COM6906-001

140127647

The University Of Sheffield.

Individual Assessed Work Coversheet

Assessment Code: COM6906

Description: Final Dissertation

Staff Member Responsible:

Due Date: 10-09-2015 12:00:00

☑ **I certify that the attached is all my own work, except where specifically stated and confirm that I have read and understood the University's rules relating to plagiarism.**

I understand that the Department reserves the right to run spot checks on all coursework using plagiarism software.

Student Registration Number:

140127647

Assessment Code:

COM6906-001






University of Shefield

Cracking CAPTCHAs

Jennifer Parak

Supervisor: Dr. Richard Clayton

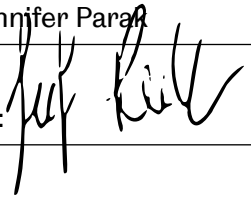
A report submitted in fulfillment of the requirements
for the degree of MSc in Advanced Computer Science
in the
Department of Computer Science

10th September 2015

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Jennifer Parak

Signature: 

Date: 10th September 2015

Acknowledgements

I would like to express my deepest appreciation to the University of Sheffield for giving me the opportunity to complete my MSc in Advanced Computer Science. Thanks to the Postgraduate Scholarship and the great support offered by the Department of Computer Science, I was able to fulfill my dream and graduate from one of the leading Russel Group Universities in the UK. I would also like to thank my supervisor, Richard Clayton, who supported my with great advice and feedback throughout this project.

My sincere gratitude goes to my friends who have specifically helped me through ups and downs during this summer: Harrie Mort, for being my best friend and helping me with this project, as well as the rest of "the Gang" Nicole Joann Toolseram and Sheryl Ng, for being there when I needed them the most. All my love to Andrew Maclean, who has managed to turn things around whenever I reached a low and didn't hesitate once when I needed help with this project. I would also like to thank Dan Sumner, who has been one of the most important people to me throughout University, and volunteered to proofread my second dissertation. Not to forget, Sarah Wilson, who made the hours spend in lecture halls and the library a lot more enjoyable. Furthermore, thank you to all the friends I made living abroad.

Finally, I would like to thank my family and my friends at home in Austria - may my compass always lead me back to you!

Abstract

Completely Automated Turing test to tell Computers and Humans Apart (CAPTCHAs) are widely used on the Internet to keep it safe from bots and spammers. CAPTCHAs present a challenge-response test, based on a hard Artificial Intelligence problem, which is supposed to be difficult to solve for computers, but easy to solve for humans. This research project was focused on cracking CAPTCHAs. From the literature review we concluded that research on text-based CAPTCHAs is exhausted and therefore drew our attention to breaking image-based CAPTCHAs and explored research in the field of image recognition. By attempting to crack this type of CAPTCHA, we were simultaneously attempting to create a programme, which is able to solve a hard computer vision problem of image labelling. This project should be seen as a proof of concept, exploring whether the new Google image-based CAPTCHAs are vulnerable to such an attack. In-depth research was carried out, in order to (1) examine approaches that have been used to classify images, (2) point out problems and difficulties of cracking an image recognition CAPTCHA, (3) identify the features of a 'strong' image recognition CAPTCHA and (4) use methods identified to break a simple image recognition CAPTCHA. We developed and pre-trained a Convolutional Neural Network and tested it on unseen test data from the STL-10 dataset. With an accuracy of 70%, we successfully implemented a model, capable of deciphering image-based CAPTCHAs and integrated it into a simple Graphical User Interface for passing CAPTCHA challenges. Based on our research, we derived certain features that could ensure the security of image recognition CAPTCHAs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

”CAPTCHAs are not only annoying, they also kill 10 seconds of your time” [33]. Annoying or not, the Completely Automated Turing test to tell Computers and Humans Apart is an essential challenge-response test, which protects websites against, for example, abuse in online polls, free email services and also prevent spam, worms and dictionary attacks [2]. The challenges presented to the user to verify if he / she is a human are based on ”something computers are yet not able to do very well” [33], such as recognising characters, audio or images. Just like the difficulty of factoring large primes ensures the security of the RSA algorithm, CAPTCHAs rely on the hardness of a given Artificial Intelligence Problem. Cryptographers of the popular Automated Turing test, even encourage people to attempt to break CAPTCHAs for the sake of research [2]. A programme that can break a CAPTCHA, can also solve an unsolved Artificial Intelligence Problem. Most recent research shows that even the most difficult text-based CAPTCHAs can be deciphered using Artificial Intelligence technology with an accuracy of 99.8% [14]. Since adding more distorted text or background noise would not only make the CAPTCHA harder to solve for bots[1] but also for humans. Due to criticism regarding the lack of accessibility [31], for visually impaired people for example, in the long run the technology needs to go beyond these types of CAPTCHAs.

Over a year ago, Google introduced a new risk-analysis technology which is said to encounter bots on the Web and simultaneously be easier for humans to solve [30]. The new Google CAPTCHAs integrate image-based CAPTCHAs based on the hard AI problem of image labelling. Image labelling is still said to be an unsolved AI problem, since most humans can easily identify a cat in a picture, whereas computers cannot. However, the the most recent advancements in image classification advise us that we’re close [29].

The purpose of this project is to review research and identify machine learning techniques

[1]The term bots is often used in literature referring to software applications which run tasks automatically on the Internet

and algorithms used for cracking[2] CAPTCHAs. Since text-based CAPTCHAs are considered vulnerable [14], we will turn our focus on breaking image-based CAPTCHAs. This project can be seen as a 'Proof Of Concept', aiming to answer the research question: "Can we develop a programme that can solve the hard Artifical Intelligence problem of image labelling?"

1.1 Roadmap

The second chapter gives a short *BACKGROUND INFORMATION* on CAPTCHAs, focusing on text-based and image-based CAPTCHAs. It also introduces the new risk analysis protocol by Google called NO CAPTCHA RECAPTCHA.

The *LITERATURE REVIEW* in Chapter 3 reviews relevant literature and research conducted in the area of breaking CAPTCHAs. Divided into five different sections, it covers (1) research on breaking text-based CAPTCHAs, (2) research on breaking image-based CAPTCHAs, (3) methods for evaluating the effectiveness of a CAPTCHA machine learning techniques used for breaking CAPTCHAs, (4) research on developments in image recognition and (5) research on machine learning and classifiers used in image recognition.

Given the gap found in the current literature, Chapter 4 *REQUIREMENTS AND ANALYSIS* presents the aims of the project, breaks them down into feasible requirements and analyses them individually. Requirements needed to accomplish the project aims, as well as restrictions of the project and evaluation methods are specified.
Chapter 5 *SYSTEM DESIGN* discusses and justifies approaches used for the system implemented. The main focus is on explaining mathematical properties of algorithms used, as well as their advantages and disadvantages.

The following is an explanation of how these approaches mentioned in the previous chapter are implemented. Furthermore, Chapter 6 *IMPLEMENTATION and FUNCTIONAL TESTING* describes how the system was tested on functionality.

The core part of this project is Chapter 7 *EXPERIMENT and RESULTS*. It explains how the system is tested during experiments and reveals the results which are discussed and interpreted.

The final chapter 8 *DISCUSSION and CONCLUSION* provides a summary of the main achievements to date and conclusions drawn from the analysis of the problem. The vital section of this chapter includes recommendations to make image recognition CAPTCHAs more secure.

[2]We shall use the terms 'cracking' or 'breaking CAPTCHAs' referring to solving the given AI problem by an automated programme rather than using attacks on the protocol such as launching a man-in-the-middle attack or other attacks to exploit vulnerabilities

Chapter 2

Background

2.1 CAPTCHAs

Completely Automated Public Turing test to tell Computers and Humans Apart is a type of challenge-response test to determine whether a given user is a human or not. A CAPTCHA is a "cryptographic protocol whose underlying hardness assumption is based on an AI problem" [2, p. 296]. The AI problem presented should be easy to solve for humans but hard for computers. The paper suggests that any program that has high success over a CAPTCHA can be used to solve an unsolved Artificial Intelligence problem.

There are various different types of CAPTCHAs, all of which have the same purpose but differ in the way they present the challenge to the user. This can be text-based, image-based, audio-based, video-based or puzzle-based (e.g. solving a mathematical problem). For this report, we will only cover text-based and image-based CAPTCHAS.

In their paper, Luis van Ahn et al. describe three basic requirements of a CAPTCHA [2]:

- It must be easy for humans to pass
- It must be easy to generate automatically
- It must be hard for a computer program to pass

The paper further emphasises that CAPTCHAs don't need to be 100% effective at rejecting malicious, automatised programmes, as long as the increase they work factor, i.e. the cost of breaking the CAPTCHA should be higher than the gain of breaking it.

2.1.1 Text-based CAPTCHAs

The most commonly used type of CAPTCHAs on the Internet are text-based CAPTCHAs, which typically consists of an image containing a sequence of distorted and rendered letters and/or numbers [31]. More difficult text-based CAPTCHAs also contain background noise. The user is asked to type the characters into a textfield to pass the challenge. Theoretically, this can be done easily by a human and not by a computer. Since challenges as short as five characters (including uppercase, lowercase letters and digits) are robust against random guessing (i.e. 62^5 is roughly 912 million possible solutions), it provides a secure system against spammers [31, p. 222].

Figure 2.1: Example of a text-based CAPTCHA.

Several methods e.g. Optical Character Recognition or segmentation techniques have become more and more successful at cracking these type of CAPTCHAs. To encounter those methods, more noise and distortion has been added to CAPTCHAs. Critique has been put forward, as text-based CAPTCHAs adversely affect visually impaired or people with disabilities such as dyslexia [31].

2.1.2 Image-based CAPTCHAs

These types of CAPTCHAs represent an image recognition task to the user. Since text-based CAPTCHAs have been criticised due to their lack of accessibility and they are impractical to solve on a small mobile device, image-based CAPTCHAs seem to be a more mobile-friendly alternative [30]. The effectiveness of this CAPTCHA is based on a computer vision problem of image labelling. The user is presented six different photos and a question, such as ”Which of these photos includes a cat?”. To pass the challenge, the user has to tap all suitable photos e.g. all those including a cat.

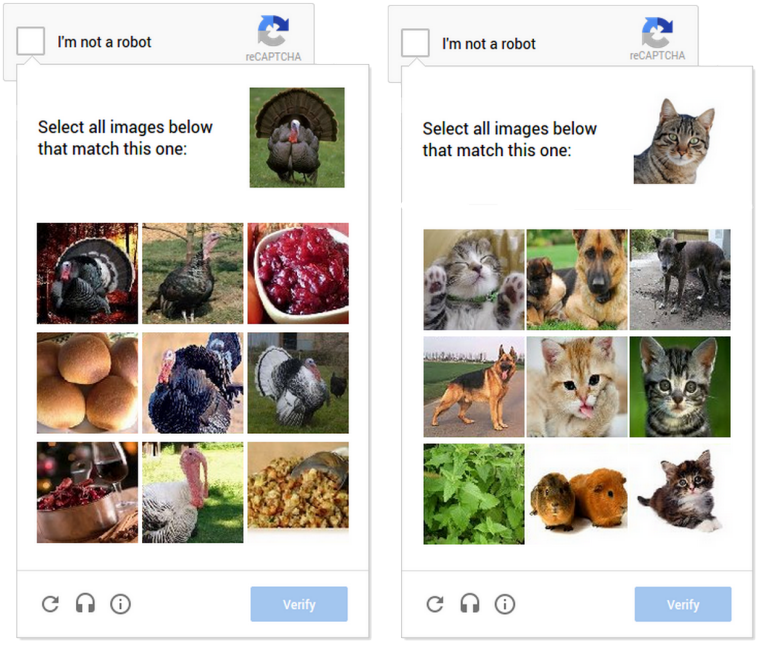


Figure 2.2: Example of an image-based CAPTCHA.

2.2 NO CAPTCHA reCAPTCHA

Research conducted by Google showed that today's Artificial Intelligence technology is able to decipher 99.8% of text-based CAPTCHAs [14]. This time last year, Google therefore announced a brand new advanced risk analysis tool which is a protocol tracking the user's behaviour before, while and just after solving the CAPTCHA [30]..

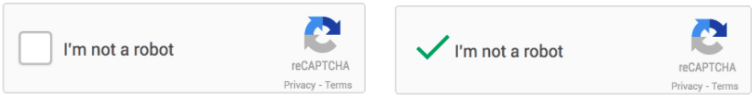


Figure 2.3: NO CAPTCHA reCAPTCHA: The user is asked to tick a box to verify that he/she is human.

The parameters taken into account are kept secret. If the analysis cannot give a confident prediction, the user will still be presented either a text-based or image-based CAPTCHA. On testing the analysis tool on Wordpress myself, I discovered that one of the parameters must be whether a user is logged into the given website or not (the fact that a user is logged in results in the fact that the user is human), however, only assumptions can be made.

Chapter 3

Literature Review

The aim of this section is to review relevant research conducted in the area of breaking text- and image-based CAPTCHAs, in order to find gaps in the literature. The literature review is divided into five sections: (1) Breaking text-based CAPTCHAs, (2) Breaking image-based CAPTCHAs, (3) Evaluation Methods for CAPTCHAs, (4) Solving Image Recognition Problems and (5) Machine Learning Techniques and Classifiers. The first three sections discuss specific topics and research areas on CAPTCHAs, whilst the fourth section aims at giving an overview of the most recent developments in Image Recognition and the last section summarises Machine Learning techniques, which we came across on reviewing the literature and evaluates them critically.

3.1 Breaking text-based CAPTCHAs

Chandavale et al [8] suggest an algorithm to break text-based CAPTCHAs which consists of four different phases: (1) preprocessing, (2) segmentation, (3) feature extraction and (4) character recognition. The paper concludes that feature extraction was more successful than other pattern matching techniques. However, it is necessary to point out that CAPTCHAs tested in this paper did not contain **negative kerning**, i.e. overlapping characters and were therefore easy to segment.

Bursztein et al provide the first systematic study on visual CAPTCHAs and suggest various recommendations for attackers and CAPTCHA designers[7]. The paper extends the 3 stage process, which was commonly used in the research area, and discovered five generic steps which can be used to decipher text-based CAPTCHAs automatically namely (1) pre-processing, (2) segmentation, (3) post-segmentation, (4) recognition and (5) post-processing. The end-product was a piece of software, called DeCAPTCHA, which includes a framework for image manipulation, machine learning algorithms such as Support Vector Machines and K-nearest neighbour classifiers (see Section 3.3.2) used to break visual CAPTCHAs. The success rate fluctuates substantially depending on the difficulty of the CAPTCHA. The programme failed to break Google's ReCAPTCHA

CAPTCHAs. This is due to the fact that this type of CAPTCHAs relies more on lines and collapsing rather than background noise. This shows that again, the problem of segmentation could not be solved.

"As of 2013, negative kerning is considered the most secure method for preventing segmentation, because it has successfully withstood years of attacks". This is a quote from a more recent paper by Bursztein et al which provides a novel approach to improve quality of character segmentation [6]. The paper criticises the standard approach, mentioned above, due to its mostly hand-crafted techniques to overcome distorted text. The approach presented in the paper breaks CAPTCHAs using machine learning algorithms including k-nearest neighbours algorithm (KNN) to handle segmentation and recognition simultaneously. The main difference in this paper is that instead of supervised learning, i.e. training the model based on labelled input, it uses a reinforcement learning approach in which a human indicates when labels are misclassified, labels them correctly and therefore teaches the model from the feedback given. The evaluation shows good results and the paper suggests that the future of CAPTCHAs will have to move to a more complex domain, e.g. image recognition CAPTCHAs.

Ben Boyter [5] published code samples in Python to decode CAPTCHAs for promotion of his book on the same topic. He uses three main steps: (1) color detection to separate letters from background, (2) a recognition algorithm to get coordinates of where the letters start and end, (3) vector space search engine to assess the likelihood of each of the letters cut out from the image compared to training set.

A break-through research paper by Google [14] has found a way to identify house numbers from a Street View image based on a deep convolutional neural network. To train a probabilistic model given a set of images, the following approach is used: Let S be the output sequence and X be the input image, therefore the goal is to learn a model of $P(S \mid X)$ by maximising the log of $P(S \mid X)$ on the training set using stochastic gradient descent, which is an efficient approximate optimisation method involving separating each gradient according to each separate observation [27].

Based on the results from the Street View dataset, the neural network was modified and tested on an internal CAPTCHA dataset. It is clear to see that the main advantage is Google's large internal dataset, which leads to the fact that overfitting is not an issue and further increases both training and testing results of the neural network. Eventually, the model managed to generate a 99.8% success rate even when tested on the most difficult reCAPTCHAs. The results show that "the utility of distorted text as a reverse turing test by itself is significantly diminished" [14, p.8]

3.2 Breaking image-based CAPTCHAs

Chew and Tygar suggest two steps for building a model that can recognise unseen images [9]: (1) building a database of labelled images from Google Images using a handcrafted dictionary, (2) search the database for the images presented during the image recognition CAPTCHA challenge. The downside of this approach, however, is that a large amount of pre-labelled images is necessary. What's more, in order for this approach to be efficient, several databases would be required, since image recognition CAPTCHAs may provide different types of challenges (e.g. labelling types of food, animals, etc.), which would lead to considerable memory requirements. Chew and Tygar further discuss issues faced by humans when solving image-based CAPTCHAs, such as accessibility and usability issues and point out recommendations for CAPTCHA designers, in order to make CAPTCHAs more secure against such a dictionary-type attack. Among those recommendations are pieces of advice such as increase the number of challenges, e.g. by increasing the number of images per keyword, use a dynamic database for generating CAPTCHA challenges, degrade the quality of the images presented during the challenge, such that the attacker cannot find an exact match easily. It is necessary to point out, however, that the latter recommendation could also lead to a negative effect on human performance.

In his paper, Golle presents a machine learning attack which successfully breaks a specific image-based CAPTCHA called ASIRRA (Animal Species Image Recognition for Restricting Access) CAPTCHA [13]. The security of ASSIRA CAPTCHAs is based on the presumption that distinguishing and classifying images containing cats and dogs automatically is difficult for computers. The paper presents very good results with an accuracy of 82.74%. The model is based on Support Vector Machine classifiers which are trained to extract colour and texture features of images. The author stresses the fact that the system only requires pre-labelled training images (i.e. supervised learning) to solve the CAPTCHA challenge automatically. The paper further analyses various algorithms used when generating captcha challenges such as the partial credit algorithm (PCA), which grants access to users who nearly passed the challenge, or the token bucket scheme (TB), which punishes users who fail a lot of ASIRRA challenges in a row. PCA gives attackers an considerable advantage and improves success probability, whilst the TB algorithm has a negative effect on the usability. Golle suggests IP monitoring schemes, in order to prevent against Machine Learning attacks and advises against degrading the quality of the images, since it would decrease the usability and therefore neglect its original purpose of being more accessible than text-based CAPTCHAs. One point to criticise in this paper is that the author neglects the fact that the effectiveness of the attack can easily diminished by adding several different types of animals in the CAPTCHA challenge. Furthermore, there is a lack of explanation how the model was trained and tested.

Zhu et al examine all existing image recognition CAPTCHAs (IRCs) and evaluate their security individually [36]. Specifically they focus on three IRCs, namely, IMAGINATION, ARTiFACIAL

and Asirra CAPTCHAs, of which the latter is most commonly used amongst the web. The paper identified that CAPTCHAs using binary classification, such as Asirra CAPTCHAs, (e.g. Does the image in question contain a dog or a cat?) can often be broken by just computing low-level features, such as colour, shape, texture of the object. Machine Learning techniques, such as Support Vector Machine classifiers, can be used to identify typical low-level features on empirical training data, which makes Asirra CAPTCHAS more vulnerable. To make an IR CAPTCHA more secure, the CAPTCHA challenges presented should be independent from past challenges, as Golle [13] also pointed out in his paper. Based on the analysis conducted, the authors developed a novel CAPTCHA called Cortcha, which fully automises image source collection and CAPTCHA challenge generation.

Lorenzi et al [23] evaluate three types of image-based CAPTCHAs and develop a methodology for breaking each of them individually. The authors propose the following steps to crack an image recognition CAPTCHA, such as the ASIRRA CAPTCHAs: (1) Extract images from the challenge presented, (2) build and train an Hierarchical Temporal Memory (HTM), (3) pass extracted images through the HTM network, (4) use the resulting probabilities to select the solution with the highest probability and (5) use remaining probabilities to assess the remaining images.

3.3 Evaluation of the effectiveness of a CAPTCHA

3.3.1 Success Probability

Using the binomial distribution, we can derive the probability $P(y)$ of observing a certain number of successes in N rounds, for example, counting the of heads given N coin tosses [27, p.54]:

$$P(Y = y) = P(y) = \binom{N}{y} q^y (1-q)^{N-y} \tag{3.1}$$

In his paper, Golle defines the success probability of classifying a single image (where $0 < p < 1$)[13]. Given a 12-image ASIRRA CAPTCHA challenge, a classifier with an accuracy as high as in Golle's paper succeeds with probability p^{12}, which gives a success probability of 10.3%, providing that the challenges are independent from each other.

3.3.2 CAPTCHA Efficacy Metric

Chew and Tygar derived a CAPTCHA efficacy metric G indicating the probability that in the time it takes a human to solve a CAPTCHA, the human will pass and a computer will not [9].

Let p = probability that a human user will pass a round
q = probability that a computer will pass a round
n = number of times faster a computer is than a human
m = number of rounds
k = threshold number of rounds

$$G = \sum_{i=k}^{m} \binom{m}{i} p^i (1-p)^{m-i} * [1 - \sum_{i=k}^{m} \binom{m}{i} q^i (1-q)^{m-i}]^n \tag{3.2}$$

The higher the efficacy metric G, the harder the CAPTCHA challenge presented.

Note that the first part of the equation is in fact the Binomial Distribution mentioned above, in this case, the probability that a human will pass the challenge, whilst the second part is the negative probability indicating the probability that a computer will not pass the challenge successfuly. Zhu et al propose that "if on average an attack produces a response within 30s with a success rate of 0.6% or higher, the attack is claimed to be effective; otherwise ineffective" [36, p.188], providing that the CAPTCHA system implements a token bucket scheme or similar IP monitoring methods.

3.4 Solving Image Recognition Problems

3.4.1 Most recent Developments

ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2014) is the largest challenge in computer vision held annually to test state-of-the-art-technology in the field [29] and provides a benchmark on object category classification and detection on hundreds of object categories and million of images. ILSVRC consists of 3 competitive tracks: 1) image classification, 2) single-object localisation and 3) object detection. In 2014, the winning team in classification, GoogLeNet, developed a 22-layer deep convolutional neural network which managed to achieve an error rate of only 6.5% [32]. In order to increase both depth and width of the network, the team added additional dimension reduction layers without causing significant computational overhead. This resulted in an overall 4.2x reduction in image classification error between ILSVRC 2010 and ILSVRC 2014 [29].

Vinyals et al have developed an end-to-end system called NIC (Neural Image Caption) to automatically generate natural sentences describing an image [35]. The system is based on a convolutional neural network (CNN) which takes an image as an input and outputs it as a compact representation, followed by a recurrent neural network (RNN) that generates a corresponding sentence. The novelty in this paper is that a single joint model was used to generate image captions, whereas previous research was based on stitching together existing

solutions. The paper concluded that one reason for its success was the availability of a huge amount of data available and suggests that the bigger the size of the database, the better the performance [35]. The problem of overfitting was prevented by using techniques such as generalisation, where the weights of the CNN component are initialised according to a pre-trained model and drop out methods, where data is held out in order to validate the model [27].

Simulteanously, Karpathy and Fei-fei presented a model which generates natural language descriptions of images in their regions (and reason about the given contents) [16]. The models combines a (1) convolutional neural network (CNN) over image regions, (2) a bidirectional recurrent neural network (RNN) over sentences and (3) structured objective aligning both modalities. The paper stresses and promotes the emergence of CNNs for image classification and object detection. During training, the model is fed with set of images and corresponding sentence descriptions. The neural network maps words and image regions to a common, multimodal embedding, such that the objective can learn the embedding representations. As before, in order to prevent overfitting, cross-validation and dropout regularisation is used.

3.4.2 Problems and Difficulties Associated with Image Recognition

Easy and Hard Object Classes in Image Recognition

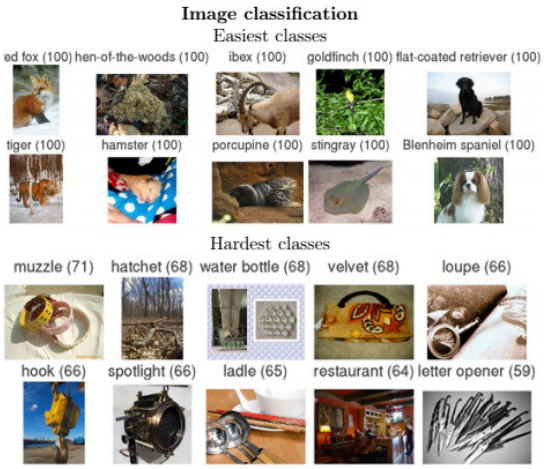


Figure 3.1: Examples of easiest and hardest classes for Image classification.

The analysis of the ILSVRC 2014 competition shows that 121 image classes out of a 1000 gave a 100% accuracy, which suggests that there are classes which are easy and some which are hard to recognise by automated systems [29]. A majority of classes which are easy to recognise in image classification applies to mammals such as "red fox" and animals with distinctive structures such as "stingray". Compared to the hardest classes (with accuracy as low as 59%), which include metallic and see-through-objects, such as "hook" and "water bootle", as well as highly general scenes such as "restaurant" [29].

Russakovsky et al. identified a strong difference in accuracy depending on whether the object is natural (higher accuracy) or

man-made (lower accuracy) [28]. The paper further defined five additional properties inspired by human vision which affected the performance of algorithms in image classification: (1) Objects which are larger in the real-world are easier to detect than smaller objects, (2) deformable objects such water snakes are easier to detect than rigid objects such as mugs, (3) Objects with a distinguishable texture and (5) distinguishable colour are easier to classify.

Other Difficulties associated Image Recognition

In recent a TEDTalk, Li Fei-Fei points out that the best-performing deep neural networks, have become more and more powerful when detecting and classifying objects, however, they still struggle to interpret scenes like human beings do [11]. The world's most enormous trained Convolutional Neural Network may be able to detect a car in an image and be capable of telling its type and year, however it is still not able to associate objects with art and history, interpret emotions in a scene or detect and appreciate the beauty of nature like human beings would do.

A very recent paper reveals that deep neural network can be easily fooled and forced to misclassiy certain images with a 99.9% confidence level [26], which are completely unrecogniseable to the human eye, rather than images of a similar class. Specifically, the researchers use an architecture called "Alexnet", which is a Convolutional Neural Network pre-trained on the IMAGENET database and is publicly available. Their model achieves 0.94% error rate on the MINST dataset, a dataset for handwritten character recognition. The unrecognisable images are produced by directly optimising the posterior probability for existing classes using gradient ascent. Furthermore, the authors show that by modifying regularisation terms slightly, the unrecognisable images can be made easier to recognise.

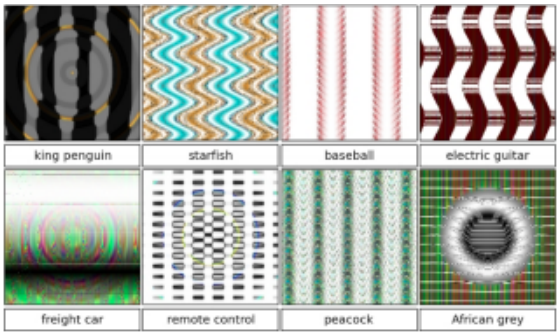


Figure 3.2: Images which fool Convolutional Neural Networks

3.4.3 Evaluation Techniques in Image Recognition

To evaluate the performance of the algorithms submitted for the ILSVRC14 competition, the error of a prediction was calculated [29]. For the task, the algorithm was allowed to identify up to 5 objects in an image without being penalised provided that one of the named objects

corresponds to the ground truth label [1] To be more precise: Each image i is given a class label C_i. An algorithm may return 5 labels $c_i1, ...c_i2$ and is considered correct if $c_ij = C_i$ for some j. Let the error of a prediction $d_ij = d(c_ij, C_i)$ be 1 if $c_ij = / = C_i$ and 0 otherwise. The prediction error mentioned above is therefore the fraction of test images on which the algorithm makes a mistake

$$error = \frac{1}{N} \sum_{i=1}^{N} \min_j d_ij \tag{3.3}$$

Additional measures of error were used, in order to retrieve more detailed information on the algorithms' performance. The commonly used Top-1 Error is defined as follows: If the highest-confidence output label c_i1 did not match ground truth class C_i, penalty was higher. Another metric, the hierarchical error lowers penalty if two nearby classes were confused, such as the breed of a dog.

3.5 Machine Learning techniques and Classifiers

3.5.1 Techniques used for text-based CAPTCHAs

Artificial Neural Networks are inspired by the structure and behaviour of biological networks and represent a non-linear computational model which is mainly applied to solving machine learning and data mining problems of classification, e.g. for handwriting recognition [4]. The model consists of various nodes (nerve cells) connecting to each other. These nodes are distributed among the layers: input, (one or more) hidden and output layer. Mathematically, each node acts according to a transfer function which calculates the inner product of a given input and weight vector to produce a scalar result. A major advantage of a neural network is that it adjusts the weight scalars during the learning process by itself and furthermore, has the ability to generalise the results which is particularly useful for character recognition. Even though Neural Networks seem quite magical, they can lead to over-fitting of the model and the output can be hard to translate into probabilistic results. Another major drawback is that understanding the inner-workings of a Neural Network can be quite challenging.

Vector Space Model, also known as Vector Space Search Engines, is widely used to rank documents according to their relevance given a certain keyword [5]. Document and query similarity can be expressed by looking at the proximity between the corresponding vectors in the spatial domain, i.e. the cosine similarity. Boyter [5] makes use of this model to create an 'image recogniser' aimed to classify data. The main advantages over Neural Networks are mainly that VSM's require less iterations and are less prone to overfitting. What's more,

[1] In machine learning, the term "ground truth" refers to the accuracy of the training set's classification for supervised learning techniques. This is used in statistical models to prove or disprove research hypotheses.

if necessary, they can be edited live. Nevertheless, due to the fact that the model is very calculation intensive, it can be slower and less flexible than Neural Networks.

3.5.2 Techniques used for image-based CAPTCHAs

Hierarchical Temporal Memory networks are a form of neural networks and common in the field of handling image recognition tasks [23]. During training, the HTM network receives an image as input and performs pre-processing, such as noise reduction or changing colours. It then passes the image through multiple levels until it eventually represents the original input into its most abstract way, up until the highest level which represents global properties, such as shape. This is done by converting extracted patterns from the input into 'sparse distributed presentations' [23, p.329], which have desireable properties such as robustness to noise, high capacity and ability to encode multiple interpretations. Lorenzi et al further suggest that the high-level representations learnt may be used to feed them into a supervised learning classifier. This network sounds incredibly promising, however, on conducting more in-depth research, we found that there has been a lot of criticism towards HTM networks since they are said to be more difficult to analyse, as their structure tries to maintain the link between the neurons and network topologies and known structures in the brain. We recommend this blog post for further reading on this topic [15].

Support Vector Machine is linear, supervised learning model (i.e. they require labelled training data) which takes two sets of training vectors (representing positive and negative instances) and creates a hyperplane in order to classify them into one of two separate groups [4]. The main advantage of the SVM is that any local minima located during the training process will give a unique solution, compared to other classifiers where the local minima might not have the lowest error. SVMs are also less prone to overfitting compares to Artifical Neural Networks. Even though SVMs may be extended for multi-class classification, the problem would have to be formulated into several binary classification problems.

3.5.3 Techniques used for Image Classification

Traditionally, pattern recognition was a 2-stage process including a feature extractor and a trainable classifier [20]. The most recent trends, however, seem to change back to fully-connected layers [18], due to simplicity and optimisation of parallel computing. Nevertheless, unsupervised networks are still used for pre-training and results show that pre-training significantly improves results [1].

Unsupervised Learning for Pre-training

Restricted Boltz Machines: represent a two-layer graphical model, compromising a set of binary hidden units and a set of visible units. The layers are connected symmetrically by

a weight matrix W. The model has been widely used for different types of data including unlabeled images, speech, documents or user ratings and more importantly, they represent a learning module for deep blief networks. Given an input, the RBM learns a probability distribution, specifically, it aims to maximise the product of probabilites on the training set. RBMs have the disadvantage that they are difficult to optimise using methods other than stochastic gradient descent, due to the biased nature of the gradient and intractability of the objective function [19].

Sparse Autoencoders: represent a simply 3-layer type of ANNs used for unsupervised learning. Sparse coding algorithms aim at capturing high-level features from unlabeled data through learning basis functions [22]. Applying this algorithm on natural images, it can be found that the 'learnt bases resemble the receptive fields of neurons in the visual cortex'. One attractive property of sparse coding is that it has the capability of learning basis sets where the number of bases exceeds the input dimensions. One of the main drawbacks of sparse autoencoders, however, is that they are computationally expensive.

Supervised Learning for Fine-tuning

Apart from supervised learning models already mentioned, such as SVMs, we came across a few more, which seem to be popular in the field of image classification.

Convolutional Neural Networks are a type of Artifical Neural Networks. They consist of three main layers: (1) The Convolutional Layer, (2) Pooling Layer and (3) Classification Layer. The hierarchical model alternates between the convolutional and subsampling layers similar to cells in the primary visual cortex [34]. Unlike Artificial Neural Networks discussed in Section 3.5.1, Convolutional Neural Networks (CNNs) are more suitable for receiving images as input, since they contain neurons arranged in 3 dimensions (width, height, depth). That means that they can easily accept coloured images consisting of 3 channels (R, G, B). CNNs also have the nice property of having neurons locally connected to the input (receptive fields).

Logistic Regression: is a popular approach to binary classification [27]. The aim is to model the distribution of discrete, dependent variables by estimating probabilities using a logistic function; the assumption is to model the log-odds of the basis function. We define the odds as the ratio of the probability of a positive outcome, to the probability of a negative outcome.

3.6 Gaps in the Literature

Having reviewed relevant literature in the field of general image classification and recognition, as well as specific research papers on breaking text- and image-based CAPTCHAs, there is a perceivable movement away from hand-crafted solutions and towards supervised Machine Learning approaches, as well as attempts to implement unsupervised learning algorithms. The literature review shows that research on text-based CAPTCHAs is fairly exhausted. Goodfellow

[14] conclude that even difficult text-based CAPTCHAs are vulnerable to attacks using machine learning techniques and that other CAPTCHA technologies will have to be used in the future.

The attention is therefore drawn to image-based CAPTCHAs, since they present a difficult computer vision problem which is yet hard to solve by computers. Among the earliest research papers, Chew and Tygar [9] suggest building a database of labelled images and comparing them with the image in question. To date most research on image-based CAPTCHAs has focused on ASIRRA CAPTCHAs. This led to the development of models that can successfully break these types of CAPTCHAs using machine learning techniques [13] [23] [36], mainly using Support Vector Machine classifiers or a Hierarchical Temporal Memory network. In contrast there has been no research conducted on the new Google CAPTCHAs (see Chapter 2), and therefore will be the main focus of the project.

Combining the state-of-the-art advancements in image recognition with well-grounded research on image-based captchas, this project aims to fill the gap and provide a proof of concept exploring whether the combination of both can help developing a system that can be used to crack CAPTCHAs. Additionally, through this experience, we will explore problems and difficulties in image recognition and provide suggestions to improve the security of image-based CAPTCHAs.

Chapter 4

Requirements and Analysis

This chapter explores and analyses the core requirements essential for the development of a system which can break a simple CAPTCHA. We shall highlight the project aims and break them down into scalable and feasible requirements. These are separated into functional requirements, which are tangible requirements affecting the behaviour of our system and non-functional requirements, which are requirements that do not change the inner-workings of the system developed. The requirements stated are analysed individually and restrictions, which limit the scope of the project will be pointed out in this chapter.

4.1 Aims

- A1 To examine machine learning approaches that have been used to classify images
- A2 To implement some of these approaches in an attempt to break a very simple CAPTCHA
- A3 To point out problems and difficulties encountered when attempting to crack an image recognition CAPTCHA
- A4 To identify the features of a 'strong' image recognition CAPTCHA

4.2 Requirements

#	Requirement	Priority
Prerequisites		
0.1	Which machine learning approach is the most appropriates?	High
0.2	How do we define a 'very simple CAPTCHA'?	High
Functional Requirements		
1	Unsupervised Feature Extraction	High
2	Fine-tuning and Training	High
3	Image Classification	High
4	Sample patches for Feature Extraction	Medium
5	Perform Data Pre-processing	Medium
6	Create Training and Test Dataset	Medium
Non-Functional Requirements		
7	Create a Graphical User Interface for Demonstration	Low

Table 4.1: System Requirements.

4.3 Analysis

4.3.1 Prerequesites

Which machine learning approach is the most appropriate for classifying images?

From the literature review, we can conclude that supervised models such as Convolutional Neural Networks and Support Vector Machines are commonly used in image classification. The decision-making process is discussed further in the next chapter.

How do we define a 'very simple' CAPTCHA?

To meet our aim A2, it is necessary to develop evaluation techniques, in order to rank the difficulties of the image recognition CAPTCHAs, which we attempt to break. Chew and Tygar [9] have derived an efficacy metric to indicate the probability that in the time it takes a human to pass a CAPTCHA, a computer will not pass. This can therefore be used to evaluate whether a CAPTCHA is relatively easy to pass.

4.3.2 Functional Requirements

The system can be broken down into 3 core parts, namely, (1) Unsupervised Feature Extraction for pre-training our model, (2) Fine-tuning by training a model on a labeled training set and (3) Using a Classifier to classify images. Given randomly sampled patches, we want to extract features and perform pre-processing in advance, in order to encourage unsupervised learning. The output will be used to initialise the weights and biases when training our model to retrieve the optimal parameters. These will eventually be used to (3) classify unseen images from our

test set.

(1) To extract features, we require sampled patches from our dataset. (2) For improving results, it is necessary to apply pre-processing on the patches, such as PCA Whitening. Finally, we require (3) a training set in order to learn the optimal parameters with our model and a test set including unseen test images, in order to test the performance of the model chosen.

4.3.3 Non-Functional Requirements

For demonstration, CAPTCHA challenges shall be generated and presented in a Graphical User Interface. The CAPTCHA challenge will be solved by the model and presents the results to the user. The interface is considered to be a non-functional requirement, since it aims to present the results of the system and will not affect the behaviour of the system.

4.4 Project Restrictions

This project aims to act as a proof of concept, showing that the vast developments in the area of image recognition could leave this type of CAPTCHAs increasingly vulnerable to machine learning attacks. From the literature review, we saw that state-of-the-art models have achieved high accuracy results by using a large amount of data during training [35]. Since this project is subject to time and memory constraints, we are not striving for perfection.

We will therefore set certain priorities when designing the system, which will be discussed in the following chapter.

4.5 Evaluation techniques

Foremost, we explore whether or not we can develop a model that can classify images correctly and pass a given CAPTCHA challenge. It is further necessary to develop evaluation techniques, in order to rate the performance of our system. In line with the common machine learning evaluation techniques, we can evaluate our performance using the accuracy, also known as 0/1 loss, [27] of the system. Providing that the system achieves reasonable results, we shall evaluate our system by computing the accuracy, as well as the corresponding confusion matrix. The graphical user interface will help evaluating whether the system can pass a given CAPTCHA challenge successfully. Inspired by literature in the field, we shall derive the success probability of our system given a CAPTCHA challenge [13], as well as computing the prediction-error, top-1 error and hierarchical error [29].

Chapter 5

System Design

The previous chapter analysed what was required to satisfy the project aims and broke down the problem into feasible steps. This chapter will highlight and justify the technical approaches used to meet the project requirements. Specifically, we will discuss and analyse the structure and mathematical properties of the system which was necessary for implementation.

5.1 CrackedIT - A system for Cracking Captchas

To satisfy the requirements of the system, we designed an Autoencoder for feature extraction and a Convolutional Neural Network including a Softmax Classifier for Image Classification. As for our design philosophy, we gave priority to two factors: Simplicity and Extendability. The system was designed in a simplistic manner aiming at developing a functioning system which returns 'reasonable' results to meet our project aims. Furthermore the system should be flexible enough, such that it could be extended for further research purposes, such as increasing performance and results by implementing the system on a GPU.

5.1.1 Feature Extraction

From the literature we found that pre-training can be done by using an already pre-trained model on a large dataset which is very often done for research or by developing our own pre-training system. The latter was chosen, for our own learning purposes and in order to prevent alienation of the inner-workings of the system. For the purpose of extracting essential features from unlabeled data, we had considered various options, such as Restricted Boltz Machines (RBMs), Principal Component Analysis (PCA) and Autoencoders. Compared to RBM's, Autoencoders are simpler to implement and optimise when running on a GPU, since algorithms such as L-BFGS or conjugate gradient can be used [19]. PCA provides a fairly simple feature extraction method, however, for the purpose of extendability, we favour the implementation of an Autoencoder. This is because we promote a layer-wise pre-training, as

it can achieve better results on large-scale data. Autoencoders therefore satisfied the two major requirements of our design philosophy: simplicity and extendibility. Another advantage of Autoencoders is that the nature of the model allowed us to explore the way Neural Networks work. There are multiple types of Autoencoders, however, from further reading we identified that Sparse Autoencoders are an efficient way to pre-train a model, since their output can be used for deep neural networks [19]. By imposing a sparseness constraint on the hidden units in the network, we could avoid overfitting on the training data [34]. Having chosen this model, it could easily be made more efficient. An example of this would be adding noise to the input to create a denoising autoencoder [3]. We performed unsupervised feature learning, according to Coates framework [10, p. 217]:

1. Extract random patches from unlabeled training images
2. Apply pre-processing to the patches
3. Learn a feature-mapping using an unsupervised learning algorithm

Input, Data Structures and Pre-Processing

The input can be defined as follows [10, p. 217]: Let $x^{(i)}$ be the i-th patch randomly sampled from our training set T, where $x^{(i)}$ has dimension $d*d$ and c colour channels. We shall refer to d as the 'receptive field size'. Each $x^{(i)}$ is represented as a vector R^N, where $N = d*d*c$. The dataset used for feature extraction is therefore $X = x^{(1)}, ..., x^{(m)}$, where m is the number of patches. Pre-processing was performed on the dataset X, specficially we chose to apply ZCA Whitening on the data [17, p.48], since ZCA whitened patches look similar to the original patches, which is favourable when implemented with Convolutional Neural Networks.

Learn feature mapping using an unsupervised learning algorithm

When training Artificial Neural Networks, backpropagation is commonly used for computing the cost function as well as its derivative with respect to the weights and biases[4, p.241]. The algorithm involves the following steps [4, p.244]:

1. Forward propagate through the network to find activations of hidden and output units
2. Compute the cost function which is to be minimised
3. Generate the errors of the output activations
4. Backpropagate these to generate the errors of the hidden activations
5. Derive the partial derivatives of the cost function

During the first step, we performed a **feedforward propagation to compute the output activations**, meaning the output value of the given layer. The input activations $a^{(1)}$ are equal to the input data

$$z^{(1)} = x = a^{(1)} = x \tag{5.1}$$

The hidden layer $z^{(2)}$ represents the weighted sum of the inputs. Hidden activations are computed by passing the output from the hidden layer into our activation function f,

$$\begin{aligned} z^{(2)} &= W^{(1)}a^{(1)} + b^{(1)} \\ a^{(2)} &= f(z^{(2)}) \end{aligned} \tag{5.2}$$

where W is a weight matrix, b is a bias vector and f is our activation function. Here, we chose f to be the sigmoid function where $f(z) = 1/1 + e^{-z}$. The aim of the activation function is to rescale the output from the hidden layer such that the values are in the range of [0, 1]. Recent literature reveals that rectifier linear units (ReLU) can increase the performance, however, the sigmoid function was chosen, since it had been implemented successfully for years [19]. We invite researchers to make use of ReLUs, in order to increase the system's performance.

The output layer computes the weighted sum of the hidden units, its activations are equal to the outputs from the output layer, which represents our hypothesis $h_{W,b}(x)$:

$$\begin{aligned} z^3 &= W^{(2)}a^{(1)} + b^{(2)} \\ a^3 &= z^{(3)} = h_{W,b}(x) \end{aligned} \tag{5.3}$$

Note that we did not compute the activations using the sigmoid function, but used a linear activation function instead, since it had proven to yield better generalisation on image data [12, p.315] along with a quadratic cost. Given a set of m patches, we computed the cost function $J(W, b)$ which we aimed to minimise [24, p.1740]:

$$J(W,b) = [\frac{1}{m}\sum_{i=1}^{m}(\frac{1}{2}||h_{w,b}(x^{(i)} - \hat{x}^{(i)})] + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{ji}^{(l)})^2 + \beta\sum_{j=1}^{s_2}KL(\rho||\hat{\rho}_j) \tag{5.4}$$

The cost function consists of three terms, whereas the first term computes the mean-square error, the second term introduces a weight decay to prevent overfitting and the third term forces sparsity on our network[24]: The mean-square error included in the first term aims to minimise the difference between the output and the input. By introducing a weight decay term, we can keep the magnitude of our weights low, in order to prevent overfitting. The influence of this term on the cost is controlled by the weight decay parameter λ. L indicates the number of layers in the network, which is 3 in our case, since out network consists of an input, hidden and one output layer. The number of units for each layer l, is given by s_l. $W_{ji}^{(l)}$ denotes the weight between the i-th unit of layer l and the j-th unit of layer $l + 1$, whereas b_i^l is the bias associated with unit i in layer $l + 1$. Note that the aim of the weight decay term is to keep the magnitude of the weights low and is therefore only applied to the weights and not to the biases. The last term is a sparsity penalty term and has a very important role. Our network learns the function h_W, b such that the output $\hat{x}$ is similar to the input x. By placing a constraint on the network, we can force it to learn a compressed representation and only the most important features. The sparsity penalty term includes three parameters:

β controls relative influence of this term on the cost function, ρ is a sparsity parameter and $\hat{\rho}_j$ giving the average activation of hidden unit j. $KL(.)$ stands for Kullback-Leibler divergence (see equation 5.5). KL divergence measures the difference between two functions and was chosen due to its appealing property that $KL(\rho||\hat{\rho}_j) = 0$ providing that $\rho = \hat{\rho}_j$ and increases constantly as ρ moves away from $\hat{\rho}_j$. This worked in our favour, due to the fact that when minimising the sparsity penality term, $\hat{\rho}_j$ will be close to ρ, meaning the average activation of each hidden unit j will be close to 0. By using KL divergence, we were able to guarantee that the cost function stated above penalises $\hat{\rho}_j$ if it is too far away from ρ and we can therefore enforce a sparsity constraint on our network

$$KL(\rho||\hat{\rho}_j) = \rho log\frac{\rho}{\hat{\rho}_j} + (1-\rho)log\frac{(1-\rho)}{(1-\hat{\rho}_j)} \tag{5.5}$$

To satisfy step 3 and 4, we **generated the errors from the computed output activations**. The error terms, sometimes referred to as δ -terms, give a measure of how much of the error can be traced back to the given unit in the network [4, p.242].

The error terms from the output layer are given by:

$$\delta_i^{(3)} = -(y_i - a_i^{(3)}) \tag{5.6}$$

The error terms for each node i in the hidden layer $(l = 2)$ are given by:

$$\delta_i^{(2)} = \sum_{j=1}^{s_3} W_{ji}^{(2)} \delta_j^{(3)} f'(z_i^{(2)}) \tag{5.7}$$

The key step of the backpropagation algorithm is to **compute the partial derivative of the cost function with respect to the parameters**, in order to update the weights and the biases. It turns out that the derivative of the cost function can also be computed using the errors we derived in 5.6 and 5.7, which is useful for implementation.

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = [\frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)})] + \lambda W_{ij}^{(l)} \tag{5.8}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W,b;x^{(i)},y^{(i)}) \tag{5.9}$$

Optimisation: Since the cost function cannot be solved analytically, we had to resort to an iterative optimisation algorithm for identifying the minimum. Stochastic gradient descent is one of the most popular optimisations algorithms in machine learning, since it is easy to implement and works fast on a large dataset [19]. Nevertheless, the algorithm still comes

with a few disadvantages. Firstly, stochastic gradient descent requires manual tuning of the parameters, such as the learning rate and secondly, it works sequentially and would therefore be difficult to implement on a GPU. Among other optimisation algorithms, we also considered Limited-Memory-BFGS (L-BFGS) and Conjugate Gradient (CG). Whilst experimental results show that CG outperforms other algorithms on high dimensional problems[19], for extracting low-level features such as colour and edges, L-BFGS represents a more competitive option. Due to time constraints, it was not possible for us to validate our model using different variations and therefore rely on the results from Le et al [19]. Further research into the performance is welcome and discussed in the last chapter.

Output

The Autoencoder was designed to yield a function f that transforms an input patch $x \in R^N$ to a new representation $y = f(x) \in R^K$ [10, p. 218].

5.1.2 Image Classification

Having reviewed the current literature and undertaken our own research, we came to the conclusion that Convolutional Neural Networks (CNN) provide state-of-the art results in image classification. The network consists of three main building blocks, namely, convolutional layer, pooling layer and fully-connected (classification) layer. Being a type of Neural Network, we trained our CNN using by performing forward propagation (in the convolutional and pooling layer) and backpropagation (in the Classification Layer) [21]. Since the algorithms have been mentioned above, we will keep our focus on the layer-wise fashion of the CNN.

Convolution Layer

Convolution in the mathematical sense, essentially means performing dot products between the filters and the local region of the input to measure their similarity [10, p. 218]. Having learnt a transformation function f, we can now attempt to compute a representation $y \in R^K$ given any $w * w$ patch. This means that we can represent the original $n * n * c$ image by applying our function f to many sub-patches. Specifically, for every $n * n$ image, we compute y for every $w * w$ sub-patch of the input image.

Pooling Layer

Pooling is necessary, in order to reduce dimensionality of our image representation and improve robustness to distortions [10, p. 218]. There are various types of pooling, for simplicity we chose average pooling using four non-overlapping windows in our feature maps, as suggested by Coates et al [10]. This is accomplished by placing several windows of a certain dimension

(pool dimension) over our convolved features. From each window, we took the average value in order to subsample each region in our feature map.

Classification Layer

We shall use Multiclass Logistic Regression (or also known as Softmax regression) to classify images in our last layer. Softmax regression is a generalisation of logistic regression and caters for multiple classes, which is a required property of our system [4, p.209]. Since our labels are mutually exclusive, Softmax Regression seems to be a suitable classifier. The goal is to estimate the probability that $p(y = j|x)$ given an input x for each class $j = 1, \ldots, k$. Essentially, we are trying to retrieve the probabilities of the input x belonging to one of the k class labels.

To learn the optimum parameters, we minimised the following cost function on the convolved and pooled features from the training set [24]:

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^{(i)} = j\} log e^{\frac{\theta_j^T x^{(i)}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}]}}} + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2 \tag{5.10}$$

Whilst the first term computes the cost of our function, the second term represents a weight decay term, which penalises large values of the parameters. This is important in order to avoid numerical problems caused by the overparameterised nature of softmax regression[24]. The weight decay also ensures that the minimum computed is guaranteed to be a unique solution, rather than a local minimum. Since we cannot solve for the minimum of $J(\theta)$ analytically, we also need to derive the associated gradients, in order to apply an optimisation algorithm:

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[x^{(i)}(1\{y^{(i)} = j\} - p(y^{(i)} = j|x^{(i)}; \theta))] + \lambda\theta_j \tag{5.11}$$

In the last step, the trained model is used to classify the pooled and convolved features to the given class labels [24, p.1741]:

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)}; \theta) \\ p(y^{(i)} = 2|x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)}; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix} \tag{5.12}$$

where $\theta \in R^{k*(d+1)}$ represents the parameters of our model.

5.1.3 Cracking CAPTCHAs:

The design of the Graphical User Interface was inspired by Lorenzi et al's research on image-based CAPTCHAs [23], with the difference that we used a Convolutional Neural Network, instead of a Hierarchical Temporal Memory. As suggested we extracted images from the CAPTCHA challenge, passed these through a trained model and selected the image with the highest probability.

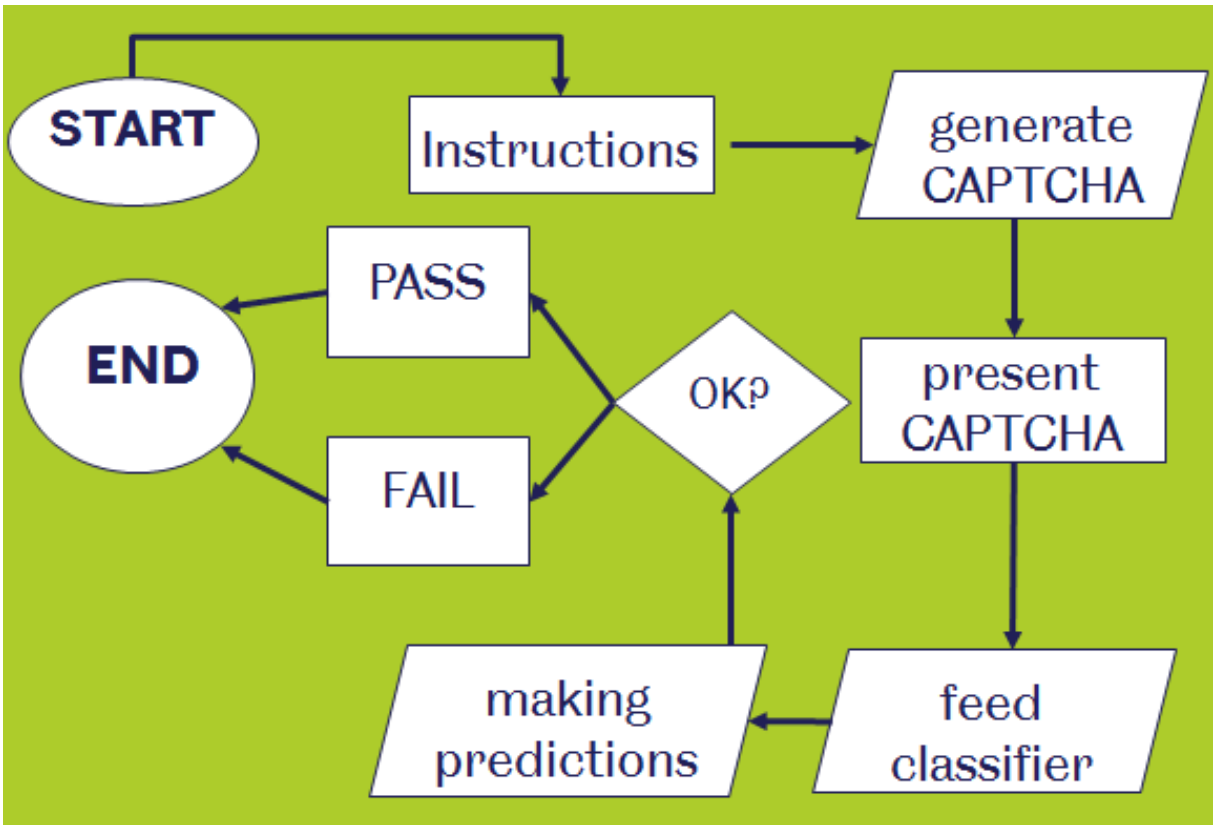


Figure 5.1: Flow Chart.

Chapter 6

Implementation and Functional Testing

Whilst the previous chapter, discussed and justified approaches used in this project, chapter 6 will explain how these approaches have been implemented into our system. The first section explains the choice of programming language. The data set chosen will be presented and specifically, we shall point out which classes had been used. The third section describes the implemented system in more detail focusing on data structures and parameters and step-by-step implementation of algorithms mentioned in the last chapter. The last section described how the system was tested on its functionality.

6.1 Programming Language

Even though, the system could have been implemented in various other programming languages, we chose to implement the whole system in Python. Python is an interpreted, interactive and pseudo-object oriented scripting language. It is especially renowned for its focus on code readability by using a simple syntax design. An Apple Macbook (Mac) has been used throughout this project. Since Python interpreters are pre-installed on Macs, it was a more attractive alternative to MATLAB. We were already familiar with programming Machine Learning algorithms in Python and were used to performing mathematical tasks using libraries such as 'numpy', 'matplotlib' or 'scipy'. Choosing Python as our programming language, also meant that we could use one programming language to implement various parts of systems, as well as developing a Graphical User Interface. This would have been impossible to do in MATLAB. Java and C++ would have provided a more structured design approach, however, due to the ease of mathematical processing and coding, Python was considered to be the more suitable choice.

6.2 Dataset

6.2.1 Overview

Inspired by the CIFAR-10 dataset, the STL-10 dataset is openly available for researchers in the field of image recognition, specialising in unsupervised feature learning and deep learning algorithms. The dataset is recommended for use in unsupervised learning, as it includes more unlabelled training examples than in CIFAR-10.

- 10 classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck.
- Images are 96x96 pixels, color.
- 500 training images (10 pre-defined folds), 800 test images per class.
- 100000 unlabeled images for unsupervised learning. These examples are extracted from a similar but broader distribution of images. For instance, it contains other types of animals (bears, rabbits, etc.) and vehicles (trains, buses, etc.) in addition to the ones in the labeled set.
- Images were acquired from labeled examples on ImageNet.

6.2.2 Data Selection

In courtesy of Standford University Machine Learning Team, we were allowed to use the dataset available online [25]. For pre-training we use 100,000 sampled patches from random unlabelled images from the original dataset.

As for fine-tuning our model, we only selected 4 out of the 10 available classes from the entire dataset, in order to make the project computationally feasible. The classes chosen were airplanes, cars, cats and dogs. Each class consists of 500 training images and 800 test images, resulting in a training set of 2,000 images and a test set of 3,200 images.

Figure 6.1: Left: Patches used for Pre-Training, Right: Categories in the training and test set.

6.3 Implemented System

6.3.1 Pre-Training with a Sparse Linear Autoencoder

Input, Data Structures and Pre-Processing

As discussed in the last chapter, we used patches extracted randomly from the original $96*96$ images $t^{(i)}$ in our training set $T = t^{(i)}, ..., t^{(n)}$, where $n = 2,000$. Each patch $x^{(i)}$ has dimension $8 * 8$, and is represented in a vector $P^{(M)}$, where $M = 8 * 8 * 3 = 192$ to generate a dataset of m randomly sampled patches: $X = x^{(i)}, ..., x^{(m)}$, where $m = 100,000$.

Initially, pre-processing is applied to the patches. Specifically, we first rescale the pixel values from [0, 255] to [0, 1] by simply dividing the image data matrix by 255. Secondly, since 'natural images' are stationary, meaning that the statistics in each dimension is equally distributed, we want to normalise the image values. Therefore, we center the data by subtracting the mean from each patch. Having normalised our data, the third step of pre-processing is Zero Component Analysis, also refered to as ZCA whitening. ZCA whitening is widely used with the purpose of improving algorithms and enable them to learn representative features. The motivation comes from forcing the algorithm to learn higher-order correlations (by removing 1st and 2nd order correlations) rather than focusing on nearby pixel values. To whiten our data, we take the following steps[17, p.48]:

1. Compute the covariance matrix, which is given by $\Sigma = \frac{1}{n-1} X X^T$ (providing that the data is centered). Note that if data points in x are correlated, Σ will not be diagonal.
2. We therefore want to decorrelate our data by diagonalising the covariance matrix and creating a decorrelation matrix D. Using the numpy function 'np.linalg.svd', we can easily compute the eigenvalues and eigenvectors of the covariance matrix. Note that we use 'np.linalg.svd' instead of 'np.linalg.eigh', since it is more numerically stable.
3. To compute the whitening matrix Z we compute $(U \cdot D^{-1/2}) \cdot U^T$, where U represents the associated eigenvectors

Feature Extraction

We initialised the weights from a uniform distribution within an interval of $[-4*\sqrt{\frac{6}{(fan-in+fan-out}} + 4*\sqrt{\frac{6}{(fan-in+fan-out}}]$ and all biases at zero [3], for the purpose of symmetry breaking, meaning that we force our network to not just At training time, the autoencoder learnt the hypothesis function $h_W, b(x) = x$, given an input $x^{(1)}$, where $x^{(i)}$ gives the pixel intensity values from the i-th patch. Since each patch has a dimension of $8x8$, we received 64 input units per patch, i.e. $n = 64,000$. Recall that the input units is equal to the number of output units. For simplicity, the Sparse Autoencoder only comprimises 1 hidden layer, including 400 hidden units.

We trained our model with a weight decay of $\lambda = 0.003$, a sparsity rate (average activation of hidden units) $\rho = 0.035$ and a sparsity penalty of $\beta = 5$.

Using the python library scipy, we were able to easily implement the L-BFGS algorithm to optimse our cost function. The scipy function 'scipy.optimize.minimize' is parameterised by

1. Squared reconstruction error
2. Parameter vector
3. Chosen method, i.e. LBFGS
4. A boolean option to use the gradient of objective function. We set this to true, since we computed both the costs and gradients in our objective function, otherwise gradient would be estimated numerically
5. Maximum number of iterations, set to 400

6.3.2 Fine-Tuning with a Convolutional Neural Network

Layer 0: Image Processing Layer

The resulting optimum parameters learnt from our Linear Sparse Autoencoder, are fed into the input layer of the network. Specifically, we extracted the weights, biases, the mean of the patched data as well as the whitening matrix. For preprocessing purposes and in order to improve our results, we compute the weights and biases as followed: $W_{preprocessed} = W \cdot Z$ where Z is the whitening matrix and $b_{preprocessed} = b - W \cdot \bar{x}$.

Layer 1: Convolution Layer

The convolution layer was implemented in the form of matrix multiplications, using Tutorials from Stanford University [25] and LeCun et al as a guidance [21]:

1. Parameters required are passed into the convolution layer: (1) The number of features were set equal to the number of hidden units in our system, which we set to 400. (2) We used a $8 * 8$ sized filter ('kernel') for convolution and 0 padding. We further passed on the (3) weights and (4) bias from the input to the hidden layer as well as (5) the training images.
2. Using three nested for-loops, we iterated over every training image $t(i)$, every feature $f(j)$ and every image channel (R,G,B) $c(k)$, in order to extract the weights from feature $f(j)$ resulting in a feature matrix, also known as "kernel". To convolve the image with the feature correctly, we had to flip the matrix such that its rows and columns are reversed.
3. Then we convolved each feature with the given subregion of the image, using the scipy function 'scipy.signal.convolve2d',

4. Added the bias to the convolved image,
5. And passed it through our activation (sigmoid) function.
6. To extract the convolved features, we simply extracted them from the convolved image.

We used 400 $8*8$ kernels learnt by the Sparse Autoencoder and convolved each $64*64$ input image resulting in a $57*57*400$-dimensional feature matrix[1], an output size of $3,249$. As our classifier was designed to receive a vector as input, this would result in a vector of $1,299,600$ features per example.

Layer 2: Subsampling Layer

The paragraph above, shows that it was necessary to decrease the size of feature maps, in order to make our programming computationally feasible and the classifier less prone to overfitting. As discussed in the last chapter, we chose to implement mean subsampling over four non-overlapping regions ($19*19$) for simplicity. Again, we used Lecun's formulation [21] to implement the pooling layer: We implement two nested for-loops to iterate over every training image $t(i)$ and every a feature $f(j)$ and divided each feature map into four sections and output the maximum activation value of each quarter, which reduced the size our feature map to a dimension of $3*3*400$. For computational reasons, we only convolved and pooled 25 features at a time.

Layer 3: Fully-Connected Layer

Having convolved and pooled features from the training set $(x^{(1)}, y^{()}), ..., (x^{(m)}, y^{(m)})$, we fed our classifier with the processed features, corresponding labels, number of classes and weight decay. As mentioned in the last chapter, to make this process computationally feasible, we only used 4 classes of the entire dataset. We initialised the parameter vector randomly and used a weight decay of $1e-4$. The cost and gradients of the softmax cost function as were computed as follows:

1. A ground truth matrix G was computed for every class and every training example.
2. We computed a matrix M including all the weighted data and subtract the maximum value of M each time to avoid numerical overflow.
3. The class probabilities were derived by computing $\frac{e^M}{\sum e^M}$
4. We computed the averaged cost including the weight decay term using equation 5.10
5. A matrix, including all the gradients for all examples and all classes, was computed using equation 5.11

[1] $(64-8+1)*(64-8+1)=57$

As before, we used the 'L-BFGS' algorithm to optimse the squared reconstruction cost function iteratively with a maximum of 400 iterations. Having reached the maximum number of iterations, the output was a trained model including the optimal parameters, input size of the data and the number of classes. In the next chapter, we will further describe how predictions were performed.

6.3.3 Graphical User Interface

The GUI was implemented as suggested in the Flow Chart in Chapter 4. The aesthetics of the GUI is kept very simple. The interface contains 3 frames and for orientation, we printed out the processes in the terminal.

1. Instruction screen gives instructions to the user.
2. CAPTCHA challenge presentation frame retrieves a generated CAPTCHA challenge and presents it to the user. The user is instructed to click the button, in order to allow the model to solve the challenge.
3. The predictions are compared with the true labels and results are evaluated.

Example screenshots of the interface are given in Section 7.3.

6.3.4 Class Diagram

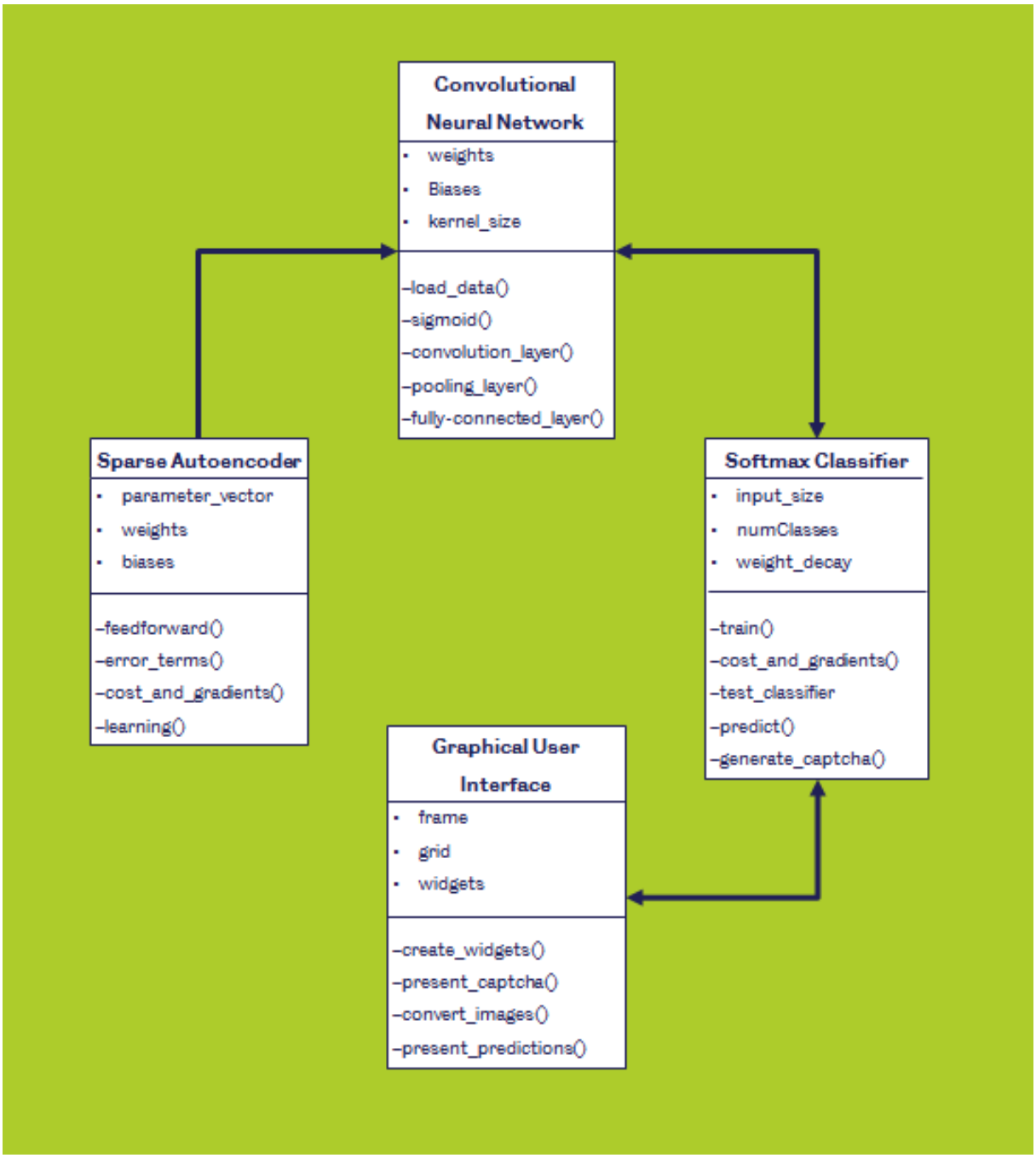


Figure 6.2: Class Diagramme.

6.4 Functional Testing

Along with the programmes developed, we also created test cases as follows. We wrote a test case for every function, testing the correctness of **data structures used**, as well as

the **expected values** of important values. These tests were conducted, since we found that aligning matrices and ensuring the correct dimensions was the greatest challenge during the programming phase. **Gradient Checking** was a vital sanity check when computing the gradients during backpropagation. We used the derivative of the second order terms of the Taylor expansion with respect to a scalar x to approximate our gradient by a finite difference approximation. We set $\epsilon = 10^{-4}$ for our learning rate, since it is still small enough to detect bugs [3, p.21]:

$$\frac{\partial f(x)}{\partial x} = \frac{f(x+\epsilon) - f(x-\epsilon)}{2 * \epsilon} + \sigma(\epsilon^2) \tag{6.1}$$

We compared the analytically and numerically derived gradients and checked for a difference smaller than $1e-9$.

It is common practise to use **visualistation** for debugging unsupervised and supervised learning tools [3]. We therefore visualised the output for testing the functionality of our System developed. The following output was created for testing, which can be found in the Appendix:

- Sparse Autoencoder: Whitened patches, Whitening matrix
- Convolutional Neural Network: Convolved images, Pooled images,
- Graphical User Interface: Screenshots

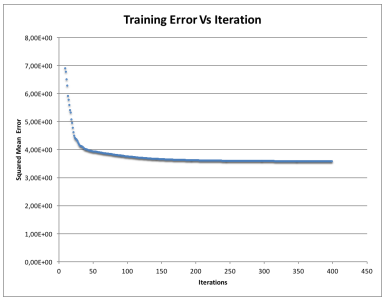
Chapter 7

Experiments and Results

Primarily, in this section we present conducted tests and experiments, which were used to rate the performance of our system and to answer our research question: Can our system break a CAPTCHA? We shall first present the performance of our Sparse Autoencoder, compute the accuracy of our classifier on the test set and explore whether pre-training had an effect on our results. And finally, we will be testing whether we could meet our aim A2, by attempting to break a simple CAPTCHA using our system developed. By using metrics derived from the Literature Review, we will be able to evaluate the results.

Iterating 400 times, the Autoencoder can take up to 7 hours when running on a simple 2.5 GHz machine. The graph below shows the development of the mean squared error during optimisation using the L-BFGS algorithm.

Having constantly updated the weights and biases after each iteration, the system saved the optimised parameters output the learnt features. The figure below shows the weights, also known as 'filters' or 'kernels', learnt by the Sparse Autoencoder through minimising the reconstruction error.



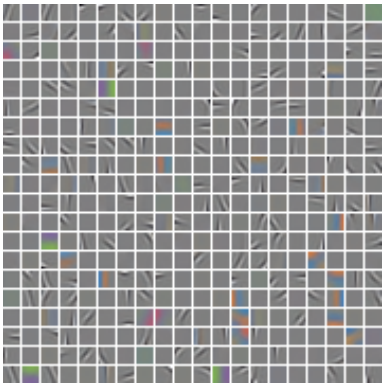


Figure 7.1: Left: Mean Squared Error, Right: Extracted Features.

In general, well trained networks generate smooth filters, whereas a buggy implementation

usually returns a noisy pattern [17]. Since the filters computed during training looked promising, they were use to initialise the weights of our Convolutional Neural Network.

7.1 Testing the Classifier on the Testset

Our Softmax Classifier was trained by minimising the mean squared error in 400 iterations using the L-BFGS algorithm. The figure below shows the trend of the error during optimisation:

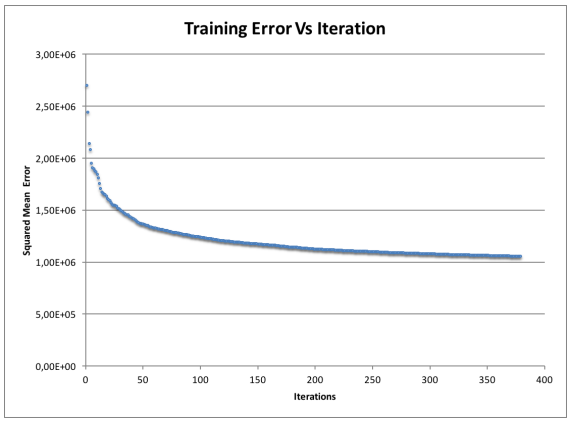


Figure 7.2: Training Error vs Epoch.

For this experiment, we used the same 4 selected classes as used in the training set, namely, airplanes, cars, cats and dogs. The test set included 3,200 images. The trained model, parameterised by the optimum parameter vector, input size of the data and number of classes, was tested by feeding the classifier with the pooled and convolved features and corresponding test labels from the test set. We used equation 5.12 in the design chapter to generate a matrix M including the class probabilities. Predictions were computed by extracting the maximum of matrix M. The output was therefore a vector of numbers indicating the predicted class which the input is most likely to belong to.

The predicted labels generated by the model were compared to the true test labels. By taking the sum over the number of correct predictions and averaging on our dataset, we could compute the accuracy [27, p.198], as well as the error rate (the inverse) of our model on the test set:

$$\frac{\sum(t_n = \hat{t}_n)}{N} * 100 = accuracy\%$$

$$\frac{\sum(t_n = / = \hat{t}_n)}{N} * 100 = error\%$$

7.1.1 Performance on the Testset

The model was tested in two rounds. For each round we calculate the accuracy and the error of our model as mentioned above on the test set 1) with pre-training and pre-processing, 2) without pre-training, without pre-processing respectively.

Test Round 1

During this round, we tested our Classifer including the results gained from pre-processing and pre-training. Convolution, Pooling and calculating the prediction takes 718.907516956 seconds. Evaluating the results, we found that our Model #1 achieves an accuracy of 70%.

Test Round 2

Opinions and suggestions vary when it comes to whether or not it is necessary to pre-train a model. In this round, we tested our model in the same fashion as above, however, this time we initialise the weights randomly from a uniform distribution within an interval of $[-4 * \sqrt{\frac{6}{(fan-in+fan-out}} + 4 * \sqrt{\frac{6}{(fan-in+fan-out}}]$ and the biases to zero, in order to verify whether pre-training had a significant effect on our model's performance. Our model showed an accuracy of 55%, which justifies the use of pre-training, in order to achieve reasonable results.

Test Round	Accuracy	Error	Time (in seconds)
Model with Pre-training	70	30	718.907516956
Model without Pre-training	55	45	658.3940921
Difference	15	15	60.51342486

Table 7.1: Error and Accuracy on Test set.

Evaluation

This project should be seen as a proof-of-concept aiming to explore the use of a Convolutional Neural Network for cracking CAPTCHAs. From literature reviewed, we found that Golle [13] achieved an accuracy of about 82% on a test set including 2 different classes, namely cats and dogs. Recall that Chew and Tygar [9] pointed out that binary classification could easily be broken by extracting low-level features such a colour and shape. Comparing those results with an achieved accuracy of 70% on a dataset which includes 4 different classes, we consider our results on the test set a success.

To explore the results further, we extended our analysis on the best performing model in order to make sense of incorrect predictions and discover why they arose. This can be visualised in the form of a confusion matrix. The confusion matrix shows the given 'true' and 'negative' labels [27], where each row represents instances of the true class and each column represents the predicted class. The colourbar to the right indicates the probability (the darker the shade of red colour represented, the higher the probability).

Having computed the confusion matrix, it is clear that there is indeed a confusion in between the four classes. For further analysis, a normalised confusion matrix was computed, in order

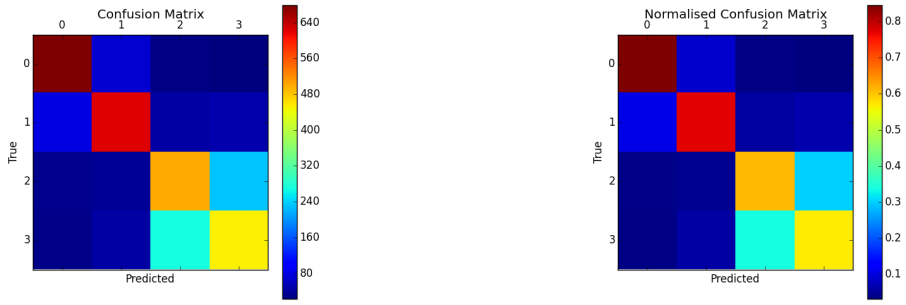


Figure 7.3: Left: Confusion Matrix, Right: Normalised Confusion Matrix.

to see the relative confusion in between the classes. The results were not surprising. The model returned incorrect predictions for the two classes belonging to the hierarchical class 'animals'. Interestingly, dogs got slightly more often mixed up (273 out of 800) than cats (232 out of 800). We received similar results for the classes 'airplanes' and 'cars', which both count as 'vehicles'. Here we can see that 70 images containing an airplane were confused as a car and 85 car-labled images were confused as an airplane.

We invite researchers to conduct further research by considering the hierarchical classes in the form of a binary classification, however, for our purposes, we are only required to know that these confusions exist and will not explore them any further.

7.2 Cracking CAPTCHAs

The Graphical User Interface discussed in the previous chapters was used to test our best model on a CAPTCHA challenge. We generated CAPTCHAs randomly and let our model predict the solution. Specifically, we randomly picked one image $x^{(1)}$, where $x \in 3,200$ with label $y^{(1)}$, where $y \in 1,2,3,4$, representing the solution of the challenge. We then picked m random images from the test set $x^{(m)} \in 3,200 - m + 1$ with an associated label $y^{(m)}$, where $y \in 1,2,3,4 (\neq y^{(1)})$. Each challenge $c^{(i)}$ was combined into a single captchachallenge vector. Given the challenge, each image was passed through the Convolutional Neural Network, where the images were convolved, pooled and fed into our Classifier. The model generated the class probabilities for each label. Since we were only interested in the class appearing in the question (which is equal to the label belonging to the first image), we only required the class probabilites of label $y^{(1)}$. The maximum value and therefore the highest probability was returned and represents the prediction of our model. We compared the solution to the actual true label, if both are equal, the model passes the CAPTCHA challenge successfully. Please note that we only generated one solution for now, in order to make the CAPTCHA relatively easy to pass. For the purpose of extendability, the number of solutions can easily be modified.

For comparison, CAPTCHAs of 4 different difficulties were generated as follows:

1. Very Easy CATPCHAs consist of one correct image $x^{(1)}$ and one dummy image $x^{(m)}$, where $m = 1$
2. Easy CATPCHAs consist of one correct image $x^{(1)}$ and three dummy images $x^{(m)}$, where $m = 3$
3. Difficult CATPCHAs consist of one correct image $x^{(1)}$ and four dummy images $x^{(m)}$, where $m = 5$
4. Very Difficult CATPCHAs consist of one correct image $x^{(1)}$ and five dummy images $x^{(m)}$, where $m = 8$

7.2.1 Performance on Cracking Captchas

We evaluated the performance of the Classifier on the CAPTCHA challenge using the evaluation metrics from research on image-based CAPTCHAs and image classification, as well as metrics commonly used in machine learning.

Before explaining the results gained during the test rounds, we shall present two metrics that shall be used to rate the difficulty of the CAPTCHA challenge in question. As suggested in Golle's paper [13], we derived the **success probability** for passing a CAPTCHA using equation 3.1 from the Literature Review. For demonstration, we shall use an easy CAPTCHA, where 4 images are presented to the model, three of which are dummy images and one of which represents the solution. Success Probabilities for other difficulty levels can be seen in the table below. Recall that the success probability defines the probability of achieving i number of successes during N number of rounds. Having achieved a 70 % accuracy on the test set, our success probability is:

$$P(Y = y) = P(y) = \binom{4}{4} 0.7^1 (1 - 0.7)^{4-4} = 49\% \tag{7.1}$$

Difficulty of the Captcha Challenge: For comparison, it was necessary to evaluate the difficulty of the CAPTCHA challenge generated. From the Literature review, we were able to derive the efficacy metric suggested by Chew et al [9]. We used the equation 3.2 from the Literature Review to calculate the efficacy metric G. Before doing our calculations, we needed to derive a few of the parameters. We set the number of rounds m and the number of successes i to 1. According to the experiments conducted above, the probability of our system to pass a given CAPTCHA challenge is 0.7. As for the probability of a human user passing a round, the numbers vary greatly, due to accessibility and language issues. We shall therefore consider the average of a worst and best case. Microsoft claimed that the probability of a human user passing an ASSIRRA CAPTCHA would be 0.98 in under 30 seconds, whilst Chew et al. developed a CAPTCHA that can be passed by a probability of 0.9 in 42 seconds, which

results in an average probability of 0.94 in 36 seconds[9]. Our trained classifier could compute predictions for a generated CAPTCHA challenge in about 1.26279284 seconds, and is therefore 28 times faster than a human user.

$$G = \sum_{i=4}^{4} \binom{4}{4} 0.94^{9}(1-0.94)^{4-4} * [1 - \sum_{i=4}^{4} \binom{4}{4} 0.7^{4}(1-0.7)^{4-4}]^{2} 8 = 0.181 * 100 = 18.08\% \tag{7.2}$$

Test Rounds

The table below presents the results of the Classifier on 12 CAPTCHA challenges, 3 challenges per difficulty level. Inspired by evaluation metrics used during the ILSVRC competition we used the prediction error rate (P-Error), Top-1 Error (Top-1) and hierarchical error (H-Error) derived, in order to evaluate the performance of our algorithm on the image classification task. The prediction error rate was calculated using equation 3.3. In our case, the Top-1 Error checks whether the solution of the CAPTCHA was predicted successfully and gives lower penalty if the solution image was correctly classified (-0.1). The hierarchical error increases penalty for misclassification outside the hierarchical class (+0.1), for example a dog misclassified as a car would get a higher penalty than a dog misclassified as a cat. To demonstrate how difficult it is to pass the given CAPTCHA, we include the success probability (S) and the efficacy metric (G) for each difficulty.

Difficulty	P-Error	Top-1	H-Error	S	G
Test Round 1					
Very Easy	0	0	0	0.49	1.49059*10^{-9}
Easy	0.75	0.75	0.85	0.2401	1.32709*10^{-9}
Difficult	0.83	0.83	0.53	0.1176	0.0161439
Very Difficult	0.6	0.5	0.4	0.0403	0.166528
Test Round 2					
Very Easy	0	0	0	0.49	1.49059*10^{-9}
Easy	0.5	0.5	0.3	0.2401	1.32709*10^{-9}
Difficult	0.6	0.6	0.5	0.1176	0.0161439
Very Difficult	0.7	0.7	0.6	0.0403	0.166528
Test Round 3					
Very Easy	0	0	0	0.49	1.49059*10^{-9}
Easy	0.25	0.25	0.15	0.2401	1.32709*10^{-9}
Difficult	0.83	0.83	0.63	0.1176	0.0161439
Very Difficult	p	top	h	0.0403	0.166528

Table 7.2: Performance on Cracking CAPTCHAs.

Evaluation

The table gives a good overview and clearly shows that increasing the number of images in the CAPTCHA challenge, is an easy way to make a challenge more difficult. Just by looking at the success probabilities respectively, one can see that the current image-based CAPTCHAs make it difficult for attackers who use machine learning approaches to crack a CAPTCHA. However, recalling the statement from Zhu et al [36], an attack producing a success probability of 0.6% or higher within 30 seconds, is claimed to be effective. With a success probability of 4% achieved on the most difficult CAPTCHAs, we therefore conclude that we successfully implemented a system, which passes a simple CAPTCHA challenge.

Chapter 8

Discussion and Conclusion

8.0.1 Findings

Thanks to the vast progress in computer vision and image recognition, this project has successfully proven that we could use off-the-shelf approaches and technologies, in order to pass an image-based CAPTCHA challenge. Even with a relatively simple pre-trained Convolutional Neural Network, we managed to develop a system which is able to crack CAPTCHAs with an accuracy of about 70% and can therefore assume that the state-of-the-art models, which achieve as slow as 6.5%, would be able to do so even more successfully.

Do image recognition CAPTCHAs still represent a hard AI problem? Yes. No. Maybe. The development and training of a model capable of passing a CAPTCHA challenge is fairly involved. Keeping in mind that the model trained during this project is very limited in its capability, e.g. restricted to 4 different classes and is already computationally expensive. The leading systems in this field can take up to a week. Nevertheless, in the past two years, the research area has gained a lot of attention. There are numerous libraries and pre-trained models available to facilitate development of neural networks. In their paper, Luis van Ahn et al. [2] point out that a CAPTCHA doesn't need to be 100% secure, but the AI problem presented should aim to keep the work factor as high as possible. The latest paper on image-based CAPTCHAs found was published in 2008 and does not consider the state-of-the-art benchmarks. We have branched out to further more general academic reading in the field of image recognition to identify properties that could ensure the security of an image-based CAPTCHA.

To make an image recognition CAPTCHA more secure, we have identified the following three main features [1]:

1. Security based on Image Properties. We found that images have certain properties that can affect the security of an image recognition captcha. Natural Images are stationary,

[1] Note that we don't consider the trade-off between accessibility and security of a CAPTCHA

which means that the statistics are equally distributed in the image. This property is taken advantage of when extracting features during convolution, since we can use our feature extracting detector to learn features anywhere in the image rather than having to learn from the entire image, which can be very computationally expensive for images larger than $28 * 28$. Image statistics are non-stationary when it comes to specific categories [17], such as an enclosed forests or far views of city center. Another way to get round the advantage of stationary images, would be to use images including multiple objects. This would mean that the model would have to learn the whole image, in order to be able to detect one of the images correctly, which leads to high computational expense.

2. Security based on Properties of the CAPTCHA challenge. From the experiment conducted, we found that to make the CAPTCHA challenges secure, challenges have to be generated completely random. If one can predict the number of solutions to be selected or the number of different possible classes, attackers can increase their success probability. Furthermore, by increasing the number of images to be selected and challenge rounds, as well as the size of each image, the efficiency of attackers can be significantly decreased. One of the most effective ways to make image recognition CAPTCHAs less prone to Machine Learning attack is to use the images from a large and dynamic database. Through changing the types of challenges dynamically, models would have to be trained on a huge amount of data.
3. Security based on Limitations of Convolutional Neural Networks. On reviewing the literature in the field, we found that there are hard and easy classes a network can recognise, such as images that contain colours and texture features, for example, according to the results from the ILRVCS competition, red foxes, tigers or hamsters are easier to classify or detect by a neural network than ladles or water bottles. What's more, up until now, classifiers struggle to interpret scenes, describe emotions or detect the beauty of nature such as a rainbow in the background. Recent research shows that Convolutional Neural Networks could be fooled into misclassifying images with a high confidence level, which are unrecognisable to humans (see Figure 3.2). CAPTCHA designers could make use of this phenomenon.

8.0.2 Goals Achieved

The goals set before the project have all been achieved. On reviewing the literature we were able to explore novel machine learning techniques (A1) used for classifying images and discovered the main problems and difficulties of cracking CAPTCHAs (A2) through implementing some of the approaches on a simple CAPTCHA (A3). Last but not least, the section above discussed features which could help improve the security of image recognition CAPTCHAs (A4).

8.0.3 Future Work

We invite researchers to extend this project, in order to improve performance and results and the capability of cracking CAPTCHAs. We believe that there is a lot of room for improvements. By increasing the results by 20%, we could pass difficult CAPTCHAs more successfully. Building a more powerful system, one could use the features derived above to make an even more secure CAPTCHA and attempt to break it with the improvement model.

Bibliography

[1] Agrawal, P., Girshick, R. B. and Malik, J. [2014], 'Analyzing the performance of multilayer neural networks for object recognition', *CoRR* **abs/1407.1610**.
URL: *http://arxiv.org/abs/1407.1610*

[2] Ahn, L. V., Blum, M., Hopper, N. J. and Langford, J. [2003], Captcha: Using hard ai problems for security, *in* 'Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques', EUROCRYPT'03, Springer-Verlag, Berlin, Heidelberg, pp. 294–311.
URL: *http://dl.acm.org/citation.cfm?id=1766171.1766196*

[3] Bengio, Y. [2012], 'Practical recommendations for gradient-based training of deep architectures', *CoRR* **abs/1206.5533**.
URL: *http://arxiv.org/abs/1206.5533*

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

[5] Boyter, B. [2013], 'Decoding captchas', [Online]. https://www.google.com/recaptcha/api2/demo/. Last Accessed: 15/04/2015.
URL: *http://www.boyter.org/decoding-captchas/*

[6] Bursztein, E., Aigrain, J., Moscicki, A. and Mitchell, J. C. [2014], The end is nigh: Generic solving of text-based captchas, *in* '8th USENIX Workshop on Offensive Technologies (WOOT 14)', USENIX Association, San Diego, CA.
URL: *https://www.usenix.org/conference/woot14/workshop-program/presentation/bursztein*

[7] Bursztein, E., Martin, M. and Mitchell, J. [2011], Text-based captcha strengths and weaknesses, *in* 'Proceedings of the 18th ACM Conference on Computer and Communications Security', CCS '11, ACM, New York, NY, USA, pp. 125–138.
URL: *http://doi.acm.org/10.1145/2046707.2046724*

[8] Chandavale, A. A., Sapkal, A. M. and Jalnekar, R. M. [2009], Algorithm to break visual captcha., *in* 'ICETET', IEEE Computer Society, pp. 258–262.
URL: *http://dblp.uni-trier.de/db/conf/icetet/icetet2009.html#ChandavaleSJ09*

[9] Chew, M. and Tygar, J. D. [2004], Image recognition captchas, Technical report, EECS Department, University of California, Berkeley.
URL: *http://www.eecs.berkeley.edu/Pubs/TechRpts/2004/5256.html*

[10] Coates, A., Ng, A. Y. and Lee, H. [2011], An analysis of single-layer networks in unsupervised feature learning, *in* 'International conference on artificial intelligence and statistics', pp. 215–223.

[11] Fei-Fei, L. [2015], 'How we're teaching computers to understand pictures', [Online]. http://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures?language=en. Last Accessed: 01/09/2015.
URL: *http://www.ted.com/talks/fei*$_f ei_l i_h ow_w e_r e_t eaching_c omputers_t o_u nderstand_p ictures?language = en$

[12] Glorot, X., Bordes, A. and Bengio, Y. [2011], Deep sparse rectifier neural networks, *in* G. J. Gordon and D. B. Dunson, eds, 'Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)', Vol. 15, Journal of Machine Learning Research - Workshop and Conference Proceedings, pp. 315–323.
URL: *http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf*

[13] Golle, P. [2008], Machine learning attacks against the asirra captcha, *in* 'Proceedings of the 15th ACM Conference on Computer and Communications Security', CCS '08, ACM, New York, NY, USA, pp. 535–542.
URL: *http://doi.acm.org/10.1145/1455770.1455838*

[14] Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S. and Shet, V. [2013], 'Multi-digit number recognition from street view imagery using deep convolutional neural networks', *CoRR* **abs/1312.6082**.
URL: *http://arxiv.org/abs/1312.6082*

[15] Hire, F. [2014], 'A unifying view of deep networks and hierarchical temporal memory', [Online]. http://inbits.com/2014/09/a-unifying-view-of-deep-networks-and-hierarchical-temporal-memory/. Last Accessed: 15/08/2015.
URL: *http://inbits.com/2014/09/a-unifying-view-of-deep-networks-and-hierarchical-temporal-memory/*

[16] Karpathy, A. and Li, F. [2014], 'Deep visual-semantic alignments for generating image descriptions', *CoRR* **abs/1412.2306**.
URL: *http://arxiv.org/abs/1412.2306*

[17] Krizhevsky, A. and Hinton, G. [2009], 'Learning multiple layers of features from tiny images'.

[18] Krizhevsky, A., Sutskever, I. and Hinton, G. E. [2012], Imagenet classification with deep convolutional neural networks, *in* 'Advances in Neural Information Processing Systems'.

[19] Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B. and Ng, A. Y. [2011], On optimization methods for deep learning., *in* L. Getoor and T. Scheffer, eds, 'ICML', Omnipress, pp. 265–272.
URL: *http://dblp.uni-trier.de/db/conf/icml/icml2011.htmlLeNCLPN11*

[20] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. [1998], Gradient-based learning applied to document recognition, *in* 'Proceedings of the IEEE', pp. 2278–2324.

[21] LeCun, Y., Kavukcuoglu, K. and Farabet, C. [2010], Convolutional networks and applications in vision, *in* 'Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on', IEEE, pp. 253–256.

[22] Lee, H. [2010], *Unsupervised feature learning via sparse hierarchical representations*, Stanford University.

[23] Lorenzi, D., Vaidya, J., Uzun, E., Sural, S. and Atluri, V. [2012], Attacking image based captchas using image recognition techniques, *in* V. Venkatakrishnan and D. Goswami, eds, 'Information Systems Security', Vol. 7671 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 327–342.
URL: *http://dx.doi.org/10.1007/978-3-642-35130-3*$_{23}$

[24] Lu, Y., Zhang, L., Wang, B. and Yang, J. [2014], Feature ensemble learning based on sparse autoencoders for image classification, *in* 'Neural Networks (IJCNN), 2014 International Joint Conference on', pp. 1739–1745.

[25] Ng, A. [2013], 'Udfldl tutorial', [Online]. http://deeplearning.stanford.edu/tutorial/. Last Accessed: 28/07/2015.
URL: *http://ufldl.stanford.edu/tutorial/StarterCode/*

[26] Nguyen, A. M., Yosinski, J. and Clune, J. [2014], 'Deep neural networks are easily fooled: High confidence predictions for unrecognizable images', *CoRR* **abs/1412.1897**.
URL: *http://arxiv.org/abs/1412.1897*

[27] Rogers, S. and Girolami, M. [2011], *A First Course in Machine Learning*, 1st edn, Chapman & Hall/CRC.

[28] Russakovsky, O., Deng, J., Huang, Z., Berg, A. C. and Fei-Fei, L. [2013], Detecting avocados to zucchinis: what have we done, and where are we going?, *in* 'International Conference on Computer Vision (ICCV)'.

[29] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. and Fei-Fei, L. [2015], 'ImageNet Large Scale Visual Recognition Challenge', *International Journal of Computer Vision (IJCV)* pp. 1–42.

[30] Shet, V. [2014], 'Are you a robot?', [Online]. http://googleonlinesecurity.blogspot.co.uk/2014/12/are-you-robot-introducing-no-captcha.html. Last Accessed: 30/04/2015.
URL: *http://googleonlinesecurity.blogspot.co.uk/2014/12/are-you-robot-introducing-no-captcha.html*

[31] Susilo, W., Chow, Y.-W. and Zhou, H.-Y. [2010], Ste3d-cap: Stereoscopic 3d captcha., *in* S.-H. Heng, R. N. Wright and B.-M. Goi, eds, 'CANS', Vol. 6467 of *Lecture Notes in Computer Science*, Springer, pp. 221–240.
URL: *http://dblp.uni-trier.de/db/conf/cans/cans2010.htmlSusiloCZ10*

[32] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. [2014], 'Going deeper with convolutions', *CoRR* **abs/1409.4842**.
URL: *http://arxiv.org/abs/1409.4842*

[33] van Ahn, L. [2011], 'Massive-scale online collaboration', [Video: Online]. http://www.ted.com/talks/luis_von_ahn_massive_scale_online_collaboration?language=en. Last Accessed: 30/04/2015.

[34] Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A. [2008], Extracting and composing robust features with denoising autoencoders, *in* 'Proceedings of the 25th International Conference on Machine Learning', ICML '08, ACM, New York, NY, USA, pp. 1096–1103.
URL: *http://doi.acm.org/10.1145/1390156.1390294*

[35] Vinyals, O., Toshev, A., Bengio, S. and Erhan, D. [2014], 'Show and tell: A neural image caption generator', *CoRR* **abs/1411.4555**.
URL: *http://arxiv.org/abs/1411.4555*

[36] Zhu, B. B., Yan, J., Li, Q., Yang, C., Liu, J., Xu, N., Yi, M. and Cai, K. [2010], Attacks and design of image recognition captchas, *in* 'Proceedings of the 17th ACM Conference on Computer and Communications Security', CCS '10, ACM, New York, NY, USA, pp. 187–200.
URL: *http://doi.acm.org/10.1145/1866307.1866329*

Appendices

Appendix A

Appendix A

A.1 Functional Testing

A.1.1 Sparse Autoencoder

We add a small epsilon to the eigenvalues when transforming the data back. Since we are using ZCA Whitening, the whitened data still appears like the original patches. The whitening matrix was used to whiten our patches.

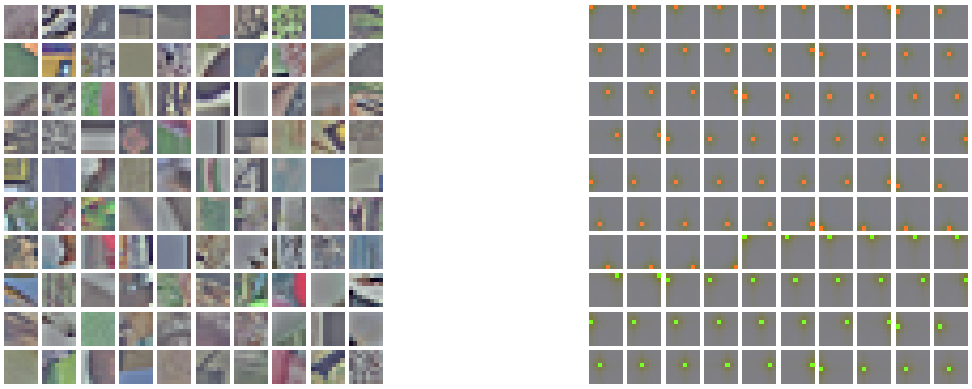


Figure A.1: Pre-Processing of Data Patches.

A.1.2 Convolutional Layer

Recall that we iterate through local regions in the original image and compute the dot product of every region and every filter. The result is a convolved image

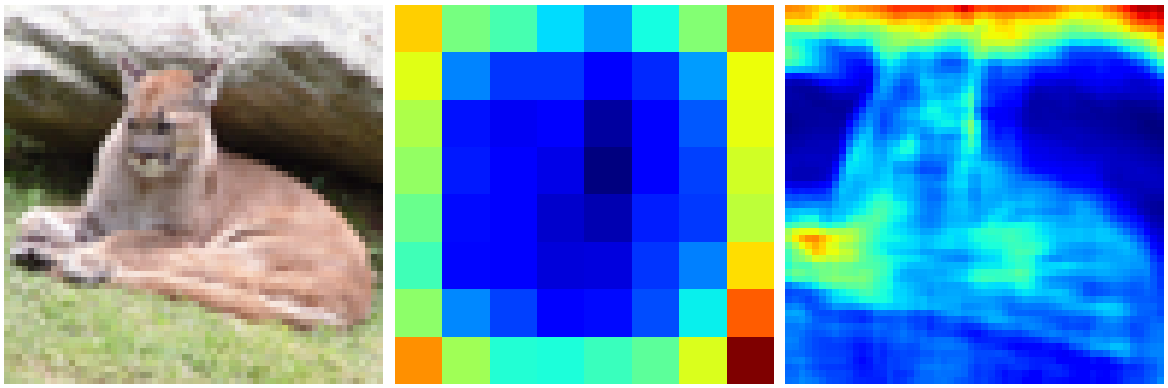


Figure A.2: Convolution Process Example 1.

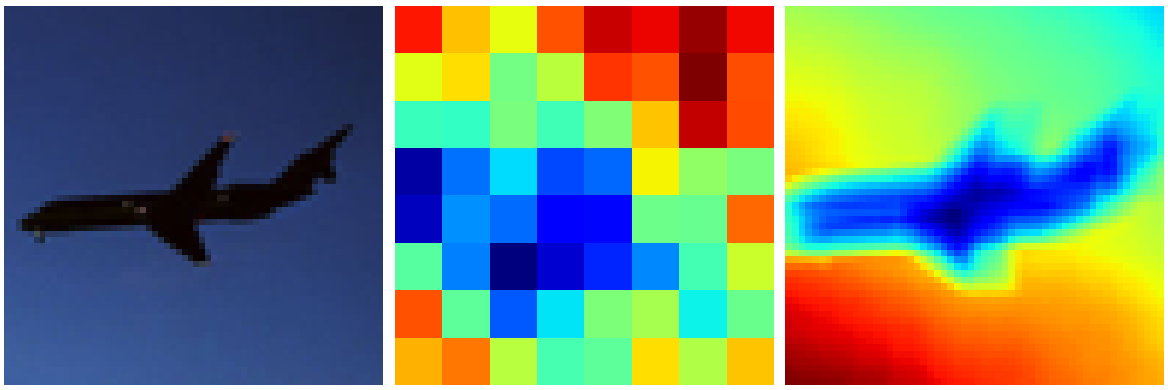


Figure A.3: Convolution Process Example 2.

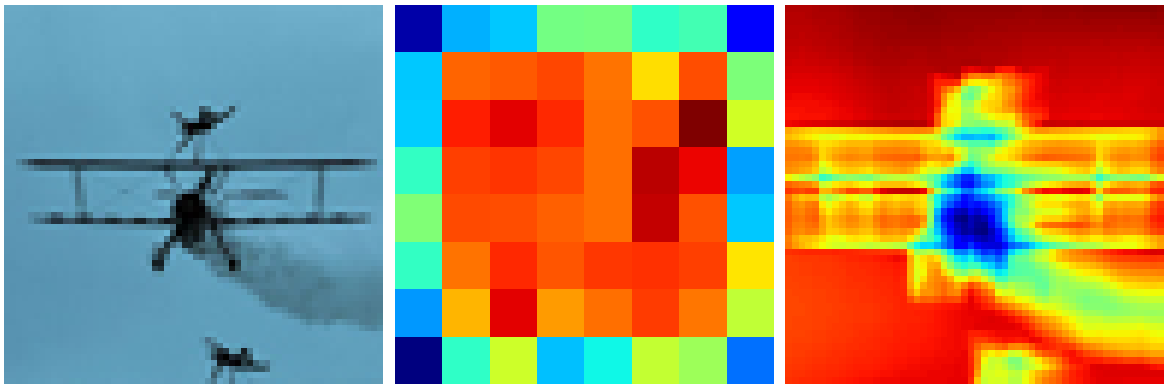


Figure A.4: Convolution Process Example 3.

A.1.3 Pooling Layer

Pooling, or also known as subsampling, is used for dimensionality reduction between convolutional layer and classification layer. Looking at the pooled features, we can see that they are barely recognisable for the human eye.

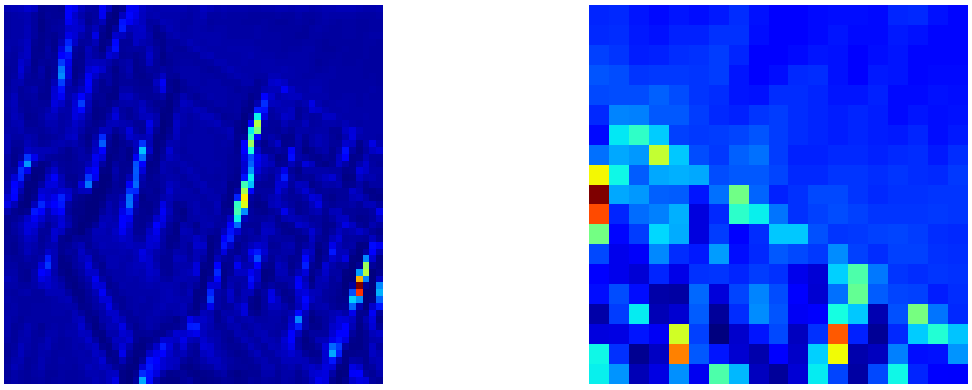


Figure A.5: Convolution Process Example 1.

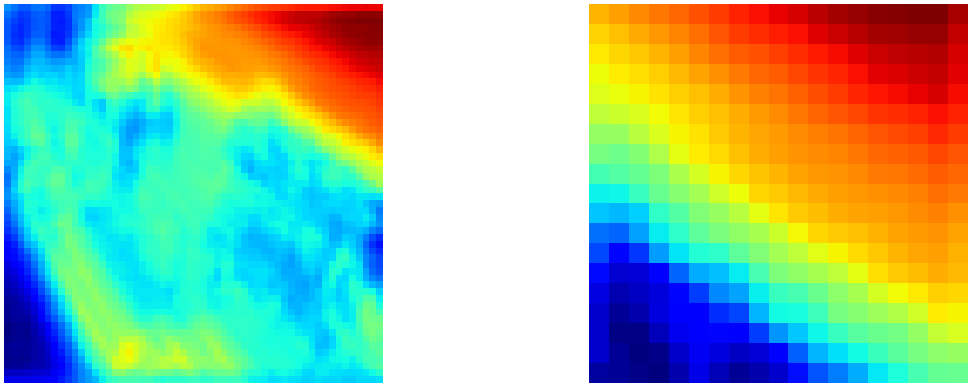


Figure A.6: Convolution Process Example 2.

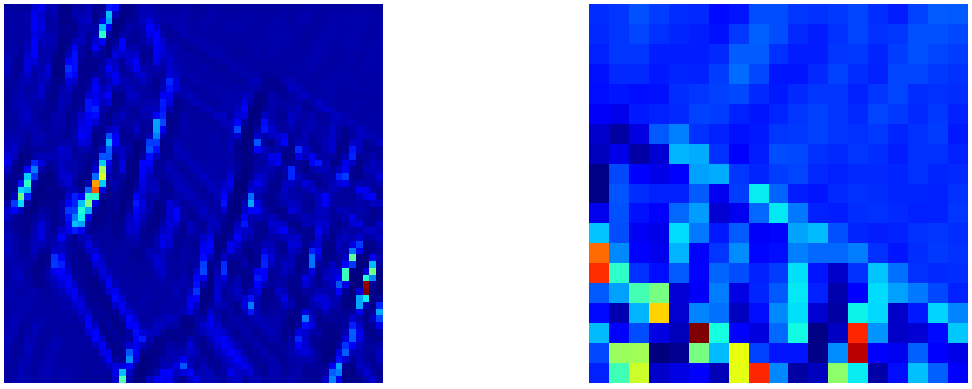


Figure A.7: Convolution Process Example 3.

A.1.4 Graphical User Interface

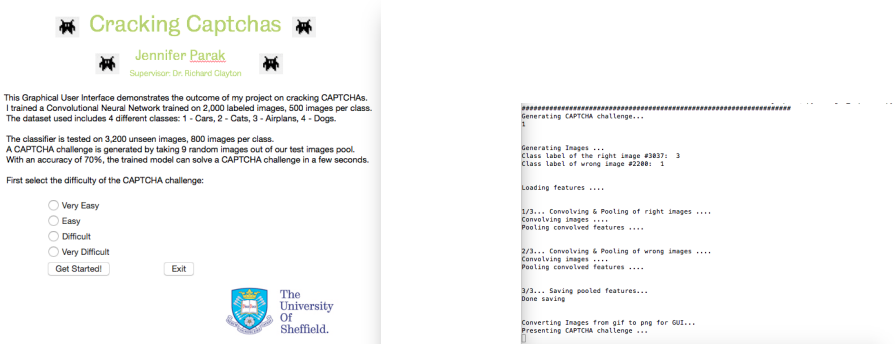


Figure A.8: Instruction Screen.

Figure A.9: Present CAPTCHAs.

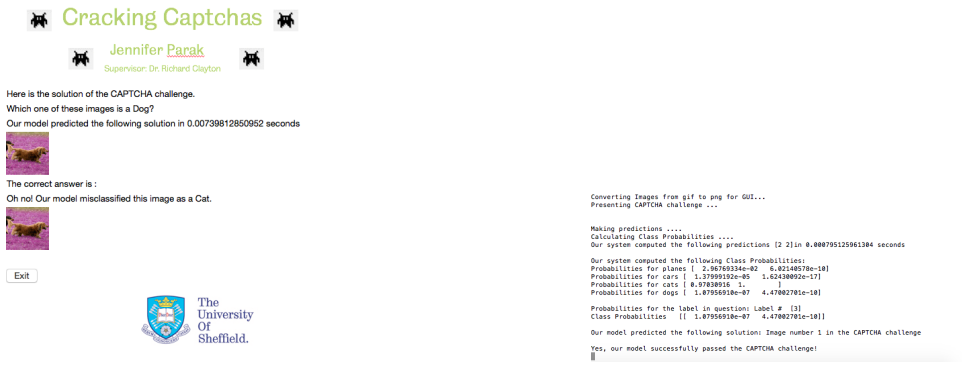


Figure A.10: Test Round 3.