FINAL REPORT DELIVERABLE

Jesse Warren

CSE-398 Computer Vision

ABSTRACT

The aim of this project is to write an algorithm that recognizes objects and has the ability to efficiently solve picture CAPTCHAs from the reCAPTCHA captcha source.

1. PROBLEM STATEMENT

This project has a couple of major motivations. Firstly, CAPTCHAs, while necessary for security, can be time-consuming and frustrating for users. Therefore, an automated CAPTCHA solver could streamline the browsing process and enhance the user experience. If successful, such technology could have commercial value, potentially appealing to businesses that seek to automate interactions with websites that use CAPTCHAs. Another motivation is to test the robustness of CAPTCHA systems. By attempting to break these systems, I can identify and address potential vulnerabilities in the system. Developing such an algorithm presents a complex challenge in the fields of machine learning and computer vision. I am motivated to apply some of the advanced concepts of computer vision I have learned from this class and others I have taken to create a project that has a real-world application.

2. PREVIOUS WORK

At the core of solving CAPTCHA challenges is the task of object detection – a critical aspect of computer vision. Recent years have seen significant progress in this field, particularly through deep learning methods. The YOLO (You Only Look Once) series, specifically YOLOv5, has emerged as a prominent framework due to its efficiency and accuracy in real-time object detection. The YOLOv5 model, hosted on GitHub, offers a pre-trained model that can be fine-tuned for specific object recognition tasks, making it ideal for CAPTCHA solving [1]. Additionally, the model's ability to be retrained on custom datasets will be beneficial for adapting to the diverse images presented in CAPTCHAs. The application of machine learning in CAPTCHA solving is not new. Several studies and projects have demonstrated the feasibility of using machine learning algorithms to bypass CAPTCHA systems. A noteworthy example is detailed in the Medium article by Cavallo which outlines a method for solving reCAPTCHA using image segmentation and machine learning [2]. This approach is particularly relevant, as it demonstrates the process of segmenting captcha challenges into individual images for object classification. Beyond object detection, the project necessitates interacting with web pages to select CAPTCHA images. This is where Selenium, a powerful tool for automating web browsers, comes into play. Selenium allows for programmatically clicking on images and navigating web pages, essential for the practical application of the CAPTCHA solving algorithm [3].

3. DATASET

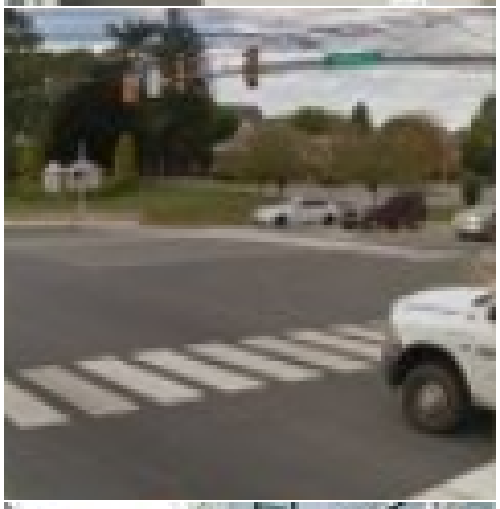
3.1. Original Dataset

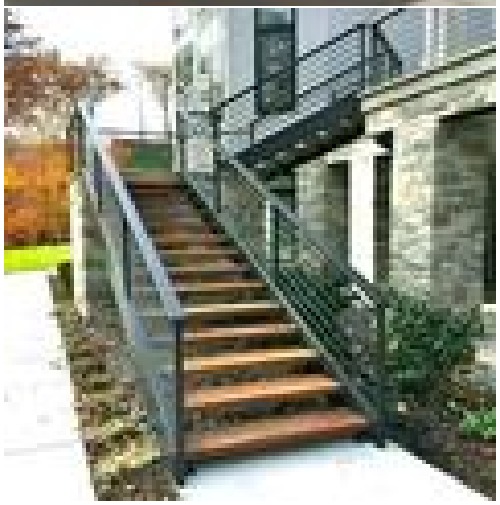
The original dataset used only consisted of images of stairs, crosswalks, and chimneys. Not only did the dataset have only 3 object classes of the 10+ classes possible in recaptcha but the quantity of images proved not to be enough early on as well. With a lack of a better dataset available publicly, the decision was made to create a dataset.

3.2. Dataset Creation

A script was written to scrape images from the official recaptcha demo. Additionally, a script was written to label the images in the yolo format. Approximately 2,500 images were hand labeled and then ran through an augmentation pipeline to essentially double the dataset to 5,000 images. The augmentation formula added random noise to each image. The Yolo model was trained on this dataset with 40 epochs,

3.3. Dataset Examples





4. APPROACH

4.1. General Overview

The main script automates the interaction with a reCAPTCHA challenge found on a webpage, specifically designed to handle Google's reCAPTCHA system. It utilizes the Selenium WebDriver for browser automation, PIL for image manipulation, and YOLO (You Only Look Once) deep learning model for object detection within the reCAPTCHA images.

4.2. Setup and Navigation

A Chrome WebDriver is initiated to automate browser interactions. The script navigates to a specified webpage hosting a reCAPTCHA challenge (in this case, Google's reCAPTCHA demo site). reCAPTCHA challenges are typically embedded in iframes. The script dynamically waits and switches context to the appropriate iframe to interact with the reCAPTCHA challenge.

4.3. Image Scraping and Processing

Once the reCAPTCHA challenge is activated, the script extracts the image(s) presented in the challenge. This involves locating the image element using its CSS selector and retrieving the image URL. Images are downloaded and segmented into smaller grid images, using the Python Imaging Library (PIL). This step is crucial for preparing the data for object detection, as reCAPTCHA images often contain multiple objects that need to be individually recognized.

4.4. Object Detection with Yolo

For each image in the challange grid, the script invokes the YOLO object detection model via a command line interface. This is handled using Python's subprocess module, which allows for the execution of shell commands directly from Python. The script processes the output from the YOLO model to determine whether the identified objects match the reCAPTCHA's query (e.g., "select all images with cars"). This involves parsing the detection results, which include object classes and their bounding boxes.

4.5. Interaction Automation

Based on the analysis of the detection results, the script automatically selects the correct images in the reCAPTCHA challenge. After selecting all relevant images, the script submits the reCAPTCHA response for verification.

5. EXPERIMENTS

5.1. Problems With the Approach

reCAPTCHA challenges have a time limit for completion, typically expiring after two minutes. The process of scraping images, performing object detection, and responding appropriately is computationally intensive and time-consuming, particularly when integrating third-party models like YOLO. The latency introduced by image downloading, processing, and the invocation of the YOLO model for each image segment results in a pipeline that often exceeds the allowable time to respond to the CAPTCHA. This delay compromises the feasibility of the approach in real-time applications where quick response times are crucial.

After making a selection, reCAPTCHA sometimes refreshes part of the image grid with new images. This dynamic change requires the automation system to continuously monitor and re-process images, further increasing response times and computational overhead. Despite the model's high intrinsic accuracy, the overall approach faces a bottleneck when it comes to real-world applicability. The combination of tight time constraints and highly variable CAPTCHA formats means that, without substantial optimizations or a different approach to image processing and decision-making, this method may not be viable for all but the most controlled environments.

5.2. New Plan

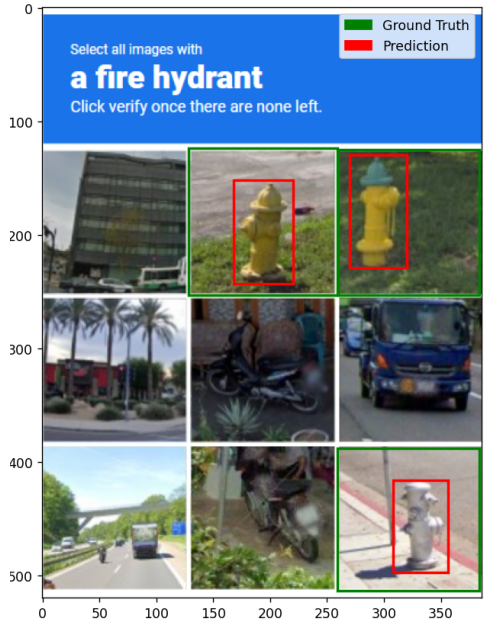
The new approach involved a couple of steps. Using similar web scraping as mentioned above, test captchas were obtained from the official recaptcha demo website. After which, the test captchas were hand labeled using the previously mentioned scripts. Next, in a similar manner to the above approach the model evaluates the images, but this time without time constraint. The results are analyzed and the model's performance is evaluated on total correct images, and total correctly solved captchas. Additionally, a script was created to visualize the ground truth labels alongside the models predictions as well as a script graphing IoU performance.

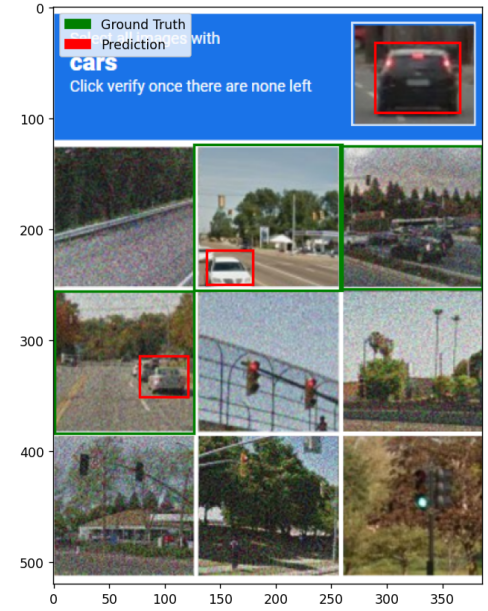
6. RESULTS AND DISCUSSION

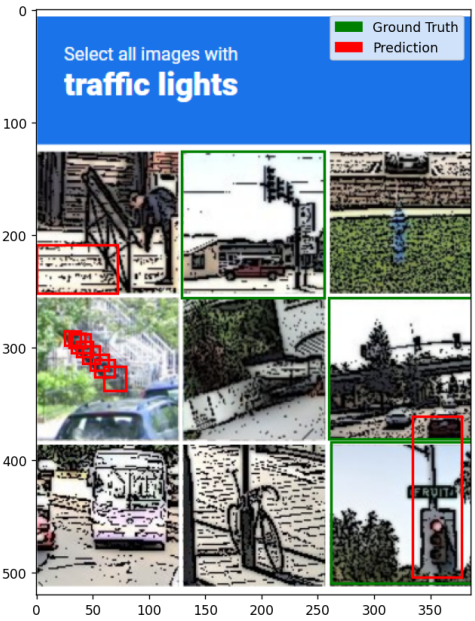
6.1. Model Performance

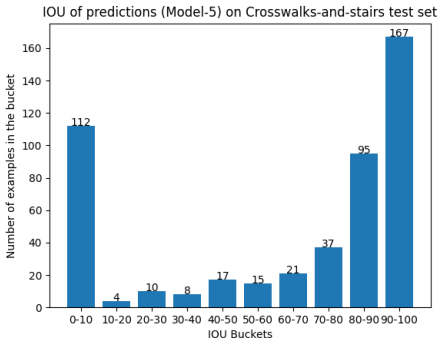
The evaluation of the captcha-solving model presents mixed results, which indicate certain challenges in the model's current capabilities and the complexity of the captcha-solving task. The test set consisted of 100 test captchas and each captcha in the test set had 3 or more objects to be detected. The model correctly identified 158 out of 304 captcha images, resulting in an accuracy of approximately 51.97 percent. This metric demonstrates that the model has a moderate ability to recognize and decipher individual captcha objects correctly in over half of the test cases. The overall effectiveness of the model in solving complete captchas accurately was lower however, with only 14 out of 100 test captchas fully solved correctly, corresponding to an accuracy of 14.00 percent. This indicates significant challenges in achieving correct detection of all captcha objects to solve the captcha. However, it makes sense for this to be lower as one missed object will result in an unsolved captcha.

6.2. Performance Visualization









6.3. Discussion

The accuracy of 51.97 percent in correct image detection highlights a partial success in understanding some of the captcha characteristics. However, this partial recognition is insufficient for practical applications where complete captcha solving is necessary. It suggests that while some features and elements are being learned effectively, others, perhaps due to their complexity or subtlety, are not being captured by the model. Additionally, The variability in captchas such as the presence of noise and distortions could significantly impact model performance. The model's training may not have encompassed sufficient diversity or volume of data to generalize effectively across the wide range of captchas encountered in the test set. This project taught me that Google uses the reCAPTCHA system for a reason. Additionally, it demonstrates the sheer quantity of data needed to accurately train a model for real world applications. It seemed the success cases here were objects such as fire hydrants, cars, and buses that had features that were more easily learned than objects such as crosswalks, stairs, and bicycles. I believe more data is necessary to achieve greater results.

7. FUTURE WORK

7.1. Possible Improvements

Developing algorithms for dynamic segmentation that can adapt to different captcha formats and layouts dynamically. This would allow the system to handle a wider variety of captcha types without manual reconfiguration.

Optimizing the system to reduce latency and increase processing speed to ensure that responses are generated within the time limits imposed by captcha systems. Implementing GPU acceleration for model inference is one potential approach to achieve faster processing times.

Creating a dataset that well represented captcha images was quite difficult, and even the current dataset of 5,000+ images did not seem to be enough. A larger dataset could be used to improve the model. By training the model on a more diverse range of data, the system can learn to generalize better to unseen captchas, reducing overfitting and enhancing its ability to adapt to new or dynamically changing captcha styles. Training on a large and varied dataset allows the model to extract and learn a richer set of features. This depth of learning enables the model to discern and recognize subtle nuances that might be missed when trained on a smaller dataset. This might mean better detection of obscured or overlapping objects, understanding contextual clues within images, or more accurately identifying minor details.

8. REFERENCES

[1] Ultralytics. "Ultralytics/Yolov5: Yolov5 in PyTorch ONNX CoreML TFLite." GitHub, github.com/ultralytics/yolov5. Accessed 4 Mar. 2024.

[2] Cavallo, James. "Solving Recaptcha with Image Segmentation Machine Learning." Medium, Medium, 13 Nov. 2023, cavalloj.medium.com/solving-recaptcha-with-image-segmentation-machine-learningdb3844d9894e.

[3] Selenium, www.selenium.dev/. Accessed 4 Mar. 2024.

[4] "ReCAPTCHA Demo." Google, Google, www.google.com /recaptcha/api2/demo. Accessed 1 Apr. 2024.

[5] Mazurov, Mike. "Google Recaptcha Image Dataset." Kaggle, 9 Aug. 2022, www.kaggle.com/datasets/mikhailma/test-dataset/data.