Unsolvable CAPTCHA Generator

Doruk Özer, Sarp Göl, Yiğit Yakar, Efe Ertem

INTRODUCTION

A Completely Automated Public Turing Test to tell Computers and Humans Apart (CAPTCHA) is used in web systems to secure the authentication and privacy of the web systems. With the developments in Machine Learning, CAPTCHAs are not able to distinguish between humans and computers as accurately as before since ML models can solve CAPTCHAs easily. Therefore, it is a necessity to build CAPTCHAs that are unsolvable by computers so the privacy is preserved. To achieve the unsolvable CAPTCHA generator, we have built a CNN Based model to solve the CAPTCHAs. After building the solver with high accuracy, we proceeded to implement three different evasion attacks using Gaussian Blur and Random Noise. While implementing evasions, we have tried to keep the images readable by humans while making them uninterpretable by machines. In the end, we have come up with successful attacks that reduced the CAPTCHAs recognition accuracy significantly.

PREPROCESSING

The process of preprocessing CAPTCHAs involves two steps. The first step is to remove noise from the image to prevent bias in the trained network and to assist in character segmentation. The second step is to segment the characters to convert the problem into a character recognition task. This process is repeated for N CAPTCHAs containing 4 characters each, resulting in 4N characters for the model's character recognition dataset.[2]

THRESHOLD

The image is first converted to grayscale and then binarized using a threshold of 230 to remove noise, which means all pixels with an intensity value less than or equal to 230 are turned into black, and all pixels with an intensity value greater than 230 are turned into white. [2]

REMOVE NOISE

Removing noise from an image before processing it in a machine learning model is crucial to ensure accurate and reliable results. Noise refers to any unwanted or irrelevant information in an image that can negatively impact the performance of the model. If noise is not removed, it can lead to bias in the trained model as well as confusion during the segmentation and recognition process. We used the “NoiseRemover” module of James Hahn’s project which deals with two types of noises present in the image, circular noise and horizontal noise. The implementation of the noise remover is as follows:

- Erosion is an image processing technique used in computer vision to shrink or erode the boundaries of bright regions in an image. The erosion process is performed by sliding a small structuring element (also known as a kernel) over the image and replacing the central pixel of the structuring element with the minimum value of the pixels under the structuring element. This process is repeated for every pixel in the image and it shrinks the white regions in an image, making the region smaller and removing small white pixels on the boundaries. In the module, erosion is applied with a small kernel of size (2, 2) for a single iteration to weaken the circular and linear noise present in the image [2].
- The median filter is a useful technique for eliminating noise in nonlinear digital signals while preserving important edges. The process of reducing noise is a common step in the pre-processing phase to enhance the final outcomes of subsequent processing tasks such as edge detection on an image. The median filter works by iterating through each entry of the signal and replacing it with the median value of its neighboring entries. This approach is highly effective in removing noise while preserving edges [1]. An example mean filtering algorithm can be seen in the figure below.

```
Require: Image $X$ of size m × n, kernel radius $\tau$
Ensure: Image $Y$ of the same size as $X$ Initialize:
  kernal histogram $H$
  for i = 1 to m do
      for j = 1 to n do
          for k = -$\tau$ to $\tau$ do
              Remove $X_{i+k,j-\tau-1}$ from $H$
              Add $X_{i+k,j+\tau}$ to $H$ end for $Y$
          $_{i,j}$ ← $median(H)$ end for end
          for
```

Figure 1. Median Filtering algorithm [1]

The median filter applied is a vertical one with a size of (5, 1) which effectively eliminates most of the line noise. This also helps to reduce the circle noise present in the image. Following this, a horizontal filter of size (1, 3) is applied to remove the remaining circle noise. However, during this process, some of the actual character data may also be removed. To address this, a dilation operation is performed. Dilation is an image processing technique that is used to increase the size of certain elements (foreground) of an image. This is done by using a small shape (structuring element) that is used to "scan" over the image. When the structuring element overlaps with a foreground element, it is used to "dilate" that element by adding pixels to it, thus increasing its size. This process can be useful for a variety of tasks, such as improving the visibility of small features in an image, removing small gaps or holes in an object, and enhancing the contrast of certain parts of an image. Using a dilation filter of size (2, 2) to increase the size of the letters. This also helps to restore any remaining character data. Despite this, some noise may still be present in the image, so a final median filter of size (3, 3) is applied to remove any weak noise that remains. [2]

SEGMENTATION

Segmentation is crucial as it divides the CAPTCHA image into smaller parts, which improves the performance of the analysis. It detects the boundaries of the CAPTCHA, such as lines or curves, and considers factors like rotation, noise, and twisted characters[3]. We used John Hahn's module once again, "characterSegmenter" module. In his implementation, characters are then segmented out of the image using OpenCV's connectedComponents [4], [5] [6] function and the watershed algorithm [7]. Any errors in character overlap or noise recognition are handled using a combination of mask removal and sub-dividing.

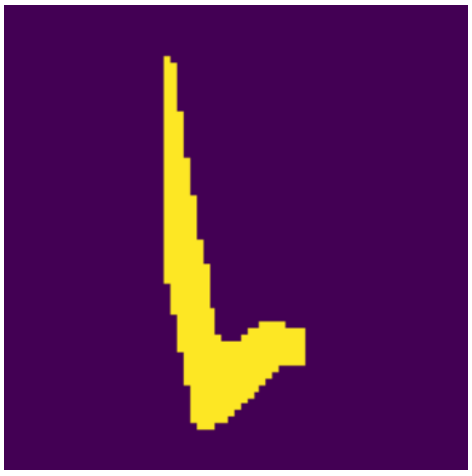


Figure 2: An example of the preprocessed image with label "L"

MODEL

Convolutional Neural Networks (CNNs) have been widely used and studied in image processing tasks and have been proven to achieve state-of-the-art performance in various image processing tasks such as image classification, object detection, and semantic segmentation. [9] CNNs are effective in image processing due to their ability to learn hierarchical representations of input data, enabling detection of simple and complex features for high accuracy in image classification tasks. [8] A CNN architecture called AlexNet achieved a top-5 error rate of 15.3% on the ImageNet dataset, which is a large dataset of labeled images. [9]

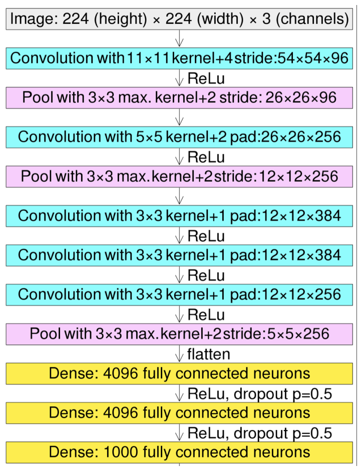


Image: 224 (height) × 224 (width) × 3 (channels)
Convolution with 11×11 kernel+4 stride:54×54×96
ReLu
Pool with 3×3 max. kernel+2 stride: 26×26×96
Convolution with 5×5 kernel+2 pad:26×26×256
ReLu
Pool with 3×3 max.kernel+2stride:12×12×256
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×256
ReLu
Pool with 3×3 max.kernel+2stride:5×5×256
flatten
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 1000 fully connected neurons

Figure 3: AlexNet architecture [10]

Even though Y.Wang et al. suggests using an adaptation of LeNet architecture [1], we determined that utilizing the AlexNet architecture would be the most appropriate approach for our project, given its deep structure and the availability of sufficient training data[2, 9]. Furthermore, we employed the Adam optimization algorithm in order to optimize the performance of our model.

Our convolutional neural network (CNN) model for image classification is created using PyTorch which allows for defining a sequential flow of layers in the model. The model consists of several layers, each with a specific function.

The first layer is a 2D convolutional layer with 20 output channels, a kernel size of (5, 5) and a stride of (1, 1). It also has padding of (4, 4) to ensure that the output has the same dimensions as the input. This layer is responsible for learning and extracting features from the input image.

The next layer is a rectified linear unit (ReLU) activation function, which is applied element-wise to the output of the previous layer. It is used to introduce non-linearity to the model, allowing it to learn more complex features.

After the ReLU activation, there is a max pooling layer with a kernel size of 2 and a stride of 2. This layer is used for down-sampling the feature maps, reducing their dimensions and retaining only the most important features.

This process is repeated with a second convolutional layer, which has 20 input channels and 50 output channels and the same kernel size, stride and padding as the first convolutional layer. This layer is also followed by a ReLU activation function and a max pooling layer. The output from the max pooling layer is then flattened, which means that it is converted into a 1D array and passed through a linear layer with 24200 input features and 500 output features. This layer is also followed by a ReLU activation function.

Finally, there is a second linear layer with 500 input features and 36 output features. This layer represents the output of the CNN, which is the predicted class for each input image. The Adam optimizer is used to update the weights of the model during training, with a learning rate of 1e-4.

TRAINING

Our dataset consisted of 7969 CAPTCHAs generated by using James Hahn's module [2], each containing four randomly generated characters from the set of "A-Z" and "0-9" in random positions. This resulted in a total of 31876 labels. The first 30000 characters were designated for training while the remaining 11976 were used for validation. Additionally, a separate test set of 4000 CAPTCHA images was created. A train_part34 method was implemented to train the model using a learning rate of 1e-4, a batch size of 128, and for a total of 23 epochs. The method utilizes cross entropy loss and evaluates the model's performance on the validation set during each epoch. The chosen values for the learning rate, batch size, and number of epochs were determined through manual experimentation and were determined to be the optimal for performance.

EVASION

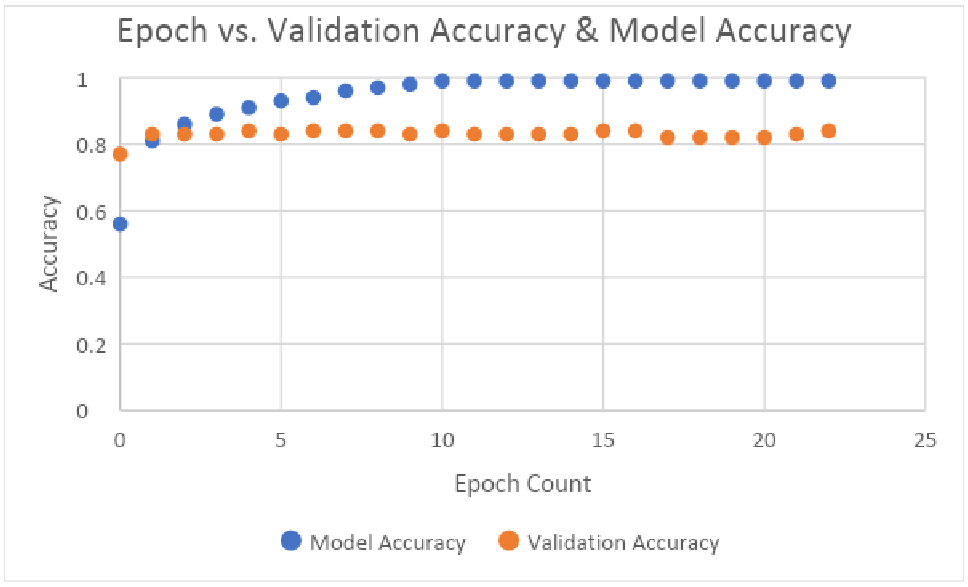
We used two types of noises to evade the CAPTCHA images: Gaussian blur and Salt & Pepper noise.

Salt and pepper noise is characterized by random black and white pixels, or "salt and pepper" in the image. It causes errors in the image acquisition process.

Gaussian blur is a type of noise that has a probability density function (PDF) equal to that of the normal distribution. One benefit of using Gaussian blur is that it is equally likely to occur in any direction. This makes it well suited for applications where the noise is not known to be coming from a specific direction. Additionally, Gaussian blur is computationally efficient to generate, and it is relatively easy to filter out using a variety of techniques. Additionally, it is a “white noise” meaning that it has a flat power spectral density.

We used OpenCV’s cv2 module to add noise. For an image with gaussian blur we used the "cv2.randn" method, with the specified mean and sigma to create the noise and then add the noise to the image after scaling with the “cv2.add” method. To add a salt-and-pepper noise to the image, we used the "random_noise" function from the “skimage” module which is the scikit-image library created for image processing in Python. The specified amount of noise is added.

TRAINING RESULTS



Test Accuracy: 0.82

3936/4096 characters

Only Gaussian Noise:

Gaussian Noise Results

Noise	Unrecognized Captchas	Recognized Characters	Correct Characters	Accuracy
0	1	76	62	0,816
10	1	76	60	0,789
20	1	76	62	0,816
30	1	76	65	0,855
40	1	76	61	0,803
50	1	76	64	0,842
60	2	72	54	0,750
70	2	72	59	0,819
80	4	64	50	0,781
90	4	64	48	0,750
100	6	56	44	0,786
120	7	52	25	0,481

Unrecognized Captcha: Captchas that were not recognized by the solver.

Correct Characters: Characters correctly predicted by the solver among the all recognized characters

Accuracy: Correct Characters/Recognized Character

Only Random(Salt and Pepper) Noise:

Random Noise Results

Noise	Unrecognized Captchas	Recognized Characters	Correct Characters	Accuracy
0,01	0	80	62	0,775
0,05	0	80	61	0,763
0,10	0	80	64	0,800
0,15	0	80	59	0,738
0,20	1	76	56	0,737
0,30	2	72	49	0,681

Combined:

Combination of Gaussian Blur and Random Noise Results					
Gaussian Noise	Random Noise	Unrecognized Captchas	Recognized Characters	Correct Characters	Accuracy
20	0,05	1	76	62	0,816
20	0,1	1	76	60	0,789
20	0,2	1	76	58	0,763
50	0,05	2	72	58	0,806
50	0,1	1	76	61	0,803
50	0,2	3	68	53	0,779
80	0,05	3	68	45	0,662
80	0,1	4	64	44	0,688
80	0,2	6	56	44	0,786
100	0,05	5	60	44	0,733
100	0,1	5	60	38	0,633
100	0,2	7	52	30	0,577

Experimental Results and Analysis:

We have run three different experiments after creating the model and evasion methods. In the first experiment, we have used Gaussian Blur as the noise. In the second experiment, we have used random noise and in the final experiment, we have used both noises combined. To decide what amount of noise should be added, we have run the code several times to see if the images are still interpretable by human-eyes after the noise is added. After running this experiment, we have realized that the maximum amount of Gaussian Blur and Random Noise should be 120 and 0.3, respectively. When they are combined, these numbers drop to 100 and 0.2 since both noises drop the interpretability level of the image cumulatively. Examples of images with Gaussian Blur and Random Noise can be found below.

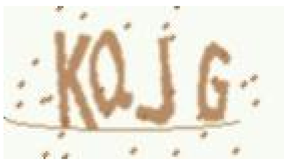


Figure 4: Without noise

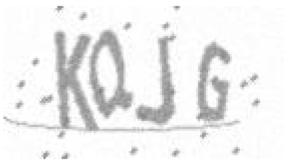


Figure 5: Gaussian-20 noise

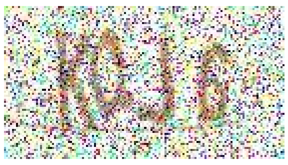


Figure 6: Salt-0.3 noise

Figure 7: [Gaussian + Salt]-0.210 noise added

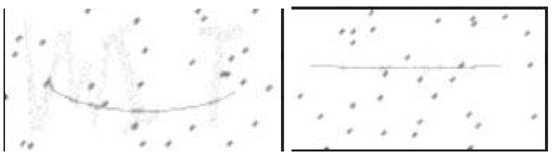


Figure 8 & 9: Unreadable Captchas

Analysis & Conclusion

While analyzing the results, it should be kept in mind that accuracy metric alone will not be sufficient to come up with sensible conclusions. We have defined accuracy as the number of correctly predicted characters over the number of recognized characters. In all experiments results, there is a column for unrecognized captchas. The data in this column represents the number of captchas that were not recognized by captcha solver. These unrecognized captchas are not included in the accuracy calculations. Since these captchas were not even read by the solver, it is safe to say that they have also evaded the solver successfully. In the end, they were not recognized so we will keep them out of accuracy calculations. Additionally, the number of correctly recognized characters is another metric that should be considered while evaluating the results as it shows the real performance drop of the solver character by character.

In the first experiment, the accuracy levels do not drop until the noise is 120. However, the number of unrecognized captchas and the correctly predicted character amounts have a clear decreasing trend that shows that the noised captchas are evading the solver more and more. The level of readability also decreases quite heavily as the noise is increased. Although readability drops, most of the captchas are still quite interpretable by human-eyes. With a few exceptions, Gaussian Blur works quite well with the solver.

In the second experiment, the sudden decrease in the accuracy from the previous experiment is not visible. Instead, there is a rather smooth trend in accuracy as well as the number of correctly predicted characters. However, the number of unrecognized captchas is at most 2. Comparing it with the first experiment, the model was able to recognize more captchas than the first experiment. The accuracy level of the recognized characters drops to 0.68 in the maximum amount of random noise. Considering the fact that the model had an accuracy of 0.82 in the captchas with no noise added, this drop is not enough to call the experiment successful.

Final experiment results are more promising than the previous experiments' results. Throughout the experiment, we have used a combination of four different Gaussian Blurs and three different Random noises. The most successful combinations were the combinations with Gaussian Blur at level 100. The number of recognized characters and accuracy levels see an all-experiment low with 30 and 0,57 respectively. Even though the readability of the images visibly dropped, most of the images are still quite readable. The images with Gaussian Blur level 80 were still performing quite well considering all of them had 45 or 44 characters that were classified correctly by the solver. The number of unrecognized captchas have followed the prior increasing trend which shows that evasion is working successfully.

Overall, it can be comfortably said that our evasion attacks were mostly successful. The experiments with Gaussian Blur were able to evade the model more than the ones with Random Noise. Yet, the best experiment was the one where both noises were combined. In the combination experiment, it is shown in the data that Gaussian Blur was the noise that had more impact on the model. In most trials of the combined experiment, the random noise reduced the accuracy only slightly. However, random noise had more effect after the Gaussian Blur level hit 100. So, combinations around this level can be used in further experiments with other types of noises.

References

1. Wang, Ye & Lu, Mi. (2018). An optimized system to solve text-based CAPTCHA.
2. Hahn, James. “Solving Noisy Text CAPTCHAs.” *Medium*, 28 Feb. 2020, medium.com/analytics-vidhya/solving-noisy-text-captchas-126734c3c717.
3. S.-Y. Huang, Y.-K. Lee, G. Bell, and Z.-h. Ou, “An efficient segmentation algorithm for captchas with line cluttering and character warping,” Multimedia Tools and Applications, vol. 48, no. 2, pp. 267–289, 2010.
4. Federico Bolelli, Stefano Allegretti, Lorenzo Baraldi, and Costantino Grana. Spaghetti Labeling: Directed Acyclic Graphs for Block-Based Connected Components Labeling. IEEE Transactions on Image Processing, 29(1):1999–2012, 2019.
5. Costantino Grana, Daniele Borghesani, and Rita Cucchiara. Optimized Block-Based Connected Components Labeling With Decision Trees. IEEE Transactions on Image Processing, 19(6):1596–1609, 2010.
6. Kesheng Wu, Ekow Otoo, and Kenji Suzuki. Optimizing two-pass connected-component labeling algorithms. Pattern Analysis and Applications, 12(2):117–135, Jun 2009.
7. “The Watershed Transformation.” *The Watershed Transformation*, people.cmm.minesparis.psl.eu/users/beucher/wtshed.html.
8. LeCun, Y., Bengio, Y. Hinton, G. Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539
9. Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.
10. “AlexNet” *AlexNet - Wikipedia*, 17 Nov. 2013, en.wikipedia.org/wiki/AlexNet.