Breaking Bots: Enhancing CAPTCHA Design with Neural Style Transfer

Rinnara Sangpisit
Stanford University
450 Jane Stanford Way, Stanford, CA 94305
rinnara@stanford.edu

Abstract

Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) systems were designed to protect online platforms from bot access. However, the rise of advanced computer vision models has rendered CAPTCHA systems less secure as modern models can solve CAPTCHAs with high accuracy. Neural style transfer has emerged as a cutting-edge method to boost the security of CAPTCHAs. By combining style representations of famous artwork with CAPTCHAs' content representations, this technique constructs augmented CAPTCHA images that remain legible to humans but challenging for machines to solve. In this project, I adopted neural style transfer to generate more robust CAPTCHAs and implemented a ResNet-9 model as the solver. My experimental results showed that the model's accuracy decreased by 12.1% on the full CAPTCHA level and 3.8% on the character level after neural style transfer, boosting the security of CAPTCHAs.

1. Introduction

Bots frequently overwhelm critical platforms with spam, fraudulent submissions, or malicious activity, including the spread of misinformation and harassment on social media. To mitigate this, many websites employ Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) systems to differentiate between humans and automated agents. For example, the U.S. visa application site uses text-based CAPTCHAs with distorted, multicolored characters, while platforms like Google employ image-click CAPTCHAs. However, the rise of advanced computer vision models has made CAPTCHA systems less secure. This project aims to understand the effectiveness of cutting-edge model architecture in solving traditional text-based CAPTCHAs and explore the role of neural style transfer in making these CAPTCHAs more resistant to bots.

To evaluate the effectiveness of modern computer vision architectures in solving text-based CAPTCHAs, I implemented a Residual Network (ResNet-9) model due to its strong performance and efficient architecture [10]. I generated train, validation, and test datasets of standard text-based CAPTCHAs modeled after those currently used on various websites to train and evaluate the model. There are 10,000 training data, 1,500 validation data, and 1,500 test data. The accuracy scores on this dataset serve as the baseline performance of the model.

Previous literature has demonstrated impressive results of neural style transfer in decreasing the effectiveness of automated CAPTCHA solvers. Inspired by these findings, I decided to implement neural style transfer on the CAPTCHA dataset to introduce greater complexity to the data [9]. The preprocessed CAPTCHA images serve as the content images and the starting point for target images. I used Vincent van Gogh's *The Starry Night* as the style image [18]. Performing neural style transfer allowed me to generate target images that resemble the structure of the preprocessed CAPTCHA images while incorporating style from *The Starry Night*. The resulting target images, referred to as stylized images, remain similarly legible to the human eye compared to non-stylized images.

After obtaining 10,000 stylized training data, 1,500 stylized validation data, and 1,500 stylized test data, I evaluated the baseline model's performance on the stylized test dataset to assess the model's robustness to adversarial transformations. Subsequently, I retrained the model on the stylized training images, with hyperparameter tuning based on stylized validation images, and evaluated its performance on the stylized test images. The results provided insight into the impact of neural style transfer on boosting the security of standard CAPTCHAs.

2. Related Work

CAPTCHA Breaking Efforts

Previous studies have employed various image recognition methods to develop CAPTCHA solvers with state-of-the-art performance. For example, Tian and Xiong introduced a generic solver combining unsupervised learn-

ing and representation learning and outperformed fully supervised models with similar architecture [16]. Ye et al. utilized a generative adversarial network-based approach to solve various text-based CAPTCHA schemes, achieving state-of-the-art results [19].

CAPTCHA Enhancements

Besides efforts to break CAPTCHAs, previous research also explored various strategies to enhance the security and robustness of text-based CAPTCHAs. Che et al. [3] emphasized the importance of data pre-processing by introducing a data selector that filters high-quality training data, followed by a data augmentor that applies four distinct image noise transformations. Their approach resulted in improved model generalization. My project builds upon this idea by adopting similar noise transformations in the data generation process.

Derea et al. [5] promoted model efficiency by using a segmentation technique that uses binary images to divide CAPTCHA images into components, allowing the use of fewer softmax output layers. This reduces the model's parameter count, improving training speed and storage efficiency. While my project does not directly adopt their segmentation method, it draws from their approach by using one softmax head for each character in the CAPTCHA for a more simplified model architecture that still maintains reliable performance.

Neural Style Transfer

Neural style transfer enables the generation of images with high perceptual complexity that combine the content of an arbitrary image with the appearance of well-known artworks [8] [9]. This algorithm iteratively optimizes a target image to match the desired content information and the artwork's style information [11]. Neural style transfer has led to a number of successful industrial applications. For example, Prisma, Ostagram, and Deep Forger offer services to transform users' images to match the appearance of famous work of art [1] [2].

Neural style transfer emerged as a cutting-edge technique to produce more robust CAPTCHAs that decrease the effectiveness of automated CAPTCHA solvers. Dinh et al. [6] combined adversarial example generation with neural style transfer to produce CAPTCHAs that are significantly more resilient to machine learning attacks while remaining readable to humans. Another paper by Cheng et al. [4] found that neural style transfer decreased the machine classification accuracy on image-based CAPTCHAs by 16%, while the difficulty for humans remains the same. These impressive findings inspired me to adopt neural style transfer to augment standard CAPTCHAs with the goal of enhancing their resistance to machine attacks.

3. Approach

3.1. Model Architecture

Due to its strong performance, Residual Networks (ResNet) have been widely adopted for image recognition tasks [10]. I adapted the core ResNet architecture to suit the CAPTCHA recognition task, as illustrated in Figure 1. Each input image first passes through an initial layer composed of a convolutional layer, batch normalization, and ReLU activation. This is followed by several ResNet v1 blocks, where each block includes two convolutional layers, batch normalization, ReLU, and a skip connection added before the final ReLU activation. Skip connections are applied only in projection blocks, where the input and output dimensions differ.

Unlike the original ResNet design that ends with global average pooling and a linear layer, I modified the prediction head to support sequence prediction for five-character alphanumeric strings. Specifically, I applied spatial average pooling independently across five spatial segments of the feature map, each representing one character. These features were then passed to five separate fully connected heads, each producing a character prediction, which were concatenated to form the final output.

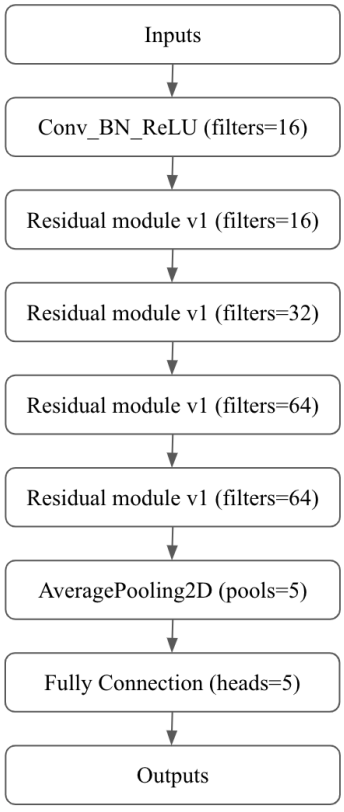

Figure 1: ResNet-9 Model Architecture

3.2. Neural Style Transfer

To enhance the robustness of the original CAPTCHA dataset, I explored neural style transfer as an approach to augment the dataset. Neural style transfer takes in a content image and a style image and aims to construct a target image that retains the structure of the content image and takes on the texture of the style image [9]. To do so, it obtains feature representations of the content, style, and target images by passing the images in a forward pass through the model to obtain feature representations after each layer of the model [8]. Let N_l denote the number of filters in layer l and M_l be the size of each filter i.e. height $\times$ width. These feature representations are used to calculate loss, which is used to compute gradients in order to construct the target image.

Content Loss. The content loss measures the content difference between the content and target images. Let $\vec{p}$ and $\vec{x}$ be the content and target images, and P^l and F^l be their respective feature representations at layer l in the model. I define the content loss as the squared-error loss between the two feature representations:

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2$$

Style Loss. To capture style representations, I need an additional feature space on top of the features that are outputted from each layer of the model. This feature space consists of correlations between the outputs from different filters. The feature correlations are given by the Gram matrix G^l, where G^l_{ij} is the inner product between the vectorized feature maps i and j in layer l:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

The style loss captures the style distance between the style and target images. It is defined as the mean squared loss between the Gram matrices of the target image and the style image. Let $\vec{a}$ and $\vec{x}$ be the style image and the target image, and A^l and G^l be their respective style representations (Gram matrices) in layer l. The style loss in layer l, represented by E_l can be computed as:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2$$

Each layer's contribution to the total style loss is set by weighting factors $w_i s$. Thus,

$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

In the scope of this project, I assumed that each layer contributes equally to the style loss. Therefore, $w_l = \frac{1}{|L|}$.

Total Loss. The total loss combines the content loss and the style loss:

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x})$$

where α and β are weighting factors of the content loss and the style loss respectively.

Optimization Loop. The ResNet-9 model has 4 ResNet v1 blocks, blocks 1-4, which comprise the 4 model layers. Each model layer contributes to either the content loss or style loss [9]. The initial layers of the model effectively recognize basic patterns of the input data, while later layers capture more subtle characteristics, allowing for a more precise approximation of the data [13] [15]. Therefore, I selected three style layers - block 1, block 2, and block 3 - for style reconstruction and one content layer - block 4 - for content reconstruction. Only the content layers contribute to the content loss, and only the style layer contributes to the style loss.

To synthesize a new image that matches the content representations of $\vec{p}$ and style representations of $\vec{a}$, I started with a base target image to optimize on. The base image could be a random white noise image or the content image itself. In this case, I use the content image as the base image because it allows for faster convergence in generating the synthesized image. The style image used in all stylized images is Vincent van Gogh's *The Starry Night* [18].

I performed repeated optimization loops on the target image. In each iteration, I used feature representations to compute the total loss and gradient to perform gradient descent. After repeating the optimization loop for 200 iterations, the best image, which minimizes the total loss, is saved as the synthesized image. Figure 2 illustrates this process.

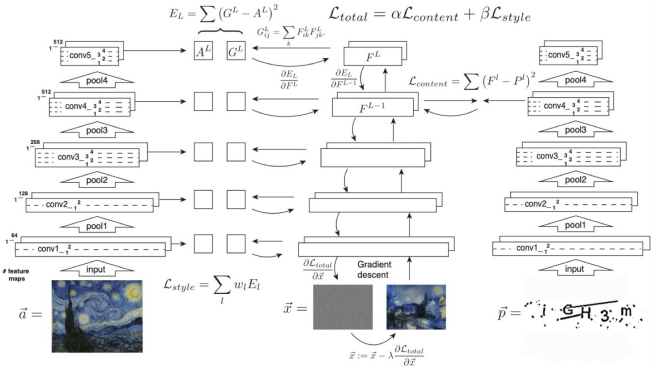

Figure 2: Neural style transfer diagram [9]

3.3. Feature Extraction

To implement Neural style transfer, feature representations of content and style images from each layer of the model are precomputed for an efficient optimization loop. Content representations are obtained by passing the content image to the model and registering the outputs after each layer using hooks. Content representations at each layer

have different shapes. For example, the output of the content image after layer 1 has shape (16, 64, 128). This means there are 16 feature maps, each with shape 64x128.

Style representations are obtained by passing the style image to the model and applying a feature space upon it. For example, after layer 1, I obtained 16 feature maps, each with shape 64x128. The feature correlations between these feature maps, given by the Gram matrix $G^1 \in \mathcal{R}^{16\times16}$, are the style representations at the 1^{st} layer.

The target representations are not precomputed but obtained and updated in each optimization loop. In each iteration, I performed a forward pass of the target image through the model to obtain its feature representations. Then, I calculated loss using the target representations and precomputed content and style representations. The loss is used to update the target image, and in turn, its feature representations for the next iteration.

4. Data

4.1. Data Generation

Based on the data generation method proposed by Kokoska et al. in [12], I implemented a data generation pipeline for text-based CAPTCHAs. Each image is a 70x250 pixel RGB canvas with a white background and contains a random five-character alphanumeric string. There are 62 character classes, composed of 26 uppercase English alphabets, 26 lowercase English alphabets, and 10 numeral digits. Characters are drawn with random font size, color, position jitter, and slight rotation to introduce randomness. To simulate real CAPTCHA distortions, I added noise and occlusions by incorporating pixels and colored lines at random positions. With this pipeline, I generated 10,000 training images, 1,500 validation images, and 1,500 test images.

Generating the CAPTCHA dataset myself ensures a consistent level of noise across all images. This consistency will also facilitate data augmentation by reducing unwanted variability that could interfere with its effectiveness. To assess the quality of the datasets, I inspected examples of generated CAPTCHAs to ensure that their labels can be appropriately deciphered with the human eye. Figures 3 and 4 show examples of generated CAPTCHAs that support the validity of the datasets.

Figure 3: CAPTCHA for “MUCtY”

Figure 4: CAPTCHA for “qBGoN”

4.2. Data Preprocessing

Following data generation, I implemented a data preprocessing pipeline consisting of three steps: 1. Grayscale conversion 2. Gaussian blurring 3. Binarization. Grayscale transformation converts a colored image to a single-channel image, which reduces the number of model parameters without significantly affecting its recognition accuracy [17]. Then, I applied Gaussian blurring to the grayscale image to reduce noise [14]. Finally, I employed binarization with an adaptive threshold, a technique shown to enhance the speed and performance of image recognition methods, to convert the blurred image to a binary image composed solely of black and white pixels [7].

These preprocessing steps help to reduce noise and standardize relevant features, leading to faster and more stable training. The training, validation, and test datasets generated earlier were preprocessed and saved in separate folders. To visualize the preprocessing steps, Figure 5 shows an original CAPTCHA image, while Figures 6, 7, and 8 show the image after applying grayscale conversion, Gaussian blurring, and binarization, respectively.

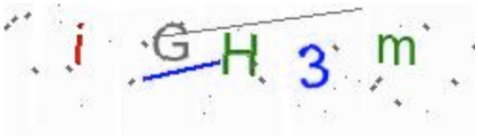

Figure 5: Original CAPTCHA image

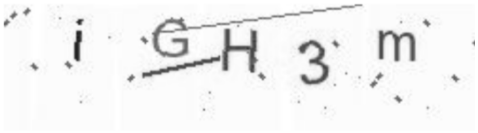

Figure 6: CAPTCHA after grayscale conversion

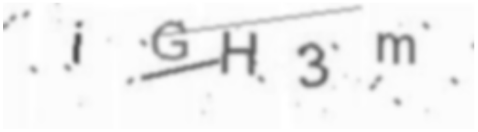

Figure 7: CAPTCHA after Gaussian blurring

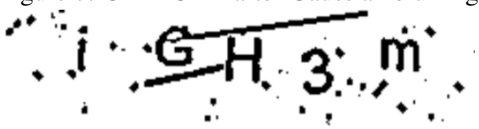

Figure 8: CAPTCHA after binarization

4.3. Style Transfer Dataset

To augment the standard CAPTCHA dataset, I performed neural style transfer on the preprocessed standard

CAPTCHAs. Neural style transfer involves extracting features from the content image (standard CAPTCHAs) and style image (*The Starry Night*) by passing them in a forward pass through the model. Therefore, when performing style transfer, I used preprocessed standard CAPTCHAs, which are compatible inputs to the model, and no longer require an additional preprocessing step afterwards.

There are 10,000 stylized training data, 1,500 stylized validation data, and 1,500 stylized test data. Stylized images incorporate the texture of *The Starry Night*, resulting in a textured background throughout the CAPTCHA image. To visualize this, Figures 9 and 10 show examples of CAPTCHA images after style transfer.

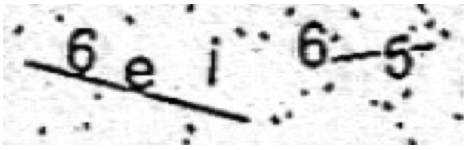

Figure 9: Stylized CAPTCHA for "6ei65"

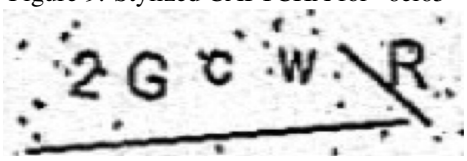

Figure 10: Stylized CAPTCHA for "2GcwR"

4.4. Feature Representations

To perform style transfer, feature representations of the content, style, and target images are used in loss optimization. Figure 11 shows an example of content representations after layer 1. Note that the output of a content image after layer 1 has shape (16, 64, 128). Thus, there are 16 feature maps, each with shape 64x128.

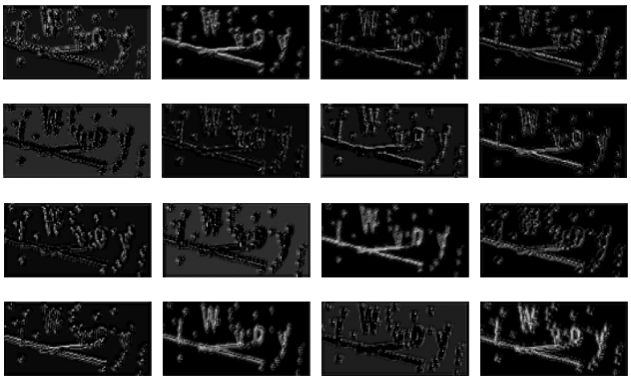

Figure 11: Content representations at the 1^{st} layer

5. Experiments

5.1. Evaluation Method

To evaluate performance, I implemented accuracy metrics at both the full CAPTCHA and character levels. Full CAPTCHA accuracy is computed as the proportion of CAPTCHAs where all five characters are correctly predicted, while character accuracy measures the proportion of correctly predicted characters across all characters in the sample.

$$\text{Full Accuracy} = \frac{\text{\# correctly predicted CAPTCHAs}}{\text{\# CAPTCHAs}}$$

$$\text{Character Accuracy} = \frac{\text{\# correctly predicted characters}}{\text{\# characters}}$$

Since partial predictions are possible, character accuracy is typically higher than full CAPTCHA accuracy. Due to its granular evaluation, character-level validation accuracy was used as the criterion for model selection during hyperparameter tuning.

5.2. Experimental Details

Baseline Model. To establish baseline performance, I first trained the ResNet-9 model on the standard CAPTCHA dataset using the cross-entropy loss function and performed inference with the provided accuracy metrics. The model was trained using the AdamW optimizer with a batch size of 64 and various combinations of learning rate and weight decay. Training was conducted for 15 epochs on an NVIDIA L4 GPU. I prevented overfitting on the training data by monitoring the training and validation loss graphs.

Neural Style Transfer Extensions. Next, I performed neural style transfer to generate stylized CAPTCHA datasets. To calculate total loss, I chose the respective weighting factors of the content and style losses to be $(\alpha, \beta) = (1,$ 1e-1). After experimenting with various combinations of loss weights, I believe this selection optimally preserves the legibility of CAPTCHA content while allowing for stylistic transfer visibility. The AdamW optimizer with a learning rate of 0.01 and weight decay of 5e-5 was used in the optimization loop to generate stylized images.

After obtaining the stylized datasets, I first performed inference on the optimized baseline model using the stylized CAPTCHAs to assess the model's robustness to adversarial perturbations. Then, I trained the model on the stylized CAPTCHA dataset using the total loss formula outlined in the Approach section, and evaluated performance with the provided accuracy metrics. Similar to the previous training runs, the model was trained using the AdamW optimizer with a batch size of 64 and various combinations of learning rate and weight decay. Training was conducted for 15 epochs on an NVIDIA L4 GPU. I monitored training and validation losses to prevent overfitting.

Hyperparameter Sweeping. I conducted hyperparameter sweeps during the training runs both before and

after style transfer extensions. During training, I varied the learning rate and weight decay used by the AdamW optimizer. After iterative refinement, I obtained the model with the highest character-level test accuracy out of all hyperparameter setups. This approach allowed me to experiment with different sets of parameters and select the optimal configurations for the model.

6. Results

Table 1 summarizes the baseline model performance on the standard CAPTCHA dataset. After hyperparameter tuning, the optimized model with a learning rate of 0.01 and weight decay of 5e-5 achieved a character-level accuracy of 95.60% and a full CAPTCHA accuracy of 80.00% on the test dataset.

Learning Rate	Weight Decay	Dev Char Accuracy	Dev Full Accuracy	Test Char Accuracy	Test Full Accuracy
0.001	1e-5	93.29%	70.40%	93.35%	69.93%
0.01	5e-5	95.12%	77.27%	**95.60**%	**80.00**%
0.02	1e-4	94.95%	77.2%	95.36%	79.27%
0.05	3e-4	94.61%	75.33%	94.79%	76.53%

Table 1. Baseline model performance on standard CAPTCHAs

Next, I evaluated the baseline model's performance on stylized CAPTCHAs to assess its ability to generalize to more adversarial CAPTCHA variations. According to Table 2, the model exhibits a drop in performance, achieving a character-level accuracy of 76.12% and a full CAPTCHA accuracy of 28.40% on the test dataset.

Learning Rate	Weight Decay	Test Char Accuracy	Test Full Accuracy
0.01	5e-5	76.12%	28.40%

Table 2. Baseline model performance on stylized CAPTCHAs

After training the model on stylized CAPTCHAs, I performed inference to evaluate performance on the stylized dataset. According to Table 3, the optimized model with a learning rate of 0.01 and weight decay of 5e-5 achieves a character-level accuracy of 91.81% and a full CAPTCHA accuracy of 67.93% on the test dataset.

Based on these results, the model performance on decoding stylized CAPTCHAs dropped by 3.8% on the character level and a striking 12.1% on the full CAPTCHA level compared to standard CAPTCHAs. The lower performance supports the hypothesis that neural style transfer is effective at making CAPTCHA architecture more secure towards bot attacks.

Learning Rate	Weight Decay	Dev Char Accuracy	Dev Full Accuracy	Test Char Accuracy	Test Full Accuracy
0.001	1e-5	92.28%	67.07%	90.63%	64.93%
0.005	3e-5	92.09%	66.67%	91.08%	64.93%
0.01	5e-5	92.79%	67.97%	**91.81**%	**67.93**%
0.05	3e-5	92.24%	66.27%	91.52%	67.20%

Table 3. Model performance on stylized CAPTCHAs

7. Analysis

7.1. Model Input Evaluation

To gain insight into why neural style transfer produces CAPTCHAs that are more robust against model-based decoding than standard CAPTCHAs, I inspected examples of both types of CAPTCHAs from the train loader.

In Figure 12, I observed that the characters of the standard CAPTCHA appear mostly clearly with coarse-grained noise scattered throughout the image. On the other hand, in Figure 13, I observed that the textured background of the stylized CAPTCHA creates fine-grained noise scattered throughout the image. This noise interferes with characters, making it more challenging to decode them. Even though stylized CAPTCHAs remain readable to humans, their fine-grained noise likely contributed to the model's compromised quantitative performance.

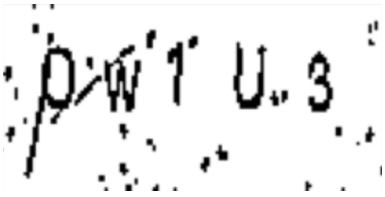

Figure 12: Standard CAPTCHA from train loader

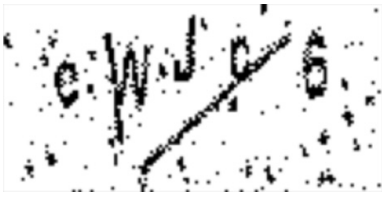

Figure 13: Stylized CAPTCHA from train loader

7.2. Error Evaluation

To understand the factors that influence model predictions, I inspected examples of stylized CAPTCHAs that the model mispredicted. In Figure 14, I observed that the fine-grained noise interferes with the fourth character 'y', making it appear as 'v' and causing the model to mispredict. In Figure 15, the fine-grained noise significantly interferes with the characters, causing mispredictions of multiple

characters in the image. These findings support the quantitative results by showing that fine-grained noise produced by style transfer undermines the model's ability to interpret CAPTCHAs.

Figure 14: Mispredicted stylized CAPTCHA (1)

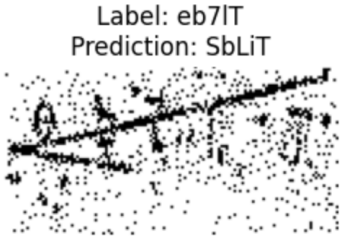

Figure 15: Mispredicted stylized CAPTCHA (2)

7.3. Saliency Maps

To visualize parts of the CAPTCHA image that most influence the model's prediction, I generated saliency maps to analyze the model's decision process. Figures 16 and 17 show the saliency maps for character positions 1 and 3 in the CAPTCHA image. The saliency maps correctly highlight relevant areas for character predictions, testifying to the model's interpretability.

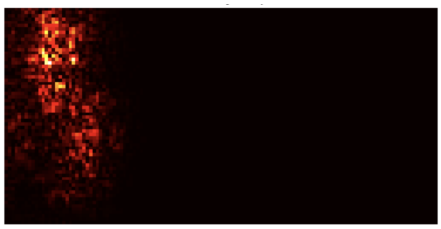

Figure 16: Saliency map for 1^{st} character

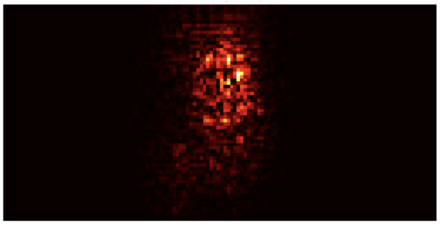

Figure 17: Saliency map for 3^{rd} character

8. Conclusion

The results of this project demonstrated that neural style transfer plays a role in enhancing CAPTCHA security, evidenced by a respective 12.1% and 3.8% decrease in accuracy at the full CAPTCHA and character level for the ResNet-9 automated solver. These findings support previous literature that showed degraded model performance after neural style transfer. Qualitative evaluation revealed that neural style transfer introduces fine-grained noise throughout CAPTCHA images, which interferes with the characters, leading to incorrect model predictions.

Future studies can build on this project by exploring the promising role of neural style transfer in other security-based applications. Although style transfer is primarily used for artistic applications today, this project highlights its potential role in security domains in the future. Furthermore, given the compelling results of combining neural style transfer with adversarial examples [6], future research can also explore the synergies between style transfer and other CAPTCHA augmentation techniques, which may uncover enhanced methods for strengthening CAPTCHA security.

References

[1] Ostagram. `https://www.ostagram.me/static_pages/lenta?last_days=1000&locale=en`.

[2] Prisma labs. `https://prisma-ai.com/`.

[3] A. Che, Y. Liu, H. Xiao, H. Wang, K. Zhang, and H.-N. Dai. Augmented data selector to initiate text-based captcha attack. *Security and Communication Networks*, 2021:1–9, 06 2021.

[4] Z. Cheng, H. Gao, Z. Liu, H. Wu, Y. Zi, and G. Pei. Image-based captchas based on neural style transfer. *The Institution of Engineering and Technology*, 2019.

[5] Z. Derea, B. Zou, A. A. Al-Shargabi, A. Thobhani, and A. Abdussalam. Deep learning based captcha recognition network with grouping strategy. *Sensors*, 23(23), 2023.

[6] N. Dinh, K. Tran-Trung, and V. Truong Hoang. Augment captcha security using adversarial examples with neural style transfer. *IEEE Access*, 11:1–1, 01 2023.

[7] A. Dionysiou and E. Athanasopoulos. Sok: Machine vs. machine – a systematic classification of automated machine learning-based captcha solvers. *Computers Security*, 97:101947, 2020.

[8] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style, 2015.

[9] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[11] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song. Neural style transfer: A review. *IEEE Transactions on Visualization and Computer Graphics*, 26(11):3365–3385, 2020.

[12] J. Kokoska, A. Mengi, and Y. Mao. Machine learning captcha solver, 2022.

[13] L. Roach, G.-M. Rignanese, A. Erriguible, and C. Aymonier. Applications of machine learning in supercritical fluids re-

search. *The Journal of Supercritical Fluids*, 202:106051, 2023.

[14] R. Roadster. The simplest way(and probably one of the best) to solve a text captcha, 2024.

[15] M. M. Taye. Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions. *Computation*, 11(3), 2023.

[16] S. Tian and T. Xiong. A generic solver combining unsupervised learning and representation learning for breaking text-based captchas. In *Proceedings of The Web Conference 2020*, WWW '20, page 860–871, New York, NY, USA, 2020. Association for Computing Machinery.

[17] X. Wan, J. Johari, and F. A. Ruslan. Adaptive captcha: A crnn-based text captcha solver with adaptive fusion filter networks. *Applied Sciences*, 14(12), 2024.

[18] Wikipedia. The Starry Night — wikipedia. `https://en.wikipedia.org/wiki/The_Starry_Night`, 2025. [Online; accessed 04-June-2025].

[19] G. Ye, Z. Tang, D. Fang, Z. Zhu, Y. Feng, P. Xu, X. Chen, and Z. Wang. Yet another text captcha solver: A generative adversarial network based approach. CCS '18, page 332–348, New York, NY, USA, 2018. Association for Computing Machinery.