Captcha Cracker: Building a Convolutional Neural Network for Security Image Decoding

Cyrus Bruce

Robert Morris University, clbst275@mail.rmu.edu

*Abstract* – Text-based CAPTCHAs were among the earliest mechanisms deployed on websites to prevent unwanted spam from automated bots. However, their design has long been criticized for inherent flaws, prompting the development of alternative CAPTCHA types such as image selection, puzzle-solving, and audio-based prompts. Convolutional Neural Networks (CNNs), a powerful deep learning approach commonly used in Optical Character Recognition (OCR), have emerged as an effective method of image classification— more specifically a CNN model optimized for interpreting complex or distorted text from images like text-based CAPTCHAs. While prior research has demonstrated that convolutional neural networks can compromise the integrity of text-based CAPTCHA systems, this study aims to evaluate the practical reproducibility and effectiveness of such techniques on a large-scale, noise-heavy CAPTCHA dataset. Rather than simply reaffirming the vulnerability of text-based CAPTCHAs, the research systematically analyzes the impact of various preprocessing techniques and model architectures— ranging from basic custom CNNs to pre-built proven models like ResNet50— on character decoding accuracy. The goal is to validate earlier findings under new constraints and to assess the real-world limitations and feasibility of automated CAPTCHA solvers. Experimental results with the Deep Learning model demonstrate that, with appropriate preprocessing, a ResNet50-based model with minimal preprocessing achieved up to 63.44% accuracy in fully decoding five-character CAPTCHAs, confirming both the feasibility of such attacks and the critical role of image preparation in model performance.

1. Introduction

To begin, it is important to emphasize the importance of CAPTCHAs and their implementation as a security practice. A CAPTCHA— an acronym which stands for *Completely Automated Public Turing test to tell Computers and Humans Apart*— uses the differences between the way humans and computer systems process information as inspiration to create a seemingly simple test that can theoretically only be solved by a human. The most basic style of CAPTCHAs are text-based and have a layout involving a series of randomized alphanumeric characters, consisting of both capitalized and lowercased letters as well as the numbers 1 through 9, with a prompt to ask the user to identify all letters visualized.

The real ingenuity is where the visualization of the characters occurs, there is a range of different noise included in the image, making it harder to visualize. The actual noise itself also varies, randomizing from a list of different tactics­­­­ that both alters the text of the image to attempt to confuse Optical Character Recognition (OCR) algorithms as to what character it is identifying and adds additional background visuals in the image to make the location of each letter more unpredictable. Some text distortion can be described by— warping text with curves around an axis or shifts in placement, skewing individual letters to alter flow of the text, and differing fonts and sizes. The background of a CAPTCHA will include randomized dots, lines, and other patterns of differing sizes to disguise the locations and shapes of each character from OCR systems.

To a human user, this noise is a simple barrier and in most cases the user will solve the CAPTCHA and get back onto their site in seconds; However, to a computer program or script masking itself as a legitimate user, this noise tends to be quite an annoyance when analyzing the image. The noise acts as an artificial barrier—one that capitalizes on the weaknesses of machine vision—to preserve system integrity by complicating automated OCR systems.

A Traditionally purposed OCR system would typically be trained on clean, well-aligned text in standardized fonts and consistent environments such as scanned documents or printed forms. As a result, when exposed to CAPTCHAs that incorporate distortions and background noise, these systems experience a dramatic drop in recognition accuracy. Since they have not been trained to process noisy, misaligned, or obscured characters, the added interference hinders their ability to generalize, often leading to misclassification or complete failure in decoding the text.

Given the functionality of a CAPTCHA and its purpose to distinguish between human users and automated programs, a common integration of these mechanisms are on the public facing sides of a network, providing a frontline defense mechanism protecting against a wide range of automated attacks. CAPTCHAs specialize in mitigating automated attacks on the network such as: *Automated Account Creation, Brute-Force Attacks, Denial-of-Service (DoS),* as well as protecting against *spam*. These protections are especially critical in environments where data security, platform integrity, and user experience are at risk. **Automated Account Creation** involves bad actors utilizing a type of bot or automated software to streamline the phishing process. Without CAPTCHA implementation on an account registration page, a single automated system could create thousands of fraudulent accounts all posing as real people to go around requesting personal details or spreading malicious links around. CAPTCHAs aid in preventing **Brute-force password cracking** by asking the potential computer / bad actor to fulfill a CAPTCHA request after so many false attempts to waste their time. Because a brute-force attack specializes in guessing a large quantity of passwords over a more thought out dictionary-attack, forcing a CAPTCHA can provide rate-limiting and make the computational toll of running a brute-force attack more burdensome than the rate of success. A **Denial of Service (DoS)** attack will overwhelm a target network with traffic and requests, making it slow down or become completely unavailable for legitimate users to access. This disruption of data integrity is not solely fixed with the inclusion of CAPTCHAs however their implementation can reduce traffic of requests and deter bot within the botnet. Many sites choose how to give out CAPTCHA requests by recording user traffic patterns to determine unusual and potentially malicious activity across the network. Fake reviews and unwanted advertisements are prohibited by the FTC as they are considered deceptive advertising practices, CAPTCHAs can combat malicious automated **Spam**, such as submission abuse under business reviews, comment sections, and other feedback. Without CAPTCHAs, such vectors me ntioned above could be exploited to spread malware, mislead users, or launch secondary phishing campaigns. While not foolproof, CAPTCHAs act as a friction layer — introducing just enough resistance to deter low-effort, automated attacks that could otherwise scale with minimal cost.

Despite their once widespread adoption within most networks and company standard security measures, the effectiveness of text-based CAPTCHAs as a foolproof solution towards automated attacks against the network is increasingly called into question— particularly as advancements in machine learning, and more specifically Optical Character Recognition models, through deep learning continue to close the gap between human perception and automated interpretation. The defenses that CAPTCHAs rely on, such as randomized distortion, misalignment, and background noise, are no longer sufficient barriers against well-trained, noise-resilient convolutional neural networks. While a traditional OCR system would struggle against a text-based CAPTCHA, deep learning models, most prominently CNNs, can preprocess captcha images to counter noise implementation and further train from the noise to adapt from it and increase its accuracy greatly. This project aims to demonstrate that text-based CAPTCHAs are not only decipherable using OCR techniques but that they can be interpreted consistently through a streamlined image classification model. By preprocessing the noise intended to distort the image and then training a Convolutional Neural Network to interpret the underlying patterns behind each alphanumeric character, this project replicates the exact kind of automated system that CAPTCHAs originally anticipated to resist. In doing so this project underscores and exploits a critical vulnerability known to discredit text-based CAPTCHAs— a deep learning model composed of enough stacked layers of convolutions, activations, and pooling operations that progressively learn to recognize abstract visual patterns, to train to both understand distorted character shapes as well as disregard irrelevant features like background noise.

1. Literature Review

This section explores my findings of relevant prior academic work relevant to CAPTCHA recognition, I chose the study *Deep-CAPTCHA: a Deep Learning Based CAPTCHA Solver for Vulnerability Assessment by Zahra Noury and Mahdi Rezaei (2020)* [1] primarily as a conceptual foundation due to the similar purpose and functionality the authors outlined for their work. While I did not aim to replicate any existing solution directly, the Deep-CAPTCHA model informed several key decisions in the development of my own architecture, including the selection of convolutional neural networks, preprocessing approaches, and multi-character output modeling. In the subsections below, I analyze and reflect on five core ideas drawn from their research and how they compared to or informed the methodology I ultimately used.

**The Use Convolutional Neural Networks in CAPTCHA Recognition**

*Noury and Rezaei* (2020) emphasize the effectiveness of CNNs in recognizing distorted, spatially complex patterns that are common in CAPTCHA images used in the model. CNNs can hierarchically detect edges and shape through convolutional filters. This research, while also concurring with earlier studies (e.g., Garg & Pollett; Stark et al.), demonstrates that even relatively shallow CNNs outperform traditional machine learning methods or RNNs in terms of speed and accuracy.

Informed by this, I implemented a comparative approach involving a custom shallow CNN and a deeper, pretrained architecture— that being Microsoft Vision Model ResNet50 [3]. ResNet50, designed for deeper feature extraction and improved generalization, was particularly well-suited to handling the greater variability present in the CAPTCHA dataset I used.

**Addressing Line-Based Noise in CAPTCHA Images**

A consistent challenge within most related research is the disruption caused by obfuscating lines across the CAPTCHA text. The solution proposed by *Noury and Rezaei* refer to morphological operations, specifically erosion and dilation to cleanly remove these artifacts before classification

Inspired by this approach, I attempted similar techniques using various OpenCV morphological filters across both python and C#. Although these operations showed some promise, the results were inconsistent across the dataset, requiring further refinement and tuning.

**Evaluating and Applying Additional Preprocessing Techniques**

Beyond the morphological operations, other preprocessing techniques discussed by *Noury and Rezaei* include Grayscale Conversion, Gaussian Blur, and Median Filtering. According to their analysis, converting to grayscale has little effect on accuracy, but it reduced data size and simplified model input. They also implemented Gaussian and median filters to smooth noise and reduce pixel-level distortion without damaging character edges.

In my own work, grayscale conversion was one of the first preprocessing steps to reduce computational load. While it may not have improved accuracy directly, it helped standardize input data and reduced computational overhead by limiting the color channels to one. Gaussian blur was used to address general noise, and adaptive thresholding played a critical role in separating characters from the background. Adaptive thresholding, in particular, was key to separating foreground text from noisy backgrounds under uneven lighting conditions. Although I tested median filtering, it proved to be too aggressive for the fine-detail preservation required in my dataset.

**Architectural Comparison: Deep-CAPTCHA vs. ResNet50**

The Deep-CAPTCHA model consisted of three convolutional blocks (with 32, 48, and 64 filters), followed by a dense layer and multiple softmax outputs—one for each character position. They explicitly compared two encoding schemes: a sigmoid-based multi-label model and a parallel softmax configuration. Their findings displayed that the softmax approach was superior for multi-character classification.

My architecture adopted a similar multi-head softmax strategy; however, I substituted their custom CNN with the deeper ResNet50 architecture pretrained on ImageNet [3]. This change allowed for better generalization across varied CAPTCHA styles while minimizing the need for manual tuning. While Noury and Rezaei reported near-perfect accuracy (~98.9%) on their synthetically generated numeric CAPTCHAs, my results were lower due to factors including increased character variety (letters and digits), real-world noise, and broader visual inconsistency.

**Additional Insights and Considerations**

An important contribution of Noury and Rezaei’s work was their vulnerability assessment, where they identified specific digits (e.g., 3, 8, 9) and rotated characters as common sources of misclassification. Building on this, I analyzed character-specific prediction errors in my model after testing to identify patterns of misclassification, which yielded similar insights.

A notable distinction between our projects is the dataset: whereas they used a custom-generated dataset of 500,000 images, I worked with a sample size of 100,000 realistically generated CAPTCHA images sourced from Kaggle. This presented the model with more variation and lower visual consistency. This difference may partially explain the performance gap and further justifies the use of a deeper ResNet50 model to compensate for greater visual complexity.

A collection of numbers and letters

AI-generated content may be incorrect.

Fig. 1: Sample of Generated CAPTCHA data

1. Dataset Description

The dataset used to build the Captcha Cracker model was sourced from Kaggle, consisting of over 113,000 randomly generated CAPTCHA images. *Figure 1* displays a random sample of potential CAPTCHA images used in the dataset, generated using Captcha Library for PHP. Each image contains a visualization of five alphanumeric characters in a colored format. To define random, both the text on each image is a randomized sequence of characters; the distortions are also not prefixed but rather randomly generated obfuscations including colored lines, arcs, overlapping shapes, and varying background intensities are present throughout the dataset. This randomness adds a substantial challenge to the recognition task, each image oftentimes containing overlapping artifacts of noise that blend into the characters themselves.

A subset of 100,000 random images were selected from the source, extracting the labels given from the image filename and converted to a structured format more suitable for preprocessing and the deep learning model.

1. Methodology

The methodological framework used to construct the deep learning model. Through trial, analysis, and error there were a lot of relevant and potential techniques for preprocessing the data; Each step in the process was carefully chosen or revised based on iterative experimentation, visual inspection, and accuracy benchmarking. The model pipeline includes a distinct preprocessing stage, an exploration of morphological transformations (many of which were ultimately discarded), and a deep CNN designed for character recognition.

Fig. 2: Display of Preprocessing Function



**Preprocessing Techniques**

Effectively preprocessing data for image classification was unique catered towards the heavy amounts of noise and distorted inputs provided alongside each image. To create a clearer image for the training model, three principal preprocessing strategies were applied to the raw data: grayscale , binarization, and gaussian blur.

The original images were provided in full color scale (RGB)— which while studies suggest creates little distraction from the objects in the CAPTCHA— introduces an unnecessary visual component given that to solve this given CAPTCHA color is not a vital factor and adds unnecessary complexity. To reduce dimensions of the data as well as improving processing efficiency, the images were converted to *grayscale* using OpenCV’s *COLOR\_BGR2GRAY [2]*.

Following grayscale, a *Gaussian Blur* was applied as a noise reduction procedure utilizing OpenCV’s *cv2.GaussianBlur().* A gaussian operation performs a convolution between the image and the Gaussian kernel, assigning each pixel a weight determined by its surrounding neighbors. The weights are derived from a 2D Gaussian distribution, which prioritizes closer pixels and diminishes the influence of those further away. The primary benefit of applying Gaussian blur prior to thresholding is the suppression of high-frequency image noise, such as compression artifacts or scattered background speckles. This smoothing step enhances the stability and consistency of subsequent binarization by reducing the likelihood that random noise will be interpreted as foreground features. It is worth noting the importance of performing Gaussian Blur on the data before binarization as the gaussian function ensures the adaptive thresholding algorithm more reliably distinguishes foreground characters from background noise. The best type of binarization for this data was adaptive thresholding. *Adaptive thresholding* proved more optimized than global binarization in this model due to its ability to dynamically adjust to local pixel intensity variations, allowing for consistent character extraction across CAPTCHA images with uneven lighting, varied backgrounds, or gradient distortions. Adaptive thresholding computes the threshold value locally for each pixel, based on the mean or weighted sum of surrounding pixel intensities within a specified window. With the randomness involved in CAPTCHA imaging, a single image may contain characters with different brightness levels or backgrounds that transition from light to dark.

A global threshold in this scenario would fail by either eliminating valid character regions or preserving irrelevant background features. Adaptive thresholding ensures that each region of the image is evaluated within its own context, resulting in a binarized output where characters are consistently represented as white lettering against a black background. This high-contrast separation greatly facilitates

downstream feature extraction and classification by the CNN, ultimately contributing to higher recognition accuracy and improved performance on the validation set.

After tuning each preprocessing technique and finalizing which strategies were most beneficial to the visualization of the data, I created a function to automatically apply these preprocessing steps and applied it to the CAPTCHA dataset. An example of the results of this function can be seen in *Figure 2,*with two random images from the dataset were plotted for demonstration purposes.

**Potential Morphological Techniques**

Although morphological transformations are considered to be very intuitive in removing interference, in my case the implementation resulted in an inconsistent performance.

A popular technique for removing line interference is to use morphological operations such as *morphologyEX* or *MORPH\_OPEN* to structure the kernel to selectively remove lines while preserving vital image features. While this technique was moderately successful at removing straight horizontal or vertical lines, it failed in scenarios where curved lines, diagonal interference, or semi-transparent obstructions were used. This limitation significantly reduced the reliability of morphological approaches while utilized in a generalized model. *Figure 3* is an example in which the morphological filtering displayed an extreme loss of character data.

A white and black image

AI-generated content may be incorrect.Other operations such as *cv2.erode()* and *cv2.dilate()* were also tested. These can either shrink or expand white regions (foreground) in a binary image. However, erosion often degraded thin characters to the point of illegibility, while dilation caused characters to merge with one another or expand into background noise. A key setback encountered in this phase was that many CAPTCHAs featured curved or non-linear obfuscation patterns, which are not effectively handled by rectangular structuring elements. Ultimately, while morphological techniques provided useful insights, their inconsistency led me to rely more heavily on well-tuned adaptive thresholding rather than additional geometric processing.

Fig. 3: Failed Preprocessing Function Results

**Model Architecture**

Initial testing was conducted using a custom-built CNN with minor improvements to test how a simpler model would handle CAPTCHA character recognition. This model featured a sequential architecture composed of multiple convolutional layers with ReLU activations, followed by max-pooling layers to reduce spatial dimensionality while retaining key features. To handle the sequential nature of multi-character CAPTCHA text, the architecture also incorporated a Long Short-Term Memory (LSTM) layer, intended to learn dependencies between character positions across the horizontal axis. The output from the LSTM was passed through a series of dense layers, culminating in multiple SoftMax classifiers—one for each character in the CAPTCHA sequence. To mitigate overfitting, dropout layers were interleaved throughout the network.

Despite its theoretically suitable design, this initial architecture suffered from several performance issues. The model quickly overfit to the training data, failing to generalize to unseen samples. Its limited capacity to capture complex distortions and noisy backgrounds, combined with excessive reliance on the temporal modeling of character positions, led to unstable training and poor convergence. Ultimately, the model achieved a test accuracy of only 0.8%, rendering it impractical for further development. This result prompted a pivot toward more robust, pretrained architectures such as ResNet50, which demonstrated superior feature extraction and generalization in the subsequent phases of experimentation.

A modified ResNet50 architecture was chosen based on the reputation of the architecture’s proven capacity for image classification while still allowing for flexible output handling. The structure of the model architecture consisted of:

* **Backbone:** ResNet50 with frozen base layers during initial training
* **Head:** a multi-branch fully connected layer with five parallel SoftMax outputs corresponding to the five objects in the image
* **Regularization:** Dropout and batch normalization were used to mitigate overfitting
* **Optimizer:** Adam optimizer with an initial learning rate of 0.001 and early stopping

To design the model to properly detect the index of possible characters in the CAPTCHA, each character was mapped to a corresponding numeric output. This character encoding is referred to as IDX2CHAR, enabling label encoding of strings for training and decoding predictions during inference. This mapping became an essential step in training categorical cross-entropy loss functions on each softmax output and allowed the model to treat each position as a separate classification task.

1. Experimentation

**Performance of Initial CNN Model**

Initial experimentation was performed using a custom-built convolutional neural network integrated with an LSTM layer for sequence modeling. While this design aimed to leverage temporal dependencies between character positions, it ultimately suffered from overfitting and unstable convergence, achieving only **~0.8% accuracy** on test data. This poor performance led to its exclusion from further analysis and development.

**Final Model: ResNet50**

To leverage both spatial and sequential structure in CAPTCHA text, the final model was constructed by modifying a pretrained ResNet50 backbone and appending a sequence-processing LSTM. Below is a simplified version of the custom architecture used:

class CaptchaResNet50(nn.Module):

def \_\_init\_\_(self, num\_chars=5, num\_classes=62):

# num\_classes = number of possible characters (A-Z, a-z, 0-9)

super(CaptchaResNet50, self).\_\_init\_\_()

# Load pretrained ResNet50

self.resnet = models.resnet50(pretrained=True)

# Replace final FC layer to output one vector per character

self.resnet.fc = nn.Linear(

self.resnet.fc.in\_features, num\_classes \* num\_chars

)

# LSTM to process sequential character dependencies

self.lstm = nn.LSTM(num\_classes, 128, batch\_first=True)

# Output layer: one Softmax classifier per character

self.fc\_out = nn.Linear(128, num\_classes)

self.num\_chars = num\_chars

self.num\_classes = num\_classes

def forward(self, x):

# Extract features using ResNet

x = self.resnet(x) # [B, num\_classes \* num\_chars]

# Reshape to sequence form for LSTM input

x = x.view(x.size(0), self.num\_chars, self.num\_classes) # [B, num\_chars, num\_classes]

# Sequence modeling

x, \_ = self.lstm(x) # [B, num\_chars, 128]

# Predict character probabilities

x = self.fc\_out(x) # [B, num\_chars, num\_classes]

return x

# Define optimizer and loss function

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

criterion = nn.CrossEntropyLoss()

**A group of colorful bars

AI-generated content may be incorrect.**

Fig. 4: Most and least frequently misclassified characters based on model predictions

The model was trained using the Adam optimizer with a learning rate of 1e-3 and categorical cross-entropy loss across all predicted characters. Some other included model enhancements included:

* **Dropout Layers:** inserted between dense layers to prevent overfitting
* **Batch Normalization:** applied post-activation layers to stabilize training
* **Early Stopping:** Used to halt the training look once the validation loss plateaued. Used a patience = 5

**Evaluation Metrics**

The model was evaluated using standard classification metrics seen in *Table 1*. Since CAPTCHA images consist of 5 individual characters, the model produces 5 separate predictions per image — one for each character position. These predictions are then concatenated to reconstruct the full predicted CAPTCHA string.

*To evaluate the model's performance, I define the following metrics:*

* **3+ Accuracy**: The percentage of test images where *at least 3 out of 5 characters* were predicted correctly.
* **4+ Accuracy**: The percentage of test images with *at least 4 correct characters.*
* **5-char Exact Match**: The *true accuracy* metric — the percentage of CAPTCHAs where *all 5 characters* were predicted correctly in the correct order.

These tiered metrics help illustrate the model's practical utility even when full-string accuracy is lower.

Table. 1: Evaluation metrics for the final ResNet50 CAPTCHA recognition model

|  |  |
| --- | --- |
| **Metric** | **Score** |
| Test Loss | 2.0330 |
| Precision | 0.8788 |
| Recall | 0.8771 |
| F1 Score | 0.8775 |
| 3+ Accuracy | 93.19% |
| 4+ Accuracy | 84.30% |
| 5-char Exact | 63.44% |

**Error Analysis**

To better understand the model’s character-level weaknesses, I analyzed the frequency of incorrect predictions across the test set. Using *collections.Counter* we can create an analysis of misclassification of each character. By creating a visualization shown in *Figure 4*, we are able manually theorize the causation for some characters that have an above or below average number of incorrect predictions. The plots illustrate the *top 10 most frequently misclassified characters* (left, in red), and also the *10 characters with the fewest incorrect predictions (*right, in blue).

From the left-side chart, we observe that characters *like* **‘I’, ‘l’, ‘1’, and ‘i’** were among the most commonly misclassified. These characters share thin, vertical features and are highly ambiguous in distorted CAPTCHA fonts—particularly when noise, lines, or low resolution interferes with character boundaries. The visual similarity among these characters likely overwhelmed the model’s ability to differentiate them, especially without explicit character segmentation.

The right-side chart highlights more distinguishable characters, such as *‘W’, ‘O’,* as well as *most numeric characters* (with the exception being the number ‘1’) which were least likely to be misclassified. To a human analyst it could be said that these characters generally have unique structural features (e.g., wide horizontal spread or diagonal symmetry), making them easier for the model to classify consistently.

This disparity highlights a key limitation of our model: while it performs well on characters with high visual contrast, it struggles with fine-grained disambiguation, particularly when multiple similar-looking characters appear in close proximity or are affected by preprocessing artifacts (e.g., binarization thinning strokes too aggressively)

1. Test Results

The final evaluation of the ResNet50-based CAPTCHA recognition model demonstrated that the approach was highly effective at partial character recognition and moderately effective at full-sequence decoding. With *93.19% accuracy for at least three characters*, the model shows strong capabilities in decoding individual components of CAPTCHA images, which could still provide valuable information in disproving the security of CAPTCHA functions. However, the *63.44% true accuracy score* indicates that perfect prediction remains a challenge due to factors such as character ambiguity, image noise, and lack of segmentation.

The character-level error analysis supported these findings, showing that ambiguous characters were disproportionately misclassified. This highlights the model's sensitivity to thin or structurally similar character patterns. These misclassifications likely contributed to the observed *test loss of 2.0330*, which, while acceptable, is slightly elevated compared to expectations and reflects the model’s difficulty in confidently resolving visually similar classes. Despite these limitations, the model achieved a strong *precision of 0.8788* and *recall of 0.8771*, indicating a balanced ability to correctly identify relevant characters while minimizing false negatives. The corresponding *F1 score of 0.8775* further confirms this balance between precision and recall, suggesting that the model performs consistently across a wide range of character types, even in the presence of challenging distortions.

1. Conclusion

This study set out to assess the practical viability of using deep learning, specifically convolutional neural networks, to bypass text-based CAPTCHA systems by accurately decoding their alphanumeric sequences. Through the construction of a custom preprocessing pipeline and implementation of a modified ResNet50 architecture paired with an LSTM sequence model, this project demonstrated that it is indeed feasible to automate the recognition of distorted CAPTCHA text with a high degree of partial accuracy and moderate full-sequence success.

The model was able to correctly predict at least three characters in 93.19% of test samples and achieve full 5-character sequence recognition in 63.44% of cases. These results, while lower than those reported in prior literature such as Noury and Rezaei’s Deep-CAPTCHA study, still affirm the broader concern that text-based CAPTCHAs no longer provide a robust defense against modern machine vision systems. In particular, the project revealed how adaptive preprocessing—especially the use of gaussian blur and adaptive thresholding—was critical in enhancing model performance by reducing irrelevant visual noise.

It is also important to consider the subjective nature of dataset complexity when interpreting model performance across different studies. In Deep-CAPTCHA, the authors trained on synthetically generated datasets consisting exclusively of numeric characters, which inherently reduced the difficulty of the classification task. In contrast, this study worked with a real-world dataset of five-character CAPTCHAs containing both uppercase and lowercase letters in addition to digits. Error analysis confirmed that most numeric characters, aside from the ambiguous '1', were consistently easier to classify than alphabetic characters—suggesting that the broader character space and increased visual similarity in this dataset introduced a significantly higher level of challenge. This likely contributed to the lower full-sequence accuracy and slightly elevated test loss, further emphasizing the impact of dataset variability on real-world model performance.

In closing, the findings of this research reinforce the growing vulnerability of text-based CAPTCHA systems in the age of deep learning. While the model developed here does not achieve perfect decoding, it demonstrates that a ResNet50-based architecture—when coupled with strong preprocessing—can effectively interpret even noisy, distorted alphanumeric CAPTCHA images. Future work may explore extending CNN-based or multimodal AI approaches to tackle more modern CAPTCHA formats, such as image selection CAPTCHAs, audio-based CAPTCHAs, and puzzle-style CAPTCHAs that require interpreting patterns or solving basic logic tasks. These systems increasingly rely on the assumption that such challenges are intuitive only to human users, but advancements in AI are narrowing this gap. As machine learning continues to evolve, so must our understanding of what constitutes secure, reliable human verification systems online.

References

[1] Z. Noury and M. Rezaei, “Deep-CAPTCHA: A deep learning-based CAPTCHA solver for vulnerability assessment,” *J. Ambient Intell. Humaniz. Comput.*, 2020. [Online]. Available: [https://arxiv.org/abs/2006.08296](https://arxiv.org/abs/2006.08296%20)

[2] OpenCV Team, “OpenCV documentation,” [Online]. Available: <https://docs.opencv.org>

[3] A. Paszke *et al*., “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” *NeurIPS 2019*. [Online]. Available: <https://pytorch.org>

**Note:** *This project is a work in progress. Additional scholarly sources and peer-reviewed literature are currently being reviewed and will be incorporated in future iterations of this research to strengthen the theoretical framework and expand on comparative methodologies.*