Koç University

College of Engineering

COMP 430
Data Privacy and Security Fall'24
Final Project Report

Breaking and Building CAPTCHAs: A Dual Approach to CAPTCHA Security Using Machine Learning and Adversarial Techniques

Batu Orhun Gündüz

İdil Görgülü

Aylin Melek

Eren Can

Breaking and Building CAPTCHAs: A Dual Approach to CAPTCHA Security Using Machine Learning and Adversarial Techniques

İdil Görgülü Batu Orhun Gündüz Aylin Melek Eren Can

https://github.com/batuogkoc/comp430-project

Introduction

CAPTCHAs (Completely Automated Public Turing tests to tell Computers and Humans Apart) are defence mechanisms which are useful for differentiating humans and automated bots. These tests have become an integral part of online security and are widely being employed in online platforms to protect websites from malicious activities, including unauthorized data extraction, spam attacks and credential thefts [1]. However, due to the rapidly developing machine learning and artificial intelligence fields, the level of security of CAPTCHA systems has started to decrease and their effectiveness has been threatened. State-of-the-art models like Convolutional Neural Networks (CNN) and other deep learning architectures have shown considerably accurate results in recognizing and correctly guessing labels in CAPTCHAs. The performances of these models raise several concerns about whether CAPTCHAs are still effective in ensuring online security and showcase the need for more advanced mechanisms to prevent the model's successful predictions.

In our project, we explore the dual aspect of CAPTCHA security: breaking existing CAPTCHA systems and building more resilient ones. Our approach involves designing and training a CAPTCHA solver using a CNN-based architecture and publicly available datasets and evaluating the performance of our model. We then explore ways to confuse our own model with evasion attacks by using mainly two adversarial techniques: random patching and the Fast Gradient Sign Method (FGSM) [2]. By testing these methodologies, we aimed to assess the vulnerabilities of our solver model and propose the usage of the perturbed images as a tool for improving CAPTCHA security.

The findings from this study highlight potential implementations to increase the security of CAPTCHA systems by utilizing slightly perturbed images which are recognizable by humans but hard to detect for machine learning based adversarial models. This would be helpful to mitigate the risk posed by the current technological advancements. Through the dual approach we followed, we aim to contribute to the ongoing efforts in strengthening CAPTCHA systems against evolving cyber threats.

Methods and Techniques

The methods and techniques we utilized for the implementation of our project are divided into two primary objectives: developing a CAPTCHA solver and generating unsolvable CAPTCHAs.

Our CAPTCHA solver follows a deep learning approach to accurate alphanumeric characters within CAPTCHA images with high accuracy. Our initial model design approach had a limited depth with few layers, and thus resulted in with (~11%), which we found insufficient and decided that we are in need of a more expressive model architecture. To overcome our issue, we utilized ResNet-101, a pretrained deep residual learning model [3] and fine-tuned it such that it is best suitable for CAPTCHA encoding. We adapted the output layer to predict 5-digit CAPTCHA sequences, with each digit being drawn from 62 possible alphanumeric characters (26 uppercase, 26 lowercase, and 10 digits). The key steps we followed during our CAPTCHA solver implementation are as follows:

- **Dataset Preparation:** We combined the "Captcha Dataset" [4] and "Large Captcha Dataset" [5] and the two datasets combined produce >195k images. Due to most of the images being already pre-processed, no preprocessing was applied other than resizing and normalization. We than split the set for training, validation, and testing (80/10/10).
- **Model Architecture:** ResNet-101 extracts feature maps from input images and afterwards an average pooling layer reduces the feature map. The architecture is then followed by a fully connected layer generating output logits for digit prediction. The model architecture is explained in further detail through the upcoming sections of the report.
- **Training Process:** The model was trained on Koç University's VALAR HPC cluster using and for performance evaluation, cross-entropy loss was applied per digit. Training was halted early as accuracy plateaued.

Our final model achieved 85.35% CAPTCHA sequence accuracy on a test set aside from our combined dataset. Digit-base accuracy was considerably higher compared to overall accuracy. The exact accuracy results are showcased in the experimentation section.

In the second part of our project, we utilized various adversarial image perturbation techniques to examine the performance of our model architecture under evasion attacks. The methodology we followed through this section is as follows:

- **Random Patch Based Perturbation:** This technique included the generation of random patches of various shapes (e.g., rectangles, circles, polygons) with varying sizes, colors, transparency, and positions. Semi-transparent patches were blended onto the image using Image.alpha_composite. In order to further distort the image, a slight blur is also added to the image.
- **Fast Gradient Sign Method (FGSM):** In this method, the image gets perturbed based on model gradients. The loss gradient with respect to image pixels is calculated and input images are updated using the formula $x_{perturbed} = x + \epsilon \times sign(\nabla_x loss)$ [2]. After the application of this formula, pixel values get modified to maintain valid ranges.

- **Evaluation Metrics:** The effectiveness of both our CAPTCHA solver model and evasion attack implementations were evaluated using two distinct metrics: Top-1 digit accuracy and the accuracy of the entire 5-digit sequence. Top-1 digit accuracy corresponds to the accuracy of correctly identifying individual characters in sequences whereas the success rate in the entire sequence accuracy is decided by whether all 5 digits in a sequence are correctly identified or not.

The success of our adversarial attacks was quantified by observing the drops in accuracy for our perturbed image samples. The script evaluate_performance.py in our GitHub repository is designed for this specific purpose.

Experiments

- **CAPTCHA Solver Implementation**

Our CAPTCHA Solver is a machine-learning model that is capable of detecting the letters within a CAPTCHA image. Our models were trained on a combination of two datasets found through Kaggle, named “Captcha Dataset” [4] and “Large Captcha Dataset” [5]. These datasets consist of 82.3k and 113k images of 5-digit CAPTCHAs, respectively. Each digit is alphanumeric, taking 62 different values (26 lowercase, 26 uppercase, and 10 digits). Very minimal preprocessing was done to the datasets, with just a 224x224 rescaling and a normalization. The combined and shuffled datasets were split into train, validation and test splits with a ratio of 80/10/10.

Initially, a homegrown CNN-based network with very few layers were used to implement the captcha solver (initial_attempt.ipynb). However, the performance was very weak, with around 11% accuracy on average per digit. This was mainly caused due to the low expressivity of the model caused by its small size. After this initial failure, it was decided that fine-tuning a larger, pretrained model would be better suited for the job. For this, the ResNet-101 model’s backbone was chosen [3]. This backbone consists of a total of 101 convolutional, pooling and normalization layers. The famous residual connections which were introduced in this paper are utilised to allow optimal gradient flow for the very deep model. The backbone takes a 224x224 image with 3 color channels and outputs a 7x7x2048 feature map. We use 7x7 average pooing to reduce this feature map into a single 2048-dimensional feature vector. After this step, we have a single linear layer that turns this feature vector into 5x62 logits. Each digit is predicted separately, with 62 logits corresponding to each possible value of the digit. As the digit number is fixed within the dataset, this fixed number of digits were deemed sufficient. However, it is possible to replace the linear layer with a small LSTM which could predict a variable number of digits.

The model was trained on the VALAR High Performance Computing cluster of Koç University. The training was done on nodes possessing 64 GBs of RAM and a single NVIDIA A40 GPU with

48 GBs of VRAM. The training was planned to be ran for 20 epochs, however, was stopped early due to the plateauing accuracy. The Adam optimizer was used with a learning rate of 3e-3 and an exponential learning rate scheduler with gamma=0.82. Cross entropy loss was applied to each digit separately, and the results were averaged to compute the final loss. The resulting training graphs are provided in Figures 1,2 and 3. It should be noted that the top-1 accuracy values are the average accuracies of all of the digits, rather than the entire CAPTCHA.

Figure 1: Training Cross Entropy Loss of the Fine-Tuned Model

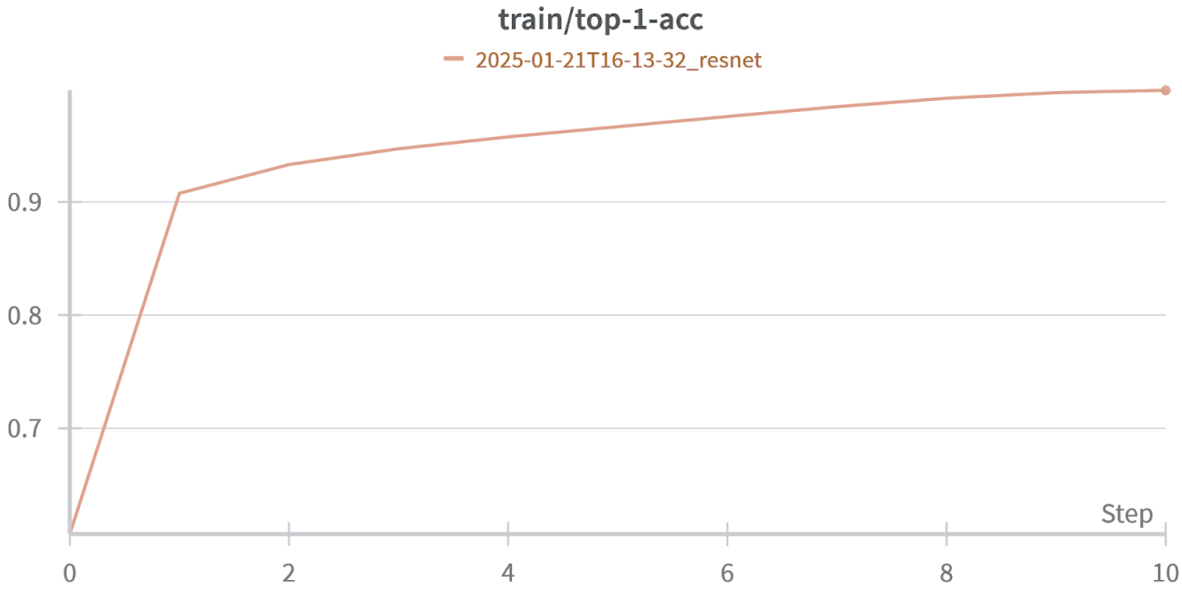

Figure 2: Training Top-1 Accuracy of the Fine-Tuned Model

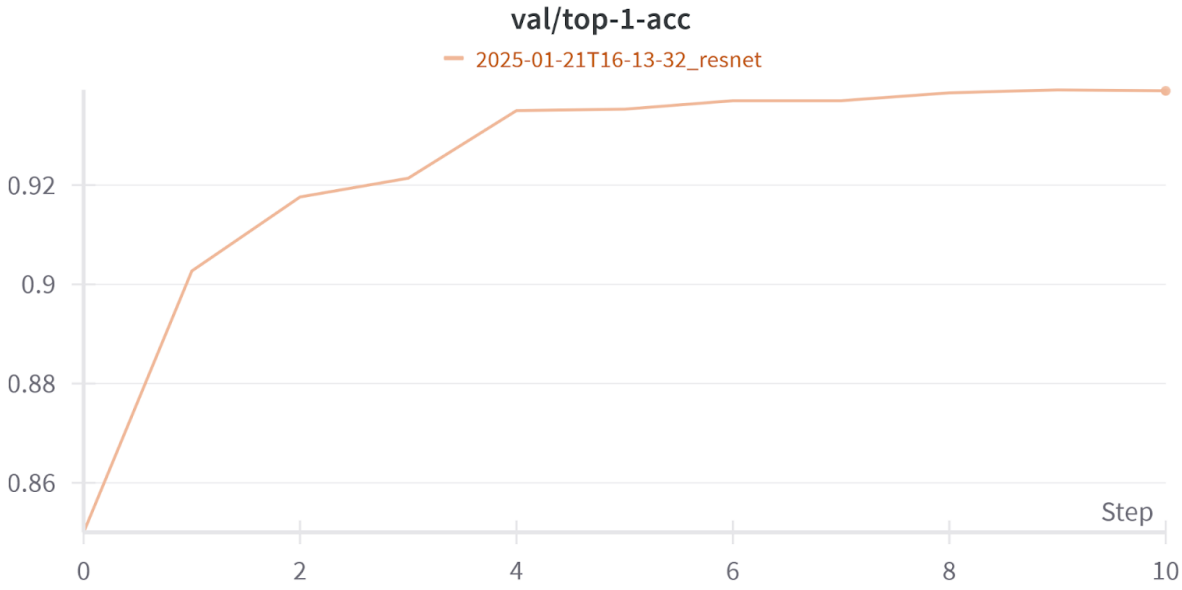

Figure 3: Validation Top-1 Accuracy of the Fine-Tuned Model

The final average Top-1 accuracies calculated per digit and the Top-1 accuracy for the entire CAPTCHA sequences on the three dataset splits are provided in Table 1.

Split	Digit Top-1 Accuracy	Entire Sequence Top-1 Accuracy
Training	0.9995	0.9976
Validation	0.9477	0.8506
Test	0.9483	0.8535

Table 1: The Digit and Sequence Accuracy Results in Splits of the Dataset

In the experimentation process, we acknowledge a slight overfitting to the training data. This is most likely due to the low variability in the data used. Regardless, 85% accuracy of the entire sequence on the test set was deemed a success, and the model was ready for use in the upcoming parts.

- **Unsolvable CAPTCHA generation**

1. Random Patch Based Perturbation

The first technique we utilized in order to generate CAPTCHAs which can't be identified by our model was applying random patches to images. The script apply_patch.py in our GitHub repository produces visually perturbed images by applying random shapes in random colors and sizes. These patches aim to disable the model from correctly processing and classifying the images and therefore making the CAPTCHAs resilient to machine learning models.

In "apply_patch.py", the function "apply_random_patch()" takes an image as input and outputs a perturbed version of that image with random patches. First, a random number of patches having various geometric shapes are generated for the input image. The image sized are modified so that they fit inside the image frame. The patches are created with random RGB values, semi-transparency with a random alpha value between 50-200, and a random shape type which can be rectangle, circle, ellipse, polygon, line and star. The locations of the images are determined in a way that they are centered near the middle of the image but this process is implemented with some randomness. The patch is overlaid on the image by using alpha blending which is done with Image.alpha_composite. After these steps, a random Gaussian blur is applied to the final image. This makes the image more distorted while maintaining human readability. Although generally not implemented during attacks including adversarial patches, this technique was found useful for increasing resilience against our CNN based CAPTCHA solver architecture.

This accuracy evaluation of this approach was achieved through generating 200 perturbed samples using apply_patch.py and measuring the accuracy of our model's prediction with evaluate_performance.by. The per-digit average was still relatively high due to the fact that the patches generally target a single character and sequence prediction usually fails by the wrong prediction of a single character instead of the entire sequence. The single character accuracy was measured as 82.40% whereas the overall accuracy of sequences decreased from approximately 85% to 42%. There is a more than 50% drop in accuracy, which is a considerable result.

Figure 4 displays example image samples from the generated adversarial images using random patch generation with the predicted labels. The predicted labels showcase the success of our evasion attack implementation. More adversarial image samples can be found in our GitHub repository under the folder "perturbed_images/patch". New sample images can also be generated by utilizing apply-patch.py.

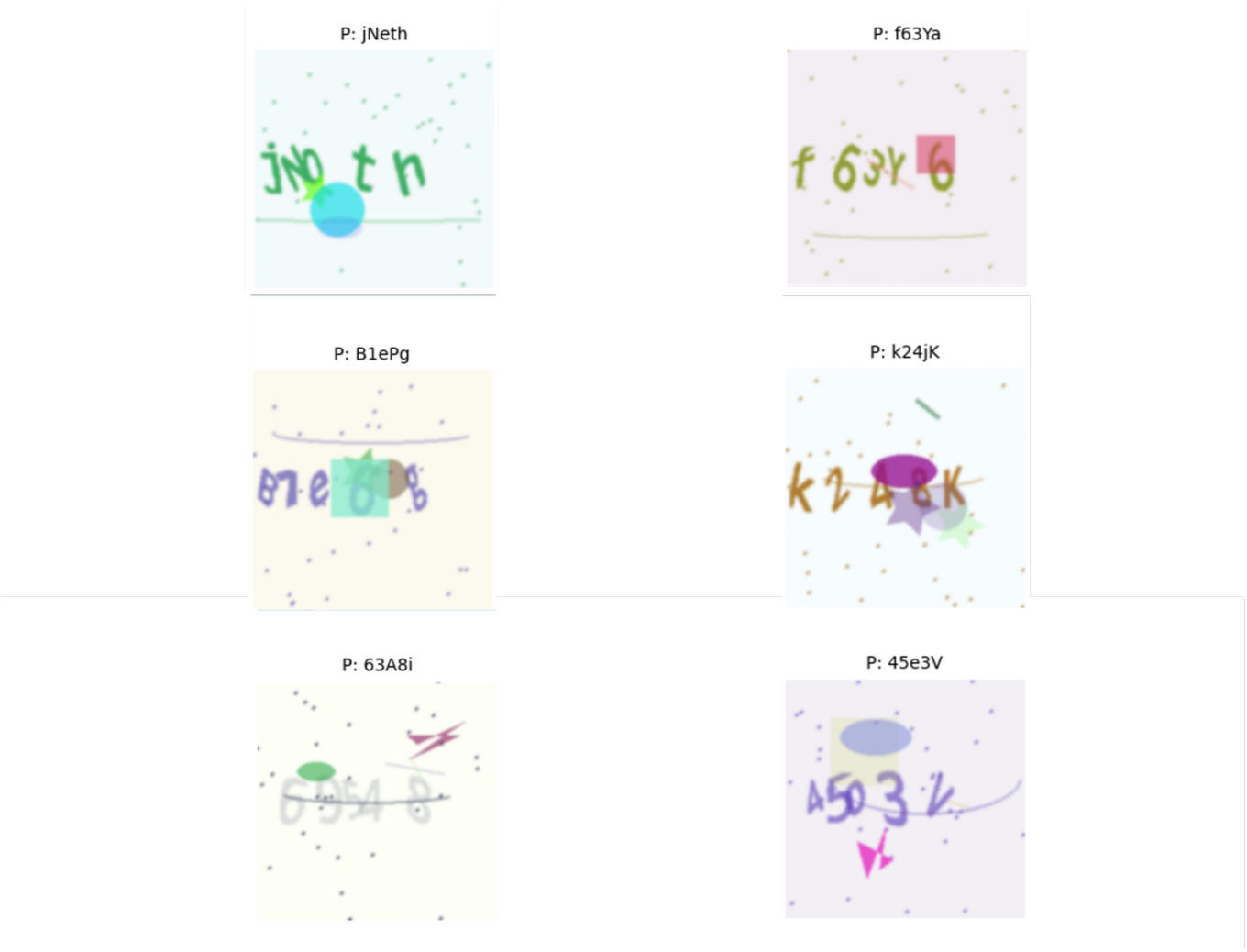

Figure 4: Model Prediction Results for CAPTCHAs Perturbed with Random Patch Generation

2. Fast Gradient Sign Method (FGSM)

This method aims to create adversarial examples for our trained model by applying perturbations to the input images using gradients which are calculated from image pixels. The script which generates adversarial images using FGSM in our GitHub repository is "fast_gradient_sign.py". The script is capable of evaluating any pre-trained CAPTCHA recognition model's ability to handle adversarial perturbations by loading the pre-trained model and the CAPTCHA dataset, then by creating adversarial examples by using FGSM and then comparing the model's predictions on the original and perturbed input images.

The first function, load_checkpoint(path), takes a file path as input and it is the path to the pre-trained model checkpoint, the output is the loaded model and the loss function. It first loads the

checkpoint and extracts the model state dictionary, then initializes a ResNetWrapper model with predefined parameters and finally loads the model's weights and returns the model and loss function.

The model evaluation function takes the pre-trained model and the input image tensor as input and outputs the flattened logits from the model which are reshaped to match the dimensions for characters and digits. The function first normalizes the input images to match the model's expected preprocessing, then passes the inputs through the model. Afterwards, extracts the hidden states and applies the model's final classification layer and finally reshapes the output to [batch_size, num_chars, num_digits].

The final function fast_gradient_sign(model,x ,y) takes the pre-trained model, input image tensor and ground truth labels as input and the output becomes the perturbed image tensor (x_perturbed). The function first creates a clone of the input image with gradients being enabled, then computes the predictions and the loss for the input. Afterwards, it back propagates the loss to calculate gradients with respect to the input and updates the input by using FGSM (x_perturbed = x + epsilon * sign(gradient), where epsilon is set to 0.1. Finally, it clamps the perturbed input to a valid pixel range [0,1].

$$\boldsymbol{x_{perturbed} = x + \epsilon \times sign(\nabla_x loss)}$$

Equation 1: Fast Gradient Sign Method

Although Fast Gradient Sign Method (FGSM) has a relatively simple implementation, it is considerably more successful in decreasing the overall accuracy of our model. Part of this effect is due to the fact that we can the parameter ϵ as we wish. The parameter ϵ is what controls the magnitude of the perturbation. The perturbed images generated with the Fast Gradient Sign Method are also more human-readable compared to the evasion attack images produced with random patch generation. The accuracy drop results of the FGSM can be seen as displayed in the Discussion Section of this report in Figure 6.

Figure 5 showcases how perturbed images generated with the Fast Gradient Sign Method change with varying ϵ values as well as how the model prediction changes as ϵ increases. Further visualizations can be generated through fast_gradient_sign.py in our GitHub repository.

Figure 5: FGSM Attack Results with Varying ϵ Values (GT: Ground Truth, P: Prediction)

- **Further Notes on Evasion Attacks**

Evasion attacks are adversarial "test-time" attacks designed to test the robustness of predictive models by providing them with crafted inputs to achieve the adversary's goal. As it was also explained in the previous parts of this report, in our project, we performed the evasion attacks by generating adversarial examples, which are modified versions of instances from our testing dataset, and using two distinct methods: patch attacks and FGSM (Fast Gradient Sign Method). Our main adversary goal was to fool the trained model by classifying the modified test instances falsely, while the CAPTCHA images remain human-readable so that it is harder to tell these pictures are adversaries and we could avoid detection. The modified adversarial examples were then fed into

the CAPTCHA-solving model to evaluate its performance and susceptibility under different attack scenarios.

The random patch-based attack is a black-box adversarial strategy. By adding random shape perturbations, this patch method disrupts the model's ability to interpret the test CAPTCHA images without needing any information about the model's architecture or parameters. This makes our attack case highly applicable to real-world scenarios where attackers may not have access to the model.

On the other hand, the FGSM attack is a white-box adversarial method. It exploits the gradient information of the trained model to generate adversarial examples that specifically target the model's weaknesses. Having access to the model's computations, this strategy shows how sophisticated attacks can be used to disrupt CAPTCHA-solving models.

Discussion

The evaluation of the evasion attacks shows key vulnerabilities in the CAPTCHA-solving model. For our random patch-based attack, the success rate was influenced by the size, shape, and transparency of the patches that were added to the images. Despite the simplicity of this technique, it proved effective in decreasing the model's test accuracy, particularly when the patch is larger or more opaque. However, it is also noted that successful attacks with large and opaque patch modifications on the test instances could be harder to detected by human eyes. Additionally, the random nature of the patches meant that some CAPTCHAs remained solvable. Thus, the degree of randomness in random patch generation affected our attack performance negatively. Therefore, the effectiveness of the random patch generation method could help us evade the predictive model with the trade-off between solvability and avoidance.

The FGSM attacks, by contrast, achieved a much higher success rate in evasion. This was the expected outcome due to the targeted nature of the attack, as the adversary modifications are explicitly designed to exploit the model's learned features. Although the CAPTCHA's remain human readable, the perturbations generated by FGSM often led to confident yet incorrect predictions with higher epsilon values which align with our attack goal. Figure 6 displays the drop in accuracy with increasing epsilon values in FGSM attacks.

Modifying all test instances with high epsilon FGSM modifications could lead to a significant decrease in the test accuracy score, this would make character detection harder and therefore the attack successful, however as epsilon increases the image is also harder to recognize with human eye. One policy we can try while implementing this attack could be using a higher FGSM epsilon value for a few instances, which would benefit us more fooling the model.

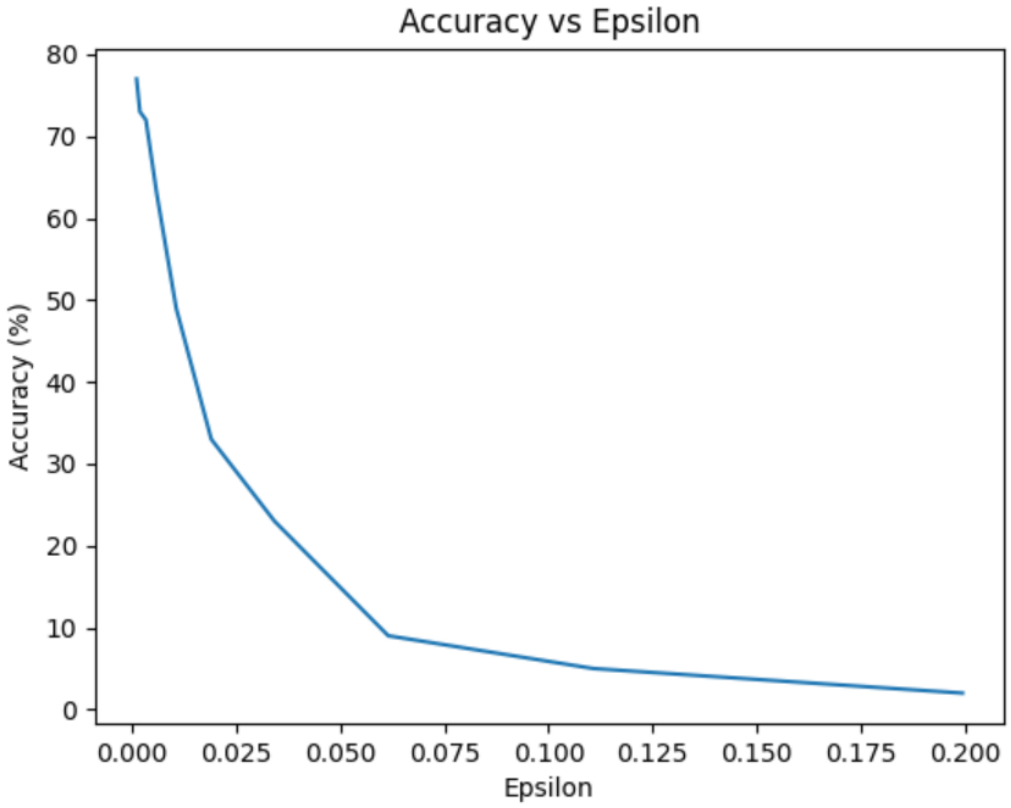

Figure 6: Accuracy% vs Epsilon (ϵ) for FGSM Attack Implementations on the Model

Conclusion

The experiments conducted in this project highlight the susceptibility of CAPTCHA-solving models to evasion attacks. While the patch-based black-box approach poses a straightforward and accessible threat, the FGSM white-box method demonstrates the extent to which an attacker with access to the model can compromise its functionality. These findings emphasize the importance of incorporating adversarial training and robust preprocessing techniques to mitigate such vulnerabilities.

In conclusion, the dual approach of breaking and building CAPTCHAs has provided valuable insights into the security challenges faced by CAPTCHA systems. Future work could explore additional adversarial techniques, including other gradient-based attacks or generative adversarial networks, as well as countermeasures such as adversarial training, input preprocessing, and model architecture enhancements to improve resilience against these threats.

Bibliography

[1] Z. Noury and M. Rezaei, “Deep-CAPTCHA: A Deep Learning based CAPTCHA SOLVER for vulnerability assessment,” arXiv.org, [Online]. Available: https://arxiv.org/abs/2006.08296

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” arXiv.org. Available: https://arxiv.org/abs/1412.6572

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” arXiv.org. Available: https://arxiv.org/abs/1512.03385

[4] P. Samadnejad, “Captcha Dataset”, Kaggle. Available: https://www.kaggle.com/datasets/parsasam/captcha-dataset/data

[5] A. Guna, “Large Captcha Dataset”, Kaggle. Available: https://www.kaggle.com/datasets/akashguna/large-captcha-dataset