Fermi National
Accelerator Laboratory

Summer internship

Final report

Applying Deep Learning techniques to improve semantic image CAPTCHAs

Author:
Paolo Didier Alfano

Supervisor:
Mine Altunay
Jeny Teheran

October 24, 2018

Abstract

New developments in artificial intelligence and deep learning are offering advanced solutions to mitigate the increasing security threats; for example, malware detection is being boosted by neural networks that are trained to distinguish a file with malicious content in real-time and at a detection rate higher than the traditional detection mechanisms based on signatures and heuristics.
CAPTCHA is a common mechanism to stop bots from accessing a web service and perform malicious actions like automated password hacking, spam propagation, etc. Neural networks have been used with great success to defeat CAPTCHAs leveraging on deep learning technologies.
The aim of this project is to write deep learning algorithms to extract information from a set of images presented in CAPTCHAs to identify which of the given images are semantically similar to the sample image. We also produced a series of recommendations to built more robust CAPTCHAs.
This project also expand our knowledge on the implementation of deep learning techniques that could be applied to malicious code detection.

Contents

1 Introduction

A very common tool used in cybersecurity to distinguish human from computers is *CAPTCHA*. This term refers to a generic challenge-response algorithm used to determine whether the user is human.
The first type of CAPTCHA was invented in 1997, although the most common type of CAPTCHA, the one made with distorted characters and numbers, was introduced in 2003.
As we will see, during the next fifteen years CAPTCHA evolved in different directions and at the present time we have different type of challenge.
At the same time, during the last few years, deep learning techniques were developed and used in a very large number of research field: computer science, engineering, physics...
Since 2012, a particular type of neural network is often used to solve image analysis tasks. This kind o neural network is called *convolutional neural network*. With the victory of the *ImageNet Large Scale Visual Recognition Competition*(ILSVRC-2012) by a convolutional neural network developed by Google, an increasing attention has been focused over this particular model.

Throughout this report we'll see how this particular type of network could be used to solve the CAPTCHA problem and how we can develop some recommendations to make more robust CAPTCHA against this type of deep learning attacks.

2 CAPTCHAs

Every time there is an online service a human being can use, this service can be also used in a illicit way from an automated program called *bot*. In fact, without limiting only to the users the access to a service, a bot could do how many requests it wants, resulting in a illicit use of the service or reducing the access to the service itself.
CAPTCHA is a common tool that wants to prevent this situations by giving the user a challenge to solve, to certify that on the client side there's really a human being.
In the following page we'll see the most common CAPTCHA types used in the past and nowaday.

2.1 CAPTCHA types

Although the first commercial use of CAPTCHA was the *Gausebeck-Levchin test* in 2003 to protect the login page of a website, CAPTCHAs were used for the first time in 1997. Their implementation was based on reverting the guidelines of readability in the *optical character recognition*(OCR). These guidelines were about having similar typefaces, plain backgrounds...
By reverting these guidelines, the firsts CAPTCHAs were similar to what we can see in Figure 1.

This type of CAPTCHA bring some disadvantages:

- User experience: usually this type of CAPTCHA can be hard to solve even for human. Usually they are not perceived like an easy and intuitive tool to use.
  At the same time, in certain jurisdictions, site owners could become targets of litigation as this type of CAPTCHA can discriminate certain people with disabilities.

- Security: with the recent machine vision development, this text-based

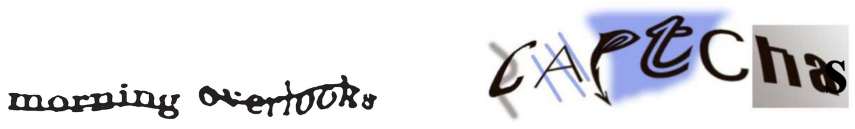


Figure 1: Two examples of CAPTCHA

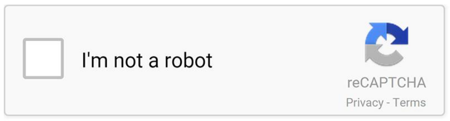


Figure 2: A reCaptcha widget

CAPTCHAs are not considered secure anymore, as characters recognition challenge are easy to solve nowaday.

2.2 Google reCAPTCHA

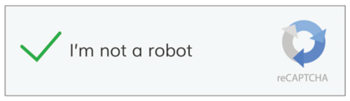
Developed by the Carnegie Mellon University and later acquired by Google, reCAPTCHA is a new type of CAPTCHA that doesn't make use of textual CAPTCHA[1]. A reCAPTCHA widget it's showed in Figure 2

Every time we click over the reCAPTCHA widget, a score is assigned to the user, based over his Google's login information, how the user has moved the mouse pointer through the web page and other information.
Basing on this score, Google can sends two different challenge:

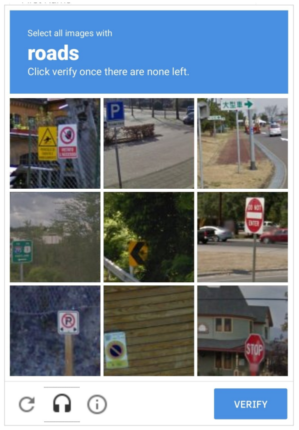
- No CAPTCHA reCAPTCHA: when the user receives an high score he doesn't need to solve any CAPTCHA at all. This situation is showed on the left part of Figure 3
- reCAPTCHA challenge: every time a user is not fully trusted, Google sends a challenge the user must solve to proceed in using the service. Usually is a sequence of visual challenges about recognizing whether an image contains an object. A typical reCAPTCHA challenge is showed in the right part of Figure 3

One important thing to notice is that every reCAPTCHA challenge is related to a "category". For example the user could be asked to recognize cars, crosswalks, road signs...
The most common categories are the following:

[1]Textual CAPTCHA were used in the reCAPTCHA system during it's early months but the textual challenge was removed later.



(a) No CAPTCHA reCAPTCHA challenge



(b) One of the challenge received when the user is not fully trusted

Figure 3: The two possible situations in a reCAPTCHA challenge

- Road signs
- Store fronts
- Cars
- Vehicles
- Motorcycles
- Roads
- Bicycle
- Bus
- A fire hydrant
- Mountains or hills
- Taxis
- Sidewalk

In the following pages the category is also called *hint* or *semantic hint*.

2.3 Attack CAPTCHAs

Since their inception, solving CAPTCHAs has been one of the most interesting and complex challenge in the computing security field.
One of the most popular technique used to solve CAPTCHAs in the past was cheap human labor. After some time automated techniques were developed to solve the firsts type of CAPTCHA.
As usually happens in the security field, after the inception of reCAPTCHA,

different methods to solve them were proposed. One of the most known method is the one proposed by Sivakorn et al.[1].
Our purpose is to understand whether a deep learning tool can solve this challenge, in order to produce some security recommendations to make CAPTCHAs more robust.

The tool we decided to use is one of the most popular and effective in the image analysis field: *convolutional neural network*.

3 Designing the architecture

To verify whether our idea was good, we started by designing the architecture.

3.1 General architecture

The first thing we do was to design an architecture able to solve the reCAPTCHA problem.
As we notice that the categories number is quite small, let's say n, we thought about two different designs, both solving a classification problem:

1. One neural network with $n + 1$ classes: as we have n different categories we can have one class for every category with another generic "negative" category representing every image that doesn't belong to any category.

2. n neural networks with two categories each: with this second approach we develop a different neural network for every category.
   Every neural network has two classes: *positive* representing the image containing the entity of the category, *negative* representing the images not containing that entity.
   This second architecture is showed in Figure 4

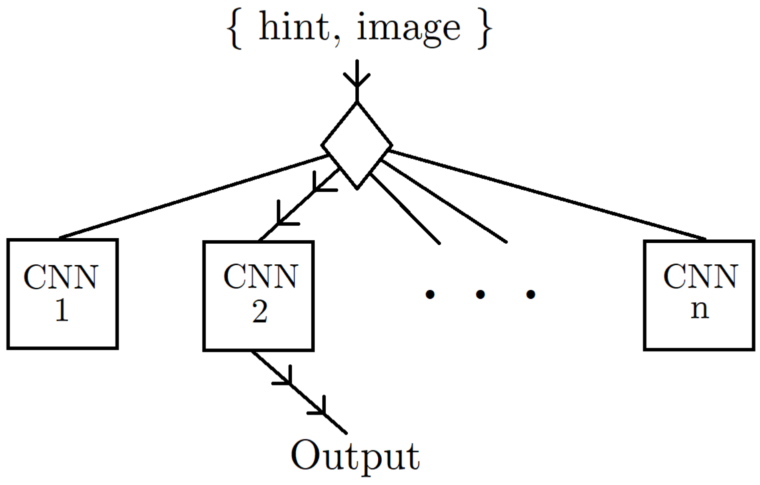


Figure 4: General architecture to solve the reCAPTCHA problem

Even if theoretically the two previous architectures solve the same problem, we decided to use the second architecture with n neural networks for the following reasons:

- Hint, semantic information: the semantic information that comes with the hint(i.e. the category name) is easier to use in the second architecture. In fact, as we can see in Figure 4 our program receives as input the image and the hint. With this informations it has to decide whether the image contains the entity indicated in the hint.
  With the architecture shown in Figure 4, is quite simple to use the semantic information as we can run one of the n neural networks by using the hint.

- Easier to test: as we had just a couple of weeks to study the problem, the second architecture seems easier to develop and test. In fact in neural networks the importance of having a proper dataset is well known and the second architecture allows us to build/retrieve just two dataset and start developing and testing over a reduced reCAPTCHA problem.

As we decided to use the second architecture we started by selecting one category of the reCAPTCHA challenge to develop a neural network solving that problem.
In the following pages we'll study the solution to just one of the category of the reCAPTCHA challenge, assuming that if we are able to solve the problem over one category, then we can probably solve it for every category.
So we'll study just the retrieval of one dataset, the developing of one neural network...
The category we decided to use is *road signs*.
By choosing this category we identified our task: given an image, determine whether the image contains a road sign.

3.2 Dataset

Once we decided to work with road signs, the first problem we had to solve was how to get a dataset that we could use to train our neural network.
As our problem is to determine whether an image contains a road signs or not, we needed to teach our neural network what's a road sign and what a road sign is not. To accomplish this task we needed two different types

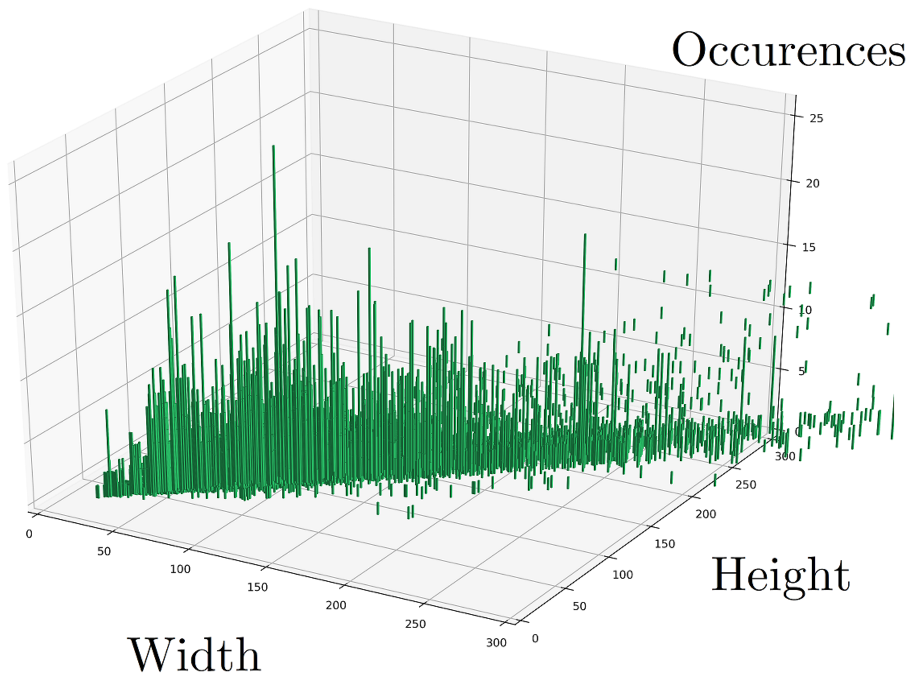


Figure 5: Size distribution of the positive class

of training data. To be more clear, from now on we'll identify the images containing a road sign as *positive* and the images not containing road signs as *negatives*.

- Positive class: to teach the neural network what a road sign is, we built a dataset by merging two different road signs dataset found on the web. The images of the first dataset were taken in the United States, the images from the second dataset from China. It's important to have road sign from different part of the world, as the images of the reCAPTCHA challenge were collected with street view from every country Google can reach.
  By merging this two dataset we obtained a larger dataset made by approximately 10^4 images. The size of these images it's not uniform and it can range between dozens to hundreds of pixels. The size distribution is shown by Figure 5.

- Negative class: it's easy to understand that building a dataset of negative images it's more difficult as it's easier to teach what a road sign

is, but it's not as easy to teach(even to a human!) what a road sign is not. That's because the best way we have to explain what a road sign is not is to take a lot of images not containing road signs.
In order to solve this problem we decided to use training data that are as more similar as possible to the images of the reCAPTCHA challenge: images from Google's Street view.
We retrieve approximately $2 \cdot 10^4$ Google's Street view images from an online dataset. The size of all these images is 205×256

4 Implementation

In order to solve the problem, we started by building one model and then by changing it to obtain better performances. With this approach we can divide our work in four different neural network ”generations”.

4.1 First generation

The first generation of neural network we decided to train was very simple and used just twelve layers[2].

```
# Layer 1
CNN_model.add(Conv2D(16, kernel_size=(3, 3),
    activation='relu',padding='same',input_shape=(imageWidth,
    imageHeight, channels)))
# Layer 2
CNN_model.add(Dropout(0.2))

# Layer 3
CNN_model.add(Conv2D(32, (3, 3), activation='relu',padding='same'))
# Layer 4
CNN_model.add(MaxPooling2D((2, 2),padding='same'))
# Layer 5
CNN_model.add(Dropout(0.2))

# Layer 6
CNN_model.add(Conv2D(64, (3, 3), activation='relu',padding='same'))
# Layer 7
CNN_model.add(MaxPooling2D((2, 2),padding='same'))
# Layer 8
CNN_model.add(Dropout(0.2))

# Layer 9
CNN_model.add(Flatten())
# Layer 10
CNN_model.add(Dense(32, activation='relu',
    kernel_regularizer=regularizers.l2(0.01)))
# Layer 11
```

[2]Considering the convolutional, pooling, batch normalization and dropout layers.

```
CNN_model.add(Dropout(0.2))
# Layer 12
CNN_model.add(Dense(num_classes, activation='softmax'))
```

As we can see, in this simple model we have the first two layers dealing with the input. In the next six layers we have a repeated structure:

- A convolutional layer
- A max pooling layer
- A dropout layer

In the last four layers we have a flatten layer followed by the only one dense layer in the network. After another dropout layer there is the softmax activation function.

By using this first model we obtained the following *accuracy* results

Train	Validation	Test
0.72	0.62	0.61

Table 1: Accuracy results for the first generation

The results showed in Table 1 must be read as following: our neural network is able to distinguish correctly whether the image contains a road sign or not in the 61% of the situations.
Even if we're already doing better than the random guess, the most important thing to notice is the performance degradation within the training accuracy and the validation/test accuracy.
This severe degradation can be considered a sign of overfitting.

4.2 Data augmentation

When we have to deal with situations like the one presented in Table 1 we can be quite convinced that we are dealing with an overfitting situation as the degradation is probably due to a partial memorization of the input in the net. This memorization make the net unable to generalize over new data.
To solve this situation we decided to change both the model and the data:

- Data: adding data using the data augmentation technique

- Model: by increasing the dropout value in the dropout layers

The most important modification is the one about the data.
The *data augmentation* technique consists in ”augmenting” artificially the amount of data we have by creating new images originated by the application of some linear transformations and brightness transformations to the dataset.

We decided to augment our dataset by using an *ImageDataGenerator* object included in the Keras's Image Preprocessing library.
The code of the ImageDataGenerator we used is the following:

```
datagen = ImageDataGenerator(
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        fill_mode='nearest')
```

The code we just showed is saying that every time we generate a new image it can be rotated by 30 degrees, with a vertical/horiziontal shifting of 20% of its size...
We can show in Figure 6 the result of data augmentation.

In the top left part of the image we can see the original image. In the top right a new image that is rotated clockwise respect to the original. In the

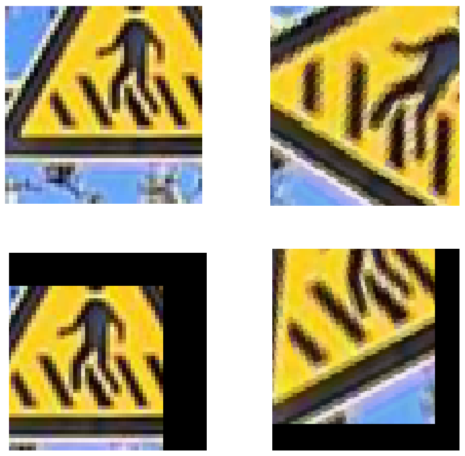


Figure 6: A typical augmentation

bottom left part we can see a shifted image. In the bottom right we can see a rotated and shifted image. As we can see by using this simple technique we can obtain a dataset that is much larger than the original.
After we implemented the data augmentation technique we could -potentially- obtain how many data we wanted[3].
We decided to create a new dataset that is four times larger than the original.

4.3 Second generation

By generating more data and with minor model modifications we obtained a significant improvement in the network performance:

Train	Validation	Test
0.79	0.79	0.78

Table 2: Accuracy results for the second generation

The results presented in Table 2 shows that the overfitting situations seems solved.
In fact, aside of the general improvement, the performance degradation between training set and validation/test set is quite small.

The gap reduction between train and validation/test is a very good news but the general "low" accuracy performance is seeming to show us an underfitting situation.
This "inversion" from overfitting to underfitting is related to the augmentation we performed.

4.4 A more complex model

Usually we have an underfitting situation every time our model is too simple to understand the data we have.
This shouldn't be very surprising as we saw in subsection 4.1 that the model we used was very simple, with just three convolutional layers.
When a model is too simple this means essentially that the network needs more parameters to understand the data. In order to have a more parameters we can basically do as follow:

[3]We have to remember that with too much augmentation we can have some problems due to redundancy.

- Add layers: every time we add a new layer we are increasing considerably the number of parameters in the network.
- Add neurons: also having layers with more neurons means having more parameters.
- Increasing batch size: the "batch size" number tell us how many examples we consider before updating the weight in the network. As the network use a stochastic gradient descent technique to determine how to change the parameters, increasing the batch size can lead to a more precise evaluation of the gradient. This technique contrasts underfitting.
- Reducing dropout: by reducing the dropout value we can forget less about the data. This makes our model more complex.

All these and other guidelines can be found in the well-known deep learning book written by Bengio, Courville and Goodfellow[2].

4.5 Third generation

In order to obtain a more complex model we decided to use a new neural network made by twenty two layers:

```
# Layer 1
CNN_model.add(Conv2D(16, kernel_size=(3, 3),
    activation='relu',padding='same',input_shape=(imageWidth,
    imageHeight, channels)))
# Layer 2
CNN_model.add(Dropout(0.2))

# Layer 3
CNN_model.add(Conv2D(32, (3, 3), activation='relu',padding='same'))
# Layer 4
CNN_model.add(MaxPooling2D((2, 2),padding='same'))
# Layer 5
CNN_model.add(Dropout(0.15))

# Layer 6
```

```
CNN_model.add(Conv2D(64, (3, 3), activation='relu',padding='same'))
# Layer 7
CNN_model.add(BatchNormalization())
# Layer 8
CNN_model.add(Dropout(0.2))

# Layer 9
CNN_model.add(Conv2D(96, (3, 3), activation='relu',padding='same'))
# Layer 10
CNN_model.add(Dropout(0.15))

# Layer 11
CNN_model.add(Conv2D(128, (3, 3),
    activation='relu',padding='same'))
# Layer 12
CNN_model.add(MaxPooling2D((2, 2),padding='same'))
# Layer 13
CNN_model.add(Dropout(0.2))

# Layer 14
CNN_model.add(Conv2D(96, (3, 3), activation='relu',padding='same'))
# Layer 15
CNN_model.add(BatchNormalization())
# Layer 16
CNN_model.add(Dropout(0.2))

# Layer 17
CNN_model.add(Conv2D(64, (3, 3), activation='relu',padding='same'))
# Layer 18
CNN_model.add(Dropout(0.15))

# Layer 19
CNN_model.add(Flatten())
# Layer 20
CNN_model.add(Dense(32, activation='relu',
    kernel_regularizer=regularizers.l2(0.01)))
# Layer 21
CNN_model.add(Dropout(0.2))
# Layer 22
CNN_model.add(Dense(num_classes, activation='softmax'))
```

The structure used has some similarities with the one showed in subsection 4.1. In fact we have again:

- Two first layers dealing with the input
- A six-times repeated structure
  - Convolutional layer
  - Other layers like max pooling, batch normalization or dropout layers

  This part of the network involve the layers from layer 3 to layer 18.
- Four final layers dealing with the output

After the implementation of this new model we obtain another performance improvement:

Train	Validation	Test
0.82	0.81	0.81

Table 3: Accuracy results for the third generation

In order to obtain better performance we decided to change some others features of the network including the optimizer, learning rate, learning epochs number. We can show part of the test we made in Table 4.

As we can see, after some modifications, the best result we were able to obtain was the one showed in Table 5.

We consider the best result the one showed in Table 5 as the accuracy over train and validation is quite similar to the others but the accuracy over the test set is better than all the others[4].

As we can see, the improvement obtained is significant but after all this testing with the model hyperparameters/parameters we came to the conclusion that if we wanted to obtain new improvements we should have work over the data, not over the model.
In order to improve our data we apply a normalization technique to the data.

[4]The only other model able to reach similar performance is the one with 36 layers that is quite complicated. So we consider this model worse than the other one.

#Layers	Optimizer	Learn. rate	Batch	Train	Valid.	Test
16	Adadelta	1.0	32	0.79	0.76	0.81
16	Adadelta	1.0	64	0.81	0.76	0.80
16	Adadelta	1.0	128	0.77	0.76	0.81
19	Adam	0.001	64	0.77	0.75	0.81
19	Adam	0.0015	64	0.80	0.81	0.85
16	Adam	0.001	64	0.80	0.76	0.80
19	Adam	0.002	64	0.76	0.73	0.76
22	Adam	0.0015	64	0.78	0.78	0.83
22	Adam	0.002	64	0.77	0.77	0.82
22	Adam	0.003	64	0.76	0.76	0.8
22	Adam	0.005	64	0.50	0.49	0.47
22	Adam	0.001	32	0.77	0.78	0.83
36	Adam	0.0015	64	0.78	0.80	0.84

Table 4: Accuracy results for the third generation

#Layers	Optimizer	Learn. rate	Batch	Train	Valid.	Test
19	Adam	0.0015	64	0.80	0.81	0.85

Table 5: Accuracy results for the best result obtained in the third generation

4.6 Data normalization

Data normalization is a quite common technique in machine learning used to obtain more ”homogeneous” data.
In literature we can find different types of data normalization:

- Min-Max normalization: $x' = \frac{x - min(X)}{max(X) - min(X)}$
- Mean normalization: $x' = \frac{x - \mu(X)}{max(X) - min(X)}$
- Standardization: $x' = \frac{x - \mu(X)}{\sigma(X)}$
- Scaling: $x' = \frac{x}{\parallel x \parallel}$

In all the formulas showed above x' is the normalized data, x is the original data, X is the set of data, μ is the mean and σ the standard deviation.

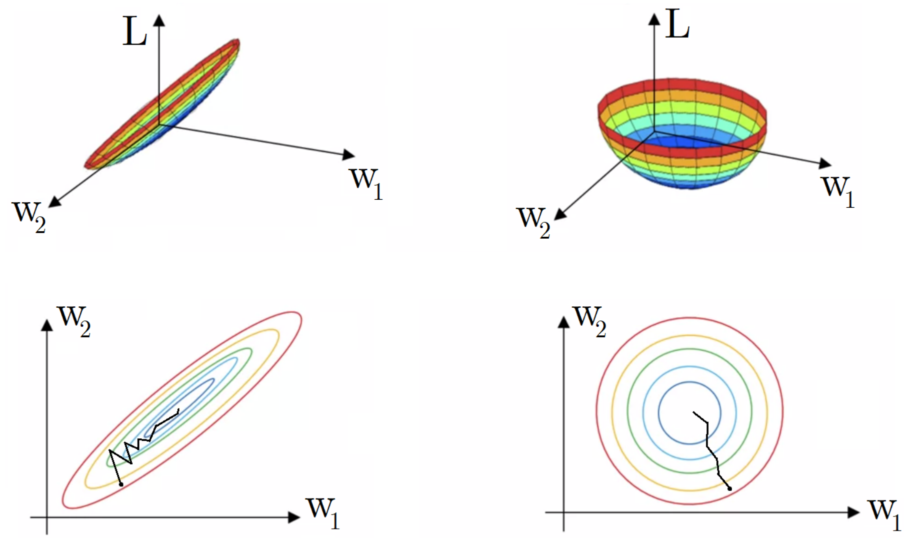


Figure 7: The normalization process

Before explaining which technique we decided to use, we must say that the normalization process allows the network to learn something more about the data. The downside of this approach is that the normalization is data-dependent[5], so every time we apply this technique to the data we obtain a new dataset taht is very problem-specific.
The improvement obtained is related to many factors like the data used, the machine learning technique used and others.
Normally the improvement obtained with the normalization is due to the greater ease the stochastic gradient descent technique is performed. We can give an intuition of this with Figure 7.

In the top-left part of the image we can see an hypothetical three dimensional representation of the loss in a neural network with just two parameters, w_1 and w_2, with unnormalized data. In the bottom left part we have the same plot seen from above.
As we can see, with this "stretched" surface we need more steps to reach the loss minimum, as with the gradient evaluation we don't move toward the "center" of the surface, where the minimum is.

[5]By "data dependent" we mean that the process depends from the data minimum, maximum, mean, standard deviation...

In the top-right and bottom-right part of the image we can see the plot of the loss function after the normalization. We can see that is easier to find the function minimum for the stochastic gradient descent technique because of a more regular shape of the function.

The normalization technique we decided to use is standardization.
To standardize our data we took every image in the dataset and, for every color channel, we calculate the mean and the standard deviation.
One thing to notice is that the output of the standardization can be smaller than zero or greater than one. The problem is that by having negative value we lose the perception of the input. We don't have images anymore, but just number. Usually we have two solutions to this problem:

- Every value smaller than zero is set to zero. Every value greater than one is set to one. By doing this we'll still have an image as output.

- We use the value as they are. By doing this we'll have just number as input to the neural network

We decided to use the first approach. We can show two standardized images in Figure 8. After we apply the standardization technique to all the data we obtain a significant accuracy improvement.

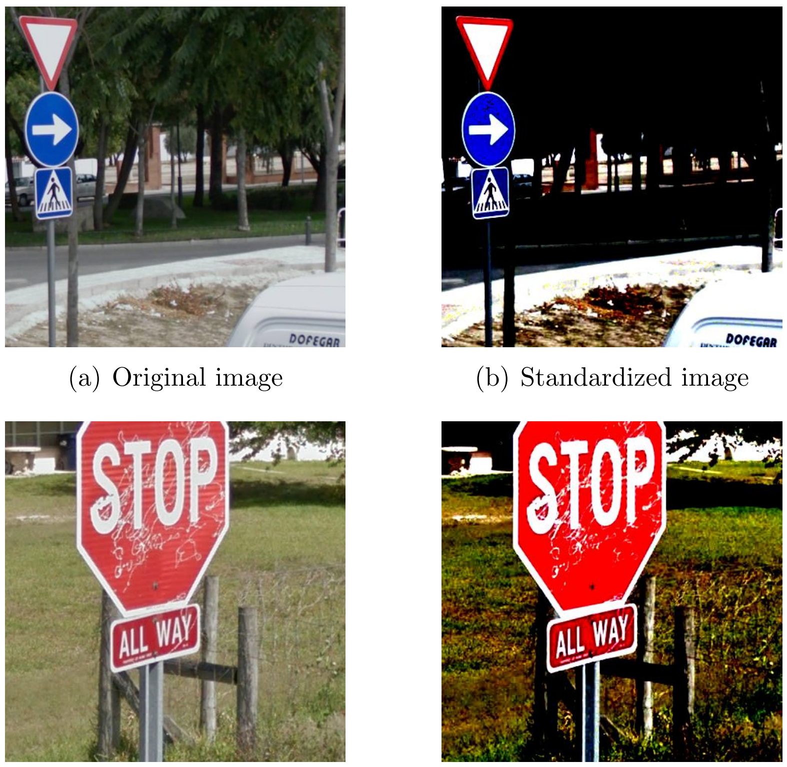
4.7 Fourth generation

After we standardize the data, we start to work again over the model. By applying some minor modifications to the best model obtained in subsection 4.5 we noticed another accuracy improvement:

Train	Validation	Test
0.91	0.91	0.92

Table 6: Accuracy results for the fourth generation

As we can see the performance obtained by our neural network are, at this time, quite good. Having a neural network that is able to tell correctly whether an image contains a road sign or not in the 92% of the situations is a very good result.



(a) Original image (b) Standardized image

(c) Original image (d) Standardized image

Figure 8: The result of the standardization process

5 Google's data

After we developed our neural network, we decided to test it offline over Google's challenges.
When we talk about recognizing road signs in Google's challenges, the most common one is made by a single image separated in a 4×4 grid like the image shown in Figure 9

We collect 800 images forming 50 road signs reCAPTCHA challenge. As these images weren't labeled, we labeled them manually.
From now on we'll use the following -and very important- terms:

- *acc*: the accuracy obtained by the neural network over our dataset. This value is equal to the accuracy obtained by the network over the test set as showed in Table 1, Table 2, Table 5, Table 6.

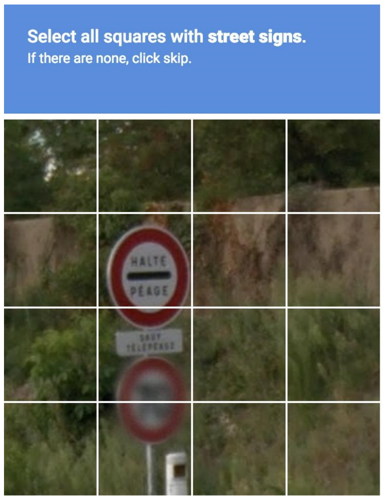

Select all squares with street signs.
If there are none, click skip.
HALTE
PEAGE

Figure 9: A typical road signs challenge

- acc_{Google}: the accuracy obtained by the neural network over the 800 images we received from Google.
- *solved*: as the 800 images forms a pool of 50 reCAPTCHA challenge, with the term *solved* we refer to the percentage of challenge solved.

We also need to use the two following quantities: *precision* and *recall*.
The precision is the portion of the images that our network classify as positive and that are truly positive.
The recall is the portion of the truly positive images our network was able to find.
Let's clarify this by using an example.

Example : let's suppose that we have a road signs reCAPTCHA challenge that is made by a 4×4 grid, so we have 16 images.
Let's suppose that our neural network tells us that four images are positive, and the other twelve are negative. Let's suppose that one of the positive was misclassified. The precision is $\frac{3}{4} = 75\%$
Let's suppose that the "real" positive images were five. As the neural network found just three of them, the recall is $\frac{3}{5} = 60\%$

As the attack was offline we needed some criteria to tell when a reCAPTCHA challenge was correctly solved. We noticed the following two things:

- Every time a single square of the challenge is classified as positive when the real value was negative, the challenge is considered failed.
- Even if the user it's not able to find all the road signs in the challenge, if it's able to find most of them(four over five, five over six...) the challenge is considered passed

Considering the two things above we decided to consider a reCAPTCHA challenge *solved* every time:

- The precision is exactly 1: we are telling that every time our network is saying an image contains a road sign, this "positive" evaluation it must be correct.
- The recall is at least 0.8: we are asking the neural network to find at least the 80% of all the road signs.

With all this considerations in mind we obtained the following results:

Generation	acc	acc_{Google}	$solved$
First	0.61	–	–
Second	0.78	–	–
Third	0.85	0.84	0.08
Fourth	0.92	0.63	0.02

Table 7: Results obtained by different generations

It's possible to see that we didn't test the two first generations over Google's data.
The first important thing to notice is the severe performance degradation between the third and the fourth column. We must remember that the third column is simply telling us how often the neural network classify correctly over a *single* image of the challenge. As the challenge is made by sixteen different images we have that:

$$solved \propto (acc_{Google})^{16}$$

There are still two important things to notice:

1. The third generation's model obtains similar percentages over the single images of our dataset(0.85) and over the single images taken from

Google(0.84). As this two dataset can be very different one from another we should have expected more degradation. As we collect just 800 images we cannot exclude that the degradation didn't happened just because of the data pool we used.

2. The fourth generation's model is doing better of the previous one over our data (0.92 versus 0.85) but we have a severe performance degradation between our data (0.92) and Google's data(0.63).
   This is probably due to the standardization process described in subsection 4.6. As we already said, every time we go trough a normalization process, the data become more problem-specific. So, when we use data that are quite different from the one we used during the training[6], a performance degradation like the one we experienced is always possible.

The fourth generation model was able to classify correctly just the 63% of the images taken from Google. This low value explains why that model was able to solve just the 2% of the reCAPTCHA challenges.
As we'll see in section 6 the gap between acc and acc_{Google} can probably be filled. The considerations we'll make allows us to think that CNN are still an effective tool against reCAPTCHA.

[6]We have to remember that we use a dataset with road signs from United States and China but Google is using Street view images coming from many parts of the world!

6 Future works

Even if the results presented in section 5 are not so good-looking we can make some considerations about what we could do next to get others performance improvement.

One thing to do could be to collect more and more diversified data.
As we said, Google is using images that comes from every part of the world, while our data comes from two dataset about United States and China.
As the percentage obtained by the neural network over our data is quite good (0.92) we can suppose that the network is able to learn even in the complex context of the street view's images. By adding more images we should be able to fill -at least part of- the gap between *acc* and acc_{Google}.

Another thing to do could be to use the standardization process that doesn't produce images as output, but just numerical output data. This could be important because with the process we are using right now, every value lower than zero is set to zero and every value greater than one is set to one. This produce an information loss. This technique could probably improve the *acc* value.

Another very important thing to change is the dependency between the images. Right now our neural network is using a training set made of single road signs cropped images. This images are not related one to each other.
Probably the best way to train the network should be to having as input a whole challenge (made by sixteen images) and as label an array made by 16 different binary values. Every value should tell us when a road sign is present in a square or not. For example the label of Figure 9 should be:

$$\begin{array}{l} [\ 0, 0, 0, 0, \\ \ \ 0, 1, 0, 0, \\ \ \ 0, 1, 0, 0, \\ \ \ 0, 1, 0, 0\ ] \end{array}$$

The importance of this approach is that we can do what humans usually do: consider spatial dependency! In fact every time we find a road sign in a square, is more probable that the surrounding squares contains a road sign too.

Right now our network would consider the sixteen images as independent. This way is possible, but it's quite harder to do! We should use the dependency information that comes from the mutual positions of the road signs.
The difficulty with this approach is that on the web is not so easy to find generic road signs dataset that comes from different countries, and it would be impossible to find a dataset whose label are the one we described above. A dataset like the one we are describing should be created manually.
This new type of dataset could produce a huge improvement in the acc_{Google} value, filling the gap between acc and acc_{Google}.
We also have to say that this new attack would be immune to two of three recommendations we suggest in section 7, so it's more difficult to prevent.

7 Security recommendations

By studying the reCAPTCHA challenge we found some security weaknesses against a convolutional neural network attack. We can summarize them as follow:

- Few categories: we noticed that there is a very reduced categories number in the reCAPTCHA challenge. A small amount of categories like this can be a weakness because an hacker team could design a convolutional neural network that performs very good over one of this category. By doing this the percentage p of reCAPTCHA challenge that the net could potentially solve, would be:
  $p \propto \frac{1}{c}$
  where c is the number of categories used. As c is currently a low value, the portion p could be quite high.

- Twenty five better than sixteen: when the convolutional neural network used is like the one we used, considering all the images in the challenge as independent one from another, the neural network must solve sixteen different classification problems. By using the same challenge with a 5×5 grid instead of a 4×4 grid, the network should solve twenty five classification problems instead of sixteen. Obviously this should be more difficult.
  Of course this recommendations would be useless against convolutional neural network that uses the whole challenge as training input, like we explained in section 6.

- Road sign distribution: another improvement could be about the data distribution. By now, the most common number of road signs images in a sixteen-images reCAPTCHA challenge like the one showed in Figure 9 is around three or four.
  Knowing this we decided to build a road signs dataset that is as more similar as possible to the one we should have found in the reCAPTCHA challenge. This means that we built a dataset with approximately $\frac{4}{16} = \frac{1}{4}$ images containing a road sign. This is important because usually the network recognize the data distribution and is quite difficult that a network trained with some distribution will produce an output very different from that distribution.
  One thing to make more difficult the network's predictions could be to

use a more uniform road signs number distribution in the challenges.
Let's suppose we can have a number of road signs between 0 and 16. By doing this it becomes difficult to decide the proportion between positive and negative images in the dataset. In fact if the attacker decides to train with half of the images as road signs the network will probably performs bad over all the situations with a reduced number of road signs and in all the situations with a very large number of road signs.
Like the previous recommendation, we have to say that this suggestions it would be useless against the network that uses the whole challenge as training input.

8 Conclusions

In this report we studied how to approach a semantic image security system by using deep learning techniques.
First thing we have to say is despite of all the weaknesses, breaking reCAPTCHA requires a considerable effort.
Breaking reCAPTCHA with deep learning techniques requires different tools and expertise:

- Dataset: every time we want to use a deep learning system, we need to find a huge amount of data. As reCAPTCHA proposes approximately ten categories, we need one dataset for every category. Anyway, the reduced categories number makes the problem feasible to solve.

- Model: to use the neural network in a efficient way we must find the proper network model. This means that we have to find a good hyper-parameter configuration.

- Computational resources: to train a neural network we need computational power. One single training can requires some hours even with a dedicated GPU machine. We also have to say that this machine are quite cheap to rent, so every attacker can train its neural network in a low cost way.

Aside of this -surmountable- difficulties, the results we got are encouraging. The fourth generation's model was able to obtain a 92% accuracy over our data. Of course the performance degradation over Google's data is remarkable and to get other improvements we should consider what we said in section 6; but the most important thing to notice is that with the proper tools, data and expertise a visual security system like the one used in reCAPTCHA can probably be broken. This can question the security of this security visual systems against deep learning techniques.

References

[1] : Suphannee Sivakorn, Iasonas Polakis, Angelos D. Keromytis: *I Am Robot: (Deep) Learning to Break Semantic Image CAPTCHAs*, 2016 IEEE European Symposium on Security and Privacy, 2016

[2] : Yoshua Bengio, Aaron Courville and Ian Goodfellow: *Deep Learning*, MIT press, 2016