**CAPTCHA solver for the Croatian cadaster**

Roberto Anić Banić

**Introduction**

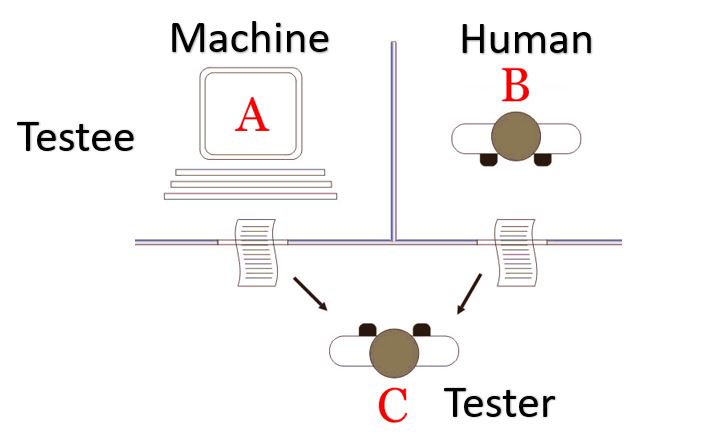
Nowadays, probably everyone with a computer or a smartphone and with access to the Internet has heard of the term CAPTCHA. In simpler words explained, it is one of those annoying popups that appear on websites where you have to do a simple task (e.g., write the random letters which are in a given picture, or click on some boxes containing a certain object) to prove that you are human. These tasks may appear silly to us and some people probably wonder why some websites (like, for example, <https://katastar.hr/#/>) use them.

In this project, I’ll explain more precisely what a CAPTCHA is, which type of CAPTCHAs exist and what their purpose is. I’ll analyze a concrete website and its CAPTCHA, namely, the Croatian cadaster, try to make a deep-learning based solver for it and analyze the security aspects of this concrete CAPTCHA.

**What is a CAPTCHA?**

CAPTCHA is an acronym that stands for “Completely Automated Public Turing test to tell Computers and Humans Apart”.

The Turing test, or as the creator called it, the imitation game, was proposed by Alan Turing in 1950. The Turing test can be explained as followed: there are three participants in this “imitation game”, player A, player B, and interrogator C. The goal of player A and player B is to prove to the interrogator that they are human, even though only one of them will be human. The other player will be a machine. The interrogator’s task is to determine which player is the human and which is the machine. He does that by asking each some questions and analyzing which answers are most human-like.



*Figure 1:* Turing test

The CAPTCHA technology can be furthermore be explained as the following: “a test to prevent automated software from accessing a website or data by requiring visitors to the site to solve a simple puzzle (typically by reading and transcribing a series of numbers or letters from a distorted image) in order to gain access”.

**Purpose of CAPTCHAs**

As mentioned in the definition, CAPTCHA’s purpose is to prevent automated software (e.g., bots) from frequently accessing a website and thus prevent server overload and possible misuse of data found on some websites.

For example, one might want to have a CAPTCHA in the following scenarios:

1. **Preventing spam** on forum-like websites.
2. **Limiting registration** of accounts for a service or social media website to prevent fake accounts and possible frauds.
3. **Stop brute force attacks on accounts** where a hacker would try to access another user’s account by randomly guessing their password.
4. **Protecting poll** **integrity** of online voting where someone might want to try rigging the results by using bots for voting.

This is an incomplete list of cases where one may want to consider using a CAPTCHA for their website since there are a lot of other use cases. Still, this list gives us an idea of why we need such a security mechanism in the first place.

**Types of CAPTCHAs**

There are many possible categorizations of CAPTCHA. Once such categorization would be the following:

1. **Text CAPTCHA**. The user needs to write the letters seen on a picture into a text box.
2. **Math problem**. These CAPTCHAs request the user to solve a simple math equation.
3. **Word problem**. Similar to math problem CAPTCHAs but this time, the math problem is given with words instead of equations.
4. **Time-based**. These CAPTCHAs calculate how much time a user needed for solving a problem and determine whether or not the user is human or not depending on the result.
5. **Confidence CAPTCHA**. This is the type of CAPTCHA where the user is asked to click on all images containing a certain object.
6. **Sweet CAPTCHA**. Similar to the confident CAPTCHA, the user will be asked to click on images but this time, the user is required to click on all boxes containing a certain object.
7. **Biometric CAPTCHA**. One such example would be a fingerprint to unlock a smartphone.
8. **reCAPTCHA**. This is the one where a user needs to click on a checkbox. But in reality, it works by using browser cookie tracking and other technologies in the background.
9. **Honeypot**. A CAPTCHA that adds hidden fields on a website which only a bot could mistakenly type into.

**Problem analysis**

Now, after having an overview of what a CAPTCHA is, why it is needed and which types there are, let’s focus on my specific task.

I chose to implement a solver to the CAPTCHA found on the Croatian cadaster website. It is a simple-looking text CAPTCHA.

Whether or not this CAPTCHA really is easy to bypass will be determined by the results of this project.

In the following chapters, I’ll discuss the input dataset, the implemented model and give a few remarks regarding the security aspects of this specific CAPTCHA.

**Data analysis**

The input data I'll be working with are images similar to this one:



*Figure 2:* CAPTCHA example

These pictures are generated on the following [website](https://oss.uredjenazemlja.hr/servlets/kaptcha.jpg).

After generating a few pictures, I noticed these regularities:

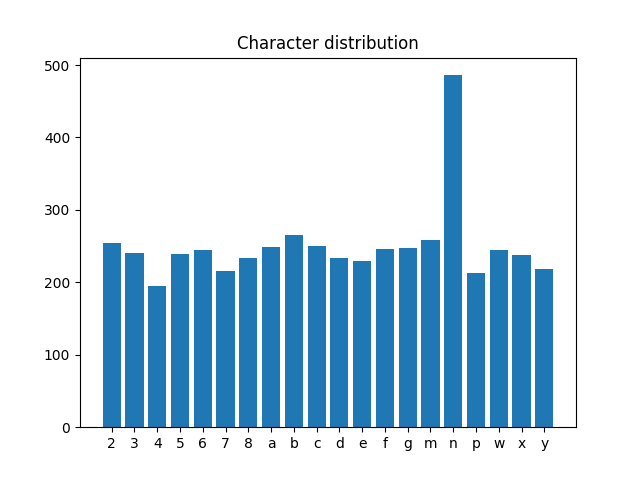
1. The pictures are always the **same size**, 276 x 50.
2. There is always a **border** which is **1 pixel thick** but JPEG artefacts can expand it into 2 pixels.
3. The pictures always have **5 characters** which are centered **in the middle**.
4. The characters in the picture are always either **numbers** or **lowercase English letters**. This gives us a total of 365 possible outputs.
5. The pictures are **grayscale**, which simplifies the task a lot. If we had a picture colored with a colored background, it’d be much harder to filter noise from actual information.

The noises and irregularities in the pictures are also somewhat regular:

1. The **background** is **gray to white**, going from left to right.
2. Some **letters** are a little **blurry**. Usually the 3 centered letters. Letters can sometimes be so blurry that they are not recognizable.
3. Some **letters** (but not all) will be **very close to each other** which might make it harder to detect them as separate objects.
4. These pictures always have **two lines** as a distraction but ONLY one of them strikes through the characters.
5. Characters in the pictures are **slightly rotated, translated, curved**, and/or differently **scaled**.

**Dataset creation**

By using a basic Python script I proceeded to personally download and label exactly 1000 captcha images. This yielded a dataset from which I could theoretically extract statistically significant data. Using these labels, I found out that the captcha only uses characters displayed in *Figure 3*. From this histogram we can see that a lot of characters from the standard english alphabet are missing, alongside the digits 0, 1, and 9. An outlier that is most certainly visible is the character „n“.



*Figure 3*: Character distribution

I have found that there are 2 distinct types of characters that the captcha validator treats like the character „n“. These are a normal „n“ and a very vertically squished „h“. These are visible on *Figure 4*.



*Figure 4:* N variations

**Data preprocessing and noise removal**

Before we give the input data to any neural network, we first need to preprocess the data and remove as much noise as possible.

The first thing to concentrate on is to make the characters as dark as possible and the background as light as possible.

Removing the background turned out to be an easy task. The gradient is easily removed by filtering out characters lighter than a specific grayscale color, namely 180. I found that number after trying out a few ones on a few pictures. In a more general case, with a less hardcoded CAPTCHA, it'd take some time to optimize this hyperparameter (a parameter we, as humans, input into our models). Theoretically speaking, one possible way would be to just try out different numbers and see for which one the model yields the best results. Another approach would be to let the model learn which the best value for this hyperparameter is. This falls into the category of unsupervised learning and would probably be an overkill since this hyperparameter space is not large (at most 256 values but always less than that, since we do not want to lighten the characters on the picture).

Borders were removed by cutting off 1 pixel on each side of the image (cutting off 2 resulted in too big of a data loss, and it did not matter for the CNN), and a persistent top right corner JPEG artifact was manually removed, as it can be seen on the following code snippet:

img: ndarray = cv2.imread('./kaptcha.jpg', 0)

# clean border

img[0:, 0:1] = 255

img[0:, -1:] = 255

img[0:1, 0:] = 255

img[-1:, 0:] = 255

# clean background

img[img > 180] = 255

img[0:15, 0:15] = 255

After that, we have to proceed to line removal. At this time, I have identified the beginning of both lines and the end of the longer one using the following code:

start\_col = next(filter(lambda x: min(x[1]) < 255, list(enumerate(img.T))))[0]  
start\_row\_one = next(filter(lambda x: x[1] < 255, list(enumerate(img.T[start\_col]))))[0]  
start\_row\_two = next(filter(lambda x: x[1] < 255, reversed(list(enumerate(img.T[start\_col])))))[0]  
end\_col = next(filter(lambda x: min(x[1]) < 255, reversed(list(enumerate(img.T)))))[0]  
end\_row = next(filter(lambda x: x[1] < 255, list(enumerate(img.T[end\_col]))))[0]

It seems that only the removal of the bigger one is going to pose a task because the smaller one NEVER goes past column 75 (starting from 1) after the first cleaning of the image was done. This allows us to clean the image to that column. This can be simply done using the following code:

img[:, :75] = 255

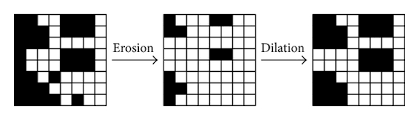
This yields the results shown in *Figure 5.*



**

*Figure 5*: Before and after background and one line removal

The next and more challenging task is to remove the line striking through the characters. After researching how to approach this task, I found out about *morphological image processing*. It is a collection of non-linear operations which affect the shape or morphology of features in an image. The two fundamental methods are *erosion* and *dilatation*. In simpler terms explained, erosion shrinks parts of an image, while dilatation does the opposite – it widens parts of an image.



*Figure 6*: Example of erosion and dilatation

While reading a research paper called *Deep-CAPTCHA: a deep learning-based CAPTCHA solver for vulnerability assessment,* in the *Preprocessing* chapter I learned about the term *median filter*.

I used erosion and the median filter in the following code snippet:

img = ~img  
img = cv2.erode(img, np.ones((2, 2), np.uint8), iterations=1)  
img = ~img  
img = scipy.ndimage.median\_filter(img, (4, 1))  
img = cv2.erode(img, np.ones((2, 2), np.uint8), iterations=1)  
img = scipy.ndimage.median\_filter(img, (1, 1))

The final result after applying all the mentioned noise removal methods can be seen in *Figure 6*.









*Figure 7*: Before and after noise removal

As we can see on the right side of Figure 7, most of the noise is removed – on some pictures better than on others, but still, the result is satisfying. Unfortunately, this proved to be useless when training the CNN because it only degraded the results. So, while data cleaning is a very important step, in this project the original input was better left alone if better results were to be achieved.

**Train and test dataset**

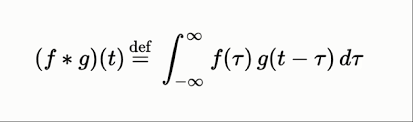
To be able to train our model and to evaluate our results, I will be dividing the initial dataset into a train and test dataset. The train dataset will contain 80% randomly sampled labeled CAPTCHAs, while the test data set will contain the remaining 20% of the original dataset.

**Model**

For solving this CAPTCHA, I used TensorFlow and implemented a convolutional neural network (CNN) with it.

The name of this network implicates that the mathematical operation convolution is performed.

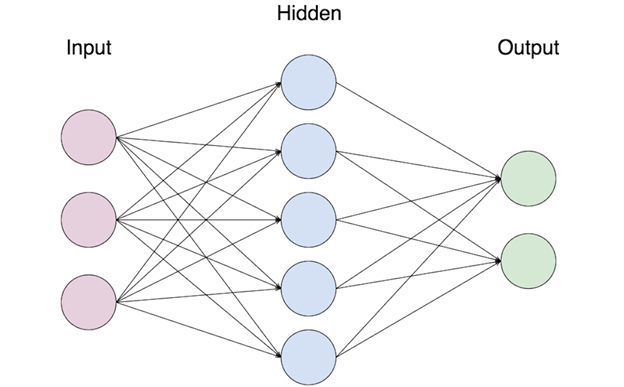
Convolution is a way to derive a single function from two given functions. The formula can be seen on *Figure 8* but I will not go much into the mathematical details.



*Figure 9*: Mathematical definition of convolution

Just like a basic feed-forward neural network, the CNN consists of three parts:

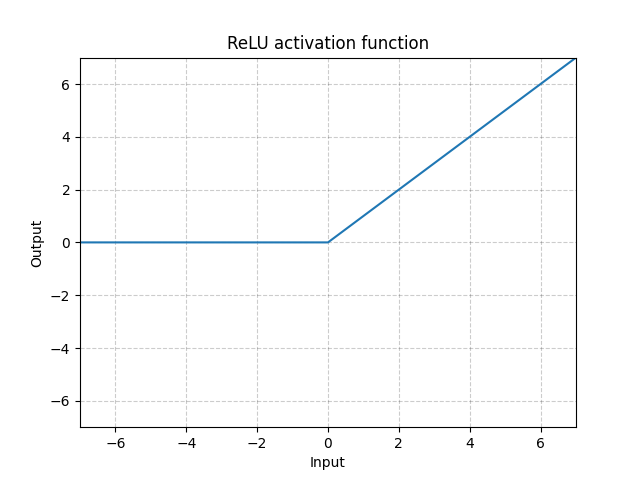
* input layer
* hidden layer
* output layer



*Figure 8*: Generic architecture of a neutral network

It is important to note that the hidden layer does not have to be a single layer – a neural network can of course have multiple hidden layers.

The input of our CNN is a CAPTCHA image and the output is 5 characters which the model predicted. The hidden layer is a bit more complicated. The hidden layer performs the convolutions and contains different types of layers, which will be described later in this chapter.

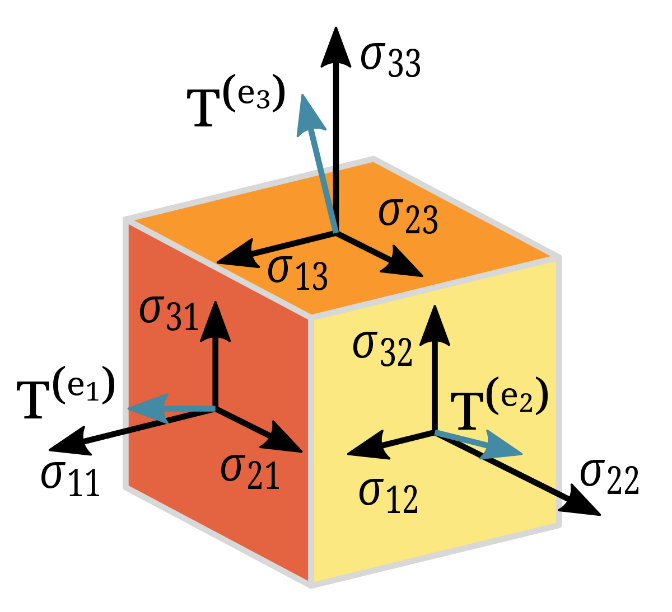
The activation function for in a CNN is typically the ReLU function (also known as the ramp function).

*Figure 10*: ReLU function

A CNN has the following hidden layers:

1. Convolutional layers
2. Polling layers
3. Fully connected layers

The input to a **convolutional layer** is *tensor*. A tensor is a dimensional mathematical object, similar to vectors and matrices.



*Figure 11*: Tensor

The shape of our tensors is the following: (number of input pictures) × (picture width) × (picture height) × (input channels).

The neural network does not know how many input pictures it’ll be given. The input channel refers to the number of colors the picture uses (e.g., RGB). One channel means that the given picture is in grayscale.

The convolutional layer applies a filter to the input and creates a feature map. The feature map gives us some information about the image, like corners or edges. Later in the network, the feature map is given to other layers to learn several other features of the input image.

The **pooling layer** reduces the dimensionality of the data by combining the outputs of neuron clusters at one layer into another single neuron in the next layer. The max pooling, which I used in my CNN, takes the maximum value of each cluster of neurons in the feature map.

The **fully connected layer** connects every neuron in one layer to every neuron in another layer.

This layer is just before the output layer and in the output layer. The flattened data goes through these layers and is then classified.

In a high-level overview, these layers (and the whole CNN) can be seen on *Figure 12*.

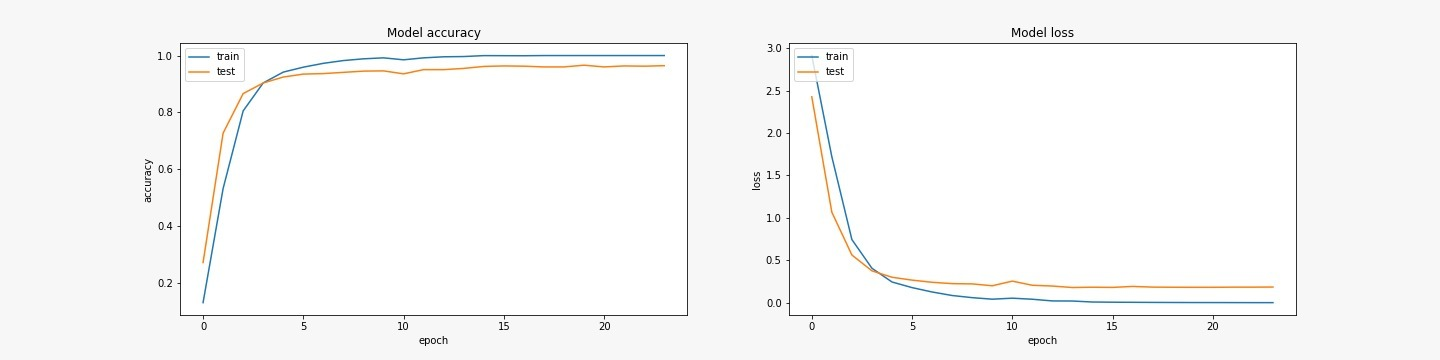


*Figure 12*: CNN architecture

**Model evaluation**

To evaluate how good our model predicts, there are two possible approaches her. One approach would be to look at how many times the model predicted all five characters correctly and treat it as a miss even if one character was wrongly predicted. The other approach is to look at the single characters and see how many times the model guesses the character correctly.

For evaluating this CNN, I used the second approach.

The accuracy I got was between 0.9 and 0.95.

*Figure 13*: Model accuracy and model loss

If we interpret these number as the possibility the model will guess the character correctly, there is a 60% - 77% chance the model will successfully solve the CAPTCHA.

**Conclusion**

With the most basic knowledge of deep learning and plenty of searching on the Internet, I would say I successfully implemented a solver for the CAPTCHA found on the Croatian cadaster website.

Even though I am satisfied with the results, there are two things I would like to comment on here.

The first would be the possible improvements of the model accuracy. The obvious one would be the size of the dataset. Since I had to label all pictures by hand, I did not have a large dataset to begin with.

The second possible improvement would be to optimize all the hyperparameters (and there are a lot of them) and to see if changing any would yield better results.

Another thing to try out would be to play with the CNN – maybe adding one or multiple layers would improve the results.

The second thing I would like to comment on are the security aspects of this CAPTCHA. The Croatian cadaster contains very confidential data, like the OIB (PIN) of Croatian citizens, which land they possess, etc.

It is in way say sad that someone with basic knowledge could crack this “security system”. Imagine just how someone with a little more deep learning knowledge could exploit these easy solvable CAPTCHAs. Text CAPTCHAs are very outdated today, since we have powerful tools which can easily solve them. It would be better to add confidence CAPTCHAs or sweet CAPTCHAs to improve the security aspects of this website.

**References**

[1] <https://en.wikipedia.org/wiki/CAPTCHA>

[2] <https://en.wikipedia.org/wiki/Turing_test>

[3] <https://www.merriam-webster.com/dictionary/captcha>

[4] <https://www.imperva.com/learn/application-security/what-is-captcha/>

[5] <https://www.tsohost.com/blog/6-reasons-why-your-website-needs-a-captcha-form>

[6] <https://www.educative.io/edpresso/what-are-the-types-of-captchas>

[7] <https://medium.com/analytics-vidhya/solving-noisy-text-captchas-126734c3c717>

[8] <https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)>

[9] <https://towardsdatascience.com/navigating-the-hyperparameter-space-32f48be2c12d>

[10]<https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm>

[11] <https://arxiv.org/pdf/2006.08296.pdf>

[12] <https://www.kaggle.com/arnrob/captcha-2-solutions-to-break-them-99>

[13]<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

[14] <https://en.wikipedia.org/wiki/Convolutional_neural_network>

[15]<https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1-convolution-operation>

[16] <https://en.wikipedia.org/wiki/Tensor>

[17]<https://medium.com/featurepreneur/understanding-the-concept-of-channels-in-an-image-6d59d4dafaa9>

[18]<https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>

[19] <https://www.upgrad.com/blog/basic-cnn-architecture/>

**Pictures**

[1] <https://www.how2shout.com/what-is/what-is-turing-test-and-it-used-for.html>

[2] <https://www.researchgate.net/figure/The-average-AR-values-of-test-images-a-JPEG-quality-b-contrast-c-Gauss-blurring_fig13_264795127>

[3]<https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1-convolution-operation>

[4] <https://epos.myesr.org/posterimage/esr/ecr2019/147732/mediagallery/828073>

[5] <https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html>

[6] <https://en.wikipedia.org/wiki/Tensor>

[7]<https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c>

[8] <https://www.upgrad.com/blog/basic-cnn-architecture/>