Adversarial AEA-CAPTCHA

Alireza Khodaie, Arman Torikoglu, Muhammed Esad Simitcioglu

Abstract

CAPTCHA is used to make decisions if the user is a human or a computer on the website. There are many different types of CAPTCHAs used today such as text-based, image-based, audio-based, etc. Recently, the asked test that enables this authentication system to distinguish between humans and computers can be easily solved using machine learning models. CAPTCHA designers have tried their best to improve security but most schemes deployed on the websites have been cracked [1].

In this project, we have benchmarked several attack strategies against a CAPTCHA solver having AlexNet architecture. Some of these attacks were among the popular attacks in the literature of adversarial examples, and some were produced by ourselves. We have generated adversarial (unsolvable) CAPTCHAs from these attacks and tried to predict them using our model. With these adversarial CAPTCHAs, we have successfully decreased the accuracy rate from 89% to 0%.

Motivation

It should be easy for individuals to solve but challenging for computers to grasp a CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart). Finding this harmony between security and usability is difficult. Automated registration, spam, and harmful bot programs are all prevented by CAPTCHA[2]. If the CAPTCHA has a human success rate of 90% or above and a computer program success rate of less than 1%, it is considered successful [3].

There are different kinds of CAPTCHA existing in the wild. Text-based CAPTCHA, Image-based CAPTCHA, Audio-based CAPTCHA, etc. In this project, we focused on Text-based ones. They are easy to solve by machine, yet it's still popular. Most of the text-based CAPTCHAs in the wild have been broken by big companies like Google and Microsoft.

Character crowding, noise arcs, two-layer structures, and other methods are used by CAPTCHA designers to increase the resilience of the CAPTCHA. However, investigations [4,5] have shown that all of these resistance mechanisms appear to be unsuccessful. These worries introduced a fresh subject to the CAPTCHA investigations. Are we actually safe when visiting websites, and if not, can cybersecurity address this issue?

You can find our implementation on this repository

Proposed Method

In the following sections, we describe our approach to present a CAPTCHA solver and generate adversarial CAPTCHAs to fool our model.

Dataset

Our dataset contains 30240 CAPTCHA images generated using this repository:
https://github.com/JackonYang/captcha-tensorflow.git

We split our dataset into train, validation, and test datasets as follows:

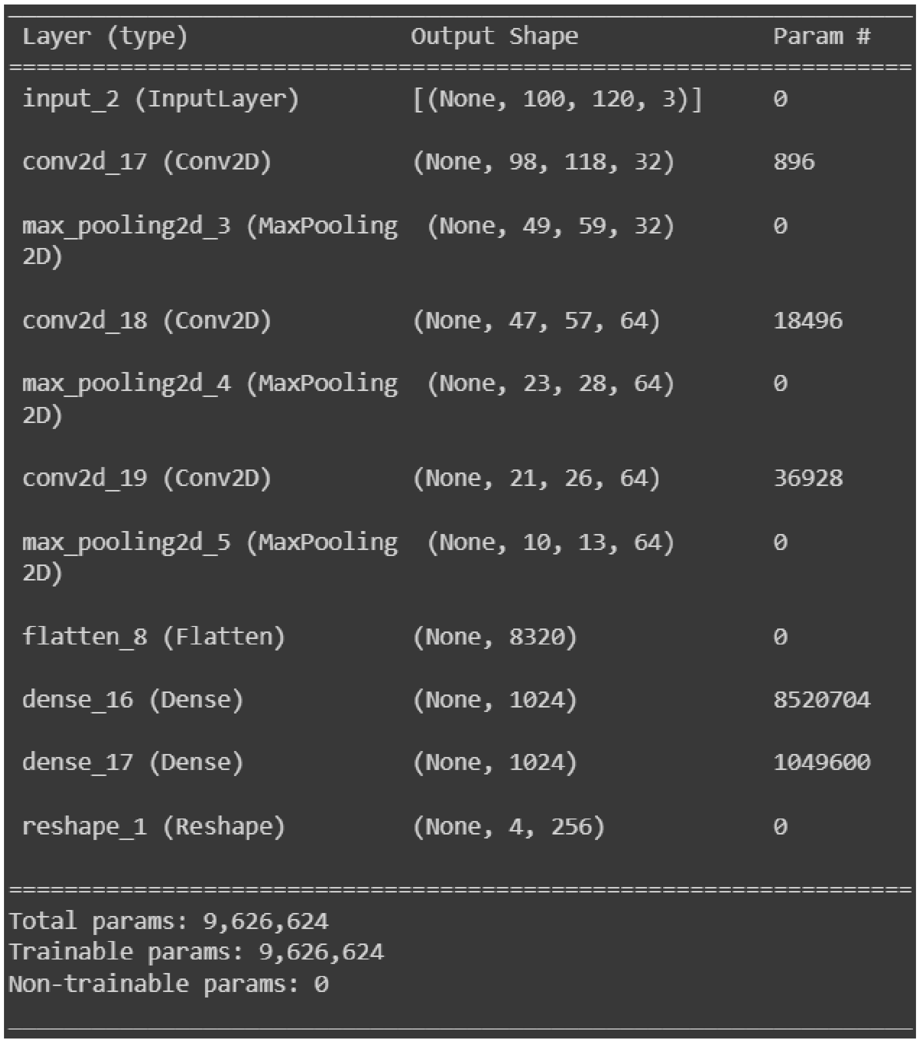
Train count: 14817, valid count: 6351, test count: 9072

The count of generated CAPTCHAs can be controlled to achieve high accuracy. We need large amounts of data to reach great accuracy, so we decided to have approximately 30,000 samples.

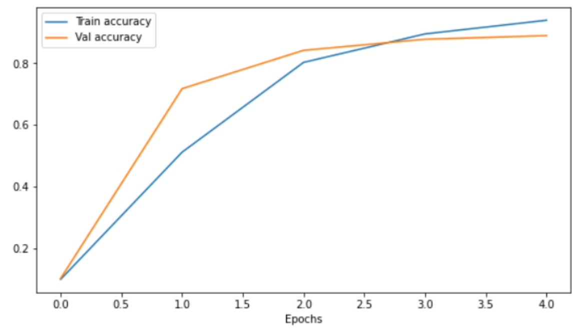
Automated CAPTCHA Solver

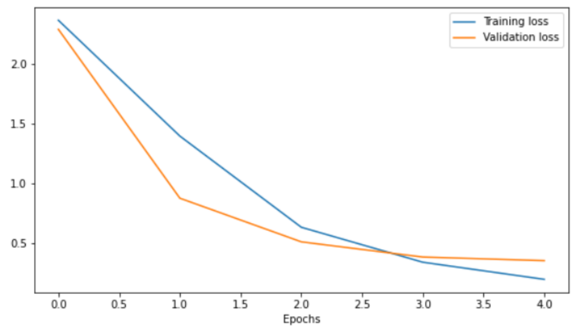
There are several CAPTCHA solver implementations with various architectures available. Most CAPTCHAs are classification challenges, and neural networks were most recently used for automatic recognition. These cutting-edge algorithms all have high accuracy rates. Our CAPTCHA solution will be implemented using Convolutional Neural Networks (CNN) as the primary architecture. Different strategies have been tried to solve CAPTCHAs by Shao et al. [6]. Our model was derived from AlexNet [7]. You can see details of our model below.

You can find our implementation on this repository



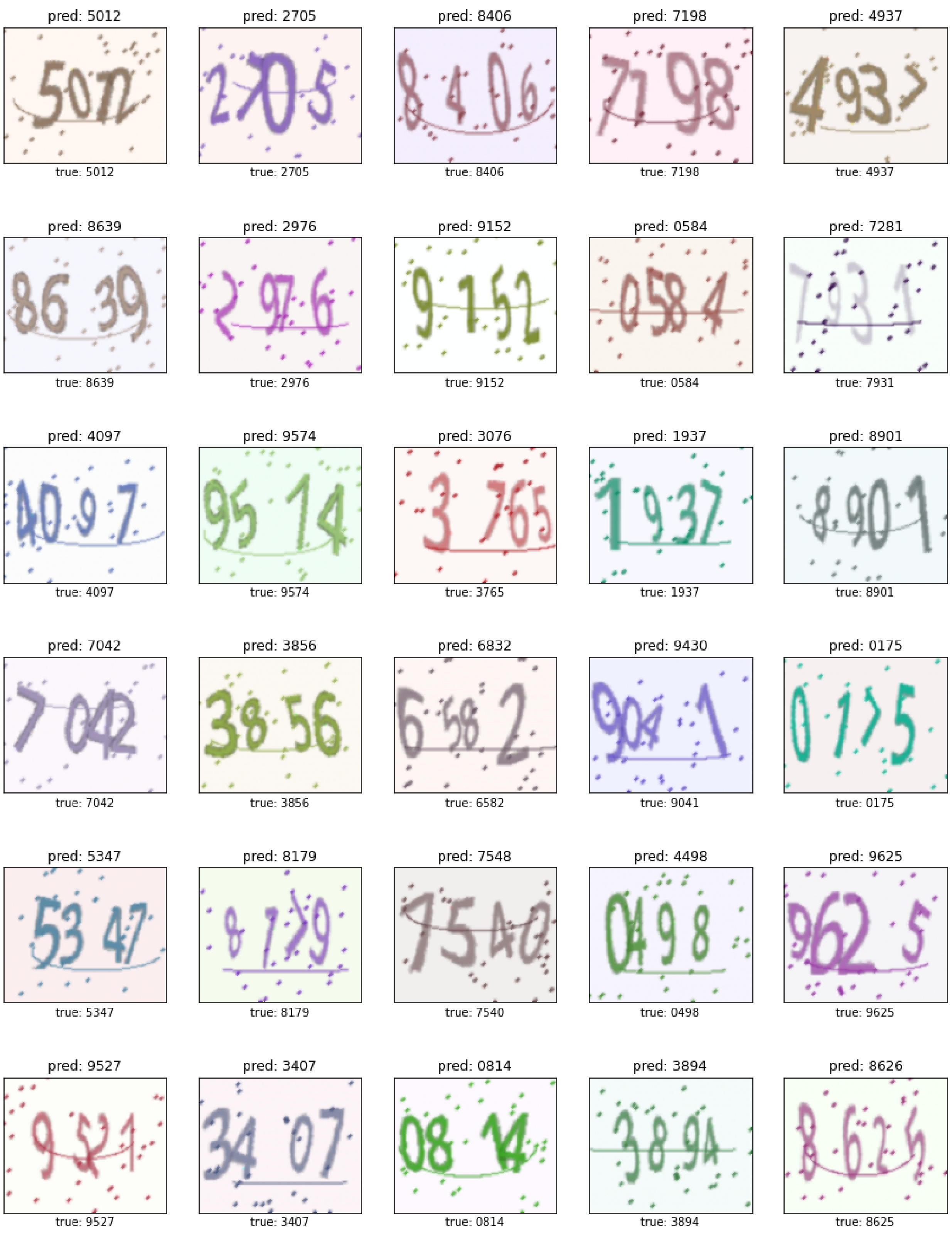
```
Layer (type)                Output Shape              Param #
=================================================================
input_2 (InputLayer)        [(None, 100, 120, 3)]     0

conv2d_17 (Conv2D)          (None, 98, 118, 32)       896

max_pooling2d_3 (MaxPooling  (None, 49, 59, 32)       0
2D)

conv2d_18 (Conv2D)          (None, 47, 57, 64)        18496

max_pooling2d_4 (MaxPooling  (None, 23, 28, 64)       0
2D)

conv2d_19 (Conv2D)          (None, 21, 26, 64)        36928

max_pooling2d_5 (MaxPooling  (None, 10, 13, 64)       0
2D)

flatten_8 (Flatten)         (None, 8320)              0

dense_16 (Dense)            (None, 1024)              8520704

dense_17 (Dense)            (None, 1024)              1049600

reshape_1 (Reshape)         (None, 4, 256)            0

=================================================================
Total params: 9,626,624
Trainable params: 9,626,624
Non-trainable params: 0
_________________________________________________________________
```





We achieved 91.56% accuracy on the training dataset, 91.24% accuracy on the validation dataset, and 89.21% on the test dataset.

You can find our implementation on this repository



Results for our CAPTCHA solver

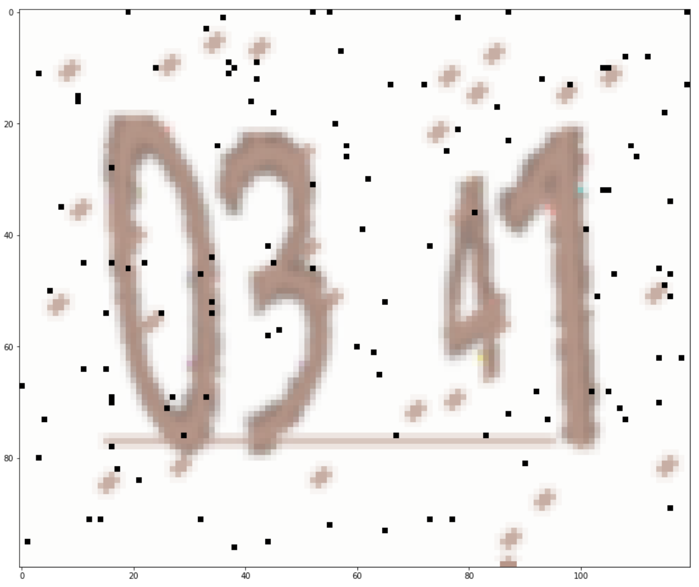
You can find our implementation on this repository

Evasion Attacks

Attacks during testing known as "evasion attacks" try to generate an error in the machine learning systems by manipulating the input data. In contrast to data poisoning, evasion attacks make use of the system's vulnerabilities and blind spots in order to create the intended mistakes.

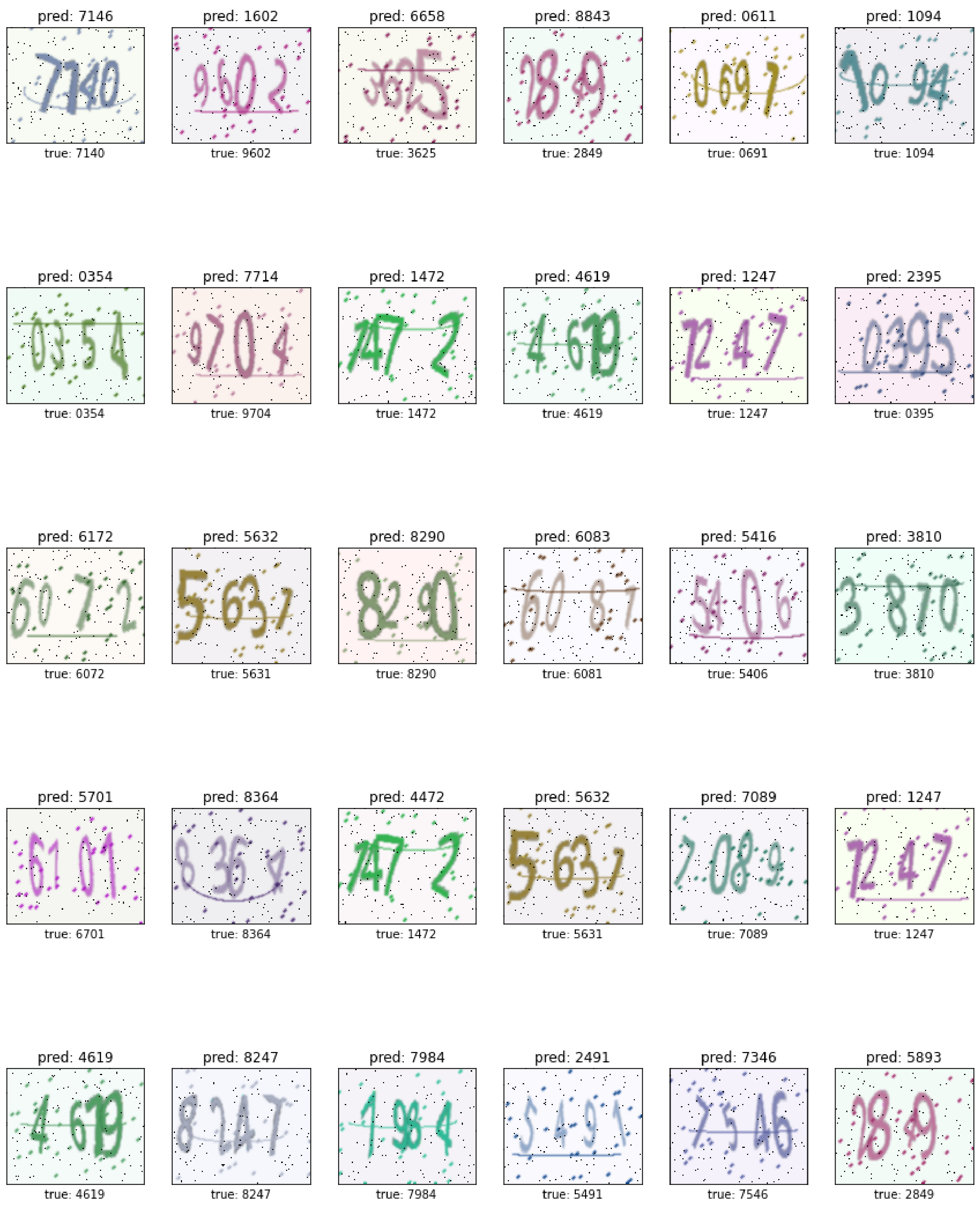
Random Perturbation

In this attack, with a one percent probability, we add random noise to the image's random location. This attack is highly visible to the human eye, but it's enough to fool the CAPTCHA solver. Using this attack, we were able to reduce the accuracy of the model to 53%.



Generated Adversarial CAPTCHA from RandomColor Attack

You can find our implementation on this repository

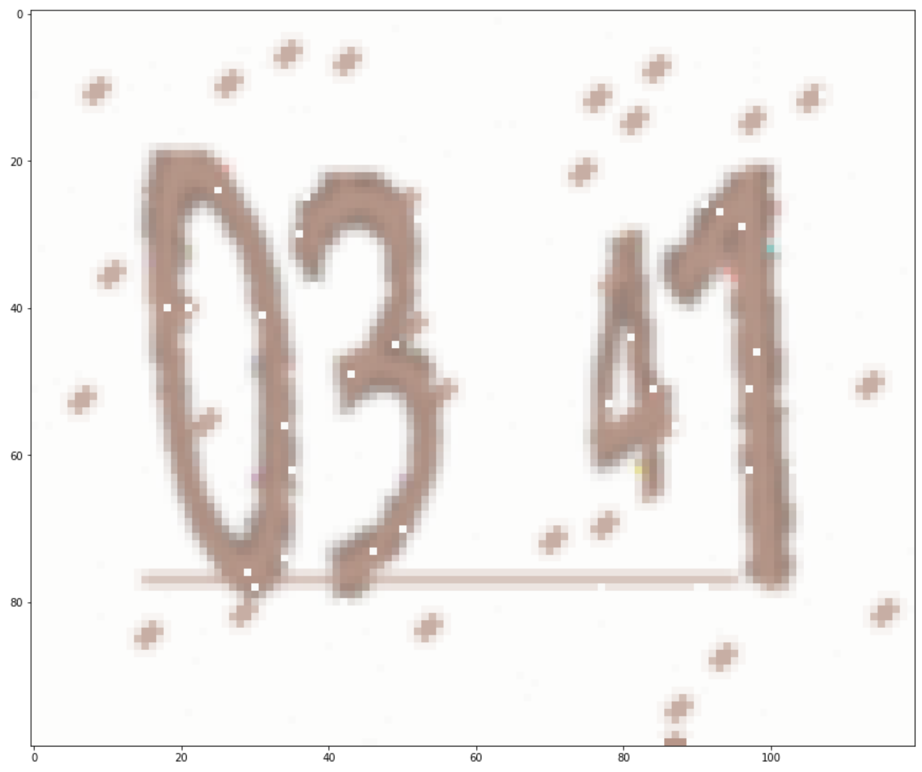


Results for Random Color Attack

You can find our implementation on this repository

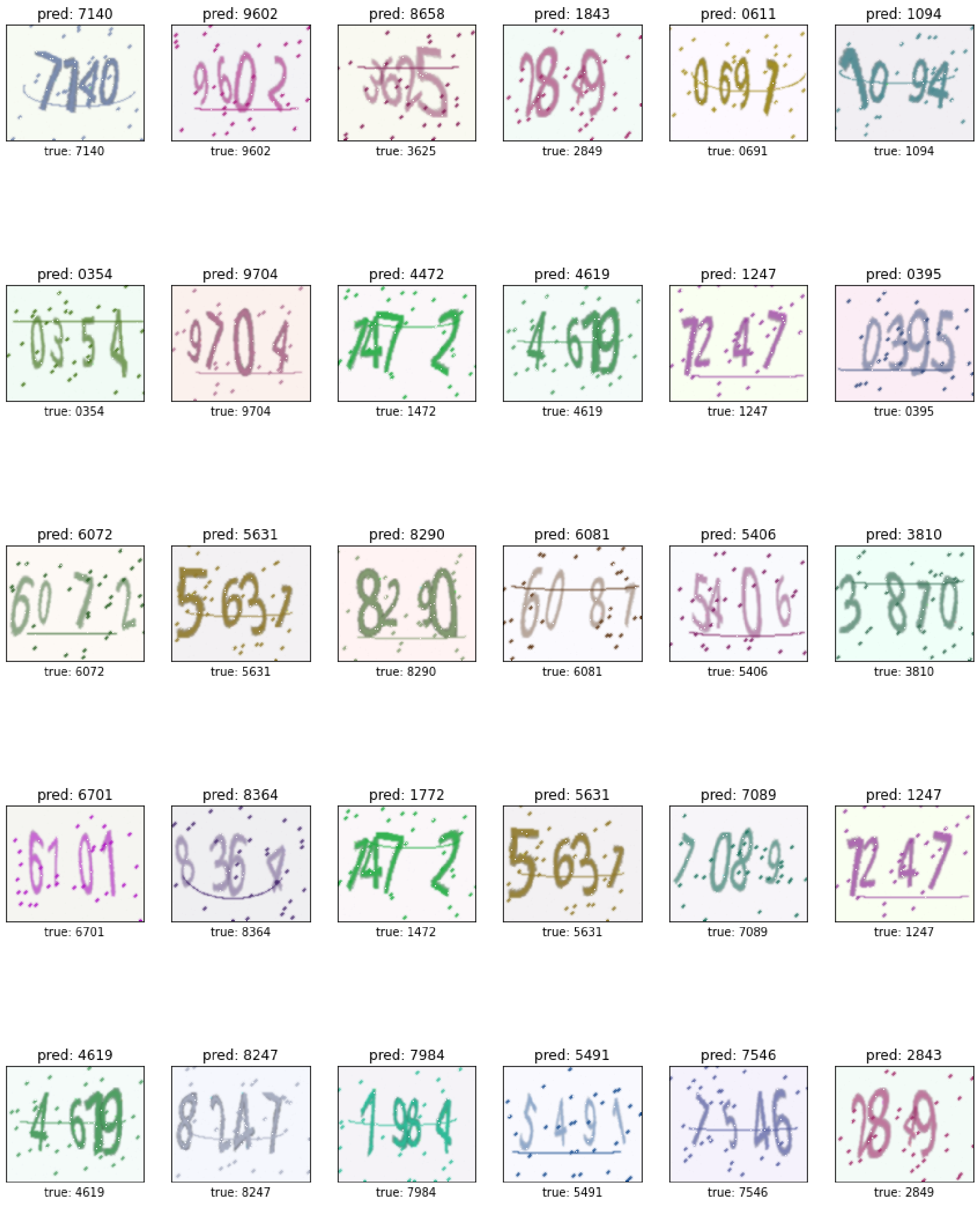
Perturbation Based on Background Color

In this attack, we pick the background color of the CAPTCHA, and then we add this pixel color to CAPTCHA's random location. The amount of added noise in this attack isn't too much, so it is not easy to be detected with human eye. The success rate of this attack is not also high. Using this attack we reduced the accuracy of the model to 80%. This attack does not have much impact on the model. It was an easy approach to the solution, and we know it is not an efficient attack to break the CAPTCHA solver, but we thought it will decrease the CAPTCHA solver's accuracy much more.



Generated Adversarial CAPTCHA from Background Color attack

You can find our implementation on this [repository]()

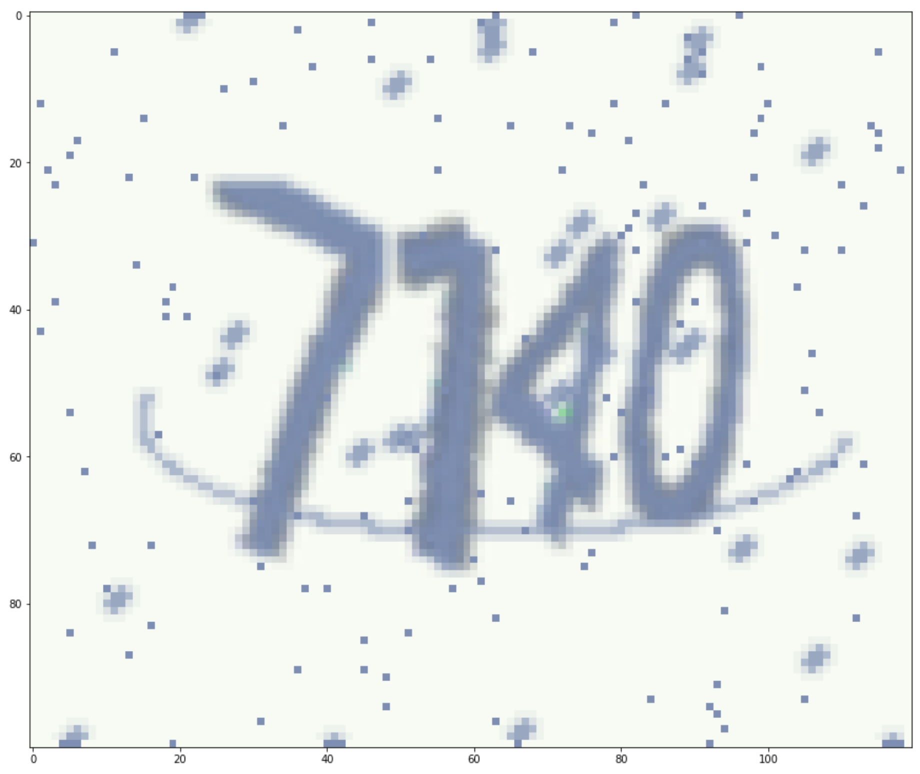


Results for Background Color attack

You can find our implementation on this repository

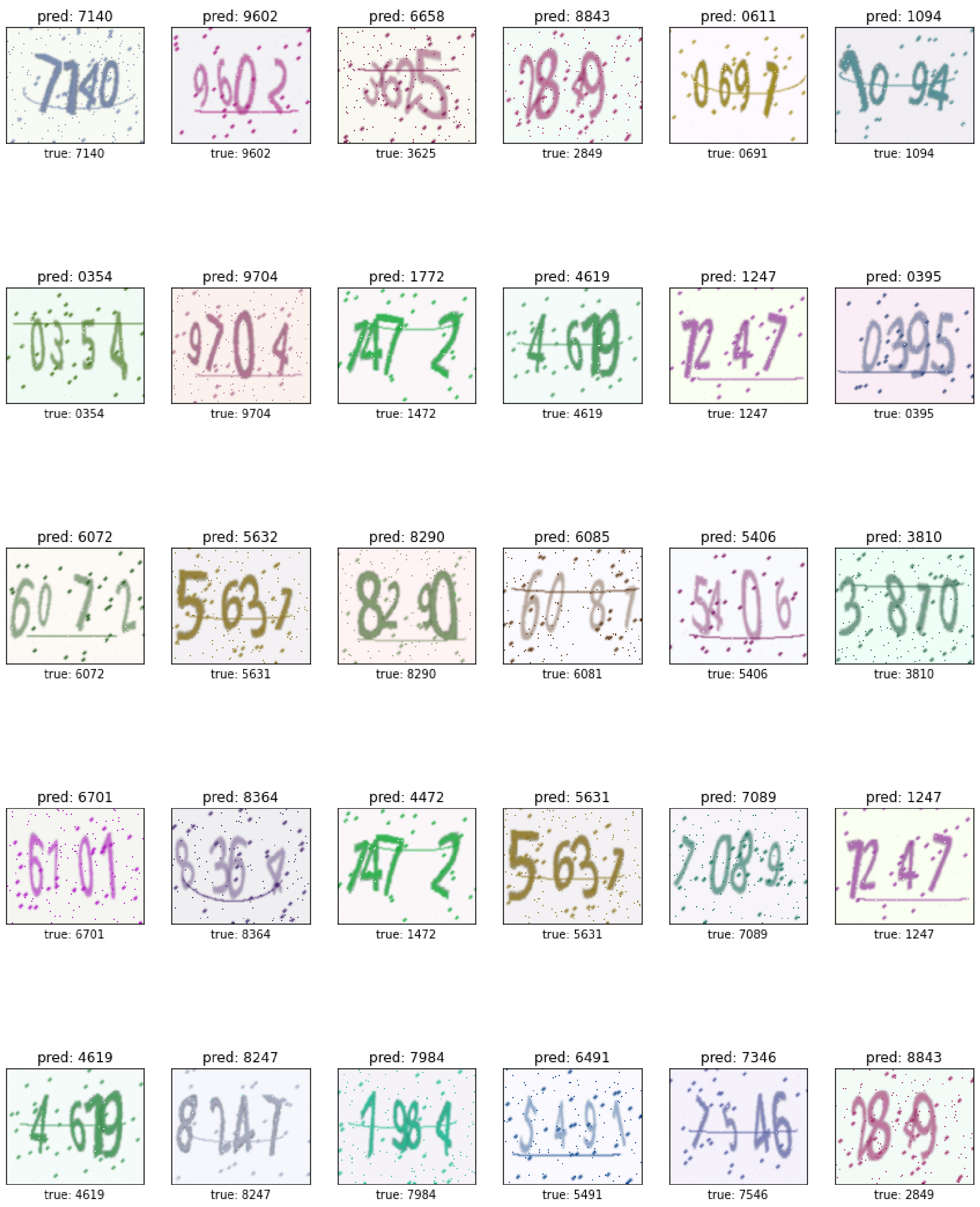
Perturbation Based on Font Color

In this attack, we pick the font color of the text of the CAPTCHA, and then we add this pixel color to CAPTCHA's random location. The amount of added noise in this attack isn't too much, so it is not easy to be detected with human eye. The success rate of this attack is not also high. Using this attack, we reduced the accuracy of the model to 73%.



Generated Adversarial CAPTCHA from Font Color attack

You can find our implementation on this repository



Results for Font Color attack

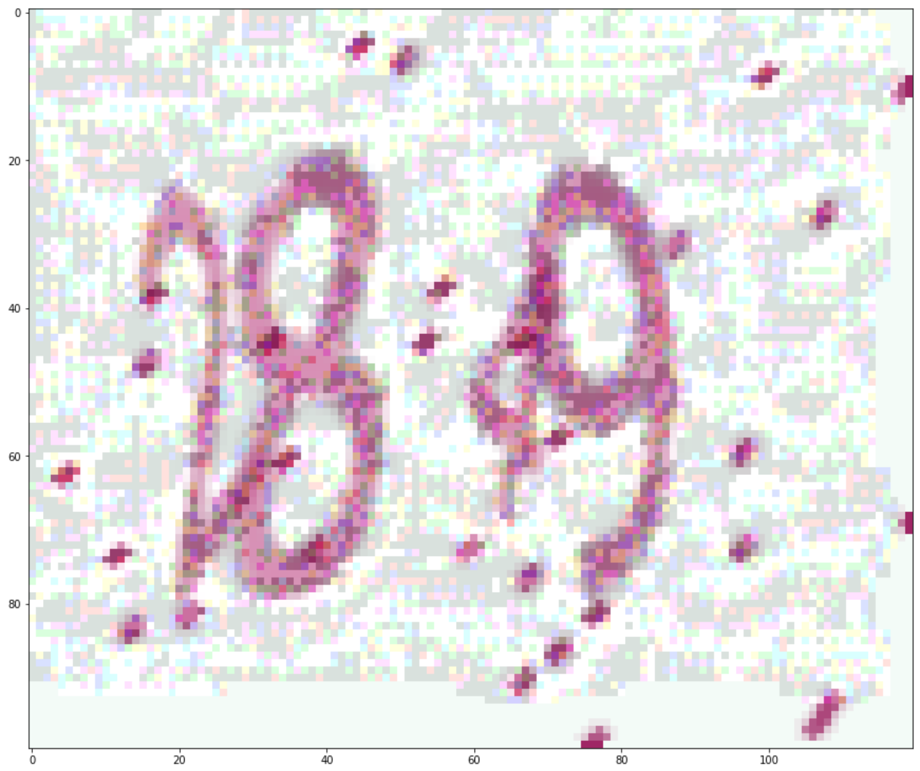
You can find our implementation on this repository

Perturbation with FGSM

The Fast Gradient Sign Method is the adversarial attack strategy we'll use (FGSM). This approach has this name because It's quick—the name says so. Computing the gradients of the loss, the sign of the gradient, and utilizing the sign to construct the adversarial image, we create the adversarial image. The algorithm work as follows:

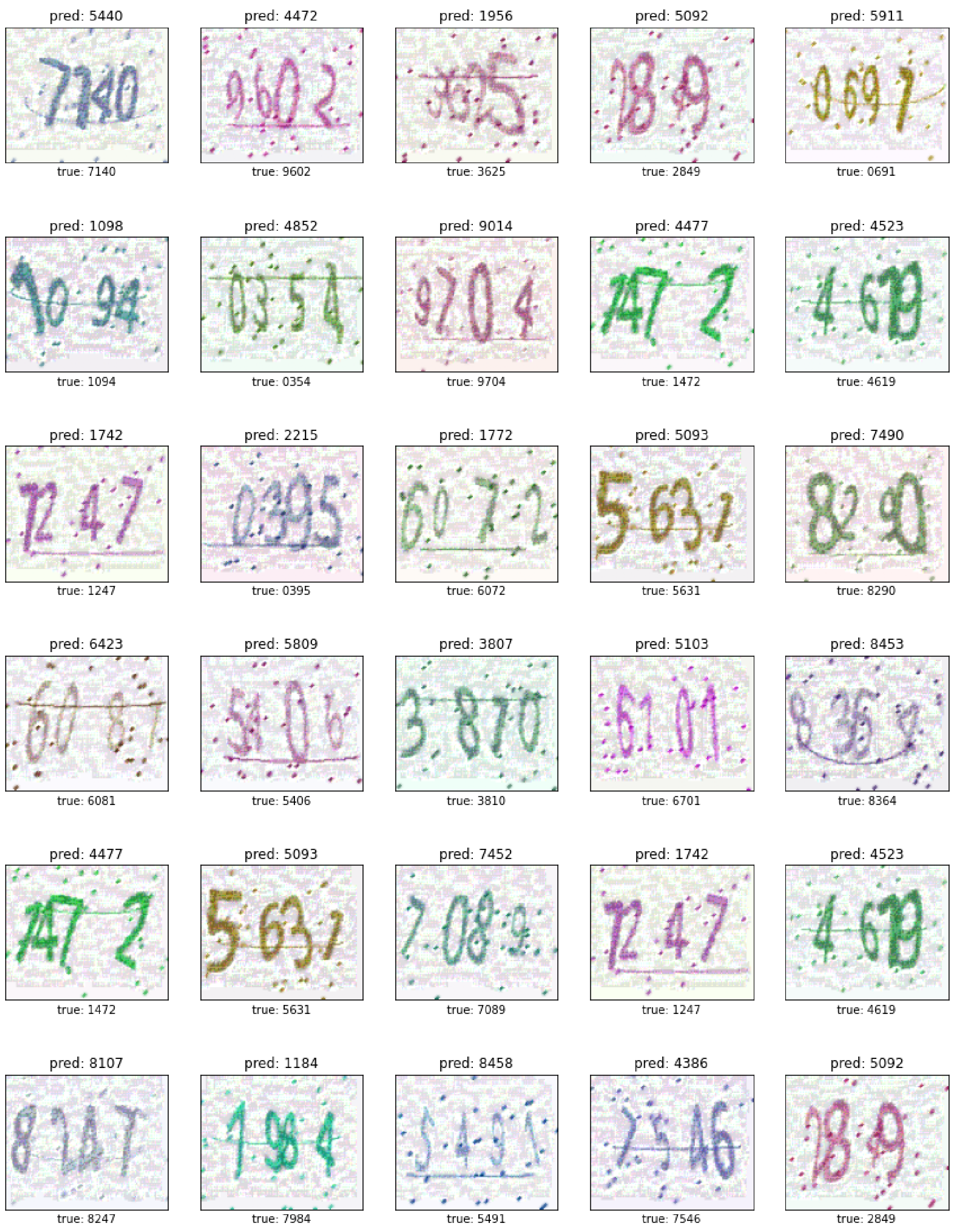
1. By adding a modest epsilon multiplier to the signed gradient. The objective is to make our gradient update large enough to incorrectly identify the input image while keeping it from being so large that the human eye can detect image manipulation.
2. After that, we add this negligible delta value to our image, which marginally modifies the image's pixel intensity values.
3. The image will seem significantly different despite the fact that these pixel changes won't be visible to the human eye, according to CNN, leading to misclassification.

Using this attack we were able to reduce the accuracy of the model to 0% which is a huge and unexpected result for us, this means adversarial CAPTCHA generated by FGSM can beat the model totally.



Generated Adversarial CAPTCHA from FGSM attack

You can find our implementation on this repository

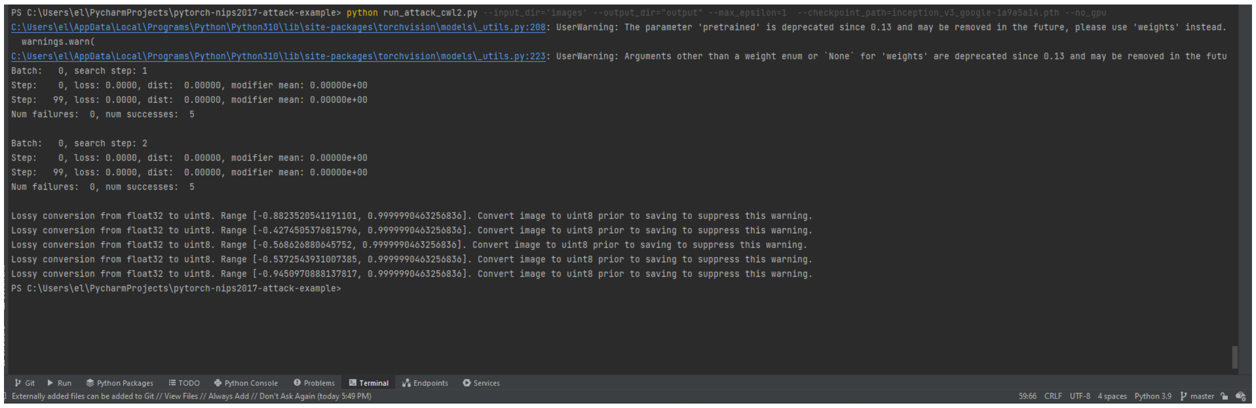


Results for FGSM attack

You can find our implementation on this repository

Carlini-Wagner

Carlini-Wagner Attack solves this optimization problem using advanced mathematical equations and proofs. The initial formula of the problem is MIN(D(x, x+ δ)): such that C(x + δ) = t and x + δ ∈ $[0,1]^n$, and this formula is later simplified to be solvable by Adam optimizer [17]. We also wrote an attack for Carlini-Wagner from https://github.com/carlini/nn_robust_attacks. Unfourtanetly, this attack failed due to our lack of resources. It is a time-consuming attack (it takes half a day to produce adversarial images according to the original paper). Our results for this attack were not promising and we decided not to include this attack in our report.



```
PS C:\Users\el\PycharmProjects\pytorch-nips2017-attack-example> python run_attack_cwl2.py --input_dir='images' --output_dir="output" --max_epsilon=1  --checkpoint_path=inception_v3_google-1a9a5a14.pth --no_gpu
C:\Users\el\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\el\AppData\Local\Programs\Python\Python310\lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the futu
Batch:   0, search step: 1
Step:    0, loss: 0.0000, dist:  0.00000, modifier mean: 0.00000e+00
Step:   99, loss: 0.0000, dist:  0.00000, modifier mean: 0.00000e+00
Num failures:  0, num successes:  5

Batch:   0, search step: 2
Step:    0, loss: 0.0000, dist:  0.00000, modifier mean: 0.00000e+00
Step:   99, loss: 0.0000, dist:  0.00000, modifier mean: 0.00000e+00
Num failures:  0, num successes:  5

Lossy conversion from float32 to uint8. Range [-0.8823520541191101, 0.9999990463256836]. Convert image to uint8 prior to saving to suppress this warning.
Lossy conversion from float32 to uint8. Range [-0.4274505376815796, 0.9999990463256836]. Convert image to uint8 prior to saving to suppress this warning.
Lossy conversion from float32 to uint8. Range [-0.568626880645752, 0.9999990463256836]. Convert image to uint8 prior to saving to suppress this warning.
Lossy conversion from float32 to uint8. Range [-0.5372543931007385, 0.9999990463256836]. Convert image to uint8 prior to saving to suppress this warning.
Lossy conversion from float32 to uint8. Range [-0.9450970888137817, 0.9999990463256836]. Convert image to uint8 prior to saving to suppress this warning.
PS C:\Users\el\PycharmProjects\pytorch-nips2017-attack-example>
```

You can find our implementation on this repository

Conclusion

In this project, we aimed to benchmark some attacks against CAPTCHA solvers. Some of these attacks were among the popular attacks in the literature of adversarial examples, and some were produced by ourselves. We have developed three successful evasion attacks against CAPTCHA solvers. Among our random perturbation attacks, black perturbations had a higher accuracy impact (lower accuracy) than other perturbations (font color and background color), the reason may be the color black is different than the colors our CNN model is trained on. The most successful and complicated one was the FGSM attack. It was more successful than our other attacks in two ways. First, the generated CAPTHCA's perturbations are not visible to the human eye, and it was hard for the CNN model to solve. So this takes us to the first definition of CAPTCHA "It should be easy for individuals to solve but challenging for computers to grasp a CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart)"

Evasion Attack Types	**Accuracy Rate of Model (Ground Truth 89%)**
Random Perturbation	53%
Perturbation Based on Background Color	80%
Perturbation Based on Font Color	73%
Perturbation with FGSM	0%

You can find our implementation on this repository

References

[1] Gao, H., Tang, M., Liu, Y., Zhang, P., & Liu, X. (2017). Research on the Security of Microsoft's Two-Layer Captcha. IEEE Transactions on Information Forensics and Security, 12(7), 1671-1685.

[2] Ahn, L., Blum, M., Hopper, N., & Langford, J. (2007). Using hard ai problems for security. In Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques (pp. 294-311).

[3] Bursztein, E., Martin, M., & Mitchell, J. (2011, October). Text-based CAPTCHA strengths and weaknesses. In Proceedings of the 18th ACM conference on Computer and communications security (pp. 125-138). ACM.

[4] Gao, H., Yan, J., Cao, F., Zhang, Z., Lei, L., Tang, M., ... & Li, J. (2016). A Simple Generic Attack on Text Captchas. In NDSS.

[5] Gao, H., Wang, X., Cao, F., Zhang, Z., Lei, L., Qi, J., & Liu, X. (2016). Robustness of text-based completely automated public turing test to tell computers and humans apart. IET Information Security, 10(1), 45-52.\

[6] Shao, R., Shi, Z., Yi, J., Chen, P., Hsiesh, C. (2021) Robust Text CAPTCHAs Using Adversarial Examples

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6):84–90, 2017.

You can find our implementation on this repository