Captcha Bypass Generation Using GANs

Muhammad Abbasi Adrian Maldonado

May 19, 2025

Abstract

This paper investigates the use of conditional generative adversarial networks (cGANs) to both synthesize and bypass CAPTCHA images, aiming to evaluate the robustness of CAPTCHA security mechanisms. We implemented a cGAN that generates 5 character grayscale CAPTCHAs of size 128×128, conditioned on ground truth labels, and train an OCR model to bypass the generated CAPTCHAs. Experimental results highlight the trade-offs between image fidelity and bypass accuracy, and underscore current limitations in GAN-based security attacks. Code is available at (github.com/mabbasii/CaptchaBypass)

1 Introduction

CAPTCHAs are widely used to distinguish human users from automated bots by presenting tasks that are easy for humans but challenging for machines. Recent advances in deep learning, particularly generative adversarial networks (GANs), have raised concerns about the potential to both generate CAPTCHAs that resist automated attacks and to bypass existing CAPTCHAs. This work explores the dual use of conditional GANs (cGANs) for CAPTCHA generation and bypass, providing insights into the robustness of current security mechanisms.

2 Related Work

Mirza and Osindero first proposed cGANs, demonstrating conditional image synthesis by feeding labels to both generator and discriminator. This also gave a foundation for starting to build the architecture of our own model, as we incorporated their idea of multiple inputs, to create our two stream discriminator. [1]. We utilized the "Large CAPTCHA Dataset" by Guna from Kaggle, which consisted of 82K CAPTCHA images [2]. Each sample from the dataset consisted of a 5 character string of uppercase letters, lowercase letters, numbers as well as noise, usually in the form of a line through the letters. Using this, enabled training on a robust dataset that allowed the model to encounter many variations of character CAPTCHAs. Lastly, recent tutorials on cGAN control over outputs [3] and advances in OCR robustness [4] informed our methodology.

3 Methods

3.1 Conditional GAN Architecture

Our original architecture consisted of a generator that took a noise vector and one hot encoded label for a 5 character CAPTCHA. It utilizes a series of ConvTranspose2d layers to output a 256x256 grayscale image. We then employed a two stream discriminator that took inputs of an image and label. We originally intended to have our discriminator output two results, a classification of the image that we would compare with the actual label, and also if the image is real or fake. This was done through a CNN composed of Conv2d layers and an Optical Character Recognition (OCR) model, that outputted the label and real vs fake respectively. We then calculated the losses using both of these outputs separately, and combined them to get the total loss of the discriminator/generator.

However, we found that using this two stream discriminator approach, was very difficult to optimize, since typically the OCR model within the discriminator would learn faster then the CNN, causing the generator to create char-

acters that are unsurpassable for humans. This will be discussed more the in the Experiments section. To compensate for this, we separated the OCR model from the discriminator and made label classification and real vs fake separate.

Our final architecture had minimal changes to the generator, but it now outputs a 128x128 image. The discriminator now is a regular one stream CNN model, that simply takes the image, and outputs whether its real or fake.

3.2 OCR Model

An independent CNN based OCR model downsamples inputs to 128×128 grayscale, predicts each character via five classification heads, and aggregates cross-entropy losses per position to compute a total loss for training.

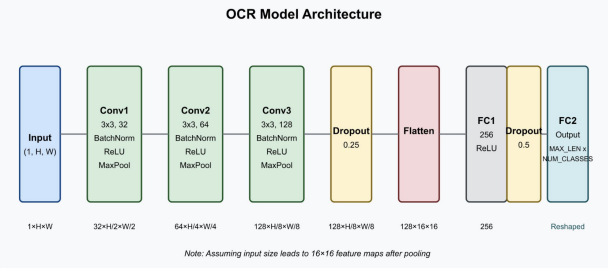

Figure 1: OCR Model Architecture

4 Experiments

4.1 Run 1

We originally conducted two major runs. Run 1 used learning rates of 0.001 for both generator and discriminator over 30 epochs; we observed rapid discriminator dominance. The generated images became cleaner as training went on, but the generator never seemed to create characters that could actually be solved.

The loss curves show that the discriminator loss dropped very early on in training and stayed relatively low throughout the entire run. This is also reflected in the generator loss, being very sporadic and also much higher than the discriminator loss.

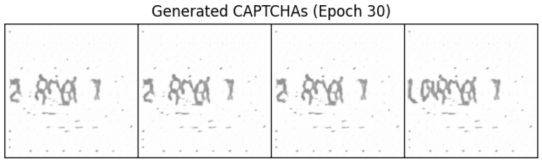

Figure 2: Run 1 Output

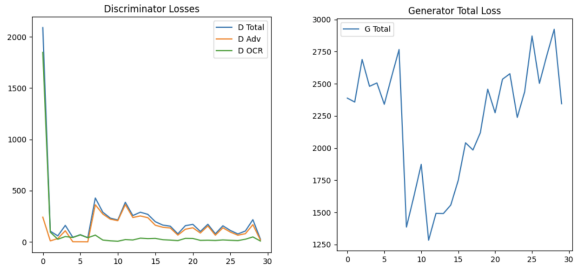

Figure 3: Run 1 Loss Curves

4.2 Run 2

To compensate for the dominating discriminator, Run 2 adjusted learning rates to 0.0005 (generator) and 0.0001 (discriminator) over 65 epochs, so the generator learning 5x faster then discriminator, thus improving generator convergence (see loss curves). The goal was to have the generator learn faster, to prevent the discriminator loss from dropping early and dominating the training. The output from this architecture was an improvement over the first run as it started creating characters that somewhat resembles real letters/numbers.

However, in general, the generated output was still unreadable for a human to solve. We theorized that the cause of this, was the use of the two stream discriminator. The OCR may have been learning faster then the CNN, causing the generator to create output that is solvable for the model, but nonsense for humans. Which is how the final architecture of having the discriminator and OCR model separate, came to be.

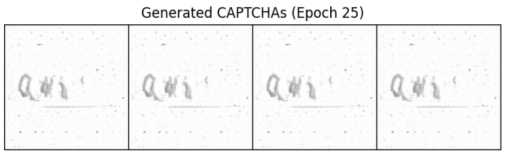

Figure 4: Run 2 Output

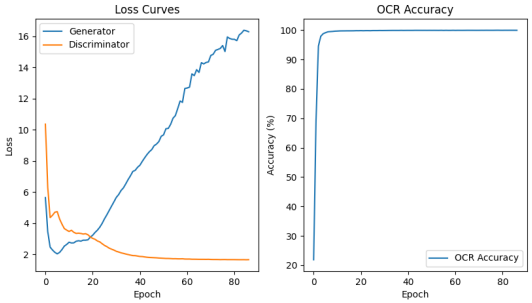

Figure 5: Run 2 Loss Curves

4.3 Final Run

The final architecture clearly had the most intelligible output, with some of the characters being very similar to real letters/numbers. Having the cGAN focus on generating CAPTCHAs that can pass as real, rather then real and hard to solve, made the model much simpler, which likely contributed to the better results. The generator loss also did not spike as much throughout training, but in the end it was still much higher than the discriminator loss. Generated output was also not suitable for training the OCR model, so we resulted in training the OCR on the "Large Captcha Dataset" instead. While training, the OCR evaluation measured both full-sequence and per-character accuracy, revealing that per-character metrics better reflect partial success.

5 Conclusion

Our cGAN framework generates CAPTCHA images, but image quality remains insufficient for reliable bypass. Discriminator tuning improved the quality of the images,

Figure 6: Final Run Output

but additional testing such as more hyperparameter tuning in both the generator and discriminator is needed in the future. Additionally, revamping our OCR architecture or even simply using a premade OCR model, could have made training easier, and allowed us to focus on fine tuning the cGAN architecture. Future work will likely explore advanced GAN regularization and incorporating OCRs into discriminators, that will help our original goal succeed.

6 Author Contributions

Muhammad Abbasi: Designed and implemented the cGAN architecture.
Adrian Maldonado: Performed data preprocessing and GAN training experiments.

References

[1] M. Mirza and S. Osindero, "Conditional generative adversarial nets," arXiv preprint arXiv:1411.1784, 2014.

[2] A. Guna, "Large CAPTCHA dataset," Kaggle, 2019. [Online]. Available: https://www.kaggle.com/datasets/akashguna/large-captcha-dataset

[3] A. Roy, "CGAN: Conditional Generative Adversarial Network — How to gain control over GAN outputs," Towards Data Science, 2019. [Online]. Avail-

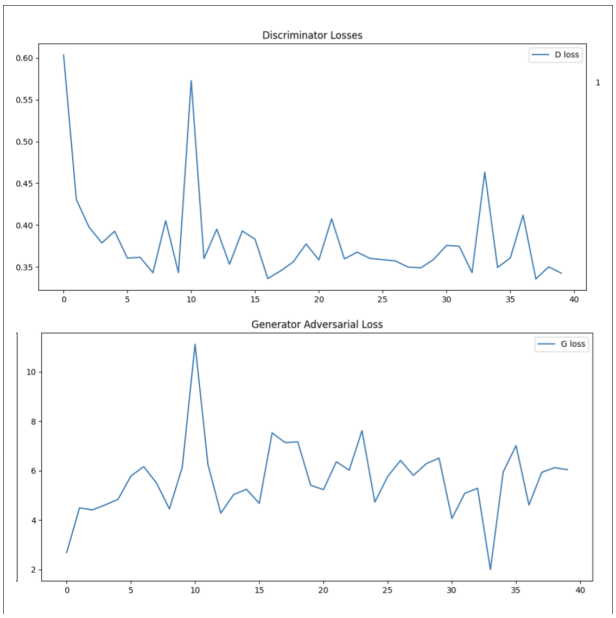

Figure 7: Final Run Loss Curves

able: https://towardsdatascience.com/cgan-conditional-generative-adversarial-network-how-to-gain-control-over-gan-outputs-b3

[4] Roboflow, "What is Optical Character Recognition (OCR)?", 2020. [Online]. Available: https://blog.roboflow.com/what-is-optical-character-recognition-ocr/