Breaking Text-Based CAPTCHA with Sparse Convolutional Neural Networks

Diogo Daniel Ferreira[1], Luís Leira[1], Petya Mihaylova[2], and Petia Georgieva[1](✉)

[1] Department of Electronics, Telecommunications and Informatics,
University of Aveiro, Aveiro, Portugal
petia@ua.pt

[2] Technical University of Sofia, Sofia, Bulgaria
petya.petkova@tu-sofia.bg

Abstract. CAPTCHA is an automated test designed to check if the user is human. Though other approaches are explored (such as object recognition), the text-based CAPTCHA is still the main test used by many web service providers, to separate human users from bots. In this paper, a sparse Convolutional Neural Network (CNN) to break text-based CAPTCHA is proposed. Unlike previous CNN solutions, which mainly use fine-tuning and transfer learning from pre-trained models, the proposed framework does not require a pre-trained model. The sparsity constraint deactivates connections between neurons in the CNN fully connected layers that leads to improved accuracy compared to the baseline approach. Visualization of the spatial distribution of neuron activity shed light on the internal learning and the effect of the sparsity constraint.

Keywords: Text-based CAPTCHA · Convolutional Neural Networks · Sparsity constraint · Neuron activity visualization

1 Introduction

CAPTCHA (Completely Automated Public Turing Test to tell Computers and Humans Apart) is an automated test designed to check if the user is human. The test is made using a challenge-response approach, where the challenge is easy for a human to solve, but hard for a machine. If the response is correct, the machine assumes that the user is human. CAPTCHAs are used by most service providers, such as email or online shopping, to prevent bots from abusing their online services. For example, to prevent a botnet to create hundreds of new email accounts per second, a CAPTCHA can be used to assure that the users creating the email accounts are humans.

This Research work was funded by National Funds through the FCT - Foundation for Science and Technology, in the context of the project UID/CEC/00127/2013.

A. Morales et al. (Eds.): IbPRIA 2019, LNCS 11868, pp. 404–415, 2019.
https://doi.org/10.1007/978-3-030-31321-0_35

The construction of CAPTCHAs is not an easy task because it is difficult to create challenges hard for machines but easily solvable by humans. Over the years, the most used CAPTCHAs are based on visual-perception tasks. Distorted characters are presented that must be typed correctly by the user. Background and foreground noise is usually added, making it almost impossible for a computer to automatically recognize the characters. However, for humans, the characters are relatively easy to recognize, due to our brain's capacity for recognizing patterns. There are three characteristics for a modern text-based CAPTCHA to be resilient:

- The large variation in the shape of letters. While there is an infinite variety of versions for the same character that the human brain can recognize, the same is not true for a computer. If all the versions of a character are different, it is hard for a computer to recognize any version not previously seen.
- Due to the large variation in the shape of characters, it can also be hard to perform segmentation for each character, mainly when the characters have no space in between.
- In specific CAPTCHAs, the context may be the key to answer correctly to the task. When the word is taken into context, it is easier for a human to answer what are the characters in the challenge, even if some of them are dubious.

The conjugation of these three characteristics makes a CAPTCHA hard to solve by a machine. However, over the last few years a number of techniques to break the text-based CAPTCHA have been proposed, [1]. Most of the solutions are based on deep (neural network) learning models trained on millions of images using clusters of computers or alternatively using fine-tuning and transfer learning from pre-trained models. The models are usually designed with huge number of parameters to account for the complexity of large scale data that they learn from. However, when it comes to production deployment on embedded or mobile devices, the network size, speed, and power consumption become an issue.

In this paper, we propose a strategy for limiting the neuron activity and show that this improves and speed up the learning compared to the baseline approach. The strategy is illustrated on Kaggle text-based CAPTCHA data set.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 explains the proposed sparse CNN framework. Simulation results and discussions are presented in Sects. 4 and 5 summarizes the work.

2 Related Work

In 2014, the authors of [2] for the first time stated that text distortion-based CAPTCHAs schemes should be considered insecure due to technological advances. They presented a general framework for solving text-based CAPTCHAs, with a multi-step algorithm based on reinforcement learning with joint phases of segmentation and text recognition. Since then, alternative CAPTCHA schemes based on object recognition have been proposed, [3],

[4], making it harder for machines to solve them, but remaining easily solvable by humans. However, there are still many implementations of insecure text-based CAPTCHAs on the web. In [5], the authors propose two approaches for text-based CAPTCHA recognition, based on pattern matching and hierarchical algorithms. Both approaches attempt to find the shape of the objects by defining key points in their structure using the Canny Edge Detector and then comparing it to the structure of each character in a local database. The first approach tries to find words in images starting with visual cues, and incorporates lexical information later (the CAPTCHAs texts are words from the dictionary). The second approach searches for entire words at once using a dictionary with all 411 words that the considered CAPTCHAs contain. The second algorithm achieved better results, with an accuracy of 92% on the EZ-Gimpy dataset. This study showed that, algorithms that deal with the whole CAPTCHA at once tend to output higher accuracy than algorithms that deal with each character separately.

In [6], the authors break the Microsoft CAPTCHA, used for systems such as MSN or Hotmail, with image segmentation and pattern matching. The segmentation and recognition combined achieved 60% accuracy. Other approaches, such as [7] or [8], also use image segmentation and pattern matching to break specific CAPTCHAs datasets.

In [9], the authors propose a two-step approach to recognize text-based CAPTCHAs. The data include CAPTCHAs from the most visited websites, like MSN, Yahoo, Google/Gmail or TicketMaster. First, segmentation is applied to separate the characters and then a Convolutional Neural Network (CNN) to recognize them. It is shown that most of the errors derive from a bad segmentation. The highest accuracy is close to 90%.

In [10], a CNN is used for CAPTCHA recognition. The network is composed by three convolution layers, three pooling layers and two fully-connected layers. The network recognizes the sequence without pre-segmentation, and with a fixed size of six characters. The problem of CNN requiring a very large training set is solved with an Active Learning mechanism. To prevent from feeding the neural network with millions of CAPTCHAs, each CAPTCHA is recognized with a certain measured uncertainty. Only the most uncertain CAPTCHAs on the test set are used for retraining the model. The algorithm reaches an accuracy of almost 90%.

In [11], a novel approach is taken to break text-based CAPTCHAs. It introduces the Recursive Cortical Network (RCN), a hierarchical probabilistic generative model with an outstanding capacity of generalization based on the human brain, designed to be trained with few examples. This model achieved an accuracy of 94.3% on character recognition on the reCAPTCHA algorithm, created and currently used by Google.

Neural networks are often over-parameterized with significant redundancy among the weights and the CNNs do not make an exception. To address this problem we propose in this paper sparsity constraint approach originated in deep autoencoders training. The idea is to limit the neuron activity and enforce learning of non-redundant information.

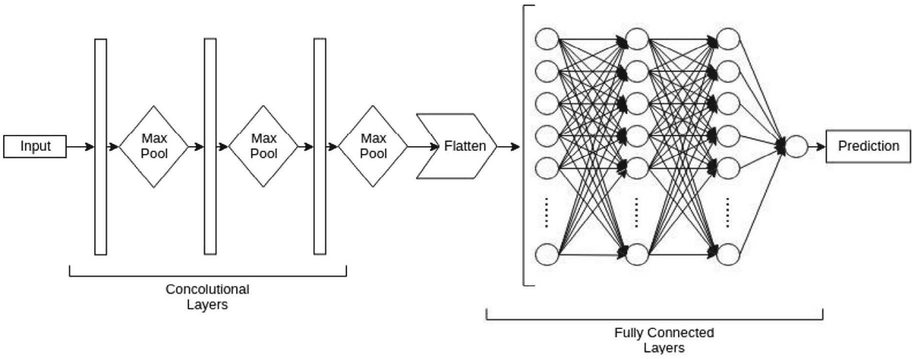


Fig. 1. CNN framework

3 The Proposed Framework

CNN combined with large-scale labeled data has become a standard recipe for achieving state-of-the-art performance on computer vision tasks in recent years. The general architecture of the CNN is given in Fig. 1). Typically, a CNN alternatively stacks convolutional (C) and sub-sampling (e.g. max-pooling) (M) layers. In a C layer, small feature extractors (kernels) sweep over the topology and transform the input into feature maps. In a M layer, activations within a neighborhood are abstracted to acquire invariance to local translations. After several C and M layers, feature maps are flattened into a feature vector, and followed by fully-connected (FC) layers. In this paper Rectified Linear Units (ReLU) are applied in the convolutional layers [12]. ReLU is formally defined as $f(x) = max(0, x)$. It has become very popular in the past couple of years since it improves significantly the convergence speed and avoids the vanishing gradient problem. The inputs of the last FC layer are passed through a softmax function, to compute the class probabilities. Given an input $x^{(i)}$ with a label $y^{(i)}$, the softmax function estimates the probability that this example belongs to each of the class labels j $= 1, 2, \ldots$ c

$$p(y^{(i)} = j|x^{(i)}; \theta) = \frac{e^{\theta_j^T x^{(i)}}}{\sum_{j=1}^{c} e^{\theta_j^T x^{(i)}}} \quad (1)$$

The network outputs c dimensional vector of the estimated probabilities, where θ is a matrix of parameters connecting the softmax layer with the previous (hidden) layer.

$$\hat{y}_{cnn}(x^{(i)}) = \frac{1}{\sum_{j=1}^{c} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \ldots \\ e^{\theta_c^T x^{(i)}} \end{bmatrix} \quad (2)$$

The denominator in Eq. (2) normalizes the distribution to sum to one. Given a batch of m training examples, the baseline softmax cost function to be minimized is

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{j=1}^{c} 1\{y^{(i)} = j\} \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{j=1}^{c} e^{\theta_j^T x^{(i)}}}]. \tag{3}$$

We propose a strategy for limiting the neuron activity in the FC layers which is imposed by sparsity constraints on the hidden units.

Let a_k denotes the activation of hidden unit k and $a_k(x)$ denotes the activation of hidden unit k when the network is given a specific input x. Further, let $\hat{\rho}_k = \frac{1}{m}\sum_{i=1}^{m}[a_k(x^{(i)})]$ be the average activation of hidden unit k (averaged over the training set). We would like to (approximately) enforce the constraint $\hat{\rho}_k = \rho$, where ρ is a sparsity parameter, typically a small value. In other words we would like the average activation of each hidden unit j to be close to ρ. This is enforced by an extra penalty term in the cost function that penalizes $\hat{\rho}_k$ if deviating significantly from ρ. Many choices of the penalty term will give reasonable results, here we choose the Kullback-Leibler (KL) divergence which is a standard function for measuring how different two distributions are [13]. KL-divergence measure between a Bernoulli random variable with mean ρ and a Bernoulli random variable with mean $\hat{\rho}_k$ is given as

$$KL(\rho||\hat{\rho}_k) = \sum_{k=1}^{s} \rho \log \frac{\rho}{\hat{\rho}_k} + (1-\rho) \log \frac{(1-\rho)}{(1-\hat{\rho}_k)} \tag{4}$$

Here s is the number of the units in the hidden layer, and the index k is summing over the hidden units of the network. The choice of ρ expresses the desired level of sparsity, here we set it to a common value of 0.1.

In the sparse cost function J_{sparse}, β controls the importance of the sparsity penalty term $KL(\rho||\hat{\rho}_k)$

$$J_{sparse} = J + \beta KL(\rho||\hat{\rho}_k)] \tag{5}$$

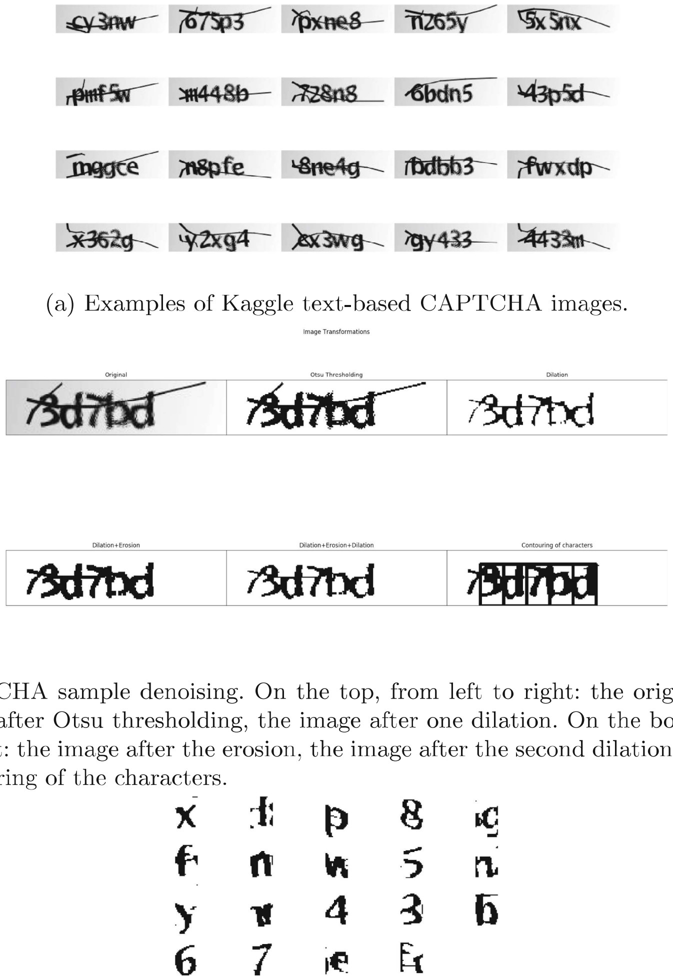
The intuition behind this optimization framework is to specialize the neurons in learning specific patterns and as a consequence enforce compression to happen.

4 Experiments and Results

4.1 Dataset

The Kaggle CAPTCHA dataset has been used to evaluate the proposed framework[1]. A few examples are given in Fig. 2(a). Each image consists of five random characters from a set of 19 characters: 2, 3, 4, 5, 6, 7, 8, b, c, d, e, f, g, m, n, p, w, x, y. The characters have the same font but rotated in different angles. Since

[1] https://www.kaggle.com/fournierp/captcha-version-2-images.



(a) Examples of Kaggle text-based CAPTCHA images.

(b) CAPTCHA sample denoising. On the top, from left to right: the original image, the image after Otsu thresholding, the image after one dilation. On the bottom, from left to right: the image after the erosion, the image after the second dilation and finally the contouring of the characters.

(c) Samples of single character images.

Fig. 2. CAPTCHA images and prepossessing

the images have been heavily corrupted by noise (black lines over the characters), a few denoising steps are taken to remove or alleviate the noise as shown in Fig. 2(b). First, the Otsu method [14] is applied to perform clustering-based image thresholding and transform the CAPTCHA into a binary image. Next, morphological transformations are applied to the image. A dilation and an erosion are applied sequentially with 3×3 kernel, followed by a second dilation with 3×1 kernel, in an attempt to eliminate the horizontal lines that create the noise in the image. Single character images were then extracted by segmentation as shown in Fig. 2(c).

Since the original dataset has been small (1070 images) we applied rotation and shifting operations to augment it. Combinations of rotations (−10°, 0°, 10°), vertical (−3 px, 0 px, 3 px) and horizontal (−3 px, 0 px, 3 px) shifts were applied to generate 27 variations of each original single character training image. After the augmentation, data grew to 5350 images per single character and 87048 examples in total. For the training 69638 items were randomly selected and the test set was limited to 17409 items.

4.2 Performance Evaluation

CNNs with varying depth have been trained, with architectures shown in Table 1. Stochastic gradient descent optimization was applied with learning rate of 0.0001 and dropout step of 0.5 to prevent overfitting. Figures 3, 4 and 5 illustrate the performance of the proposed method (with sparsity constraint) and the baseline approach (without sparsity constraint) in terms of training and testing accuracy. The testing accuracy is evaluated in each training epoch with new test data. Some observations can be made from the figures. The sparsity constraint makes the learning models less sensitive to the network dimension compared to the baseline approach. Note the similar behavior of the right side plots in Figs. 3, 4 and 5. In contrast, the left side plots show lack of learning (CNN1), overfitting (CNN2) and finally achieved a good performance while increasing the CNN complexity. The maximum testing accuracy in the baseline is 90.2 % (for CNN3). The maximum accuracy of the proposed framework is 95.7% (for CNN2). The performance of the lower complexity model (CNN1) is significantly more affected by the sparsity constraints.

The capability of CNN to model highly nonlinear functions comes with high computational and memory demands both during the model training and inference. The sparsity constraints impose connection between the neurons in the FC layers, which enforces the neurons to learn non redundant information and therefore reduces the amount of information processed, [15].

4.3 Neuron Activity Visualization

One way to understand what the CNN is learning and to asses the effect of the sparsity constraint on the internal learning process, is to visualize the representations captured by the hidden units, [16]. These representations are not always easy to understand [17], therefore we illustrate here only those that we found interpretable.

The matrix of weights between the flatten layer (256 units) and the first hidden FC layer (512 units) is denoted as $\theta^{(1)}$ (dimension 256×512). We visualize the weights collected in $\theta^{(1)}$ as representation images. Each column of $\theta^{(1)}$ is reshaped into a square 16×16 pixels image and visualized on one cell of the visualization panel shown in Fig. 6. Figure 6 illustrates the representation images

Table 1. CNN architectures

Layer type	Neurons	Kernel size
CNN1		
2 Conv layers & pooling layers	256	3×3 & 2×2
2 Fully-connected (FC) layers	512	
Softmax layer	19	
CNN2		
4 Conv layers & pooling layers	256	3×3 & 2×2
2 Fully-connected (FC) layers	512	
Softmax layer	19	
CNN3		
6 Conv layers & pooling layers	256	3×3 & 2×2
4 Fully-connected (FC) layers	512	
Softmax layer	19	

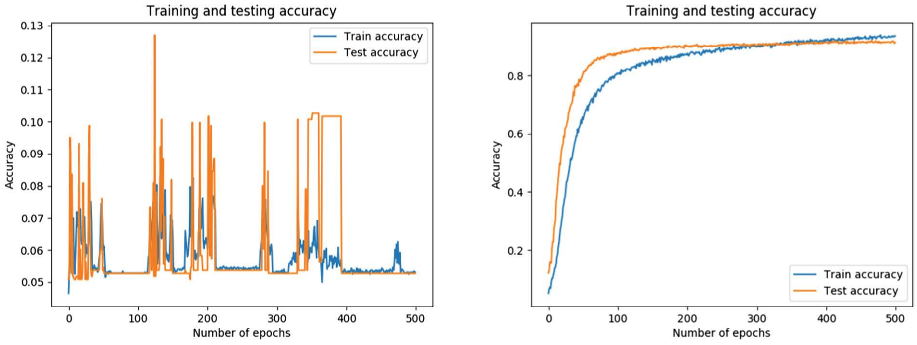


Fig. 3. CNN1: left (baseline), right (proposed framework)

of all 512 hidden units for a given visible input (the image of the character 4) as a visualization panel of 32 by 16 cells. On the left side of Fig. 6 are the results from the CNN trained with the baseline cost function J (Eq. 3) and on the right side the weights obtained after training with the sparse cost function J_{sparse} (Eq. 5). The proposed framework resulted in more “blank” cells which means that those neurons are not activated by the input image. Since the weights are fit in such a way not to violate the constraints on the neuron activity, the ‘blank’ neurons are not specialized in the specific patterns of the character ‘4’ image. The sparsity constrain acts as an inhibitor/promoter for particular stimulus and therefore favor a non-uniform distribution of the neuron activity over training examples. This hypothesis (in agreement with [18]), may explain the uniform spatial distribution of the neuron activity with the baseline training.

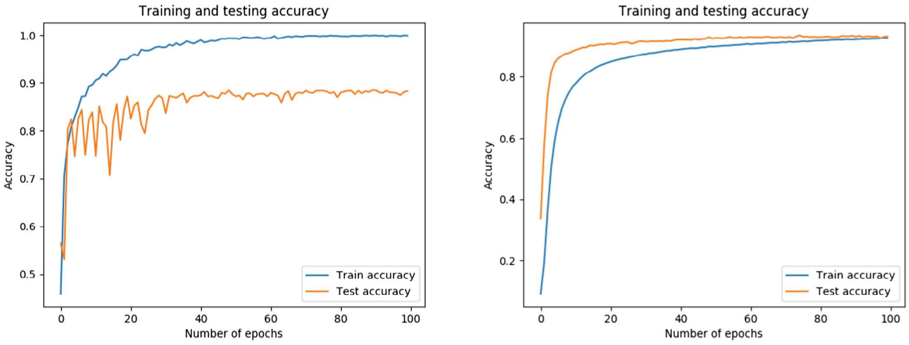


Fig. 4. CNN2: left (baseline), right (proposed framework)

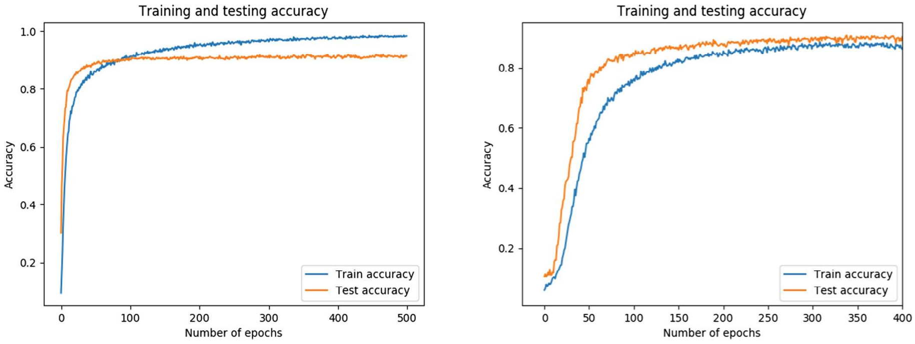


Fig. 5. CNN3: left (baseline), right (proposed framework)

Curiously, results that support the inhibitor/promoter effect of the sparsity constraint were observed also with the convolutional filters. We applied the same strategy for visualizing as before and the visualization panels (8 by 8 cells) of the 64 filters in the 3^{rd} and the 4^{th} convolutional layers are illustrated in Figs. 7 and 8. The differences are more distinct applying gray heat map. The rough granulated cells indicate specific features learned by the filters, while the smooth gray cells indicate inactivate filters. Though the convolutional filters were not explicitly constrained in Eq. (5), the constrained neuron activity in the FC layers backpropagate this effect and force the convolutional filters to be more selective to a particular stimulus. There are much more inactive filters to the specific patterns of the image of the character '4' in the right plots of Figs. 7 and 8 compared to the baseline trained filters in the left plots of the same figures.

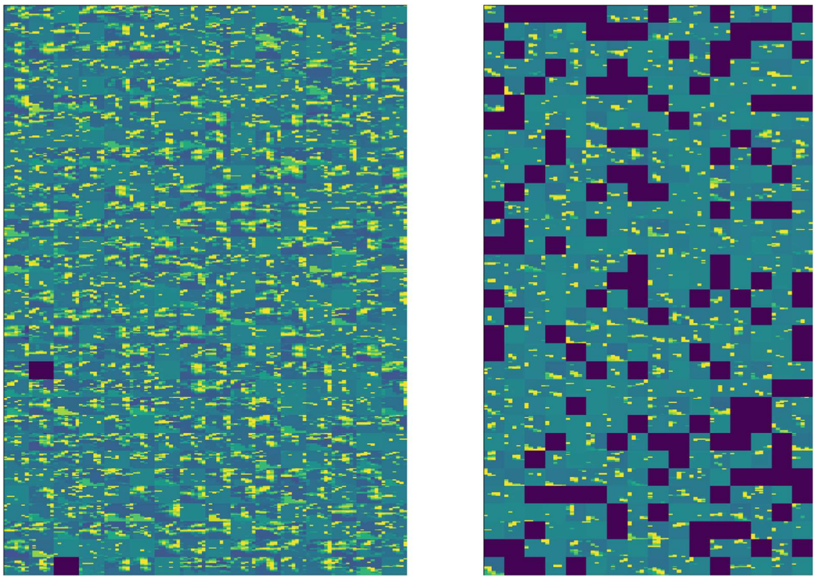


Fig. 6. CNN2. Features learning in the 1st hidden FC layer given image of character ‘4’: left (baseline); right (proposed method)

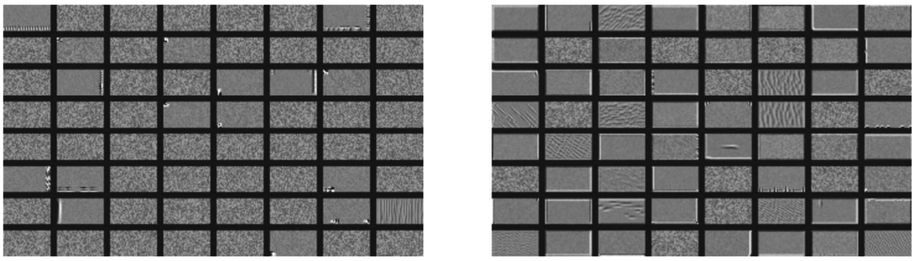


Fig. 7. CNN2. Filter patterns in 3^{rd} conv layer: left (baseline); right (proposed framework)

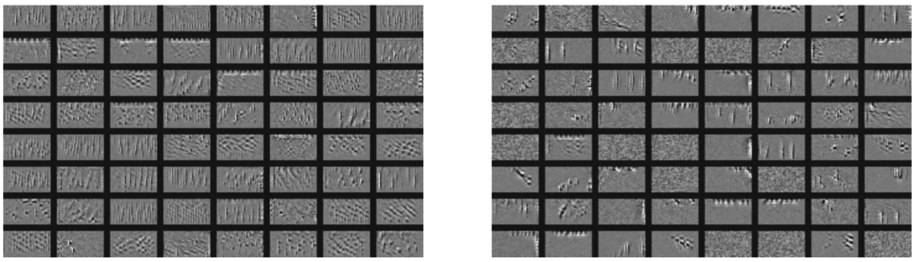


Fig. 8. CNN2. Filter patterns in 4^{th} conv layer: left (baseline); right (proposed framework)

5 Conclusions

In this paper, CNN learning framework with sparse cost function has been proposed to demonstrate a possible attack that can be made to a text-based CAPTCHA. The proposed model has a low complexity which makes its training transparent and fully controlled. Although the quantity and quality of the image dataset is not very high, the framework shows good results. The concept of sparsity, widely applied in deep autoencoders, is a good alternative to account for parameter redundancy and compressed sensing. The outcomes of experiments suggest that adding sparsity constraint can improve the network accuracy and convergence speed. Future extension of this work would be upgrading this framework to non-text based CAPTCHAs also.

References

1. Chen, J., Luo, X., Guo, Y., Zhang, Y., Gong, D.: A survey on breaking technique of text-based CAPTCHA. Secur. Commun. Netw. **2017**, 1–15 (2017)
2. Bursztein, E., Aigrain, J., Moscicki, A., Mitchell, J.C.: The end is nigh: generic solving of text-based CAPTCHAs. In: 8th USENIX Workshop on Offensive Technologies, San Diego, CA (2014)
3. Sivakorn, S., Polakis, I., Keromytis, A.D.: I am robot: (deep) learning to break semantic image CAPTCHAs. In: IEEE European Symposium on Security and Privacy (2016)
4. Zhao, B., et al.: Towards evaluating the security of real-world deployed image CAPTCHAs. In: 11th ACM Workshop on Artificial Intelligence and Security, New York, NY, USA, pp. 85–96 (2018)
5. Mori, G., Malik, J.: Recognizing objects in adversarial clutter: breaking a visual CAPTCHA. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2003)
6. Yan, J., Salah El Ahmad, A.: A low-cost attack on a Microsoft CAPTCHA. In: 5th ACM Conference on Computer and Communications Security, CCS 2008, New York, NY, USA (2008)
7. Yan, J., Salah El Ahmad, A.: Breaking visual CAPTCHAs with Naive pattern recognition algorithms. In: Twenty-Third Annual Computer Security Applications Conference (2007)
8. Li, S., Amier Haider Shah, S., Asad Usman Khan, M., Khayam, S.A., Sadeghi, A.-R., Schmitz, R.: Breaking e-banking CAPTCHAs. In: 26th Annual Computer Security Applications Conference, pp. 171–180 (2010)
9. Chellapilla, K., Simard, P.Y.: Using machine learning to break visual human interaction proofs (HIPs). In: Advances in Neural Information Processing Systems (NIPS), pp. 265–272. MIT Press (2005)
10. Stark, F., Hazırbas, C., Triebel, R., Cremers, D.: CAPTCHA recognition with active deep learning. In: German Conference on Pattern Recognition (2015)
11. George, D., et al.: A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs. Science **358**(6368), eaag2612 (2017)
12. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activation in convolution network. In: ICML Deep Learning Workshop (2015)

13. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. **22**, 79–86 (1951)
14. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Sys. Man. Cyber. **9**, 62–66 (1979)
15. Bozhkov, L., Georgieva, P.: Overview of deep learning architectures for EEG-based brain imaging. In: IEEE World Congress on Computational Intelligence - IJCNN (2018)
16. Yosinski, J., Clune, J., Nguyen, A.M., Fuchs, T.J., Lipson, H.: Understanding neural networks through deep visualization. In: 31st International Conference on Machine Learning (2015)
17. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8689, pp. 818–833. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10590-1_53
18. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-CAM: visual explanations from deep networks via gradient-based localization. In: 2017 IEEE International Conference on Computer Vision (ICCV) (2017)