CAPTCHA Solver and Creating Unsolvable CAPTHCAs

Ali Oktay 72007, Nidai Çağrı Savran 72092

Abstract

CAPTCHA is an acronym of Completely Automated Public Turing test to tell Computers and Humans Apart.It is a tool that is used to determine whether the user is a human or a bot. It was only text-based CAPTCHA at the beginning but with the development in the artificial intelligence field CAPTCHA evolved as well and nowadays CAPTCHA appears in daily life as text-based, audio-based, image-based or combinations of them. With the improvement of artificial intelligence and deep learning, bots are able to solve the basic models of CAPTCHA. To avoid that some additional security is added to the CAPTCHA that decreases the false positives but does not affect the humans, true positives i.e adding some noise that is not human detectable.

In this project, we created a model using convolutional neural network (CNN) that solves the 5 digit CAPTCHA and then created some attacks to see the performance of our model in each of them. Then using these attack models we created some unsolvable CAPTCHAs and tested our model on them and we have successfully reduced the accuracy of the model from %85 to %0. After the attack we trained the data with our initial data and some perturbed data that is created by our attack functions. Then we analyzed the performance of the model on the attacks after that training and observed it gets better on perturbed attacks.

Motivation

Creating a successful CAPTCHA which is needed to have at least %90 human success rate and lower than %1 bot success rate [1] is the main challenge of the CAPTCHA. We focused on the text-based CAPTCHAs that have 5 digits in that project to create a successful CAPTCHA and a model that beats even the successful ones. The text-based CAPTCHA is the easiest one for the bots. That is the reason why we picked that one and the 5 digit is the most commonly used type.

In our project we also used the whitebox FGSM method to create Captchas to create a dataset that any of the CAPTCHAs are not solvable by the model. Finally we also show that adding some adversarial data to the training dataset improves the performance of the model, and analyzing the difference between accuracy and loss of the model without the adversarial data and with the adversarial data.

Method

Following sections describe our approach on creating the CAPTCHA solver model and generating the adversarial examples.

Dataset

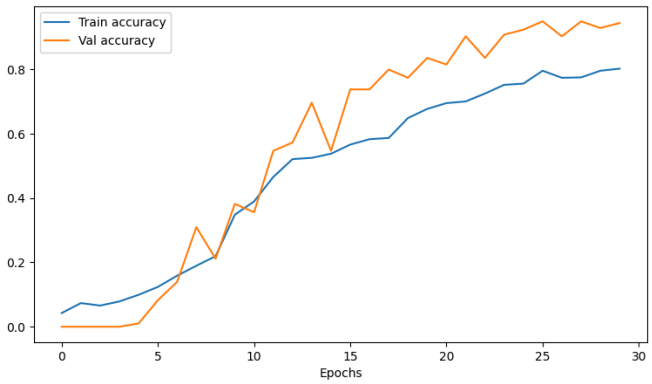
Our dataset contains 1070 sample CAPTCHAs taken from following website:
https://www.kaggle.com/datasets/fournierp/captcha-version-2-images/

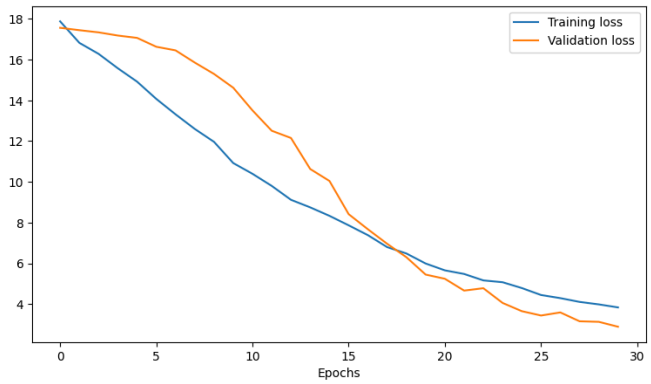
After taking the dataset, we separate it to 970 for test and validation and remaining 100 for the test. The 1070 data was enough for us to train the dataset to get desired accuracy but if we increase the size of the dataset we could get better accuracy.

CAPTCHA Solver Model

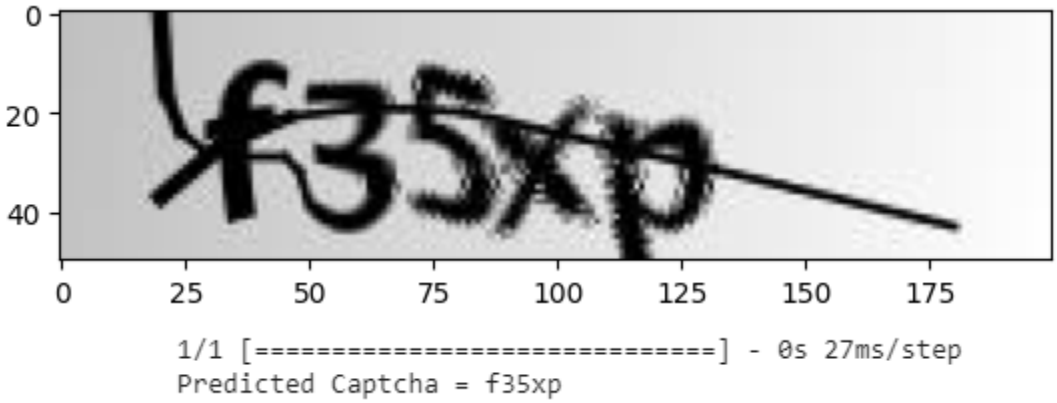
There are several CAPTCHA solvers available online which mostly use convolutional neural networks. These models have high accuracy on the usual nonadversary CAPTCHAs. I used the model below for this problem which is similar to the AlexNet [2] but modified according to our needs.


input_1 InputLayer
conv2d Conv2D
max_pooling2d MaxPooling2D
conv2d_1 Conv2D
max_pooling2d_1 MaxPooling2D
conv2d_2 Conv2D
batch_normalization BatchNormalization
max_pooling2d_2 MaxPooling2D
flatten Flatten
dense Dense
dense_2 Dense
dense_4 Dense
dense_6 Dense
dense_8 Dense
dropout Dropout
dropout_1 Dropout
dropout_2 Dropout
dropout_3 Dropout
dropout_4 Dropout
dense_1 Dense
dense_3 Dense
dense_5 Dense
dense_7 Dense
dense_9 Dense





We reached %78 accuracy on training and %94 for the validation. The test accuracy gets up to %83. And the loss strictly gets lower in every epoch which shows no signs of overfitting or underfitting so we concluded that our model and hyperparameters are a good fit for this problem and data.



This is an example input and output for our model

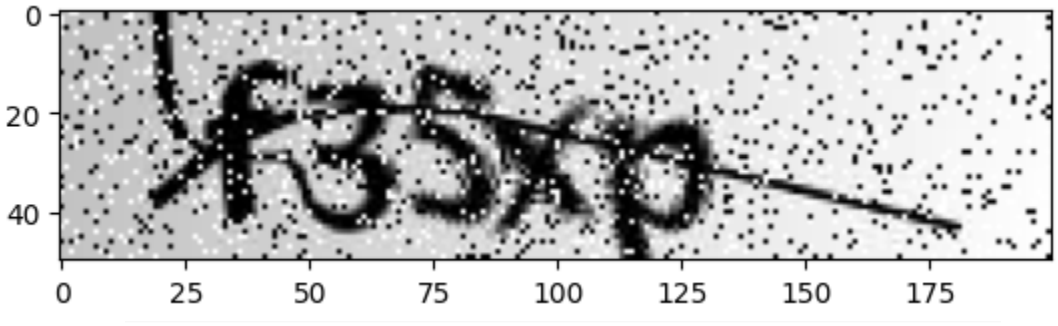
Attacks

Firstly we tried the attack method called evasion attack that is an attack type that tries to evade the model by manipulating the input data. We used two different approaches for the evasion attack: salt and pepper, and gaussian noise.

Salt and Pepper

This attack changes each pixel to white or black pixel in 5 percent probability. This attack is visible by the human eye but does not affect the solvability of the CAPTCHA. With this attack

we get accuracy %52. Below is an example CAPTCHA that is perturbed using this attack and prediction of our model..

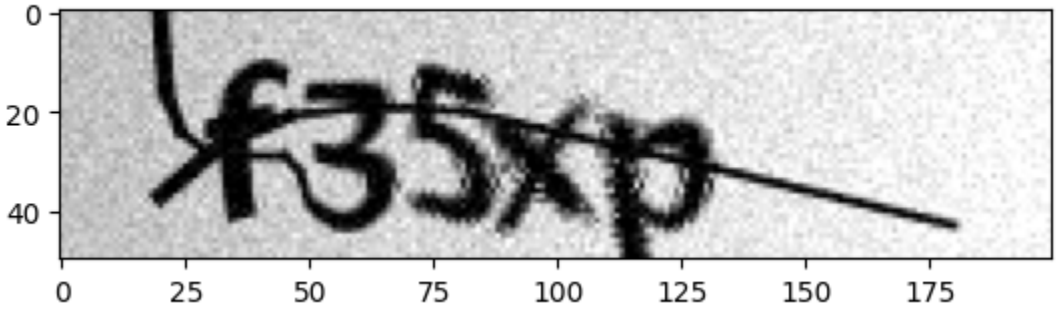


```
1/1 [==============================] - 0s 27ms/step
Predicted Captcha = f35xp
```

Gaussian Normal Distribution

This attack changes each pixel to a randomly chosen colored pixel in 5 percent probability. This attack is visible by the human eye but does not affect the solvability of the CAPTCHA. With this attack we get accuracy %69. Below is an example CAPTCHA that is perturbed using this attack and prediction of our model..

```
<matplotlib.image.AxesImage at 0x7c50dde70f40>
```



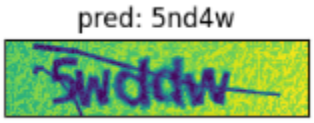
```
1/1 [==============================] - 0s 29ms/step
Predicted Captcha = f35xp
```

FGSM Attack

FGSM (fast gradient sign method) is the last attack method I tried on my model. I used white box approach in this attack which means that I provide the model to the FGSM method. It computes the gradient of the loss and sign of the gradient then uses that sign to construct the adversarial image. The working mechanism of the algorithm is like following:

1. First feed the model with an input sample x and get the prediction y.
2. Compute the loss and gradient of the model using the predicted output and actual output.
3. Take the sign of the gradient and multiply it by a small enough scalar to control the magnitude of the perturbation.
4. Add the perturbation to the input x to create the adversarial example.

Using that method we reduced the accuracy of the model to %0 which is a huge decrease that most probably caused because we used a white box approach that enables the FGSM method to find the better adversarial images. Below is an example of CAPTCHA that is created by FGSM and its prediction by our model.



Retrain with Adversarial Data

Finally we trained the model again but this time we added some adversarial data to its training and validation data to see its effect on the accuracy of the modal in non adversarial and perturbed CAPTCHAs. After we trained it again following are the new accuracies.

Input Type	Accuracy Rate
CAPTCHA without perturbation	%83.2
Salt and Pepper CAPTCHA	%82
Gaussian noisy CAPTCHA	%88
FGSM	%0

The surprising result is for us to have better accuracy on the gaussian noisy data than the CAPTCHA without any perturbation. Other than that it is clear that training the model with some

adversarial data is beneficial and has crucial benefits to the accuracy of the model on testing with perturbed data.

Conclusion

In this project we tested some attack methods and tested the effect of the adversarial training on the model. We compared the two trained models and saw that after the training using adversarial data there is a noticeable increase in the accuracy of the model. Salt and pepper has better performance at reducing the accuracy than the gaussian noise. The reason behind that is salt and pepper is strictly white and black dots but when we add gaussian noise we add more grayish colors which changes the CAPTCHA less then the salt and pepper approach. But the most successful attack is the FGSM attack by far. It reduces the accuracy to 0 immediately and it is not detectable by the human therefore it only affects the bots as we wanted when we want to create a CAPTCHA which is successful as it has a purpose of distinguishing the bot users among human users.

References

[1] Bursztein, E., Martin, M., & Mitchell, J. (2011, October). Text-based CAPTCHA strengths and weaknesses. In Proceedings of the 18th ACM conference on Computer and communications security (pp. 125-138). ACM.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017. doi:10.1145/3065386