Fall 2015

NEURAL NETWORK CAPTCHA CRACKER

Geetika Garg
San Jose State University

NEURAL NETWORK CAPTCHA CRACKER

Project Report

Presented to

The Faculty of Department of Computer Science

San Jose State University

In Partial fulfillment

Of the Requirement for the Degree

Master of Science in Computer Science

by

Geetika Garg

Fall 2015

SAN JOSE STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled

NEURAL NETWORK CAPTCHA CRACKER

By

Geetika Garg

APPROVED FOR THE DEPARTMENT OF

COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science Date

Dr. Thomas Austin, Department of Computer Science Date

Mr. James Casaletto, Department of Computer Science Date

APPROVED FOR THE UNIVERSITY

Associate Dean Office of Graduate Studies and Research Date

ABSTRACT

NEURAL NETWORK CAPTCHA CRACKER

A CAPTCHA (acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart") is a type of challenge-response test used to determine whether or not a user providing the response is human. In this project, we used a deep neural network framework for CAPTCHA recognition. The core idea of the project is to learn a model that breaks image-based CAPTCHAs. We used convolutional neural networks and recurrent neural networks instead of the conventional methods of CAPTCHA breaking based on segmenting and recognizing a CAPTCHA. Our models consist of two convolutional layers to learn image features and a recurrent layer to output character sequence. We tried different configurations, including wide and narrow layers and deep and shallow networks. We synthetically generated a CAPTCHA dataset of varying complexity and used different libraries to avoid overfitting on one library. We trained on both fixed-and variable-length CAPTCHAs and were able to get accuracy levels of 99.8% and 80%, respectively.

ACKNOWLEDGEMENTS

It gives me immense pleasure to thank my project advisor, Dr. Chris Pollett, for providing me his valuable guidance and time. He helped me in every phase of my project.

I am grateful to my committee members, Dr. Thomas Austin and Mr. James Casaletto, for their suggestions and time, without which this project would not have been possible. I also want to thank my husband, Mr. Sujeet Bansal, for his support and time throughout this project.

LIST OF FIGURES

LIST of TABLES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

A ‘Completely Automated Public Turing test to tell Computers and Humans Apart,’ or CAPTCHA, is a problem that is very easy for a human to solve, but very difficult for a computer to solve. Typically, it involves a task like recognizing a string of characters in an image. In this project, we tried to make it easy for a computer to solve a string of character CAPTCHAs.

CAPTCHAs are ubiquitous on the Internet. They are intuitive for users and are a cheap and fast way to secure a site and to ward off spam. A lot of major websites use them for security on the Internet. When CAPTCHAs were designed, there were no AI programs that could recognize them using computer vision. Recently, deep neural networks have brought major advancements in the field of AI and computer vision. They have reached state-of-the-art or better performance as compared to other methods in the fields of speech recognition [30], computer vision, natural language processing [32], and language translation [33]. We tried using deep neural networks to break CAPTCHAs to assess how secure CAPTCHA-based security systems are.

Traditionally, for object/character recognition tasks in computer vision, separate modules were created for preprocessing (noise reduction), segmentation (character segmentation), and character recognition and sequence generation, where the sequence of characters with highest probability was generated. Systems with multiple modules tend to behave poorly because each module is optimized independently and errors compound between

modules. Our system is trained to learn an end-to-end model, where we solve the entire problem in one module. Deep neural networks have multiple layers, where different layers learn features at different levels. Lower level layers learn low-level features, while higher layers learn higher-level features. The entire network is trained together so as to align the goals of individual layers. In other words, layers co-adapt to learn features that will help optimize for the single big goal of CAPTCHA recognition.

Special purpose neural networks have been studied to solve specific problems. Convolutional neural networks have achieved great success with image and video understanding and are now heavily used in computer vision. Google used convolutional neural networks [4] to process Street View images for detecting home addresses. Recurrent neural networks have enabled state-of-the-art performance in sequence processing tasks. Feed forward networks take fixed size input and generate fixed size output. Recurrent neural networks are used to feed variable length inputs and generate variable length outputs. Google used recurrent neural networks, in particular LSTMs, to generate image captions [5].

In our project, we combined the idea of two papers [4], [5] and are building a model that uses convolutional neural network to learn CAPTCHA image features and then use those features in a recurrent neural network to output the sequence of characters in the image. We combine both of the networks to get a single deep neural network that performs end-to-end CAPTCHA recognition. For our recurrent neural network, we use LSTMs [7], [22] (Long Short Term Memory. LSTMs have been around for decades [7], and lately have been used a lot in

academia and industry [5], as they solve some of the big issues with training recurrent neural networks.

This report is divided into eight chapters. Chapter 2 gives a literary reviews of CAPTCHAs, including what they are and why we need them. Chapter 3 discusses the work done in the related field of CAPTCHA recognition and what we planned to do. Chapter 4 covers an introduction to neural networks and the various components we used in our project. Chapter 5 talks about the framework we used for our project. Chapter 6 focuses on the dataset we generated for our project. We tried different kinds of models, and the details of these are discussed in Chapter 7. Chapter 8 records all the experiments and results we got after training the huge dataset. Chapter 9 concludes the report.

CHAPTER 2

INTRODUCTION TO CAPTCHAs

Before delving into the models directly, let us discuss what CAPTCHAs are. CAPTCHAs were first mentioned in a paper by Moni Naor [1] in 1996. A visual CAPTCHA is usually an image with a series of letters or numbers that prompts a user to recognize what exactly is written in the image. At present, CAPTCHAs are almost a standard security mechanism for defending against malicious and undesirable bot programs on the Internet, such as bots that could sign up for thousands of accounts a minute with free email service providers and bots that could send out thousands of spam messages each minute. They are very annoying and can cause denial of service attacks.

Where is there a need for CAPTCHAs?

The following are some scenarios in which we need to use CAPTCHAs:

Online Polling - In case of online polls, bots or computers can vote automatically. We have to prevent this. To enforce this, a CAPTCHA can be used to ensure that only humans are polling.

Free Mailing Services - There are many email services that are available for free. However, if there are a lot of accounts, it can lead to poor service. Customers would not be served properly. Again, CAPTCHAs are useful here.

Dictionary Attack Prevention - CAPTCHAs can also be used to prevent dictionary attacks in password systems. One of the ways this could be done is by preventing a computer from

iterating through the entire range of possible passwords by requiring it to solve a CAPTCHA after a certain number of unsuccessful logins.

Characteristics of a CAPTCHA:

The important characteristics of a CAPTCHA are:

- Easy for a user to solve.
- Difficult for a program or a computer to solve.

To make it happen, the following standard techniques are used. Normally, characters are used to generate CAPTCHAs. The reason why object-based CAPTCHAs are not used in CAPTCHAs is because recognition of objects requires prior knowledge of the scene, and different regions have different names for objects; therefore, it would not be universal. Instead, characters have a unique data set (for example, in the English alphabet, 26 characters and 10 numeric digits), which is present on keyboards as well, making them easier to understand. Hence, they are easy to generate. Figure 1 shows an image-based CAPTCHA in which we used the alphabet.

Figure 1: A Sample image-based CAPTCHA [36]

Initially, it was realized that CAPTCHAs could be used for security purposes because there were no AI programs that could perform the problem task. CAPTCHAs are based on the underlying assumption that it is difficult to solve this AI problem. If computers could be taught to decode a CAPTCHA, a difficult AI problem would be solved, which would help in research. It was a win-win situation.

CAPTCHAs were first used by AltaVista to prevent “bots” [29]. The motivation for this project was to show that the security methods used in many online systems are not secure and are prone to attack by hackers.

CHAPTER 3

RELATED WORK

Programmatically, breaking CAPTCHAs is not a new concept. For example, Mori and Malik [2] have broken EZ-Gimpy (92% success) and Gimpy (33% success), CAPTCHAs with sophisticated object recognition algorithms. In comparison to earlier works that were based on sophisticated computer vision algorithms, we are planning to train an end-to-end neural network system that would extract the features needed for classification with minimal hand tuning. Neural networks have shown great results recently in many domains, such as natural language processing [32], speech [30], and image processing [4][5]. Neural networks also have brought down the entry barrier in training such models, as one does not require deep domain knowledge to massage inputs in order to provide the features that a model could learn from. The hidden layers in neural networks extract the features that are useful during learning. We will be discussing more about neural networks later in the report.

In some of the papers, for example, in paper [3], the following steps were used for character recognition in a CAPTCHA:

1) Preprocessing
2) Segmentation
3) Training the model for individual character recognition
4) Generating sequence with highest probability

These steps are very difficult to do because of the following reasons:

- Segmentation is difficult, as some digits could overlap with other digits.
- Deformity of digits is also a major concern. For example, a digit “2” can have a larger loop or just a cusp.
- Unknown scale of characters. We do not know how big a character will be, so it is not known how big the segmentation boxes should be.
- Character orientation. Characters could be rotated at arbitrary angles, making recognition difficult.

Modules for each of the steps mentioned above are optimized independently, so systems combining these modules don’t work well in practice. We can instead learn a deep neural network, a single monolithic system for embedding these modules, and train the entire network together to make sure that that objectives of all the modules are aligned. We can just provide CAPTCHA images to the network and let it learn image features and how to use these features for recognition.

Relevantly, Google has published a research paper in which they used a convolutional neural network [4] for detecting home addresses using convolutional neural networks. They achieve a 96% accuracy in recognizing complete street numbers. We took the same idea for this project to train our dataset with convolutional neural networks. But in their work [4], they

fixed the length of the street number in an image unlike CAPTCHAs, in which length could be different. To tackle this problem, we propose to use Recurrent Neural Networks, which have achieved good results recently, as shown by the “show and tell” [5] paper by Google, where they generate a caption (variable length) for a given image.

The real work in decoding a CAPTCHA is to guess all the words in the CAPTCHA correctly. If even one word is wrong, we have to discard the result. Based on the papers [4] and [5], a neural network could be helpful in these scenarios.

Using these ideas, we will use CNNs to learn image features and the RNNs to predict a sequence of characters (which could also be variable) in a CAPTCHA.

Chapter 4

INTRODUCTION TO NEURAL NETWORKS

In our project, we are using neural networks to break CAPTCHAs. Neural networks are inspired by the brain. In 1943, McCulloch and Pitts [6] laid the formal foundation for the field of artificial neural networks. Because neural networks were computationally expensive and there was no good learning algorithm for training the models, they became unpopular in the 1960's. Marvin Minsky and Seymour Papert made this clear in their paper [25]. Another key advance that came later was the backpropagation algorithm, which effectively solved the neural network problem, given by Werbos [26]. In the early 1980's, researchers showed renewed interest in neural networks. From 2005 onwards, they have again become popular, as computers have now become fast enough to do large computations. People have also achieved success in training neural networks with the SGD (stochastic gradient descent) algorithm, which fortunately works, but does not have clear theoretical justification. In addition, neural network models are big, and thus require a lot of training data. With the recent advances in Big Data, it has become very easy to collect training data. They are the hottest area in the field of machine learning [24]. Neural networks work very well with different machine learning problems.

Neural networks are a type of machine learning algorithm. The basic difference between machine learning and conventional programming languages is that in conventional programming, a computer has to be explicitly programmed. We ourselves have to write and maintain the code. But in case of neural networks, the network adapts itself to the problem during training. In conventional programming style, people have to understand the problem well and research different approaches. Since a clear solution is often elusive in practical problems,

people tend to use heuristics, which work for some use cases but do not generalize well. But with machine learning, we have the model itself learn from the data. Machines tend to learn faster (*vis a vis* core research), so they take less time in solving problems. Machine learning algorithms like neural networks also do a good job with generalization. Some of the applications of neural networks include facial recognition as used by Facebook to tag photos [27], image captioning [5] by Google and etc.

Neural networks are essentially a bunch of interconnected elements called neurons. They are an information processing paradigm which is inspired by biological nervous systems. The nervous system contains around 10^{10} neurons. Figure 2 shows a single neuron.

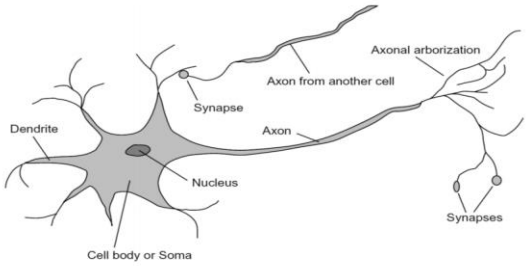


Figure 2: A Biological Neuron [37]

Each biological neuron consists of a cell body. It contains lots of dendrites, which bring electrical signals into the cell, and an axon, which transmits these signals out of the cell. A neuron fires when the collective effect of its inputs reaches a certain threshold. The axons of neurons are dependent on each other and can influence the dendrites of another neuron. Similarly, a neural

network starts with a model, as in Figure 3. It consists of several inputs and a single output. Every input is modified by a weight and multiplied with the input value.

The neuron combines these weighted inputs and, with the help of the threshold value and activation function, determines its output. The objective of the neural network is to transform the inputs into a meaningful output.

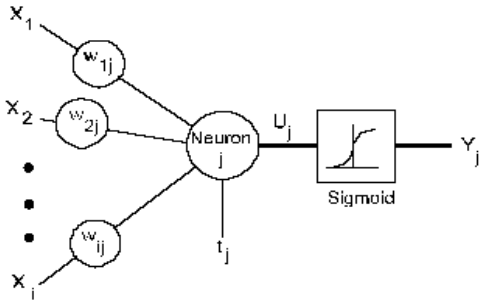


Figure 3: A Simple Neural Network Model (Perceptron) [37]

Training Neural Networks

Backpropagation in Neural Networks

Backpropagation is a technique to train neural networks. In this technique, a set of input and output pairs are assembled. The input data is then fed into the neural network via the input layer. Every neuron in the network processes the input and propagates it to the output layer. The output is then compared with the actual output. The difference between them is known as the “Error Value”. While training a neural network, we try to minimize this “Error Value”. The connection weights in the network are then adjusted gradually, working backwards from the output layer, through the hidden layer, and to the input layer, until the correct output is produced. This is the basic idea of Backpropagation. This algorithm was first introduced in the

1970s, but the importance of it was not fully appreciated until the publication of a paper in1986 [28].

The steps involved in training a neural network model are:

- Initialize a model with random weights.
- Generate output based on the weights.
- Calculate the difference between the actual output and the achieved output.
- Adjust weights according to the error and learning rate.
- Repeat the process until:
    1) Either we have reached the maximum number of epochs,
    2) Or error is not decreasing anymore.

Gradient flow in Back Propagation

Suppose we have a layer with input x and output y, computing:

$$y = f(x)$$

Define L to be the loss. To perform gradient descent on x, we calculate $\delta l/\delta x$ and plug this into the following gradient descent equation:

$$x = x - (\text{learning rate}) * \delta l/\delta x$$

To calculate $\delta l/\delta x$, the backpropagation algorithm uses the derivative chain rule.

$$\delta l/\delta x = \delta l/\delta y * \delta y/\delta x$$

So from the gradient of y, the gradient of x can be computed. This is the way gradient flows from a layer's output to input. By induction, gradients flow backwards from the top layer to the input layer. This is the main idea behind backpropagation.

Supervised Learning vs. Unsupervised Learning

Supervised Learning is a kind of learning in which some prior experience or knowledge in the machine learning problem is used to determine whether the outputs are correct or not. A model that uses this technique adjusts its parameters according to the training examples provided. In other words, the training data should be labeled, which would help the network with learning the parameters. On the other hand, in unsupervised learning, it learns only from local information. It self-organizes the data presented to it and then detects their properties. It divides the data into different clusters representing similar patterns. This is particularly useful in domains, where instances are checked to match previous scenarios. For example, in credit card fraud detection, the pattern of a case can be matched with known fraud patterns.

In the case of our CAPTCHA decoder, we classified our problem of CAPTCHA recognition as supervised learning, in which training examples have labels assigned to them, so that our network can learn how a character in a CAPTCHA may look.

Convolutional Neural Networks

In 1995, Yann LeCun Et al. [34] introduced the concept of convolutional neural networks in which they tried to recognize handwritten characters. A Convolutional Neural Network (CNN) is a variant of multilayer perceptron. CNN contains many layers, of which some could be convolutional layers. A convolutional layer is a layer that applies a convolution filter (a great way to process images for certain features). It can be seen as a sliding window function applied to input pixel matrix.

In our model, we chose to use CNN for learning image features. One of the main advantages of using a Convolutional Neural Network is that it is independent of prior knowledge. All the pixel values can be directly inputted into the network, as opposed to handcrafting the feature values by performing segmentation and filtering beforehand, as explained earlier. CNNs exploit local dependencies in images. A pixel value is more correlated with its neighbors than pixels that are farther away. A normal (dense) layer would have tried to learn all the global interdependencies. But CNN has local connections that work very well for images. Also, they are easier to train, as they have fewer parameters than fully connected networks with the same number of hidden units. They share weights in convolutional layers, which means that the same filter is used for every pixel in the layer. This leads to a reduction in the required memory size and also improves performance. Another advantage is the availability of special purpose hardware like GPUs that can perform convolution very cheaply.

A CNN can contain many convolutional layers followed by fully connected layers. Various components of the Convolutional Neural Network that we used in creating our model are explained below:

Convolutional layer:

Convolutional layer is the core building block of a Convolutional Network. It is the layer in which convolution filters are applied on an input image (or multidimensional feature vectors). In every convolutional layer in a model, different filters are applied. The same filters are applied on all the pixels. Filter application is essentially a convolution operation between input and filter (mostly 2D or 3D vectors) vectors.

If input to a convolutional layer is an image of dimensions x*y*z where x is the height, y is the width, z is the number of channels in the image, and the layer has k filters, the size of the filter is m*n*r, where m, n, and r could be equal or less than x, y and z respectively. Then the convolution layer will output a vector of dimensions x*y*k. For translational and rotational invariance, max-pooling of the feature for all the generated k feature maps is done using a patch of t*t, where t could be any number (usually it is 2 for smaller images to a maximum of 5).

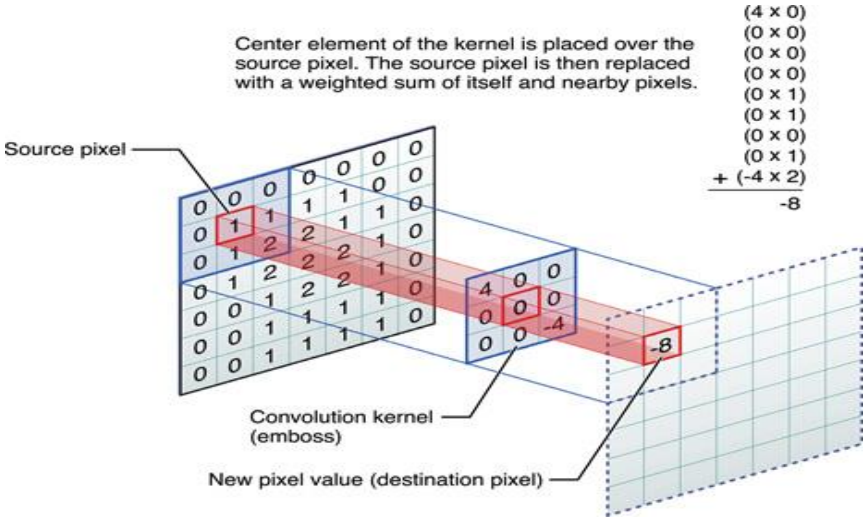


Figure 4: Convolutional Layer [17]

If we have multiple convolutional layers, then lower layers learn low-level features and higher layers learn high-level features. In particular, the first layer takes care of detecting edges, and the following layers can detect different shapes.

The weights of a filter remain the same for all the pixels. This helps in reducing the number of parameters. For example, if image size id 200*50, filter size is 5*5 and if there are 32 filters,

we have only 32*(5*5 +1) (1 for bias) = 832 weights to learn. Otherwise, it would be number_of_pixels*number_of_pixels*filters, which would be 200*50*200*50*32 = 3.2 million. This is several orders of magnitude larger and is thus intractable to learn.

Maxpool layer

The Maxpool layer averages or maxes out feature values. The main function of this layer is to reduce the spatial size of the representation in order to reduce the amount of parameters and computation, which helps in controlling overfitting. It ensures that same result will be seen, even if image features have some small translation or rotation. This feature is known as Location Invariance and Compositionality. In case of CAPTCHAs, the digit “2” could be anywhere in the image. The network tries to learn what the digit “2” looks like, not where “2” is located in the image. In other words, it does not matter where “2” is present.

Dense layer

The dense layer is simply a fully connected layer in which all the units in the layer are connected with all the units of the input layer. We do not need any special layout to apply in such a layer; it can be applied over any array of data. It is a very commonly used layer in neural networks.

Drop out

Since a fully connected layer has most of the parameters in a network, it is prone to overfitting. In order to control this, the drop out method is used. In the drop out method, nodes in

a unit are included at every training stage, with some probability that we do not know the entire network, for one particular example. This helps with preventing overfitting, as the network will

adjust to these situations. After the training is done, all the units in the network are used for inference.

Loss Function

Neural network training is essentially an objective function minimization problem. The objective function is invariably called a loss function that measures how far away predictions of the model are from actual labels. During training, the gradient of the loss function is computed with respect to each weight $w_{i,j}$. It helps us to calculate how a small change in weight can help with decreasing the loss. There are different kinds of loss functions that we can use based on the modeling problem, such as mean squared error, cross entropy, logistic loss, hinge loss, and hamming distance loss.

We used Cross entropy loss function [21] in our model. Cross entropy gives the measure of the distance between what the network believes the output distribution should be and what the actual distribution is. It is the standard loss function for multi-label classification problems. It is basically used when we need to generate a multinomial (i.e. probability for every label) distribution. It penalizes the incorrect predictions a lot, which helps in learning.

Cross entropy loss formula:

$$L_i = -\sum_j t_{i,j} \log(p_{i,j})$$

where p is prediction and t is target tensor.

Activation function

All the neurons in a neural network add their inputs and put them in a nonlinear function, which is known as the activation function. There are many different types of activation functions used in academia and industry. Some of the most commonly used are sigmoid, step function, logistics, hyperbolic, and ReLU.

Sigmoid:

$f(x) = 1/(1+\exp(-x))$

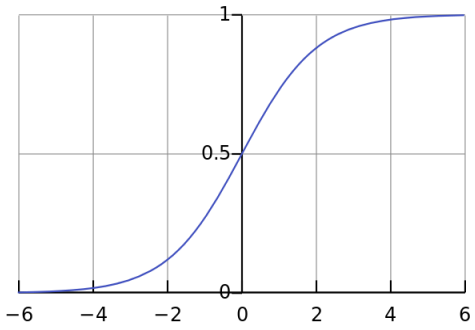


Figure 5: Sigmoid Function

Tanh unit:

f(x) = tanh(x)

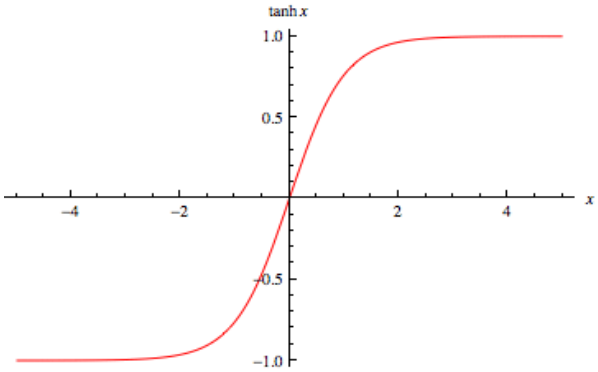
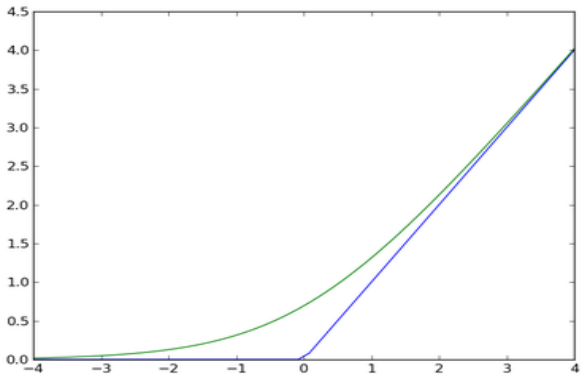


Figure 6: Tanh

LSTMs use tanh as its gradient is bounded, which helps in ameliorating the exploding gradient problem.

Rectified linear unit (ReLU):

f(x) = max(0, x)



Blue - ReLU
Green - Softplus

Figure 7: ReLU Activation Function [20]

In Figure 7, the blue line represents the ReLU activation function. ReLu works in practice, but the ReLU function is a discontinuous function, having discontinuity at 0. Softplus is similar to ReLu, but is continuous everywhere. The Softplus function is represented in the green line in figure 7.

In our models, we used ReLU activation function.

Advantages of using ReLU activation function:

- It does not lead to an exploding or vanishing gradient problem. Also, it has been shown that deep neural networks can be trained efficiently using this function without any pre-training.
- It can also be used in RBM (Restricted Boltzmann machine) to train real-valued inputs.
- They are faster to compute, as they are very simple and don't require normalization.

Softmax Layer

A softmax layer takes the activations (probabilities) calculated so far and divides each of them by the sum of all the activations. In other words, it normalizes the activations. It thereby forces the output of the layer to take the form of probability distribution, i.e. between 0-1. From these, the value with the maximum probability is chosen as the output of the network. It is a type of activation function that is applied on the last layer only.

Learning rate

Learning rate is an important concept in training a neural network, as it determines by what amount the weight should be adjusted in order to converge. A low value of learning rate will take a lot of time to converge, whereas a large value of learning rate may cause the network to bounce around the optimal parameters, as shown in the figure below:

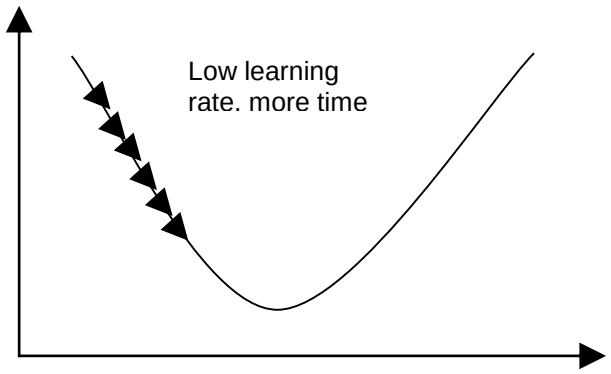


Figure 8: Impact of low learning rate

As can be seen in Figure 8, if we have a lower learning rate, the loss tends to decrease slowly. But with a high learning rate, as can be seen in Figure 9, the network could bounce around the optimal parameter values.

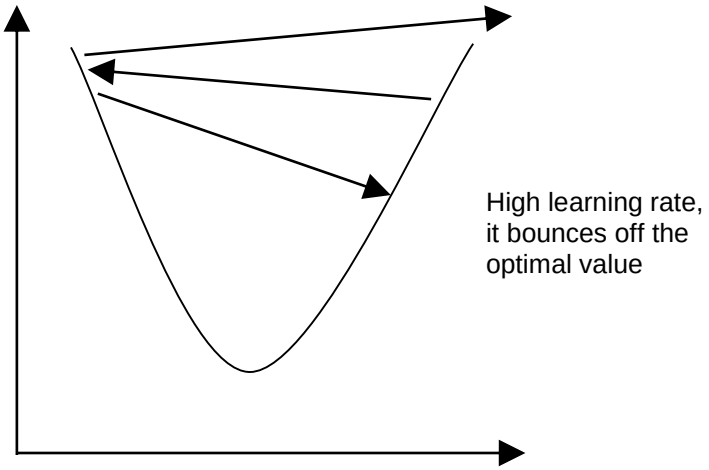


Figure 9: Impact of high learning rate.

Recurrent Neural Networks

One of the disadvantages of feedforward networks is that they accept a fixed length of input vector and generate a fixed length of output vector. Moreover, they perform mapping using the same set of layers, i.e. it does not change.

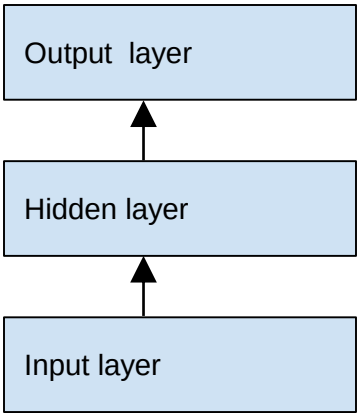


Figure 10: A Sample feed-forward network

Recurrent Neural Networks (RNNs) can be used to overcome these disadvantages. In RNNs, connections between units have a directed cycle. RNNs are called recurrent because they perform the same task for each element in a sequence with the output being dependent on the previous computations, which is not the case with feedforward networks, they just go in a forward direction independently of previously computed values. Unlike feedforward neural networks, RNNs make use of their internal memory to process arbitrary input sequences. Various applications of RNNs include handwriting recognition and speech recognition. Figure 11 shows an example of a network using RNNs.

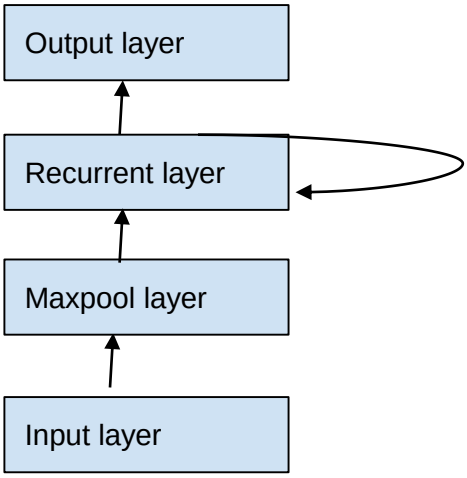


Figure 11: An example of RNN.

Figure 11 can be seen as Figure 12 after unfolding it for three steps.

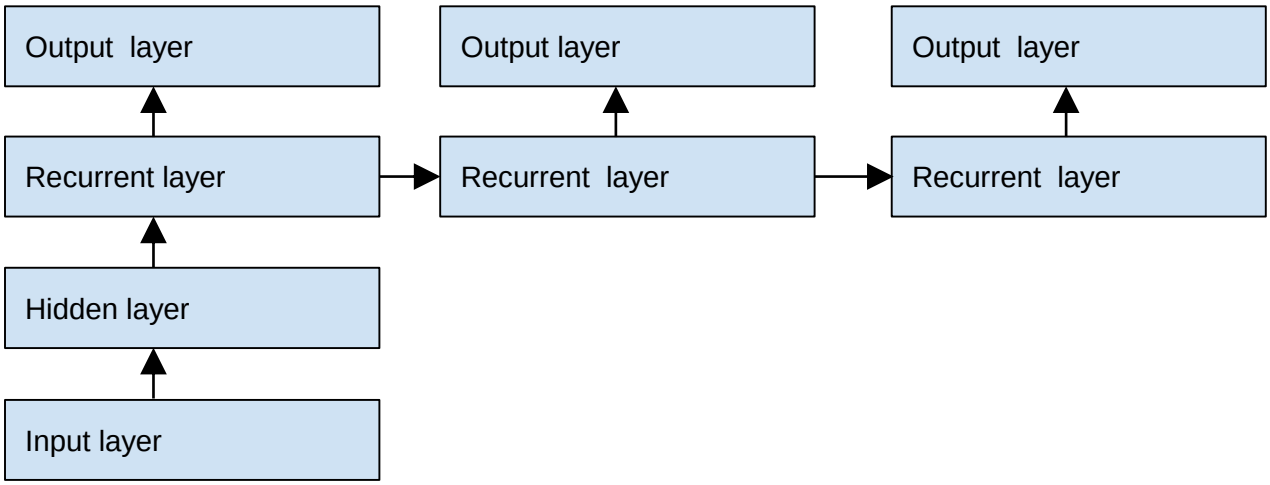


Figure 12: An example of RNN with unfolds.

RNNs have shown good results in predicting sequences of words in which the output of one field is dependent on another field [5]. The most commonly used RNNs are LSTM (Long Short Term Memory), which we have used in our project.

What are LSTMs?

Long Short Term Memory (LSTMs) networks are a special kind of RNNs. Training RNNs is difficult because of vanishing and exploding gradient issues. If gradients and weights are small, gradients can vanish quickly during backpropagation. Conversely, if gradients and weights are large, gradients can explode quickly during backpropagation. The former gradient is known as a vanishing gradient, and the latter is known as an exploding gradient. The issue is in particularly acute when the input or output sequences are long, as the gradients either vanish or explode while back propagating through long sequences. This was first found in 1991 [9][10]. LSTMs ameliorate this issue, as discussed below.

In LSTM blocks, when the error values are back propagatedu7u from the output, the error becomes trapped in the memory portion of the block. Figure 13 shows a structural diagram of the LSTM memory cell.

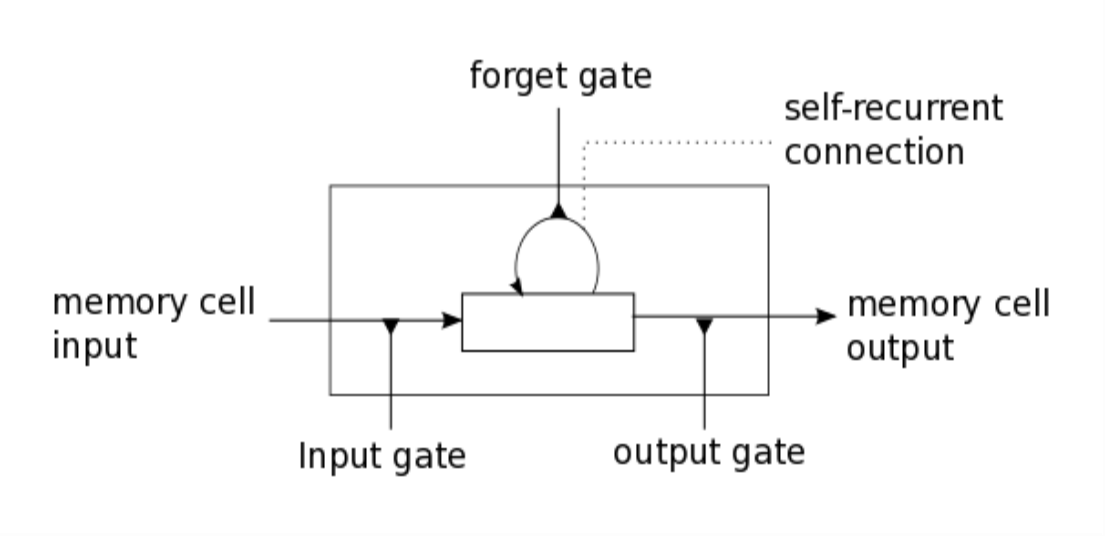


Figure 13: Memory cell in LSTMs [22]

Also, LSTMs use tanh as an activation function. A tanh gradient is always between 0 and 1, so the gradient does not explode when multiplied with gradients during backpropagation.

LSTMs are explicitly designed to avoid the long-term dependency problem. They have shown great performance in unsegmented handwriting recognition [11], machine translation, and speech recognition.

There are two types of LSTMs: forward LSTMs and backward LSTMs. In the case of forward LSTMs, we consider the output that we have so far. For example, in the case of speech recognition, consider a sentence like `I __ a girl` that a network wants to complete. In this case, we can make use of “I” to predict that it should be “am”. So, we took help of the predicted character “I”. Forward LSTMs first predict “I” and then “am” and so on.

In backward LSTMs, we try to predict backwards. For example, in the above example, it will predict “girl” first, then “a”, then “am” and lastly “I”. In bidirectional LSTMs, both forward and backward LSTMs are combined. They try to predict all the characters together. For example, `She lives in __. She speaks French`. In this sentence, we know that because of “lives”, we need to fill the name of a place. But by only looking at French, we could make a decision that it would be France. In these scenarios, bidirectional LSTMs are useful.

We also expect bidirectional LSTMs to be useful in the CAPTCHA recognition problem. For instance, in problems like "rrn", the output could be ['r', 'm'] or ['r', 'r', 'n'], so it is useful to predict if the last character is 'm' or 'n' first and then predict the previous characters.

Chapter 5

FRAMEWORKS USED

There are many publicly available frameworks that provide machine learning. Some of them are:

- Torch7
- Caffe
- Theano

Torch7 [13] is a machine learning library that was developed at New York University, IDIAP Research Institute, and NEC Laboratories America.
Caffe [14] is a machine learning library developed at UC Berkeley.

After preliminary research about the different frameworks available, we chose Theano because we found Theano to be more flexible, specifically due to libraries like Lasagne. Since Theano is in Python, it is very easy to integrate it with the rest of our system, which is also written in Python. It performs functions like input pre-processing and weights storage. Also, today Theano is one of the most popular frameworks in the Machine Learning community.

Theano

Theano [12,15] was developed by the University of Montreal. It is a Python library used to define, optimize, and evaluate mathematical expressions, especially ones with

multi-dimensional arrays. It combines the convenience of NumPy's syntax and the speed of optimized native machine language.

Theano helps in creating symbolic expressions, are beneficial when creating a neural network. Also, it supports static differentiation; that is, we just need to specify our architecture and the loss function in a declarative way to get the gradients for free.
Theano is not a neural network library, but rather a mathematical expression compiler. Its basic components are not neural network layers, but mathematical operations. But there are many wrappers around Theano that provide neural network libraries. One of them that we have used is **Lasagne**. Lasagne is a Python package used to train neural networks. It uses Theano internally. We used Lasagne because of the following reasons:

- It implements LSTM. Theano by itself does not have implementation of LSTM. Theano contains only building blocks like scan, but Lasagne has LSTM implementation.
- It implements various learning algorithms like stochastic gradient descent with momentum, adagrad etc. We use Nesterov momentum [31].
- It implements the framework to keep track of all the neural network parameters, like weights and biases. It makes it easy to save the parameters and initialize the model with pre-trained weights.

Theano takes care of optimization while compiling. Right before the expression is compiled, it is optimized. The expressions are represented as a graph of operations and variable nodes. Theano contains a lot of graph optimizers, which modify the graph to optimally produce the same result.

In addition, Theano makes use of the GPU, if present. One of the Theano's design goals is to do computations at an abstract level, so that the compiler is flexible about how to carry out the calculations on a graphic card. It supports both NVIDIA cards and OpenCL devices.

Chapter 6

DATASET

Since our CAPTCHA breaking problem is a machine learning problem, we need a huge amount of data to train our models. A standard dataset for CAPTCHAs is not publicly available, so we had to generate the dataset synthetically. We used the Simple CAPTCHA library [17], which is available in Java. It has various functions to create noise and backgrounds in a CAPTCHA image. We wrote a Java module that generates a random string of 4 to 7 characters in length and then randomly adds noise to the image. The background of an image is chosen with the same randomness. Below are a couple of images we generated through our program.

Figure 14: CAPTCHA examples

All the images generated are of same size (200*50), where 200 is the width and 50 is the height of the image. We trained our model using simple CAPTCHA images, complex CAPTCHA images, and a combination of both. Simple images are those that have very little noise, as shown in Figure 14 on the right-hand side, and complex images contain more noise and clutter, as shown on the left-hand side of Figure 14. We created around 1 million simple images, 2 million complex images of fixed length 5, and 13 million images of variable length, which is a combination of all kinds of possible CAPTCHAs using the SimpleCAPTCHA library. As we are

supplying labeled data, we have stored the name of the image as the output of that particular CAPTCHA. This acts as a label for our models.

Before supplying images for training, we convert them to grayscale using a PIL library [18] function to reduce a 3 channel image to 1 channel using the following formula:

$$L = R * 299/1000 + G * 587/1000 + B * 114/1000$$

Accuracy of neural networks depends upon the training data. If we have only one kind of image in our training data, our model would not perform well with other data or real time data. So we randomized our training data by generating images from different sources and randomly sampling images to create batches for training. To ensure that the model does not overfit, we used a PHP script to create fixed length (5) CAPTCHA images and merged it with our existing dataset. It worked well, as mentioned in Chapter 7.

Chapter 7

Our Models

We created two kinds of models: one using LSTM and another using multiple Softmax layers. We trained on both fixed length and variable length CAPTCHAs.

Main blocks of our models

CNN:

We have two convolutional layers, each with 5*5 dimensional 32 filters. After each convolutional layer, we have a maxpool layer, which applies non-overlapping 2*2 matrix to get the max of neighboring pixel values. Input images are 2D of size (height:50, width:200), as we use only 1 channel.

First convolutional layer: Number of filters = 32
Size of a filter = 5*5
Stride = 1

First Maxpool layer: Pool size = 2*2
Stride = 1

Second convolutional layer: Number of filters: = 32
Size of a filter = 5*5
Stride = 1

Second Maxpool layer: Pool size = 2*2
Stride = 1

Dense layer : No of units : 256
In dense layer, we use ReLU as an activation function.

Dropout layer : Probability 0.5, which means a unit in the dense layer is considered in a network at the time of training with a probability of 0.5.

RNNs:

We used LSTM for RNNs of size 256.

Different models

1) **Fixed length with multiple Softmax**: In this model, we used image CAPTCHAs of fixed length (5 in our models) to train the model using a fixed number (again, 5 in our case) of Softmax layers each predicting one character. Figure 15 illustrates the architecture used. We trained this model using simple as well as complex dataset images. The softmax layers at the top shared weights.

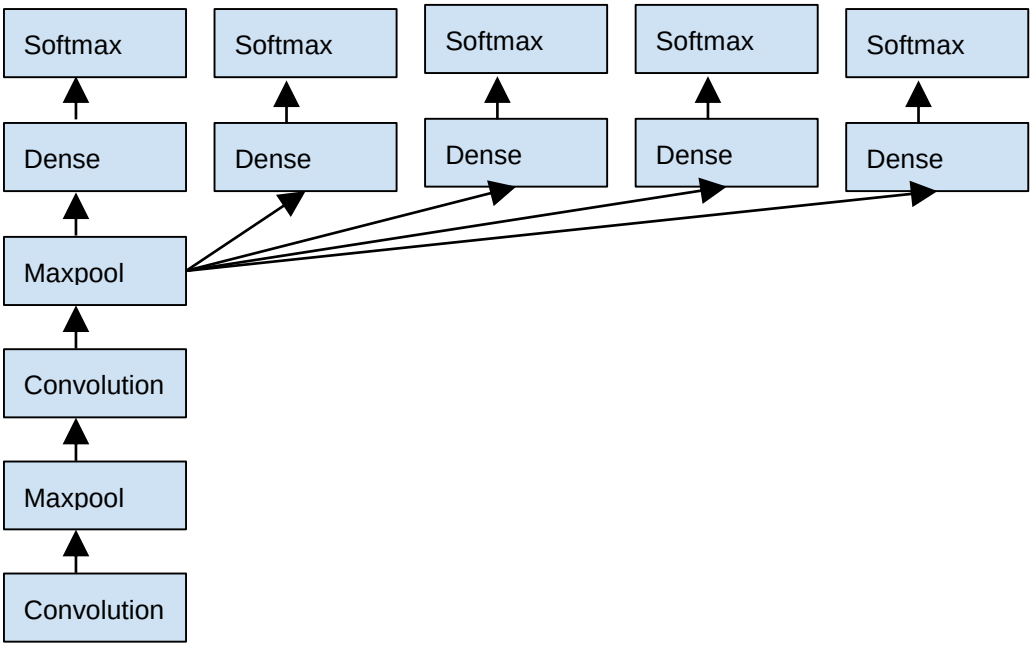


Figure 15: CNN with multiple oftmax

2) **Variable length:** For variable length, we used CNN with RNN layer, as shown in Figure 15.

In Figure 16, 3 RNN steps are taken for the sake of brevity. It should be equal to the maximum number of characters possible in a CAPTCHA image. For example, in our model, we have a maximum of 7. Figure 16 shows 3. Again, Softmax layers at the top share weights. We trained this model on the images of fixed length as well as on the images of variable length. Results are presented in the next section.

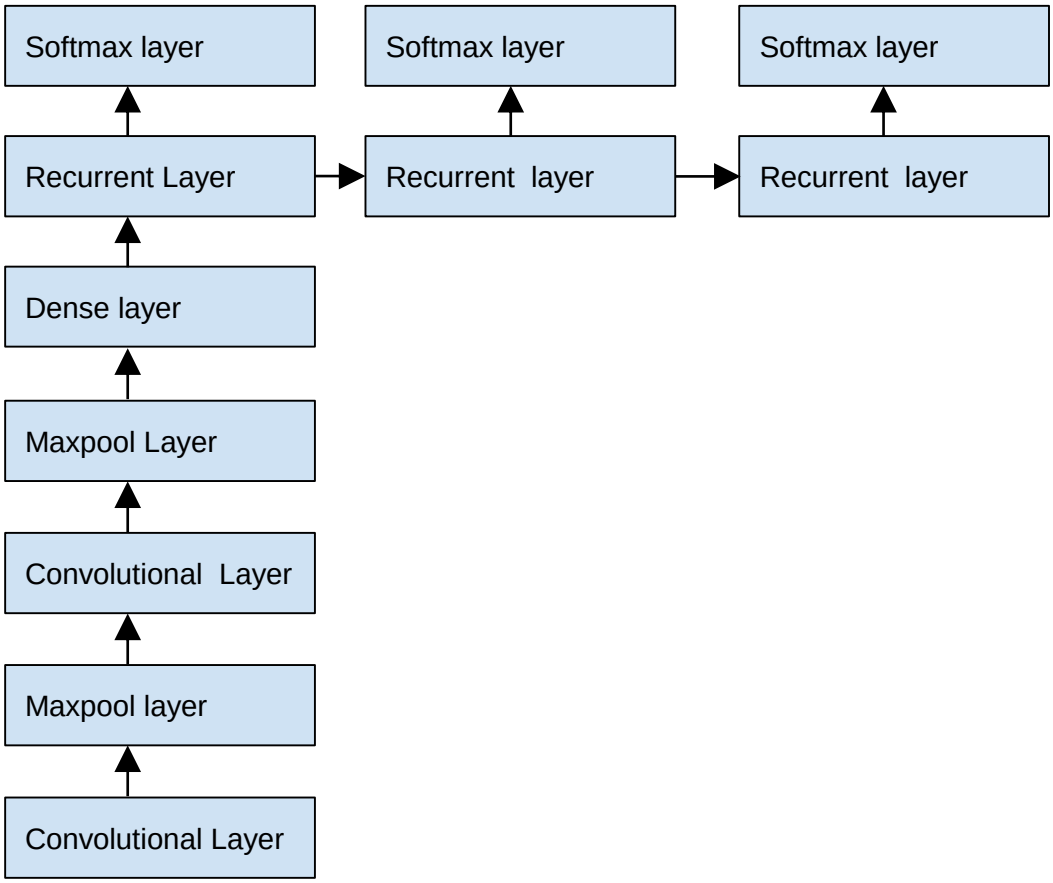


Figure 16: RNN architecture.

We assigned special label (“unk”), short for “unknown”, for termination. We mask the input. If we get one “unk”, we do not go any further, because it is faster and will take fewer steps to train.

Also, during training, exploding gradients were pulling the model too far in different directions. We tried clipping the gradients, which helped reduce this issue by dampening the exploding gradients.

3) **Fixed length LSTMs:**

This model is the same as the variable length model, but in this model, we fixed the number of RNN steps. In this case, we did not pass any “unk” label. The architecture is same as that of Figure 16. We trained this model on simple CAPTCHAs as well as on complex datasets.

The whole project is written in Python using publicly available Python libraries, as discussed in Chapter 5. The project is available on GitHub at https://github.com/bgeetika/CAPTCHA-Decoder.

Chapter 8

EXPERIMENTS AND RESULTS

After generating images, as described in Chapter 5, we began training. To train a model and make it more efficient, we packaged 20,000 images in one numpy file. This was easier to load than loading one file at a time. We specified the batch size with which we want to train the model. By default, there are 500 images.

For testing and validation, we put 20,000 images each.

The total number of epochs used in training is 20.

We have added various command line flags that help to customize our training. Below is the output of a code snippet's output showing the positional arguments used:

TrainingDirc:	Path to training data directory
ValidateDirc:	Path to validation data file
TestDirc:	Path to test data file
ModelParamsFile:	Path to the directory where params is to be stored, followed by prefix for the parameter file
optional arguments: -h, --help	Show this help message and exit
-maxsoft, --maxsoft	Provide this argument if you want to run multiple softmax
-bidirec, --bidirec	Provide this argument in order to run bidirectional
-hiddenlayers, --hiddenlayers	Number of hidden layers in the network
-learningrate, --learningrate	Learning rate

-batchsize, --batchsize	Training batch size
-testsize, --testsize	Test batch size
-includeCapital, --includeCapital	Include capital letters or not
-length, --length	Length of the characters
-rescale_in_preprocessing, --rescale	Rescale_in_preprocessing
--use_mask_input	Use_mask_input
-lstm_layer_units, --lstm_layer_units	No of units in lstm layer

Table 1: Command line arguments supplied

We have used a Google compute engine machine that had 8 cores. Because of this, we were able to train multiple models at the same time.

Different model and the individual character accuracy achieved so far:

Type of model	**Individual Character Accuracy**
LSTM fixed length (simple Dataset)	100%
LSTM fixed length (Complex dataset)	98.48%
Multiple Softmax fixed length(Simple dataset)	99.8%
Multiple Softmax fixed length(Complex dataset)	98.96%
LSTM only first character recognition	99.8%
LSTM with real data	99.2%
LSTM variable length with fixed length data	99.5%
LSTM variable length with variable length data	97.31%

Table 2: Individual character accuracy for different models

Type of model	Individual Sequence Accuracy
LSTM fixed length (simple Dataset)	99.8%
LSTM fixed length (Complex dataset)	91%
Multiple Softmax fixed length(Simple dataset)	99%
Multiple Softmax fixed length(Complex dataset)	96%
LSTM with real data	97%
LSTM variable length with fixed length data	98%
LSTM variable length with variable length data	81%

Figure 17: Training loss vs. number of images trained on

Figure 17 shows how cross entropy loss decreased during the training. This particular graph is taken from model that uses real data as well as generated data.

Number of images trained and training accuracy:

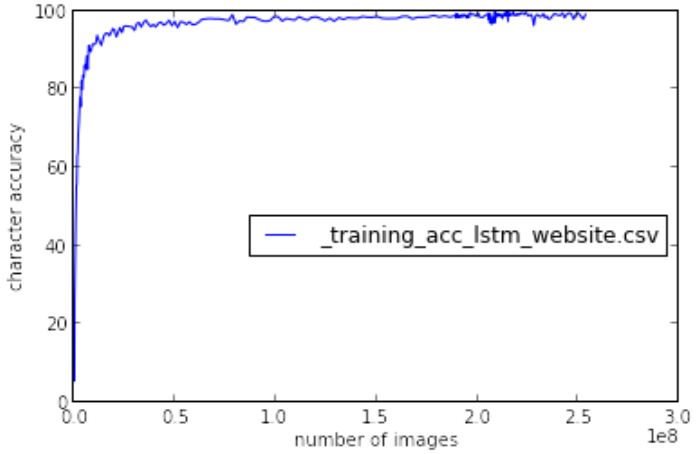


Figure 18: Training individual character accuracy vs. number of images trained on

The above graph shows how individual character accuracy improved during training. Since getting a character correct is easier than getting the whole sequence right, the curve in Figure 18 is steep as compared to the curve in Figure 19, which plots sequence training accuracy

.

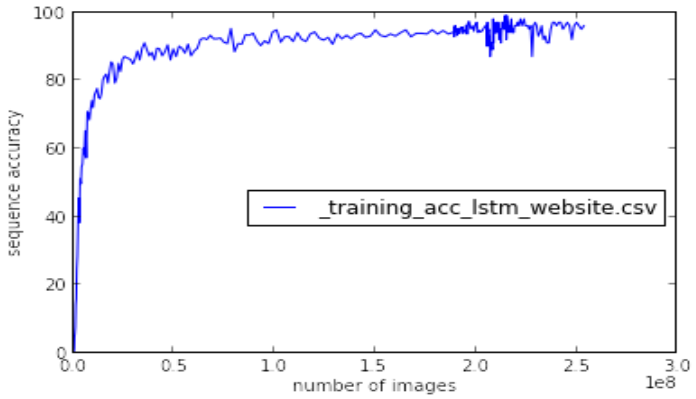


Table 3: Sequence accuracy for different models

Table 1 and Table 2 show the accuracy of models achieved so far. LSTMs on fixed size of CAPTCHA length have provided good results, as we anticipated. Model with multiple Maxsoft layers also did quite well, with 99% and 96% accuracy on simple and complex images, respectively. We have also tried merging real data that we obtained from a website. In this case as well, LSTM has provided 96.2% accuracy, with 99% individual character accuracy.

Graphs Generated:

Training loss vs. number of images trained on

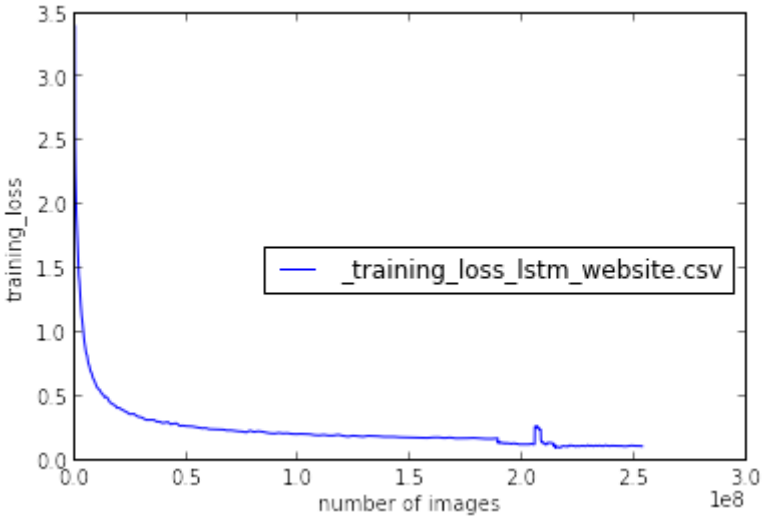


Figure 19: Training sequence accuracy vs. number of images trained on

Number of images trained on and training accuracy:

Training accuracy is generally a bit different then testing accuracy. The curves for testing accuracy and training accuracy look similar. Figure 20 shows the individual character accuracy and Figure 21 shows the sequence accuracy. Both of these curves are also generated for models trained on website data and self-generated data.

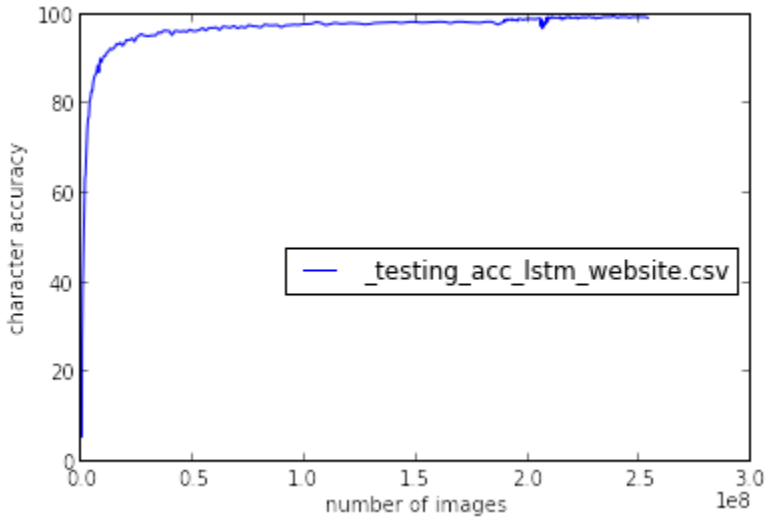


Figure 20: Testing individual character accuracy vs. number of images trained on

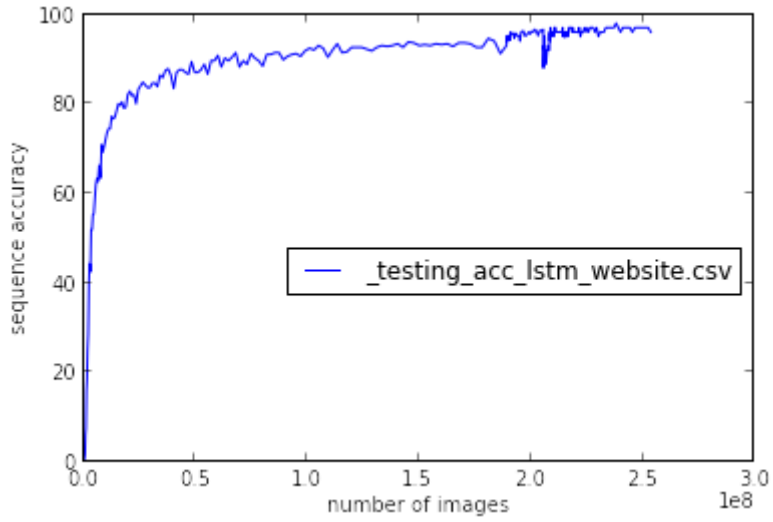


Figure 21: Testing sequence accuracy vs. number of images trained on

Chapter 9

CONCLUSION

In this project, we have tried to decode an image-based CAPTCHA using deep neural networks. We have used convolutional neural networks and recurrent neural networks instead of using the conventional approach of first cleaning a CAPTCHA image, segmenting the image, and recognizing the individual characters. For machine learning problems, we need a large amount of data, so we have generated a dataset of **13 Million** image-based CAPTCHAs. The programs generating this dataset will be publicly available so that they can be used by other people in their research. We have exploited the capabilities of CNNs to work on the images and RNNs to work with sequences. The model was trained on both simple CAPTCHAs and complex CAPTCHAs. The accuracy achieved on fixed-length CAPTCHA was very impressive (99.8% for simple images and 96% for complex images). We tried both fixed length and variable length CAPTCHAs. Both give 99and 55 % accuracy respectively. Gradient clipping helped speed up the training of LSTMs, which were otherwise very slow. Also, masking inputs helped learn models faster.

It is clear that the more kinds of CAPTCHAs we include in our training set, the more robust our model will become. We have tried to demonstrate this by using a real dataset in our training set, and were able to achieve 99% accuracy.

While it is still in early stage, our model performs better than previous work that relies on manually-generated segmentation oriented models. For example, our model beats the accuracy of [2] model. More importantly, our approach is not impacted by the fragility inherent in attacks while doing manual cleaning or segmenting of an image. This leads to our most interesting finding: neural networks can learn to perform complicated tasks such as the simultaneous localization and segmentation of ordered sequences of objects.

In the end, we are able to provide end-to-end neural network system. Given an image, we will be able to decode the CAPTCHA in that image.

Chapter 10

REFERENCES

[1] Moni Naor. Verification of a human in the loop or Identification via the Turing Test. Unpublished Manuscript, 1997.

[2] Greg Mori and Jitendra Malik. Recognising Objects in Adversarial Clutter: Breaking a Visual CAPTCHA, IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03), Vol 1, June 2003, pp.134-141.

[3] Kumar Chellapilla, Patrice Y. Simard Using Machine Learning to Break VisualHuman Interaction Proofs (HIPs) Microsoft Research, one microsoft way, WA. 2005

[4] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks,14 Apr 2014.

[5] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan, Show and Tell: A Neural Image Caption Generator, 20 Apr 2015

[6] F. Azam and H. F. VanLandingham. An efficient dynamic system identification technique using modular neural networks. Artificial Neural Networks for Intelligent Engineering, 7:225-230,1997.

[7] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Long short term memory,Technische Univ. Munich, 1991.

[8] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.

[9] H. Jaeger. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science, 2004.

[10] W. Maass, T. Natschläger, and H. Markram. A fresh look at real-time computation in generic recurrent neural circuits. Technical report, Institute for Theoretical Computer Science, TU Graz, 2002.

[11] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.

[12] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, Y. Bengio - Theano: a CPU and GPU Math Expression Compiler.

[13] Torch7. http://torch.ch/ Accessed in Oct 2015.

[14] Caffe. http://caffe.berkeleyvision.org/.Accessed in Oct 2015.

[15] Theano. http://deeplearning.net/software/theano/.Accessed in Oct 2015.

[16] Lasagne. http://lasagne.readthedocs.org/en/latest/user/installation.htm.Accessed in Oct 2015.

[17] SimpleCAPTCHA.Simplehttp://simpleCAPTCHA.sourceforge.net. Accessed in Oct 2015

[18] Python Imaging Library. http://www.pythonware.com/products/pil/. Accessed in Oct 2015

[19] Receptive field of neurons in LeNet. http://stats.stackexchange.com/questions/142606/receptive-field-of-neurons-in-lenet. Accessed in Oct 2015

[20] ReLU Rectifier (neural networks). https://en.wikipedia.org/wiki/Rectifier_(neural_networks). Accessed in Oct 2015

[21] Cross entropy. https://en.wikipedia.org/wiki/Cross_entropy. Accessed in Oct 2015

[22] LSTMS. http://deeplearning.net/tutorial/lstm.html. Accessed in Oct 2015.

[23] Tianhui Cai. CAPTCHA Solving With Neural Networks.2007-2008

[24] Artificial neural networks are changing the world. What are they?. http://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they. Accessed in Oct 2015.

[25] Minsky, M. S. Papert. An Introduction to Computational Geometry. MIT Press, 1969.

[26] Werbos, P.J. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, 1975

[27] Yaniv Taigman,Ming Yang, Marc'Aurelio Ranzato, Lior Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification, June 24, 2014.

[28] David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams. Learning representations by back-propagating errors, July 31, 1986.

[29] CAPTCHA. https://en.wikipedia.org/wiki/CAPTCHA. Accessed in Oct 2015.

[30] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. Ng. Deep Speech: Scaling up end-to-end speech recognition, 17 Dec 2014

[31] Ilya Sutskever , James Martens , George Dahl , Geoffrey Hinton. On the importance of initialization and momentum in deep learning.

[32] Tomas Mikolov Et al. Distributed Representations of Words and Phrases and their Compositionality,Oct 16, 2013.

[33] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Networks. Sep 10, 2014.

[34] Y.LeCun, B. Boser, J. S. Denke, D. Henderson, R. E. Howard, W. Hubbard and L.D. jackel. Handwritten digit recognition with a back-propagation network. 1989.

[35] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In

M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[36] http://www.lifehack.org/articles/technology/fed-with-distorted-texts-for-verification-google-offering-new-captcha.html . Accessed in Oct, 2015.

[37] http://www.neuralpower.com/technology.htm. Accessed in Oct, 2015.

[38] http://www.cheshireeng.com/Neuralyst/nnbg.htm. Accessed in Oct, 2015.