**, and **Gaussian blur**, offer some utility but are less effective in providing robust defenses against advanced attacks such as PGD and C&W.
- **Adversarial training** appears to be the most effective defensive method overall, demonstrating a greater increase in robustness than any other CV-based method.

And we have the following considerations:

- Although the most effective method, **Adversarial training**, demonstrates excellent performance, it is important to note that it was trained on a specific set of attacks, so we must be cautious when generalizing its effectiveness to other scenarios.
- Theoretically, it could be useful to test different hyperparameters for attack methods (for example, ϵ) to observe the behavior of the model in various scenarios within the same setup. However, we did not do this, since even with relatively small values of the hyperparameters we obtained our goal to drop the model's accuracy significantly.
- **This project has potential for further enhancement**. We propose several directions for improvement, such as applying other types of attacks (preferably new ones, like black-box attacks) and testing more advanced defensive strategies (such as diffusion distillation).

4 Timeline and contribution

4.1 Initial research (see D1.1)

Date: 15 September - 22 September

- Artem Matevosian: Results & reviews compilation, typesetting
- Sofia Shulyak: Theoretical literature search & review
- Ekaterina Mozhegova: Practical literature search & review, typesetting

4.2 Attack engineering (see D1.2)

Date: 22 September - 6 October

- Artem Matevosian: Target model engineering, FGSM attacks, typesetting
- Sofia Shulyak: Interim target model engineering, data engineering
- Ekaterina Mozhegova: Projected Gradient Descent, Carlini-Wagner attacks, typesetting

4.3 Building MVP (see D1.3)

Date: 13 October - 27 October

- Artem Matevosian: studying defense methods, report
- Sofia Shulyak: api+web interface implementation and deployment
- Ekaterina Mozhegova: studying defense methods, report

4.4 Finalization (you are here!)

Date: 4 November - 24 November

- Artem Matevosian: refactoring and system design (research part), attack-vs-defense battle royale, classical defense methods, typesetting
- Sofia Shulyak: integrating new features to the app, refactoring and system design (production part), typesetting
- Ekaterina Mozhegova: method research, results interpretation, typesetting

References

[1] Nicholas Carlini and David Wagner. "Towards Evaluating the Robustness of Neural Networks". In: May 2017, pp. 39–57. DOI: 10.1109/SP.2017.49.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and harnessing adversarial examples*. Mar. 2015. URL: https://doi.org/10.48550/arXiv.1412.6572.

[3] Aleksander Madry et al. *Towards deep learning models resistant to adversarial attacks*. Sept. 2019. URL: https://arxiv.org/abs/1706.06083.

[4] OpenAI. *Attacking Machine Learning with Adversarial Examples*. Accessed: 2024-10-27. 2023. URL: https://openai.com/index/attacking-machine-learning-with-adversarial-examples/.

[5] Nicolas Papernot et al. "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks". In: May 2016, pp. 582–597. DOI: 10.1109/SP.2016.41.

[6] Austin Short, Trevor La Pay, and Apurva Gandhi. "Defending Against Adversarial Examples". In: (Sept. 2019). DOI: 10.2172/1569514. URL: https://www.osti.gov/biblio/1569514.

[7] *Torchvision 0.20 documentation*. URL: https://pytorch.org/vision/stable/index.html.