CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Solving text-based CAPTCHA images with neural networks
Student:	Matěj Nikl
Supervisor:	Ing. Martin Kopp
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2017/18

Instructions

Study different types of text-based reverse Turing tests, often called CAPTCHA images (distorted texts in images), and principles of artificial neural networks (ANN). Perform a review of both CAPTCHA generating systems and existing algorithms that attempt to break the CAPTCHA systems. Focus mainly on algorithms based on ANN.

Design and use a shallow feed-forward ANN to set a baseline for the classification performance to reveal the strong and weak aspects of ANNs for the given problem and evaluate the accuracy. Design, train, and validate a deep ANN to reveal the text from CAPTCHA images with the aim to achieve better accuracy. Use your deep ANN to demonstrate the vulnerability of the CAPTCHA images, i.e., analyze the ability of your ANN to automatically recognize characters in CAPTCHA images generated by at least three different CAPTCHA systems.

References

Will be provided by the supervisor.

L.S.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague June 29, 2016

Czech Technical University in Prague

Faculty of Information Technology

Department of Theoretical Computer Science

Bachelor's thesis

Solving text-based CAPTCHA images with neural networks

Matěj Nikl

Supervisor: Ing. Martin Kopp

17th February 2017

Acknowledgements

I would like to thank my supervisor Ing. Martin Kopp for his guidance and helpful input during writing of this thesis; I would also like to thank my family and friends who have supported me all along.

Declaration

In Prague on 17th February 2017 . . . .

Citation of this thesis

Nikl, Matěj. *Solving text-based CAPTCHA images with neural networks.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Tato bakalářská práce se zabývá automatickým rozpoznáváním znaků z textových CAPTCHA obrázků. Jejím cílem je porovnání mělkých a hlubokých umělých neuronových sítí spolu s jejich dopadem na výslednou přesnost. Je navrhnut a implementován algoritmus využívající dvě umělé neuronové sítě, techniku posuvného okénka a k–means shlukování. Přesnosti přepisů jsou změřeny na 11 různých schématech se zaměřením na porovnání mělkých a hlubokých architektur.

Klíčová slova CAPTCHA, strojové učení, neuronová síť, konvoluční neuronová síť, optické rozpoznávání znaků, hluboké učení, počítačové vidění, posuvné okénko, lokalizace, Torch7

Abstract

This bachelor's thesis studies automatic character recognition from textual CAPTCHA images. Its aim is to compare shallow and deep artificial neural networks together with their impact on the resulting accuracy. An algorithm utilizing two artificial neural networks, the sliding window technique and the k–means clustering is designed and implemented. The transcription accuracies are measured on 11 different schemes with the emphasis on the comparison of shallow and deep architectures.

Keywords CAPTCHA, machine learning, neural network, convolutional neural network, optical character recognition, deep learning, computer vision, sliding window, localization, Torch7

Contents

List of Figures

List of Tables

Introduction

The ability to automatically distinguish between a real human–user and a computer software performing automated tasks has become increasingly important during the last decade. To carry out this differentiation, special challenge–response tests were designed and deployed. They are known under the acronym CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart), and are now present throughout the whole Internet – whenever there is a need to deny access of non-human users.

Lack of such limitations could result in computer bots performing various malicious actions:

- Bot can post spam messages from automatically created user profiles in message boards [5], which can potentially contain backlinks to the intended website. This way the website can be promoted among users as well as have increased rank for search engines, which can make it more likely to be displayed higher in search results.
- Advertisers are paying for their advertisement being displayed to people visiting websites. However, without testing no distinction can be made whether the visitor is human or not. This in turn means that advertisers are paying for "displaying" their advertisement to computer bots [6].
- Automated scraping of content such as email addresses [7] is an issue as well because email addresses can be abused by targeted spam. The sole act of content scraping is resource demanding on the server's side, resulting into degraded performance.
- Bots can be used for buying up good seats for various events (such as concerts, sport matches, etc.) [8], which can lead to unfulfilled demand. Therefore, brokers can resell those tickets for much higher prices.

From the list above is obvious that such actions have to be prevented and therefore, is important to have CAPTCHA that is capable of distinguishing humans and computer bots. However, as technology advances and computers are gaining human-like or even superhuman capabilities, this becomes gradually harder to achieve.

In this thesis I will address the task of automatic character recognition from multiple text-based CAPTCHAs schemes using artificial neural networks (ANNs) and a simple clustering algorithm (k-means). The goal is to take a CAPTCHA challenge as an input while outputting transcription of the text presented in the challenge. This approach should be general across multiple text-based CAPTCHA schemes with no need to modify any part of the algorithm. In the experimental part of this thesis I will compare the performance of the shallow (only one hidden layer) and deep (multiple hidden layers) ANNs.

This thesis is structured into five main chapters. The first chapter presents CAPTCHA types and introduces three CAPTCHA generators. The second chapter provides insight into artificial neural networks. The third chapter analyses current state of the art of textual CAPTCHA recognition. The fourth chapter describes the algorithm used to transcribe CAPTCHA challenges. Finally, the fifth chapter introduces performed experiments and evaluates the accuracy achieved by various ANN architectures.

Chapter 1

CAPTCHA Analysis

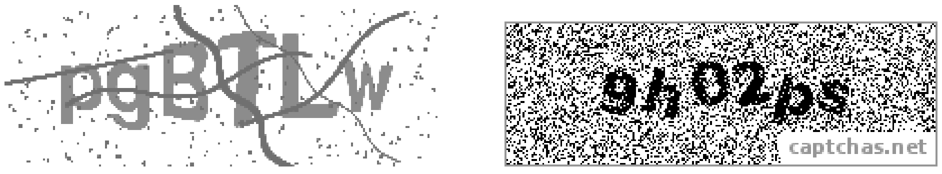
In this chapter, I will briefly go through some of the types of CAPTCHA used today, discuss different textual CAPTCHA generating software available. Finally, I will present the chosen CAPTCHA schemes which I will attempt to transcribe.

1.1 CAPTCHA Types

The abbreviation *CAPTCHA* is usually used when referring to a visual test consisting of transcribing obscured text from an image, however other types of CAPTCHA exist as well. In the following subsections I will present the most prominent ones.

1.1.1 Text-based CAPTCHA

Text-based CAPTCHA used to be the most common [4] and still is present on certain websites either as the primary means of testing or as a backup. It is based on the AI problem of recognizing characters from an image. It is now being replaced by another type of CAPTCHA, which is based on much harder, computer vision problem and thus is more secure (see section 1.1.3). I will attack this type of CAPTCHA in this bachelor thesis. For additional analysis, see Section 1.2. Figure 1.1 gives examples of such CAPTCHA.



(a) Secureimage PHP Captcha [9] (b) captchas.net [10]

Figure 1.1: Text-based CAPTCHAs

1.1.2 Speech-based CAPTCHA

This type of CAPTCHA is based on speech recognition – it is required from the testee to recognize spelled characters or spoken words over usually noisy background. It is an alternative to any visual CAPTCHA for visually impaired people, therefore it is generally available as a secondary option together with the primary CAPTCHA. Figure 1.2 shows a waveform visualization of such CAPTCHA.

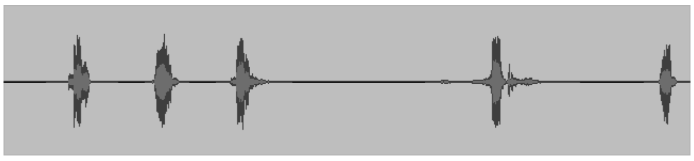


Figure 1.2: A speech-based CAPTCHA [1] spelling numbers “21261”

1.1.3 Image-based CAPTCHA

Another type of CAPTCHA is the image-based one. As of right now, they rely on the AI problem of either pattern recognition or localization, which is much harder task compared to the text recognition in text-based CAPTCHA [11]. It is required to select images which are of a certain type or containing certain objects in order to pass the test. The widely used example of this type of CAPTCHA is the reCAPTCHA [1]. Apart from fulfilling its goal, it also positively uses the effort put into solving the challenges by using the results to annotate images and create machine learning datasets. This is useful for building the next generation of Artificial Intelligence solutions [1]. Please refer to Figure 1.3 for examples.

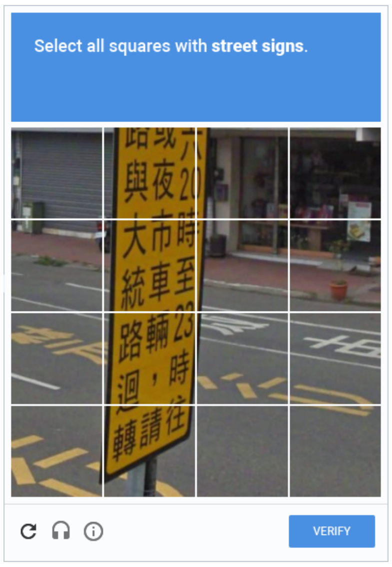
1.2 CAPTCHA Generating Software

Large companies usually develop their own CAPTCHA schemes to protect their websites. This approach has the advantage of the source code not being available to the potential attacker, however a lot of caution is necessary in order to avoid known weaknesses and to make the scheme secure.

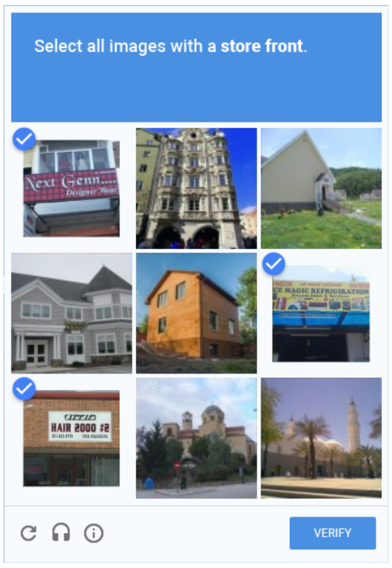
Another option is to use an open–source CAPTCHA service such as Secureimage PHP Captcha [9], or paid services such as BotDetect CAPTCHA [12] or captchas.net [10], which I will discuss more in the following subsections.

1.2.1 Secureimage PHP Captcha

This is an open-source CAPTCHA service, which means it poses no guarantees on the safety and the source codes are inherently available to all. That is a huge advantage on attacker’s side as it opens the possibility to generate as



(a) A localization CAPTCHA



(b) A pattern recognition CAPTCHA

Figure 1.3: Image-based CAPTCHAs [1]

many examples as desired while knowing correct transcriptions. To avert this possible weakness, it is designed to be highly customizable.

Default security of this CAPTCHA consist of image distortion, random lines and added noise. It is also supports any TTF font. An example of a challenge generated by this service can be seen in Figure 1.1a.

1.2.2 Captchas.net

Captchas.net is a service ran on its dedicated server, which provides generated challenges. Therefore, its source codes are not needed to be available. Still an attacker can generate thousands of examples with known transcriptions by sending enough requests to the server.

This CAPTCHA heavily relies on noise as the main security feature, while each character is rotated as well. A challenge generated by this service can be seen in Figure 1.1b.

1.2.3 BotDetect CAPTCHA

BotDetect CAPTCHA is a paid, up–to–date service supposedly used by many government institutions and companies all around the world [12]. It features as many as 60 different schemes with different security features.

Its base price starts at $99 per year however for my experimenting I have used the free version which comes with limitations such as:

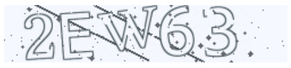
- a watermark in the generated CAPTCHA image 50 % of the time,

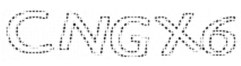
- obfuscated source codes, and
- a permanent link to the source webpage.

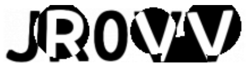
As I was able to modify its obfuscated source codes, I removed the free version limitations and also altered the challenge generation process to suit my needs. This way I am able to generate as many challenges with any specific properties as needed.

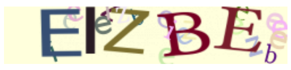
Given the reasons above I have chosen this CAPTCHA service for my experiments in this bachelor thesis.

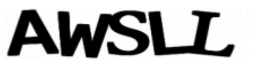
Scheme suite The BotDetect CAPTCHA is able to generate challenges in 60 schemes, out of which I have deliberately chosen 11 to be as much different as possible. See Figure 1.4 for examples of the chosen schemes.

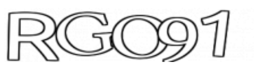

(a) Snow (s04)

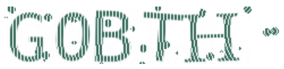

(b) Stitch (s08)

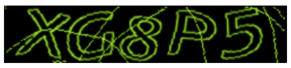

(c) Circles (s10)

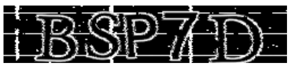

(d) Mass (s14)

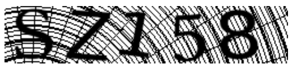

(e) BlackOverlap (s16)


(f) Overlap2 (s18)


(g) FingerPrints (s25)


(h) ThinWavyLetters (s30)


(i) Chalkboard (s31)


(j) Spiderweb (s41)

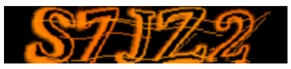

(k) MeltingHeat2 (s52)

Figure 1.4: Schemes generated by the BotDetect CAPTCHA

The *BlackOverlap* scheme acts as a baseline as it should be the easiest to solve. It has almost no protection against character recognition. It contains clear, only slightly rotated and barely touching characters.

Other schemes display various security features such as random lines and other objects occluding the characters, jagged or translucent character edges and global warp. The *Circles* scheme stands out with its color inverting randomly placed circles. This property could make it harder to recognize than others, because the solver somehow needs to account for random parts of characters and their background switching colors.

Chapter 2

Artificial Neural Network

The Artificial Neural Network (ANN) is a computational model based on large collection of simple, highly interconnected units (called artificial neurons) inspired by biological nervous systems, such as human brain, process information [13].

These artificial neurons each perform simple computation based on its inputs to produce an output – an *activation*. Nevertheless, an ANN can compute very complex functions by *layering* these simple computations.

ANNs in general are self-learning and trained instead of being explicitly programmed. This is a vital property in tasks where the solution is difficult or even impossible to express as a traditional computer program. Unfortunately, it comes at a price of ANNs being inherently very opaque when trying to interpret their inner computation processes. And without the ability to explain ANN's outputs, it is hard to be confident in the reliability of an ANN that addresses a real-world problem [14].

Even though other types of ANNs exist as well, I will solely use the feedforward ANNs, because they are well-suited for the task of character recognition from images. For example they achieve as low error rate as 0.21 % on the test set of the MNIST database of handwritten digits [11]. Given the task of character recognition, I will also only consider the use of ANNs for the task of multinomial classification.

2.1 Feedforward Neural Networks

A feedforward neural network is the simplest type of ANN. Its units are organized in layers. Data in this type of network simply flow through it from layer to layer, until they reach the output layer – which provides the output of the network. In this thesis I will study (and use) two types of feedforward ANNs:

- a multi-layer perceptron (MLP), and

- a convolutional neural network (CNN).

Another categorization used among feedforward ANNs is whether they are shallow or deep. The *shallow* ANN is defined as having only single hidden layer (therefore having two layers at most in total), while the *deep* ANN as having at least two hidden layers.

2.2 Artificial Neuron

A real neuron is the basic building block of the human brain. It weights its incoming signals in its dendrites, its soma accumulates potential from these weighted signals and when a certain threshold is met, it fires its own signal. The signal is transmitted in a form of spikes in temporal domain.

An artificial neuron is for ANNs the same as a real neuron is for our brains; it is very vaguely based on it. Specifically, in feedforward ANNs an artificial neuron computes a weighted sum of its inputs, adds its bias and conveys this value through its activation function forward to neurons connected to it. This process can be expressed by the following equation:

$$h(\mathbf{x}) = g\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right) = g\left(\mathbf{w} \cdot \mathbf{x} + b\right) \tag{2.1}$$

where $\mathbf{x}$ is vector of neuron's inputs, $\mathbf{w}$ is vector of corresponding weights, b is neuron's bias and g is the neuron's activation function.

2.3 Activation Function

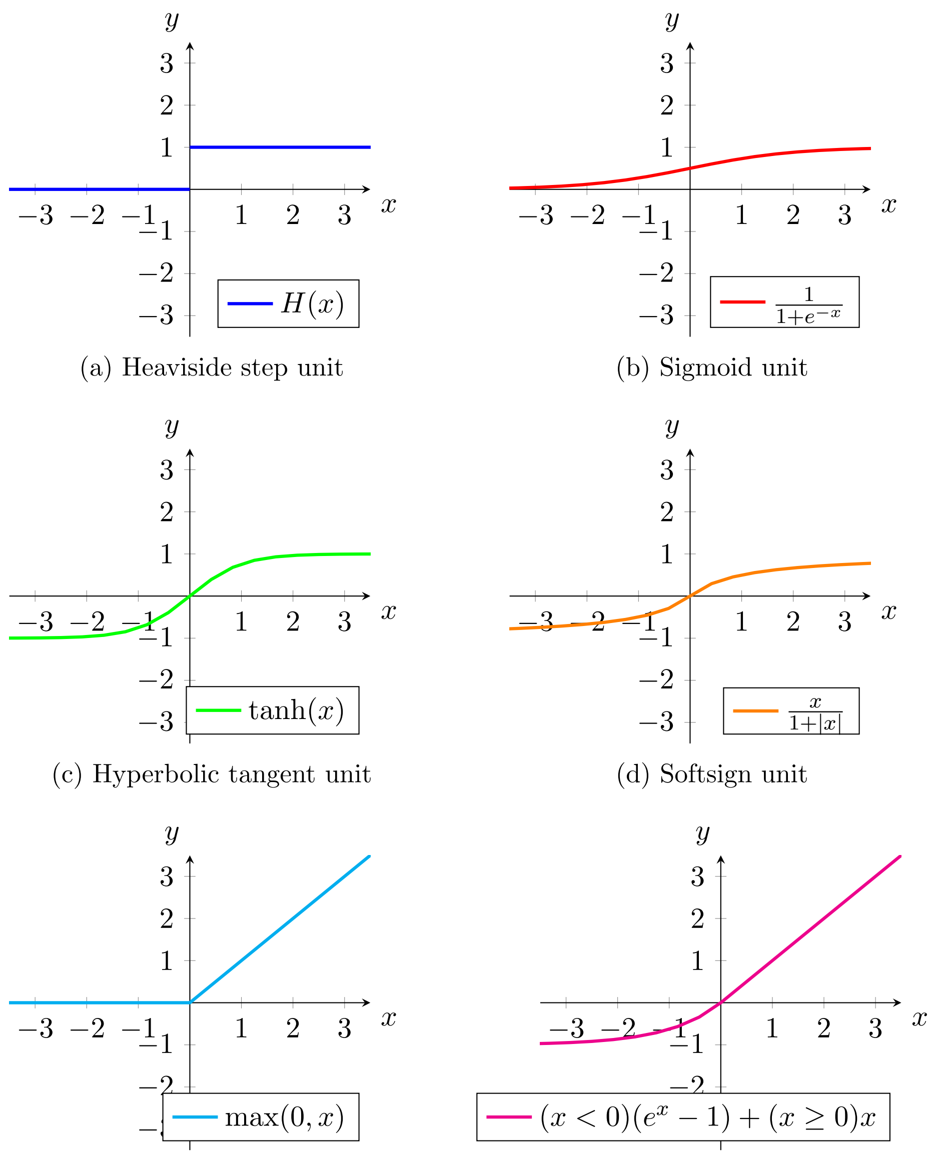
Real neuron transmits its information in the rate in which it fires discrete spikes in time. The more strongly it is "excited" about something, the more frequently it can fire.

Artificial neurons in the feedforward setting, on the other hand, do not have the option of firing in time. Their input signals come, they perform some calculations and forward just a single value – their output. The intuition behind this value is that it resembles the rate of fire of the real neuron.

An activation function in artificial neuron gives it the possibility to have a nonlinear output. Figure 2.1 shows examples of commonly used nonlinear activation functions.

2.3.1 Output Activation Function

Specifically to the multiclass classification task, it is desirable of the ANN's output to be a probability distribution over classes. The softmax function σ (a multiclass generalization of the sigmoid function [15]), when applied to a



(a) Heaviside step unit

(b) Sigmoid unit

(c) Hyperbolic tangent unit

(d) Softsign unit

(e) Rectified linear unit (ReLU)

(f) Exponential linear unit (ELU)

Figure 2.1: Activation functions used in artificial neurons

vector $\mathbf{z}$ of neurons' output values, serves exactly this purpose. For the j-th value of the resulting vector, it is defined as:

$$\sigma(\mathbf{z})_j = \frac{e^{\mathbf{z}_j}}{\sum_i e^{\mathbf{z}_i}} \tag{2.2}$$

2.4 Single-layer Perceptron

A single-layer perceptron ANN is the simplest type of feedforward ANN in the sense that it consists of just one layer of neurons. This means it can solve only linearly separable tasks (recall equation 2.1) no matter its activation function. It can be thus seen as a linear classifier.

The original perceptron uses the Heaviside step activation function (see Figure 2.1a) and the intuition behind its learning process is that it simply tries to adjust its weights in such a way that fixes all errors occurred in the training set. As a result, the training process of a perceptron ANN effectively stops whenever there are no more classification errors in the training set.

The perceptron unit can be also used with with any other reasonable activation function, its limitations however persist.

2.5 Multi-layer Perceptron

A Multi-layer perceptron (MLP) is an extension to the single-layer perceptron model. It consists of at least two layers of neurons. Typically each neuron has a connection to all neurons in the preceding layer. This is why a MLP is also known as the fully-connected ANN, Figure 2.2 depicts this fully-connectedness.

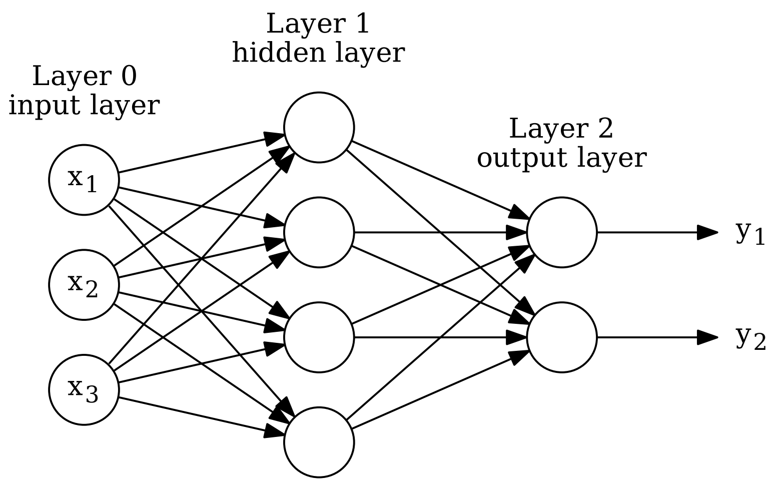


Figure 2.2: A fully-connected feedforward ANN

Each subsequent layer of neurons thus gets outputs of the previous layer, which, thanks to the activation function's nonlinearity, computes some nonlinear function given its inputs. This potentially results in computing gradually more and more complex function of the network's inputs.

Although a MLP with a nonlinear activation function under mild assumptions consisting of even just two layers (a shallow MLP) with finite number of neurons was proven to be universal approximator [16], just two layers might not be the most efficient MLP architecture. For example in order to represent certain decision surfaces with a shallow MLP, one would be forced to use exponentially more neurons then it would be required using a deep MLP [17].

2.6 Convolutional Neural Network

A Convolutional neural network (CNN) is a type of feedforward ANN that makes the explicit assumption that its input data are images. It is not much different from a standard MLP, however this assumption allows it to greatly reduce number of parameters of the network and therefore be more efficient in computation.

The first CNN called was introduced in 1980 by K. Fukushima under the name *Neocognitron*. It was developed with inspiration taken from the visual input processing part of mammalian brain – the visual cortex, whose structure it tries to mimic. Thanks to this the Neocognitron was able to achieve position-invariant pattern recognition [18].

Modern CNNs usually consist of layers of three types:

- the convolutional layer,
- the pooling layer, and
- the fully-connected layer.

Figure 2.3 gives an overview of an CNN architecture, which is using all three types of layers.

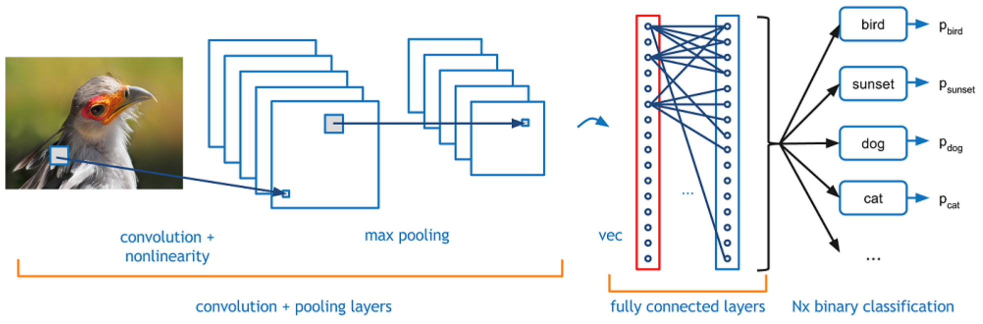


Figure 2.3: A convolutional neural network architecture [2]

2.6.1 Convolutional Layer

Artificial neurons in this layer still perform the same operation as described in Equation 2.1, however now the structure in which neurons are laid out within each layer is very important – they create stacks of *activation maps*. Also, each neuron has only spatially local connections, which form its *receptive field* of the previous layer. Please refer to the Figure 2.3 for illustration.

Moreover, all neurons within one activation map share their weights and bias. This is done to greatly reduce the number of parameters to be learned, with the motivation of it being useful to have a feature detector applied at every position of the image – it serves the position-invariance.

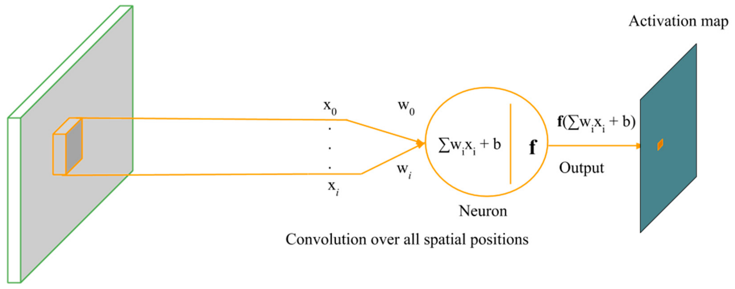


Figure 2.4: A convolutional layer [3]

The name of this layer comes from the fact, that it can be thought of as a set of *kernels* (one for each activation map) being convolved across the width and height of the layer's input, computing the dot product between the kernel's values (the neuron's weights) and the input's values. Afterwards the respective biases are added and all values are passed through an activation function. Figure 2.4 illustrates this computation. A convolutional layer thus needs these hyper-parameters to be set:

- the width and height of its kernel (the receptive field),
- the step size among both width and height axes, which specifies how far apart are the receptive fields' centers,
- the number of kernels, and
- the activation function.

2.6.2 Pooling Layer

A pooling layer is usually used within a CNN in between convolutional layers to reduce dimensionality of the CNN's intermediate activations, while (hopefully) keeping the most salient ones [2]. It has no trainable parameters and works by performing some general function of its receptive field. The usual function of choice is the maximum or the average of its input values [2]. It also helps with position- and rotation-invariance, as it gradually discards (when used multiple times within a CNN) exact spatial location [2].

Figure 2.5 gives on example of a max pooling with the receptive field of 2×2 and a step size of 2 in both axes.

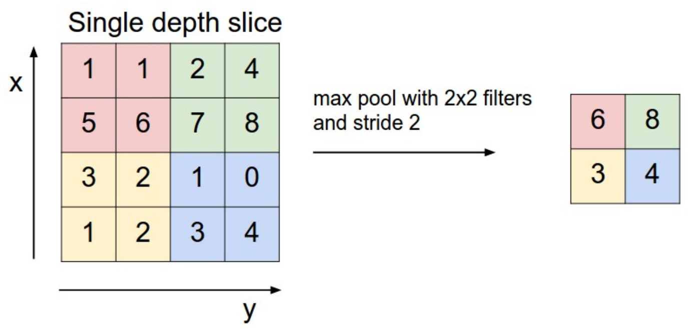


Figure 2.5: Max pooling illustration [2]

2.6.3 Fully-connected Layer

After some number of convolutional and pooling layers, a small number of fully-connected layers (a MLP) usually gets appended to facilitate classification [19], as can be seen in Figure 2.3. The intermediate activations just need to get flattened from a three-dimensional matrix into a vector in order to become valid as an input to a MLP.

2.6.4 The All Convolutional Net

It is also possible to create a CNN using convolutional layers only [19]. A convolutional layer is able to replace a pooling layer by using larger steps between receptive fields' centers; a fully-connected layer with n neurons can be replaced by a convolutional layer with n kernels of size 1×1.

2.6.5 Generalization of a Convolutional Layer

CNN's usage is not bounded only to images – to two-dimensional convolutions, It is also being used in one- and three-dimensional convolution setting:

- one-dimensional convolution for text understanding [20], and
- three-dimensional convolution for human pose estimation in space and time [21]).

2.7 ANN Learning

ANN learning is a process of adjusting ANNs parameters in such a way that when it is finished, the trained ANN preforms the intended transformation of the input data. For example, when an ANN is trained for the task of classifying images (e.g. dogs, cats, ...), it performs a transformation from an input image to a vector containing a probability distribution over all classes with the correct class's probability preferably being close to one.

An ANN can be viewed as a function $f_\theta : X \to Y$ given its parameters θ, where X is the input space and Y is the output space. Because it is not known how to determine the ANN's parameters θ explicitly for the task at hand, an alternative approach is needed – the ANN learns them itself by a process of optimization. For such process to work, the ANN needs sample data called the *training dataset.*

The process of learning can be divided into two categories based on whether the target classes for the training inputs are provided to the network or not: in the supervised learning setting the target classes are provided, while in the unsupervised learning setting they are not.

Since the task of CAPTCHA recognition (as well as the above described example) falls within the category of supervised learning, I will study only this type of ANN learning.

2.7.1 Supervised learning

The supervised version of ANN learning requires a set of n training example pairs of the form $\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \dots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$, where $\mathbf{x}^{(i)}$ is the input feature vector of the i-th example and $\mathbf{y}^{(i)}$ is its target class. Supervised learning is then the process of adjusting the ANN's parameters by optimizing its performance (by some appropriate algorithm) on the training data, based on a performance evaluation function, often called the *loss* function.

2.7.2 Loss Function

Loss function allows the evaluation of the ANN's performance on each example it is presented with. The higher the resulting loss value is, the worse the performance.

Common choices for loss functions include: the mean squared error (MSE) and the cross-entropy loss. As per [22]: "the cross-entropy is nearly always the better choice, provided the output neurons are sigmoid neurons". Since for a classification task the output neurons have a softmax function (as briefly discussed in Section 2.3.1), which is just a multiclass generalization of the sigmoid function, I will further consider only the cross-entropy loss.

Cross-entropy Loss Let $\mathbf{y}$ be a vector containing the target probability distribution for the training example $\mathbf{x}$, and $\hat{\mathbf{y}}$ be an ANN's output probability distribution for the example $\mathbf{x}$. Then, the cross-entropy loss can be computed as:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_j \mathbf{y}_j \ln \hat{\mathbf{y}}_j \tag{2.3}$$

Just for the sake of completeness, $\hat{\mathbf{y}}$ can be computed using function f_θ (as defined in Section 2.7) of the training input $\mathbf{x}$:

$$\hat{\mathbf{y}} = f_\theta(\mathbf{x})) \tag{2.4}$$

2.7.3 Backpropagation Algorithm

In order for an ANN to learn to perform a classification task, it is needed to define an objective function, minimization of which will result in learning the intended task.

Given a supervised training dataset S consisting of n example pairs $\{\mathbf{x}, \mathbf{y}\}$ (as defined in Section 2.7.1), an objective function using a cross-entropy loss can be defined as:

$$\mathcal{O}(S) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\mathbf{y}^{(i)}, f_\theta(\mathbf{x}^{(i)})) \tag{2.5}$$

Such function is differentiable w.r.t the parameters θ, however, only the gradients of the ANN's output layer can be estimated directly. Therefore, the backpropagation algorithm was invented to estimate gradients of the previeus layers.

A loss (an error) value is computed in the *forward* pass. Then, in the *backward* pass, the loss value is distributed among neurons, layer by layer (in the opposite direction to the forward pass – from the output layer all the way to the first hidden layer). While distributing the loss values for each neuron (which roughly corresponds to the neuron's contribution to the final loss value), the neuron's gradients w.r.t. to its parameters are estimated. And since the network's gradients can be estimated, a gradient-based optimization can be applied to train the network.

2.7.4 Stochastic Gradient Descent

The stochastic gradient descent (SGD) is an iterative algorithm for first-order gradient-based optimization of stochastic objective functions. The true gradient of the objective function w.r.t. its parameters is estimated by a gradient at a single example. This estimated gradient in then used to update the objective function's parameters in the direction proportional to the *negative* of the estimated gradient. A less noisy estimation is generated by taking an average of a gradient at some number of examples – a gradient of a *minibatch* of training data.

As it is possible to estimate gradients for all parameters of a feed-forward ANN (using the backpropagation algorithm), a SGD algorithm can be used to update its parameters with the goal to minimize the objective function with the side-effect of the ANN being trained for the task for which the objective function measures performance. See Figure 2.6 for an example of optimization using the SGD algorithm.

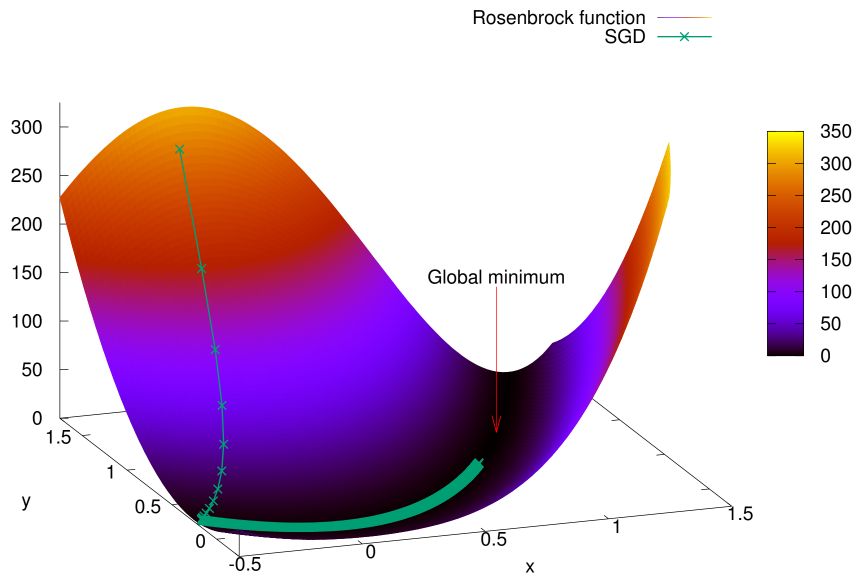


Figure 2.6: 2000 iterations of the SGD optimization

Chapter 3

State of the Art

The common approach applied in text-based CAPTCHA breaking is to use a pipeline consisting of several steps out of which only one – character recognition – is done by a general machine learning algorithm. This more algorithm-based, scheme-specific approach rose in the early 2000s and has proven to work quite well. I will discuss this approach in section 3.1.

To fix discovered vulnerabilities, more advanced CAPTCHA schemes were developed, which in turn meant new challenges for CAPTCHA solver developers. To get rid of the need for new scheme-specific algorithms, an attempt for more general CAPTCHA solver was made (sec. 3.2) and only the latest advancements in technology allowed for approaching the visual CAPTCHA transcription problem in completely general way, feeding (normalized) raw pixel data into a machine learning algorithm and expecting a transcription on the output (sec. 3.3).

3.1 Scheme-specific Approach

One can find multiple examples of the scheme-specific approach [4, 23, 24, 25] and they all share the same inherent drawback – being more or less specialized in solving only one scheme.

For example the most recent work (out of the four mentioned) [4] suggests using this five–stage pipeline (Figure 3.1 illustrates the first three stages):

1. **Pre-processing**: Background removal, line detection and removal, noise reduction, binarization.

2. **Segmentation**: Creating segments containing characters using various techniques, the most common being the Color Filling Segmentation [23].

3. **Post-segmentation**: Processing individual segments to make the recognition easier. Segments' sizes are always normalized in this stage.

4. **Recognition**: Character recognition using a classifier of choice (kNN or SVM in this case).

5. **Post-processing**: Improving the overall output when possible, for example spell checking can be performed when dictionary words are used to generate the CAPTCHA challenge.

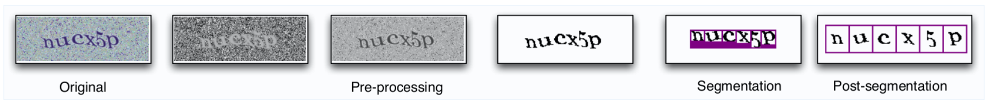


Figure 3.1: An example of first three stages of a scheme-specific pipeline [4]

Measured precision varies from CAPTCHA scheme to scheme. Their algorithm was not able to transcribe certain schemes at all, on others the accuracy of transcription was as high as 93%.

Thus this approach has been proven to work, however each CAPTCHA scheme might require different method to exploit different invariant(s) present in order to segment and recognize correctly.

3.2 General Approach

A more recent work [26] tries to address this issue by jointly performing the task of segmentation as well as character recognition by a machine learning algorithm. It consists of several parts as well:

1. **Cut-points detector** finds all possible cuts which segment a CAPTCHA into individual characters.

2. **Slicer** builds a graph of potentially meaningful segments.

3. **Scorer** traverses this graph applying OCR and assigns a recognition confidence score to each potential segment.

4. **Arbiter** selects the final value for the CAPTCHA challenge using voting weighted by the recognition score confidence.

Authors use what they call the “reinforcement learning process” where human provides important insight during training. A set of labeled CAPTCHAs is processed and all unsuccessfully classified examples are collected. For the failed CAPTCHAs, human is asked for feedback when a segment surrounded by two correctly classified segments is misclassified, because the misclassification could happen due to improper segmentation, or to bad recognition and the algorithm is unable to tell them apart by itself. The number of cases requiring manual intervention was small enough for the researchers to perform them themselves. The number of training examples was small as well –

26 examples per character, which in turn meant a training set of under 1000 CAPTCHAs.

3.3 General Approach Using Deep Learning

Another approach [27] arose from the need to recognize arbitrary multi-digit street numbers from the Google's Street View [28]. Their approach is fairly straight-forward and works on raw pixel data. Their only pre-processing step is to subtract the mean of each image and downscale it to a fixed square size. This image is then fed into a deep convolutional neural network (consisting of as much as 11 layers), which works as a feature extractor, on top of which sit 6 independent softmax classifiers:

- One classifying length of the number sequence present in the image; 7 classes: 0 through 5 plus an extra class representing "more then 5".
- Five classifying its corresponding number in the sequence; 10 classes (i.e. classifier 1 always classifying the 1^{st} digit in the sequence, classifier 2 always classifying the 2^{nd}, and so on).

The same approach was as an experiment used on a Google's (old) reCAPTCHA [29], which was at that time considered the most secure [27]. The feature extracting convolutional neural network has 9 hidden layers and is able to handle CAPTCHAs with length of up to 8 (thus having 9 independent softmax classifiers). In this experiment a remarkable accuracy of 99.7% was achieved. The drawback is the need of millions of training images and days of training time even on high performance GPUs.

Chapter 4

Algorithm Overview

My algorithm can be viewed as a three-stage pipeline consisting of these steps:

1. Create a heatmap of character locations using the sliding window technique paired with an ANN, which classifies whether there is a character in the center of its input or not.

2. Use the k-means algorithm to determine the most probable locations of characters from the heatmap.

3. Recognize the characters using a different specifically trained ANN.

4.1 Heatmap Generation

I decided to use the sliding window technique to localize characters within a CAPTCHA image. This approach is well known in the context of general object localization [30]. It allows us not only to classify individual characters but also to decide which parts of an image contain characters.

4.1.1 Sliding Window

A sliding window is a rectangular region of fixed width and height that *slides* across an image, from left to right, row by row, from top to bottom. It can be used at various scales and with various step sizes. However, no scaling is required for the challenges generated by the schemes which I will attempt to transcribe (see Figure 1.4), since the sizes of characters do not vary much. I expand the range by padding the input image with black pixels with a strip of width equal to half of the window's size. As a result the sliding window can slide out of the image by as much as half of its size in both axes.

4.1.2 Classifier

A feed-forward ANN with a single output neuron with the sigmoid activation function is used. Its output values (ranging from 0 to 1) resemble the probability of its input image having a character in the center of the image.

(a) Scheme s16 (b) Scheme s10

Figure 4.1: CAPTCHA examples with their heatmaps underneath

4.2 Clustering

I decided to use the k-means clustering algorithm to determine windows with characters close to their center. It is a simple clustering algorithm that seemed like a good choice to start with.

All points from a generated heatmap with value greater or equal to 0.5 [1] are added to a list of points to be clustered.

As there is always a constant number of characters present in all schemes, the k in the k-means (the number of centroids) is known. Also, I choose to initialize the centroids uniformly from left to right, vertically in the middle, as this provides a good initial estimation. Figure 4.2 illustrates the whole idea.

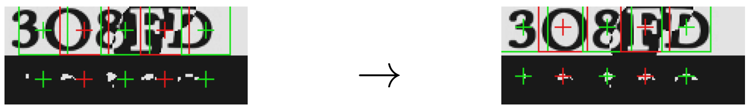
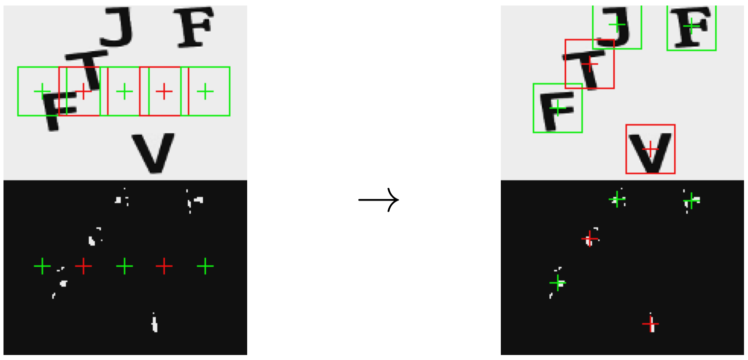

(a) Initial centroids (b) Final centroids

Figure 4.2: Heatmap clustering using k-means

It is also worth noting that this localization pipeline does assume the location of characters in any way whatsoever. It depends solely on the classifier's accuracy. Figure 4.3 demonstrates this property.

Moreover, a different clustering algorithm which would be capable of inferring the number of clusters, could effortlessly extend capabilities of this pipeline to work with any number of characters.

[1]according to the classifier there is at least a 50% chance that the window contains a centered character



(a) Initial centroids (b) Final centroids

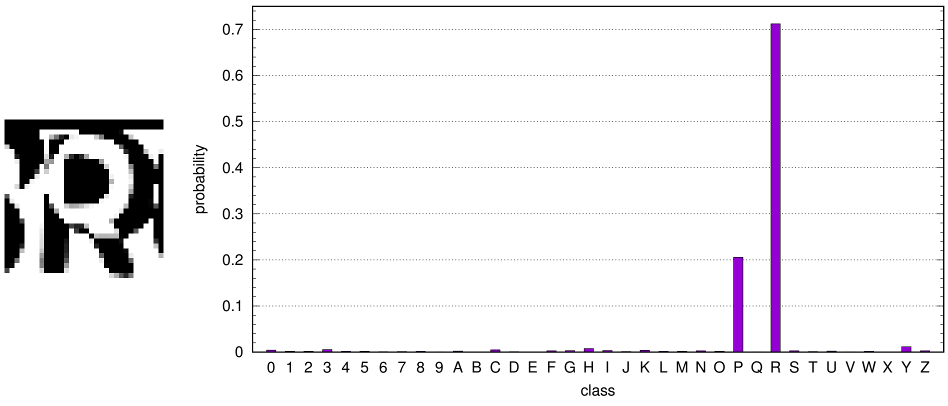
Figure 4.3: Heatmap clustering on random character locations

4.3 Recognition

Assuming that the character localization part worked well, windows containing characters are now ready to be recognized. This task is known to be easy for computers to solve; in fact, they are even better than humans [31].

Again a feed-forward ANN is used, this time with an output layer consisting of 36 neurons paired with the softmax activation function to provide probability distribution over classes: numbers 0–9 and upper-case letters A–Z.

Finally, a CAPTCHA transcription is created by writing the recognized characters in the ascending order of their x coordinates (i.e. from left to right). Figure 4.4 shows the example of a window with the character inside and its corresponding class probability distribution produced by an ANN.



(a) Input image (b) Probability distribution over classes

Figure 4.4: Classification of a character

Chapter **5**

Experiments

In this chapter, I will discuss the way the training data were generated. Then, I will briefly introduce the framework I have programmed in. And finally, I will demonstrate that the chosen CAPTCHA schemes are indeed vulnerable to such attack. I will do so by evaluating accuracy of multiple ANN architectures, including a shallow MLP as well as a deep CNN.

5.1 Training Data Generation

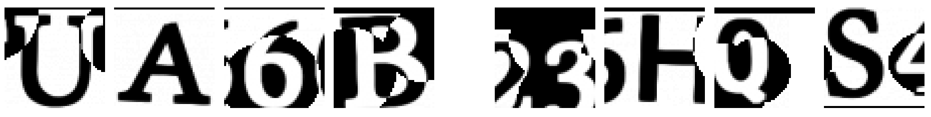
Training an ANN usually requires a lot of training examples (in the order of millions in the case of a very deep CNN, such as the one described in Section 3.3). The number of examples needed to train an ANN goes lower with decreasing number of layers. But still it is advised to have at least multiple times the number of all parameters in the network [32].

Manually downloading, cropping and labeling such high number of examples is infeasible. That is why I opted for an alternative of altering the freely available (obfuscated) code in such a way that it generates directly usable examples together with their target class. Therefore, the whole process can be automated and thus thousands of examples can be generated effortlessly.

5.1.1 Localization Data

The localization ANN, used as the classifier in the sliding window approach, has to classify images into two classes – whether a character is in the center of the image or not. Training examples thus have to be images taken from various locations of CAPTCHA images with their appropriate labels.

Moreover, it is needed to pay more attention to locations in between characters, as it is hard to tell apart cases when the window is located in between two characters and when the window is spot on a character. On the other hand an easily classifiable example is a window located above or below a character as it does not contain any character-like pixels in its top or bottom part. Figure 5.1 shows examples of such training data.

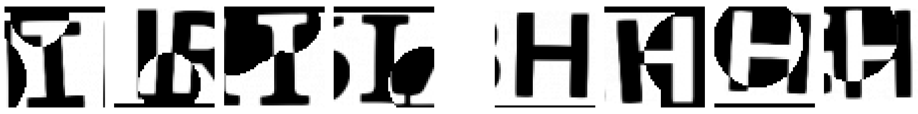


(a) Class: centered character (b) Class: character off-center

Figure 5.1: Examples used to train a localization ANN

5.1.2 Classification Data

A character recognition ANN needs enough training examples of each character. This time, however, they must be floating around across the whole window. The reason is that the inferred locations of characters are almost never perfect and the characters may therefore occur anywhere within or even slightly outside the window. Therefore, the character recognition ANN must account for such cases. Otherwise, it might fail to recognize the off-center ones.



(a) Class: character I (b) Class: character H

Figure 5.2: Examples used to train a character recognition ANN

5.2 Setup

I have used the Torch7 framework for all my experiments. As per [33]: "Its goal is to provide a flexible environment to design and train learning machines. Flexibility is obtained via Lua, an extremely lightweight scripting language. High performance is obtained via efficient OpenMP/SSE and CUDA implementations of low-level numeric routines." Creation of a MLP can be achieved in just a few lines of code, as can be seen in Listing 1.

All experiments were run on my personal laptop on its Intel Core i5-3210M CPU, on which training of even the most complex models did not take more than half an hour. The reported accuracies represent a percentage of correctly transcribed 5-letter challenges out of a set of 300. Moreover, all four combinations of shallow/deep and localization/recognition ANNs are tested to reveal more specifically the source of accuracy improvement.

All ANNs were trained on training sets of 100 000 examples using the ADAM optimizer, which is a better performing extension of the SGD optimizer [34]. The ReLU activation function is used after each fully-connected or convolutional layer, as it performs better than the conventional Tanh activation function [35]. The training stops whenever the validation accuracy does not improve for 2 consecutive iterations through the whole training dataset.

```lua
local nn  = require 'nn'
local mlp = nn.Sequential()   -- make a multi-layer perceptron

mlp:add(nn.Linear(2, 20))     -- 2 inputs, 20 (hidden) neurons
mlp:add(nn.ReLU())            -- ReLU activation function
mlp:add(nn.Linear(20, 4))     -- 20 inputs, 4 (output) neurons
mlp:add(nn.SoftMax())         -- softmax activation function

local input  = torch.rand(2)  -- vector of 2 norm. dist. values
local output = mlp:forward(input)
```

Listing 1: Creation and usage of a shallow MLP in Torch7

5.3 Determining ANN Architecture

Once trained, the architecture of feed-forward ANNs is not modular anymore. Each neuron within each layer is heavily dependent and specialized on its preceding neurons. Therefore by removing one, the ANN's performance can be degraded severely. Adding a neurons to an already trained ANN may not be beneficial as well, because complex relations between neurons have already been formed, and to include a new neurons into those relations constructively might require to retrain the whole network from scratch. Attention thus needs to be paid when designing and creating ANNs.

5.3.1 Shallow ANN

I tried different architectures for both localization and character recognition ANNs to see which combination works the best. The only possible variation in shallow, fully-connected ANN is changing the size of their hidden layer (the number of neurons within this layer). I have tried sizes of $\{15, 30, 60, 90\}$ for the localization ANN and sizes of $\{30, 60, 120, 180, 250\}$ for the character recognition ANN. The scheme s10 seemed like a good choice for this task, since it has a security feature which (in my opinion) should be difficult to overcome.

The results presented in Figure 5.3 suggest that the size of the localization ANN is not that important. What greatly affects the overall accuracy is the size of the recognition ANN. Regardless of this observation, I will use the best performing combination: the localization ANN with hidden layer of size 60 and

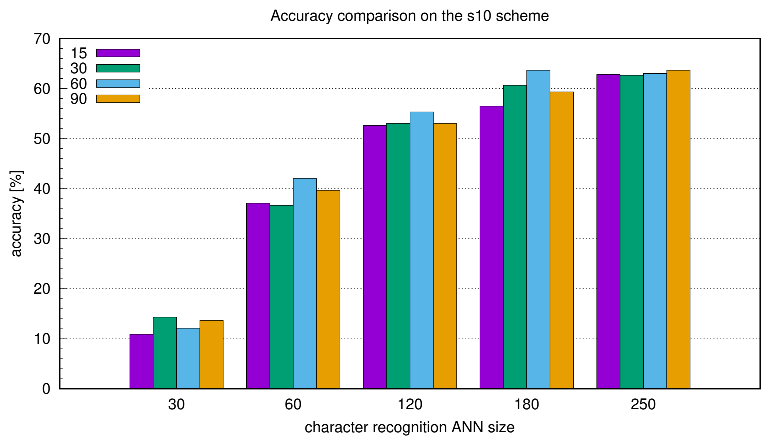


Figure 5.3: Comparison of different shallow ANN architectures on the s10 scheme

character recognition ANN with hidden layer of size 180. This combination achieves the accuracy of 63.667 % on the scheme s10.

5.3.2 Deep ANN

The LeNet-5 [36] is the main source of inspiration for both my localization and recognition deep ANNs. It is a fairly old architecture that was created for the task of handwritten digit recognition. Therefore, it is rather lightweight. Newer architectures usually focus on the more difficult task of (large-scale) image recognition and are ever so resource demanding (e.g. AlexNet [35], VGGNet [37]).

My deep localization ANN consists of the following layers:

- two convolutional layers with six and sixteen 5×5 kernels respectively,
- two 2×2 max-pooling layers, and
- a fully-connected output layer with 36 neurons.

The deep character recognition ANN contains an additional fully-connected hidden layer with 120 neurons, which is placed right before the output layer. Figure 5.4 illustrates this architecture.

5.4 Single Scheme Accuracy

This experiment corresponds to the situation of someone trying to attack a scheme of a certain website. The transcribing algorithm would typically be

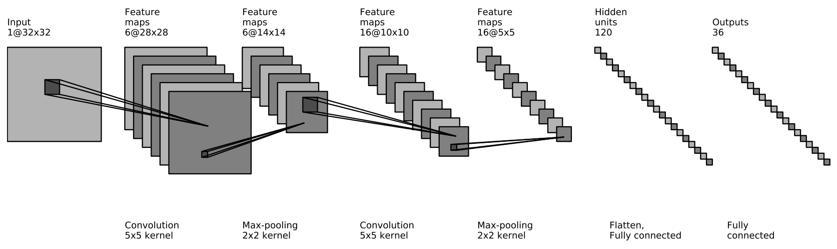


Figure 5.4: Deep CNN architecture for character recognition

designed/trained just for the targeted scheme. The same applies to the ANNs used in this experiment. They are trained and tested on just a single scheme.

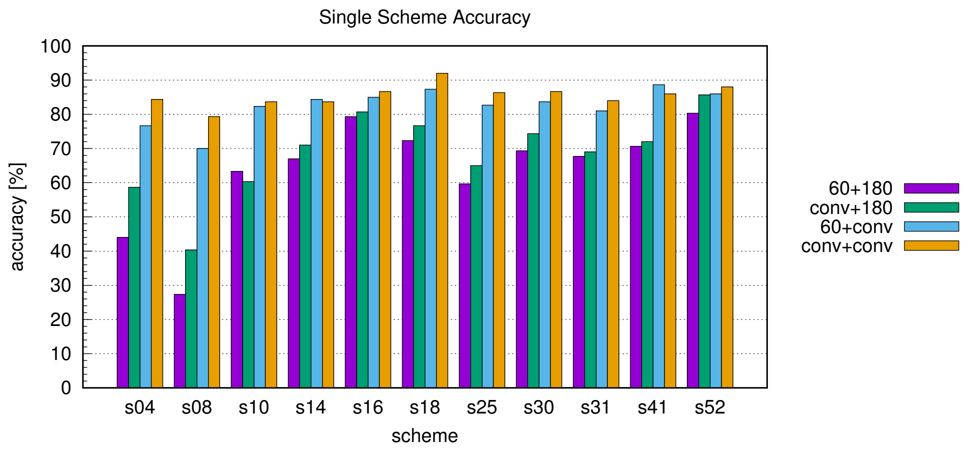
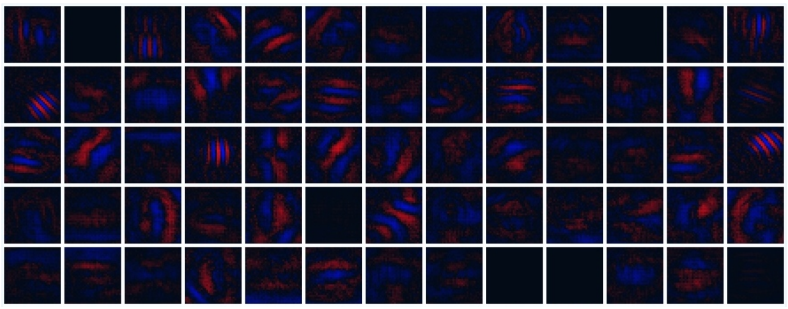


Figure 5.5: Testing accuracies for the single scheme experiment

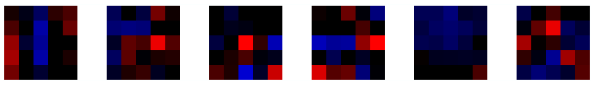
60+180	63.727 %	±	15.587
conv+180	+4.788 %	±	5.128
60+conv	+18.788 %	±	10.916
conv+conv	+21.788 %	±	13.464

Table 5.1: Summarized average accuracy gains from Figure 5.5

Table 5.1 shows that the biggest impact has the usage of the deep CNN in the character recognition part of the pipeline. Figure 5.6a visualizes a subset of neuron's parameters of the character recognition shallow ANN trained on the s08 scheme (the s08 scheme was chosen because the difference between shallow and deep architecture seems to be the most pronounced). Generally,



(a) A subset from the hidden layer of the shallow ANN



(b) The first convolutional layer of the deep CNN

Figure 5.6: Visualization of neuron's parameters of a character recognition ANNs; trained on the s08 scheme; blue represents the negative weights, red the positive ones, brightness indicates magnitude

various pairwise location-specific line detectors can be seen. The location-specificity is very important here as it means that the ANN needs to learn the same detectors multiple times at various locations. This is ineffective, and on top of that the training data might fail to contain examples of all characters at all locations. Therefore, whenever a character occurs at a previously unseen location, the shallow ANN may easily misclassify it.

On the other hand, the deep character recognition CNN needs to learn its detectors only once, as they get applied (convoluted) over each spatial location of the input image. This makes the deep CNN much more neglectful of the exact location of the character and thus achieves better accuracy. Visualization of its parameters of the first layer can be seen in Figure 5.6b.

This argument is further supported by the low accuracy differences observed when shallow/deep ANN is used for localization. The localization task is quite opposite to the character recognition task in the sense that the location is very important for it. In other words, the usage of a shallow ANN for localization does not degrade the overall accuracy as much, because location-specificity is something it is good at. Still, the deep CNN adds a few percentage points to the overall accuracy when employed for the localization task.

5.5 All Schemes Accuracy

The training dataset of this experiment contains examples of all 11 schemes. Its size, however, remains the same. Therefore, each scheme is represented by only 1/11 of the dataset size. As all schemes are trained jointly, only four

ANN instances (one for each architecture) are needed for this experiment. This experiment should test their ability to learn a common abstract representation of each class across all schemes.

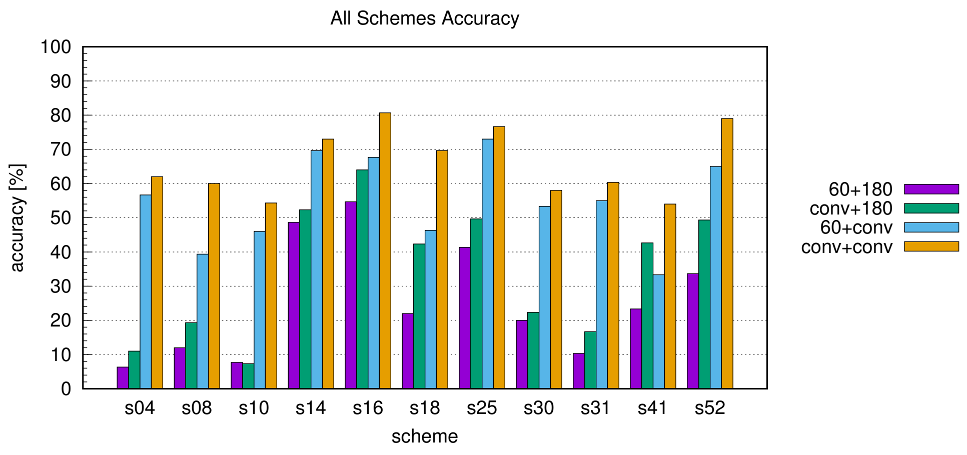


Figure 5.7: Testing accuracies for the all schemes experiment

60+180	25.455 %	±	16.881
conv+180	+8.818 %	±	6.845
60+conv	+29.576 %	±	12.333
conv+conv	+40.697 %	±	10.441

Table 5.2: Summarized average accuracy gains from Figure 5.7

Table 5.2 summarizes the average accuracy improvements against the shallow baseline. The pipelines using the character recognition CNN perform better by approximately 30 % on average. This is even a bigger improvement than the previous experiment showed.

The ability of deep CNNs to abstract from a scheme and focus on the important shapes seems to emerge even more in this experiment. This time also with the localization CNN, as the accuracy gain is on average approximately 10 %.

5.6 Leave-one-out Scheme Accuracy

This experiment simply performs the leave-one-out cross-validation. It basically pushes the previous experiment even further, as the generalization capabilities of the ANNs are explicitly tested. The ANNs are trained on 10 schemes and the one left out is used for testing.

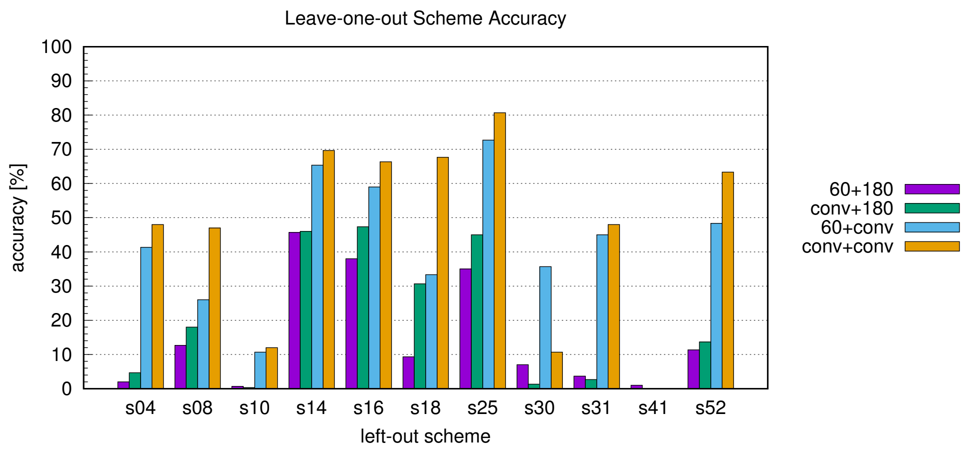


Figure 5.8: Testing accuracies for the leave-one-out scheme experiment

60+180	15.121 %	±	16.380
conv+180	+3.939 %	±	7.395
60+conv	+24.636 %	±	13.679
conv+conv	+31.545 %	±	20.070

Table 5.3: Summarized average accuracy gains from Figure 5.8

The deep CNNs confirm their ability to generalize better then the shallow ANNs (see Table 5.3). The only exception occurs for the scheme s41. Not a single challenge was transcribed correctly from this scheme by pipelines with the deep CNN(s). The probable reason is that the scheme uses very different protection which is, moreover, always present. The same reasoning goes for the results on the s10 scheme.

This all suggests that even though the pipeline with deep CNNs is capable of transcribing schemes it has never seen before, its span is still limited only to schemes fairly similar to the ones it was trained for.

Conclusion

This thesis aimed to evaluate the accuracy of shallow and deep ANNs used for the task of text-based CAPTCHA recognition. The main goal was to design and implement an algorithm capable of transcribing challenges generated by various schemes.

First, several CAPTCHA types were discussed and three different text-based CAPTCHA solutions were presented. The BotDetect CAPTCHA solution was selected as the most appropriate. Also, principles of ANNs were studied.

Then, to gain insight in the relevant area, state of the art solutions were analysed. As none of them seemed usable, an algorithm utilizing two artificial neural networks, the sliding window technique and the k–means clustering was designed and implemented.

Finally, the properties of generated training datasets for both localization and classification tasks were explained. The proposed algorithm was successfully implemented and suitable architectures for both shallow and deep ANNs were chosen. Three performed experiments facilitate the comparison of accuracies between shallow and deep architectures. An average accuracy of 63.7 % and 85.5 % was achieved using shallow and deep ANNs respectively.

The algorithm can be further extended to work with challenges containing unknown number of characters. This would require a clustering algorithm capable of inferring the number of clusters.

Bibliography

[1] reCAPTCHA: Easy on Humans, Hard on Bots [online]. 2017, [Cited 2017-01-12]. Available from: https://www.google.com/recaptcha/intro/index.html

[2] Understanding Convolutional Neural Networks for NLP [online]. 2017, [Cited 2017-02-07]. Available from: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[3] Bejiga, M. B.; Zeggada, A.; Nouffidj, A.; et al. A Convolutional Neural Network Approach for Assisting Avalanche Search and Rescue Operations with UAV Imagery. *Remote Sensing*, volume 9, no. 2, 2017, ISSN 2072-4292. Available from: http://www.mdpi.com/2072-4292/9/2/100

[4] Bursztein, E.; Martin, M.; Mitchell, J. Text-based CAPTCHA Strengths and Weaknesses. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, New York, NY, USA: ACM, 2011, ISBN 978-1-4503-0948-6, pp. 125–138. Available from: http://doi.acm.org/10.1145/2046707.2046724

[5] Forum Bot – Forum Poster and Profile Bot [online]. 2017, [Cited 2017-01-03]. Available from: http://www.forum-bot.com/

[6] The Fake Traffic Schemes That Are Rotting the Internet [online]. 2017, [Cited 2017-01-03]. Available from: https://www.bloomberg.com/features/2015-click-fraud/

[7] How to scrape emails from websites, articles, social websites? [online]. 2017, [Cited 2017-01-03]. Available from: https://anygrowth.com/blog/en/articles/how-to-scrape-emails-from-websites-articles-social-websites/Q6G4N9

[8] Powerful software to assist ticket touts widely available online [online]. 2017, [Cited 2017-01-03]. Available from: https:

//www.theguardian.com/money/2016/may/21/ticket-touts-powerful-software-assist-widely-available

[9] Secureimage PHP Captcha [online]. 2017, [Cited 2017-01-14]. Available from: https://www.phpcaptcha.org/

[10] Free CAPTCHA-Service [online]. 2017, [Cited 2017-01-14]. Available from: http://captchas.net/

[11] Classification datasets results [online]. 2017, [Cited 2017-01-13]. Available from: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[12] BotDetect CAPTCHA Generator [online]. 2017, [Cited 2017-01-14]. Available from: https://captcha.com/

[13] Stergiou, C.; Siganos, D. Neural Networks [online]. Technical report, Department of Computing – Imperial College London, [Cited 2017-02-02]. Available from: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[14] Towell, G.; Shavlik, J. W. Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In *NIPS*, volume 4, 1991, pp. 37–43.

[15] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN 0387310738, 198 pp.

[16] Sonoda, S.; Murata, N. Neural Network with Unbounded Activations is Universal Approximator. *CoRR*, volume abs/1505.03654, 2015. Available from: http://arxiv.org/abs/1505.03654

[17] Montúfar, G. F.; Pascanu, R.; Cho, K.; et al. On the Number of Linear Regions of Deep Neural Networks. *CoRR*, volume abs/1402.1869, 2014. Available from: http://arxiv.org/abs/1402.1869

[18] Fukushima, K. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, volume 36, 1980: pp. 193–202.

[19] Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; et al. Striving for Simplicity: The All Convolutional Net. *CoRR*, volume abs/1412.6806, 2014. Available from: http://arxiv.org/abs/1412.6806

[20] Zhang, X.; LeCun, Y. Text Understanding from Scratch. *CoRR*, volume abs/1502.01710, 2015. Available from: http://arxiv.org/abs/1502.01710

[21] Grinciunaite, A.; Gudi, A.; Tasli, H. E.; et al. Human Pose Estimation in Space and Time using 3D CNN. *CoRR*, volume abs/1609.00036, 2016. Available from: `http://arxiv.org/abs/1609.00036`

[22] Introducing the cross-entropy cost function [online]. 2017, [Cited 2017-02-09]. Available from: `http://neuralnetworksanddeeplearning.com/chap3.html#introducing_the_cross-entropy_cost_function`

[23] Yan, J.; El Ahmad, A. S. A Low-cost Attack on a Microsoft Captcha. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, New York, NY, USA: ACM, 2008, ISBN 978-1-59593-810-7, pp. 543–554. Available from: `http://doi.acm.org/10.1145/1455770.1455839`

[24] Mori, G.; Malik, J. Recognizing Objects in Adversarial Clutter: Breaking a Visual Captcha. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'03, Washington, DC, USA: IEEE Computer Society, 2003, ISBN 0-7695-1900-8, 978-0-7695-1900-5, pp. 134–141. Available from: `http://dl.acm.org/citation.cfm?id=1965841.1965858`

[25] Zhang, J.; Wang, X. Breaking Internet Banking CAPTCHA Based on Instance Learning. *Computational Intelligence and Design, International Symposium*, volume 01, 2010: pp. 39–43. Available from: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5692658`

[26] Bursztein, E.; Aigrain, J.; Moscicki, A.; et al. The End is Nigh: Generic Solving of Text-based CAPTCHAs. In *WOOT'14 Proceedings of the 8th USENIX conference on Offensive Technologies*, 2014. Available from: `https://www.elie.net/publication/the-end-is-nigh-generic-solving-of-text-based-captchas`

[27] Goodfellow, I. J.; Bulatov, Y.; Ibarz, J.; et al. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. *CoRR*, volume abs/1312.6082, 2013. Available from: `http://arxiv.org/abs/1312.6082`

[28] Google Street View [online]. 2016, [Cited 2016-12-18]. Available from: `https://www.google.com/streetview/`

[29] von Ahn, L.; Maurer, B.; McMillen, C.; et al. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, volume 321, no. 5895, 2008: pp. 1465–1468, ISSN 0036-8075. Available from: `http://science.sciencemag.org/content/321/5895/1465.full.pdf`

[30] Lampert, C.; Blaschko, M.; Hofmann, T. Beyond Sliding Windows: Object Localization by Efficient Subwindow Search. In *CVPR 2008*, Max-Planck-Gesellschaft, Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 1–8.

[31] Chellapilla, K.; Larson, K.; Simard, P.; et al. Computers beat Humans at Single Character Recognition in Reading based Human Interaction Proofs (HIPs). In *CEAS 2005, Conference on Email and Anti-Spam*, January 2005. Available from: `https://www.microsoft.com/en-us/research/publication/computers-beat-humans-at-single-character-recognition-in-reading-based-human-interaction-proofs-hips/`

[32] Random Ponderings: A Brief Overview of Deep Learning [online]. 2017, [Cited 2017-02-13]. Available from: `http://yyue.blogspot.cz/2015/01/a-brief-overview-of-deep-learning.html`

[33] Collobert, R.; Kavukcuoglu, K.; Farabet, C. Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, NIPS Workshop*, 2011. Available from: `https://ronan.collobert.com/pub/matos/2011_torch7_nipsw.pdf`

[34] Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. *CoRR*, volume abs/1412.6980, 2014. Available from: `http://arxiv.org/abs/1412.6980`

[35] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, edited by F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, pp. 1097–1105. Available from: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`

[36] LeCun, Y.; Bottou, L.; Bengio, Y.; et al. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, volume 86, no. 11, November 1998: pp. 2278–2324.

[37] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, volume abs/1409.1556, 2014. Available from: `http://arxiv.org/abs/1409.1556`

Appendix A

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
ELU	Exponential Linear Unit
kNN	k–Nearest Neighbors
MLP	Multi Layer Perceptron
MNIST	a database of handwritten digits
MSE	Mean Squared Error
OpenMP	Open Multi-Processing
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SIMD	Single Instruction, Multiple Data
SSE	Streaming SIMD Extensions
SVM	Support Vector Machine
TTF	True Type Font

Appendix **B**

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
src ..................................... the directory of source codes
  impl ...................................... implementation sources
  thesis ............ the directory of LaTeX source codes of the thesis
text ........................................ the thesis text directory
  BP_Nikl_Matej_2017.pdf ............. the thesis text in PDF format
```