Small Project in Artificial Intelligence and Cybersecurity

Automated CAPTCHA recognition using a Convolutional Neural Network

Thomas Mérien Student Number 12225618, Winter term 22/23

1 Introduction

CAPTCHA is an acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart". These tests are very commonly used in cybersecurity to prevent robots from falsifying cookies, creating fake accounts, or spamming comment sections for example. A wide variety of companies use them in their website.

There are several types of captchas:

• Image-based Captchas: These captchas require the user to identify objects, letters, or numbers within an image.

• Audio-based Captchas: These captchas play a series of distorted audio clips and require the user to transcribe them.

• Text-based Captchas: These captchas ask the user to type a specific word or phrase displayed on the screen.

• Math-based Captchas: These captchas require the user to solve a simple math problem, such as addition or subtraction.

• Behavior-based Captchas: These captchas analyze the user's behavior on a website, such as how they move their mouse or how quickly they complete a task, to determine if they are human or not.

• ReCAPTCHA: This is a type of captcha developed by Google that uses advanced algorithms to determine if a user is human based on their browsing behavior.

• Honeypot Captchas: These captchas use hidden fields or form elements to trick bots into revealing themselves. However, image-based captchas are the most commonly used in the web.

Recent progress in artificial intelligence and deep learning has made possible for algorithms to solve captchas, which have been specifically designed to be solvable by humans and only humans. In this project, we will create a convolutional neural network (CNN) to solve image-based captchas automatically.

All of the code is publicly available following the link:
https://github.com/tuiguillunt/Small_project.git

2 Motivation

Cracking certain types of captchas using an algorithm is already possible today, such as image-based algorithms. This is one of the most important points in cybersecurity, because they are used in some of the most critical sector in our modern societies such as banking (to prevent robots to fraudulently transfer money for example), health (to protect personal health data) or government services.

Even considering their utility, solving captchas each time we enter a website is a real annoyance, especially when we missclick or misstype the answer and have to re-do it a second time. Building an automated captcha solver was a great motivation.

There are plenty of methods to automatically solve captchas, such as:

• Recurrent Neural Networks (RNNs): which are particularly effective for solving sequential or time-series problems, such as captchas with letters or numbers that must be entered in a specific order.

• Support Vector Machines (SVMs): which are commonly used for classification problems and can be applied to certain types of captchas, such as those that involve binary classification (e.g., "click on all the images with cars").

• Genetic algorithms: which use evolutionary principles to generate solutions to problems, such as finding the correct sequence of letters or numbers in a captcha.

• Optical Character Recognition (OCR): which is a method for identifying text within an image and can be used to solve certain types of captchas, such as those with distorted or warped text.

However, as my previous attempts to work with CNNs were unsuccessful, I have decided to use a CNN to improve my understanding of this technology.

3 Dataset

The dataset contains a total of 1070 captchas. Each captcha has 5 letters or numbers written, and the test consists of recognising them. Here are some example of images:

Figure 1: 2mg87

Figure 2: 3d7bd

The dataset was downloaded on the website Kaggle and is royalty-free.
https://www.kaggle.com/datasets/fournierp/captcha-version-2-images

4 Objective of the project

The objective of this project is to create and optimise a CNN to solve captchas with the highest accuracy possible. The models proposed in the literature about captcha solving are all superior to 95% [1, 2, 3], which we will consider as the minimum accuracy to reach, in order to successfully complete this project.

In order to test our model, the data is split in two parts, with 960 captchas used for training and 110 captchas used for testing. A 90 to 10 ratio is commonly used for CNNs.

5 Implementation

5.1 Data treatment

In order to be processed by the CNN, it is necessary to sort the images into classes. Since the captchas way contain letters and numbers, there will be 36 classes: one for each letter and one for each number. An image will be associated with a list of 5 classes.

In order to sort each character separately into one of the 36 classes, it becomes necessary to detect each character and process it individually. To do this, we will detect the contours of each letter using grayscale. Each letter will be taken independently in a rectangle and put in the training data.

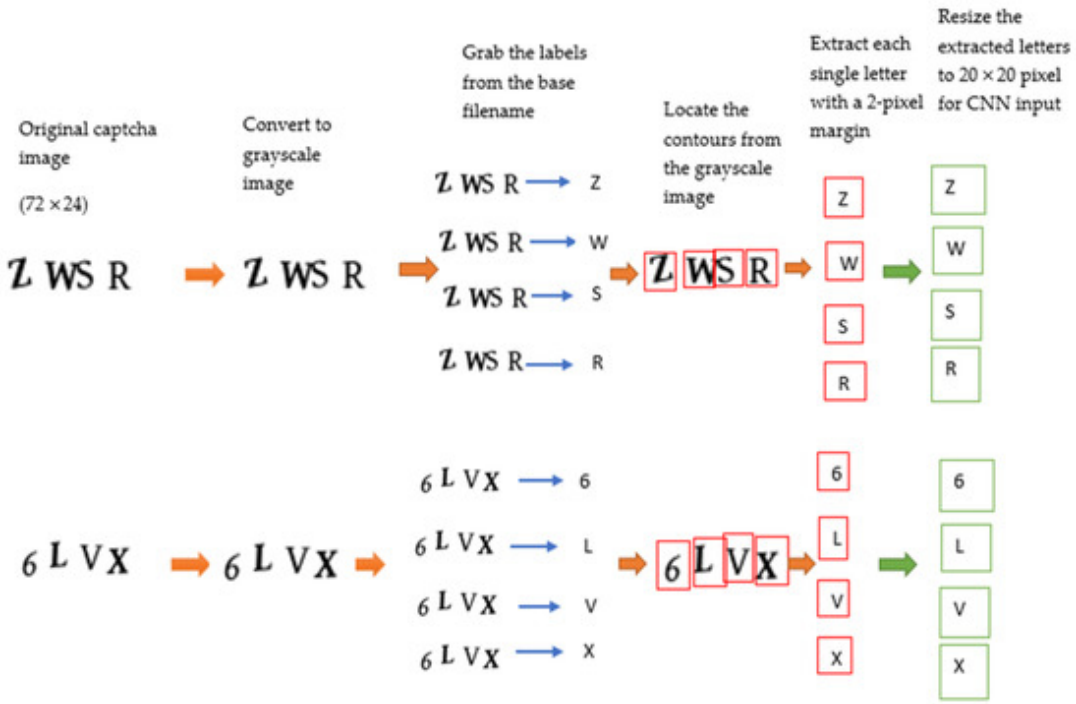


Figure 3: Grayscale method used (image from [1])

However, since characters can be linked with a line in our captchas, we will have to find the correct settings for the binarization to keep our characters divided. Otherwise, there would be only one rectangle found encircling the entirety of the text.

5.2 Description of the CNN

Input layer: The input data consists of single-channel, binarized grayscale images with a size of 20x20.

• Convolution layer C1: uses 64 3x3 convolution kernels, a ReLU activation function, and an input size of 20x20x1. This layer has $(3 \times 3 \times 1 + 1) \times 64 = 640$ trainable parameters.

• Convolution layer C2: uses 64 3x3 convolution kernels, a ReLU activation function, and an input size of 18x18x32. This layer has $(3 \times 3 \times 32 + 1) \times 64 = 36{,}928$ trainable parameters.

• MaxPooling layer P3: applies MaxPooling algorithm to the output of layer C2 for dimensionality reduction, with a pooling window size of 2x2.

• Dropout layer D1: applies Dropout to the output of layer P3 to reduce overfitting.

• Flattening layer F4: transforms the output of layer P3 into a 1D vector.

• Dense layer D5: a fully connected layer with a ReLU activation function and 256 neurons. This layer has $(18 \times 18 \times 64 + 1) \times 256 = 524{,}544$ trainable parameters.

• Dropout layer D2: applies Dropout to the output of layer D5 to reduce overfitting.

• Dense layer D6: a fully connected layer with a Softmax activation function and a number of neurons equal to the number of classes to predict.

Therefore, the model has a total of 640 + 36,928 + 524,544 = 561,152 trainable parameters.

We remind here the formulas to determine the number of trainable parameters:

• for a convolution layer: $(kernel\_size \times input\_channels + 1) \times number\_of\_filters$
• for a dense layer: $(number\_of\_input\_neurons + 1) \times number\_of\_output\_neurons$

Having more than 500,000 trainable parameters is considerably larger than what can be found in scientific literature for captcha solving which is of the order of tens of

thousands [3]. This creates a high risk of overfitting: the higher the amount of trainable parameters are, the more chance there is that the model will reject test images that were not part of the training dataset. In order to prevent this problem, we will add two dropout layers. They will allow to ignore some links to reduce overfitting and so potentially increase accuracy.

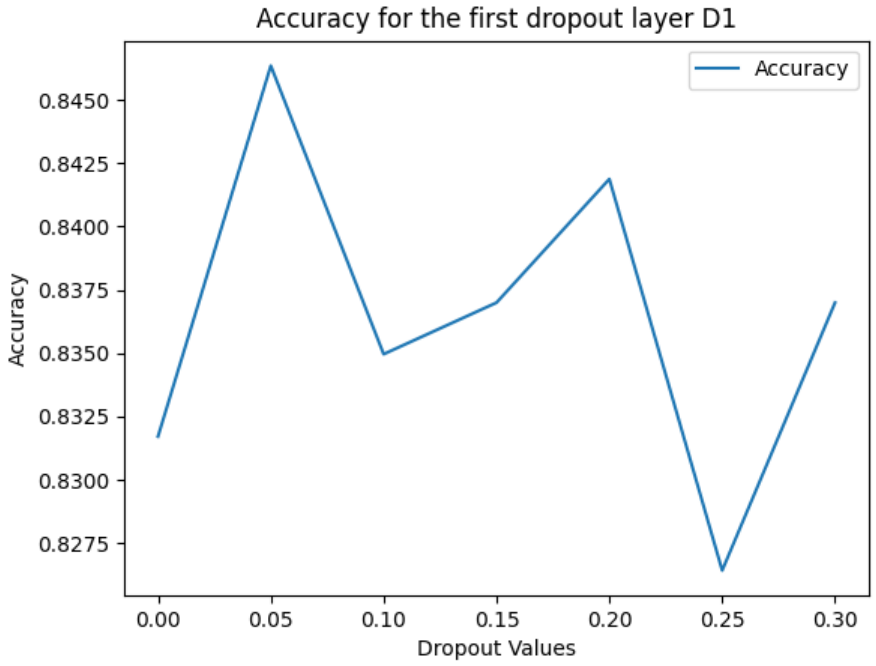
6 Determination of optimal parameter values

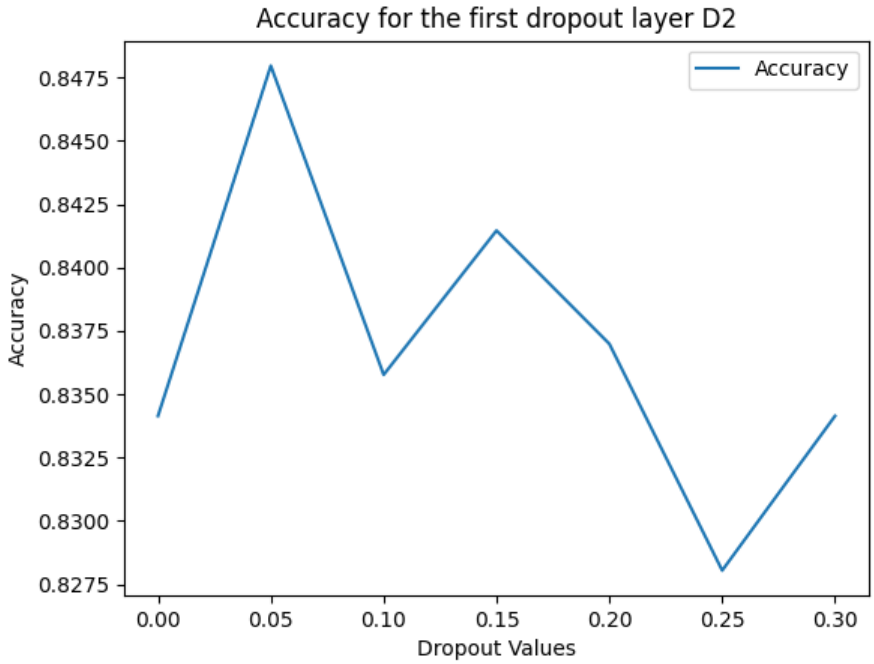
In this part, we want to determine which values are ideal parameters to maximise the accuracy of our model. Each value of accuracy presented is the average accuracy from 20 measurements. The code used to obtain such results is also provided in the archive.

For each measurement, all the other values are set to default values given in the description of the CNN

6.1 Dropout layers

Regarding the dropout layers that we added, it seems the dropout rate has a negligible influence on the accuracy.





We can see the accuracy being altered by 2% at best. This suggest that the dropout layers were superfluous. However we can see that the optimal dropout rates are 0.05 and 0.05. Be aware that the accuracy was measured, for the first dropout layer, with the second with a rate of 0. The accuracy for the second dropout layer was measured with the optimal value of 0.05 for the first dropout rate.

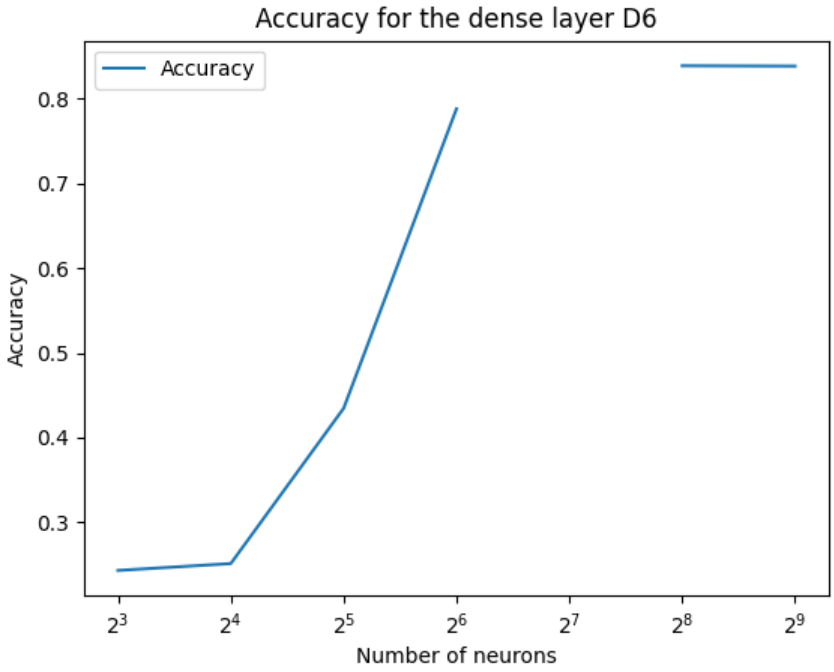
6.2 Dense layer

Having a number of neurons that is a power of 2 is not compulsory, but it is commonly used in all articles available on the subject. [1, 2, 3] It is due to practical considerations in terms of optimizing memory and algorithm performance. By using powers of 2, it is easier to optimize the use of memory and computing resources, as many processors and GPUs are designed to efficiently handle operations in powers of 2. this is also true for a convolutional layer size.

We must determine the number of neurons for the dense layer D5. this time, the more neurons we add, the higher will be the response time.

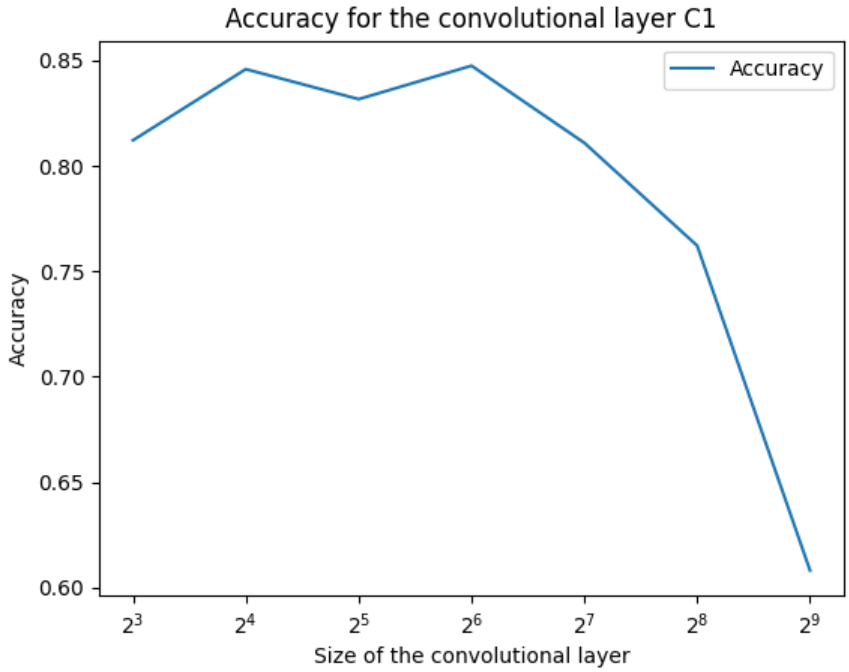
Note: there was an error on the measurement for 2^7 neurons

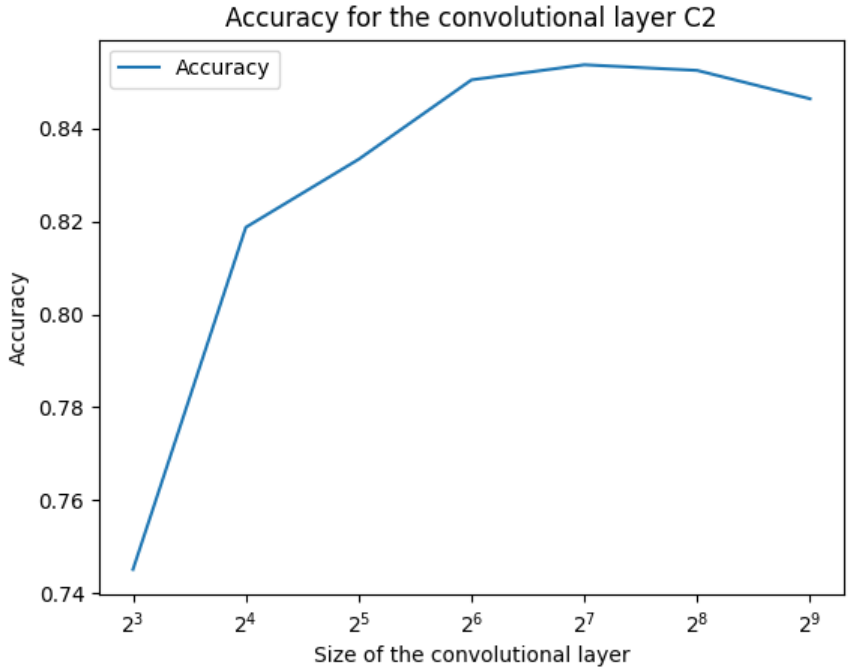
We can see that with $2^8 = 256$ neurons, we have the highest possible accuracy. The response time is reasonable, but it depends on the computing power of the machine.



6.3 Convolutional layers

The size of the convolutional layer seems to drop after $2^6 = 64$ where it is at its highest. One hypothesis to explain it is that a large convolutional layer is too receptive to noise. A large convolutional layer is still important for the quantity of informations that can be processed. Also, if the layer is too small, some informations will not be processed, so there should be another limit about the minimum size of the convolutional layer as well.


Accuracy for the convolutional layer C1
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
Accuracy
2^3
2^4
2^5
2^6
2^7
2^8
2^9
Size of the convolutional layer


Accuracy for the convolutional layer C2
Accuracy
0.84
0.82
0.80
0.78
0.76
0.74
Accuracy
2^3
2^4
2^5
2^6
2^7
2^8
2^9
Size of the convolutional layer

These figures are opposed to one another, with a drop in accuracy when the second convolutional layer is too small and a drop in the first one when the first convolutional layer is too large. It tends to confirm both previous hypothesis about small and large convolutional layers.

There seems to be an optimal range of values for the size of convolutional layers. It is dependent on each model. The optimal values seem to be 64 and 128. We will then change the default value previously set from 64 to 128.

6.4 CNN

Now that we have a CNN with its optimized parameters, the maximum accuracy can be determined. It is in average 0.86, which means the model predicts correctly 86% of the captchas tested.

7 Validation of the model

The minimal accuracy of 95% is not met, which makes this model less performing than expected.

One of the possible reasons to explain this failure could be the dataset that is composed of captchas with several characters linked with lines that aren't part of the text to recognise. This could have caused problems on some captchas when we are splitting each letter apart.

Another reason could be the structure of the model. I tried to create a simplified version compared to what was done in literature, so that I could check the optimal values i na reasonable time. However, the structure is not based on what already exist, with the number of each layer and their order.

The dropout layers are completely useless in the model. Overfitting was not a problem because the accuracy didn't change when the dropout ratio was altered. In fact, none of the articles mention a dropout layer in their model. This was a mistake to add some when I had a certain difference in the amount of parameters. If I had to restart the project, these layers would be removed.

An accuracy of 86% is still a decent result which shows that the model works correctly and is not to be thrown in the bin. If I had more time, I could have created several CNN models with a different amount/order of layers to try and improve accuracy.

As previously mentioned, the dataset is royalty-free. The articles which I referenced are available on Google Scholar and none of them specify any restriction of use. The references were made using the information specifically provided for this purpose in each article. I have put them in PDF format the 'literature' folder of my archive.

8 Conclusion

This project shows that the days of image-based captchas are counted. The deep learning technology is being improved day by day. The usage of CNN seems to be limitless, which made me wonder what else I can do with deep learning methods.

During this project, I spent three weeks looking for several subjects and searching for relevant literature on each before deciding to go on with captchas and starting to code, while implementation and testing took 4 months.

References

[1] Stephen Dankwa and Lu Yang. "An Efficient and Accurate Depth-Wise Separable Convolutional Neural Network for Cybersecurity Vulnerability Assessment Based on CAPTCHA Breaking". In: *Electronics* 10.4 (2021), p. 480. DOI: 10.3390/electronics10040480.

[2] Ashwani Kumar and Aditya Pratap Singh. "Contour Based Deep Learning Engine to Solve CAPTCHA". In: 1 (2021), pp. 723–727. DOI: 10.1109/ICACCS51430.2021.9441737.

[3] M. Irfan Uddin, Zhong Wang, and Peibei Shi. "CAPTCHA Recognition Method Based on CNN with Focal Loss". In: *Journal of Electrical and Computer Engineering* 2021 (2021). DOI: 10.1155/2021/6641329.