deCAPTCHA

Identifying Captcha text using Object Detection (ML)

Ulaganathan Natarajan
PACE, Cognizant Technology Solutions

deCAPTCHA: A Captcha solver using object detection (Machine Learning)

ULAGANATHAN NATARAJAN[1] (Performance Analyst)
[1]PACE-Cognizant technology solutions, 2nd Cross Street, Madras Export Processing Zone, Chennai, Tamil Nadu 600045, INDIA
Corresponding author: Sivakumar Kalyanaraman (Sivakumar.kalyanaraman@cognizant.com)

ABSTRACT Automation and performance testing are really important processes to measure application's performance and behaviour. In the process of performance testing or automation the common stopping stone is CAPTCHA validation. CAPTCHA's are meant to add security to your applications, so that only humans can access and will not give chances to Bot, Machine learning algorithms or attack programs to void your application. When your application needs to undergo automation or performance testing, there arises a case where you want to measure your CAPTCHA system performance or automate particular transaction that involves CAPTCHA. deCAPTCHA will provide solution for such problems and can solve almost all kinds of alphanumeric CAPTCHA's.

INDEX TERMS Machine learning, Learning systems, Supervised learning, Prediction methods, Artificial Intelligence, Performance Engineering, Performance Testing, Automation, Computer applications, scientific computing, Open source software.

I. INTRODUCTION

CAPTCHA's are widely used and considered a primary security check from the end user. The purpose is to verify that the requests or operations are made by humans and not by attack programs or bots. CAPTCHAs are generated automatically by CAPTCHA programs with random alphanumeric characters that are case sensitive. This technology is used mostly to block spammers and bots from signing in to your application, make use of forums, blogs and websites in an attacking manner. It's used to ensure that attack programs like OCRA (Optical character recognition attack) producing unwanted load on the system do not bring it down. Though using CAPTCHA helps protect the application from attackers, at times it does make it challenging for differently abled people. There are many vendors providing CAPTCHA services and very less number of applications using their own system to generate it, so you don't have the control over the CAPTCHA system even though it is implemented in your application.

The actual problem starts, when you try to automate some part of an application that has a CAPTCHA security check. It also becomes problem when you try to conduct a performance test on you application, since CAPTCHAs can't be automated by programs, you have to disable the security check and proceed further. There's will be a different performance output if you do this. Many vendors claim that their system can serve very well and will not affect the performance.

deCAPTCHA will provide solution to solve the CAPTCHAs so that the application transaction can be automated. It was developed by extending machine learning properties. Unlike other solvers, it treats each character as object, so it can dynamically learn about various types of CAPTCHA images. Object detection is a technique that works efficiently on distorted or typical CAPTCHA images to solve effectively, compared to Optical Character Recognition and Natural language processing.

II. COMPLEXITY ON IDENTIFYING CAPTCHA TEXT USING MACHINE LEARNING

CAPTCHA images are an open challenge to the AI world. It also offers well-defined challenges to the AI community, security researches and other malicious programmers to think work on advancing field of AI.

1. CHARACTER SEGMENTATION

Characters and digits are not well segmented in CAPTCHA images. This becomes problem in identifying an individual letter or digit. Most of the characters are overlapped with other thus posing a challenge to recognize characters individually.

2. CHARACTER SCALING

Character scaling is unknown here and that left machine learning and recognizing process into difficulty. We have no clue how big or small the characters are, so it's not known how big the segmentation box should be to identify it. Malformed digits is also major concern, for example the loops in digits like 6, 8 can be longer than usual.

3. CHARACTER ORIENTATION

Characters could be rotated at different angles that makes it very difficult to recognize in a standard way. Oriented characters require additional image processing like flipping, repositioning for further identification.

4. CHARACTER DISTORTION AND BACKGROUND NOISE

The major challenge is distorted characters, mostly the characters are merged or smudged with background and other characters. To increase the complexity the images may be generated with blur filters Background noises like lines over the text, image filed with dots, lines with horizontal and vertical all over the image and strong colours on specific spots. The above mentioned modules should be optimized individually. In reality the machine learning utility should handle all these modules to decode CAPTCHA text. It requires lot of image processing to remove the hurdles. We have to spend some time on image processing to optimize the image for learning and decoding.

III. ADVANTAGES OF OBJECT DETECTION OVER STANDARD OCR AND NLP IN IDENTIFYING CAPTCHA TEXT

Optical Character Recognition and Natural Language Processing are the different fields in AI and works systematically. These methodologies expect the input in a systematic way unlike object detection.

Any of these methods requires an input image that should be in a standard way, but in the world of CAPTCHAs it is not possible. When images are being processed, labelled and classified, the characters should be segmented, properly bound and clearly structured. Usually the images will be skewed, overlapped or unsegmented, thus resulting in false predictions and we may get incorrect output.

OCR, NLP and Object Detection works similarly in the following way,

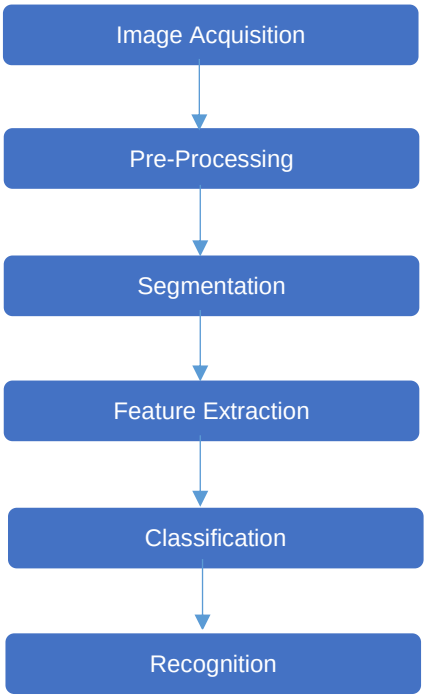

FIGURE 1. Generic process overview of OCR, NLP and CV.

The advantage in Object Detection is, we can train and identify any type of character that is skewed, smudged, broken or overlapped. The same way it is very hard to do it in OCR or NLP because it sticks to the standard way of character representation.

During the process of segmentation, extraction, classification and recognition Object Detection has upper hand compared to OCR and NLP. For example an image that contains character 'B' with a small break in the bottom, there is a high chance it can be identified as character 'R'. The damages in the characters will result in false predictions. Sometimes congested characters will lead segmentation, feature extraction algorithm to ignore some portions for processing. In Object Detection everything can be converted to an object and used to teach the machine. Even damaged or broken characters can be grouped with proper characters and be easily identified after the training.

[1] http://dlib.net/

IV. DLIB's OBJECT DETECTOR

1. DLIB INTRODUCTION

deCAPTCHA was developed using Davis E. King's machine learning library DLIB. DLIB is a modern C++ toolkit with machine learning algorithms primarily focused on the area of face and object detection. It provides wide variety of Image Processing, Optimization, Meta Programming, Computing, File Operation, UI Containers, Graphing, Networking and other miscellaneous API's.

2. FELZENSZWALB's HOG (FHOG) IMPLEMENTATION

DLIB Object Detection is designed to detect things like face, road signs, real world objects and semi-rigid objects. This object detector is a tool for learning to detect objects in images. The training process produces an object detector SVM (Support Vector Machines) which can be used to predict the locations of objects in new images. The detector learns the parameter vector by formulating the problem as a structural SVM problem.

FHOG is a histogram-of-oriented-gradient (HOG) based object detector. It's easy, super-fast and can be operated with minimum number of training images to produce best results in prediction. The HOG technique is one of the best ways to eliminate complexity in object detection. In the below figure, it focuses only on the abstract representation of the object.

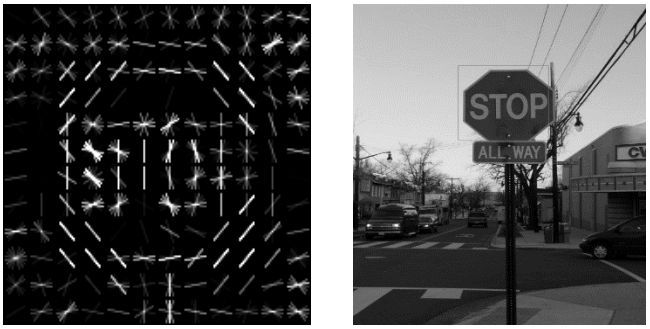

FIGURE 2. Example FHOG transformation of STOP sign, in the original image the specific part has been highlighted.

DLIB Object Detector uses the FHOG (Felzenszwalb's HOG) feature extractor. It can extract 31 dimensional version of HOG feature from an input image. The given image will be broken into cells, and within cells we compute 31 dimensional FHOG vector. The vector describes the gradient structure of a particular cell.

3. USE OF MAX-MARGIN OBJECT DETECTION (MMOD)

DLIB Object Detector internally uses the Max-Margin Object Detection technique, a technique to detect objects in images without performing any sub-sampling, but optimizing over all sub windows. MMOD can be used to improve any object detection method which is linear in the learned parameters such as HOG or bag-of-visual-word models. The MMOD approach proves it can efficiently compute and gains substantial performance. Additionally this proved that within few samples it delivered excellent detection results with less false predictions. The optimization runs until the potential improvement in average loss per training example is less than 0.15

[2] https://github.com/joaofaro/FHOG

[3] https://arxiv.org/pdf/1502.00046.pdf (MMOD)

4. USE OF STRUCTURAL SVM TRAINING ALGORITHM

The detector uses structural SVM based training algorithm to train about the objects. This HOG trainer enables it to train on all the sub windows in every image and we don't have to do any complex subsampling. This trainer can generate sufficient set of vectors within few samples, whereas many detectors require lots of images to train on. During training, this object basically runs the object detector on each image, over and over again. To speed this up, it is possible to cache the results of these detector invocations. This algorithm internally uses the structural SVM object detection problem, it learns the parameter vector by formulating the problem as a structural SVM problem.

V. deCAPTCHA PROCESS OVERVIEW

deCAPTCHA is designed as a complete machine learning utility, which can handle all CAPTCHA solving related processes. The utility is packed with a CLI Interface, Trainer, Detector, and an Image Editing Tool. It is designed and developed for easy usage and the tasks can be achieved in very few steps.

1. LIFECYCLE

The deCAPTCHA life cycle begins with the process of raw image acquisition and ends with the solved CAPTCHA. The solving process includes all necessary activities in the line, like learning from failed scenario, image editing and processing.

2. TRAINING AND DETECTION

As mentioned above in DLIB Object Detector, it uses the SVM Structural Based Training algorithm and FHOG (by extending MMOD) technique to train and detect objects.

When the CAPTCHA identification fails the image will be redirected to the image editor to eliminate any unwanted pixels that may cause a setback for learner algorithm. It's necessary to clean the noise in the image instead of training the utility to adapt the noisy image. If you choose to adapt the noisy images, you require huge amounts of dataset to train, which is a tedious process.

In the detector the length of the CAPTCHA text can be configured. When the identified text length requirement doesn't meet, it is considered as a failure scenario. After the SVMs are generated by the trainer it will be serialized and saved in disk. During the process of detection, the saved SVM detectors will be iterated to find the best match in the image. Sometimes the detectors might not find the objects due to image resolutions. CAPTCHA images are naturally smaller in resolution, so it need to be up-sampled. In some cases the image needs to be flipped to right or left to detect the objects. All these options are provided and user can configure this to achieve best results.

In terms of performance, an average image takes 4-6 seconds for training and detection with a single thread. The number of threads is configurable to achieve maximum performance of the utility.

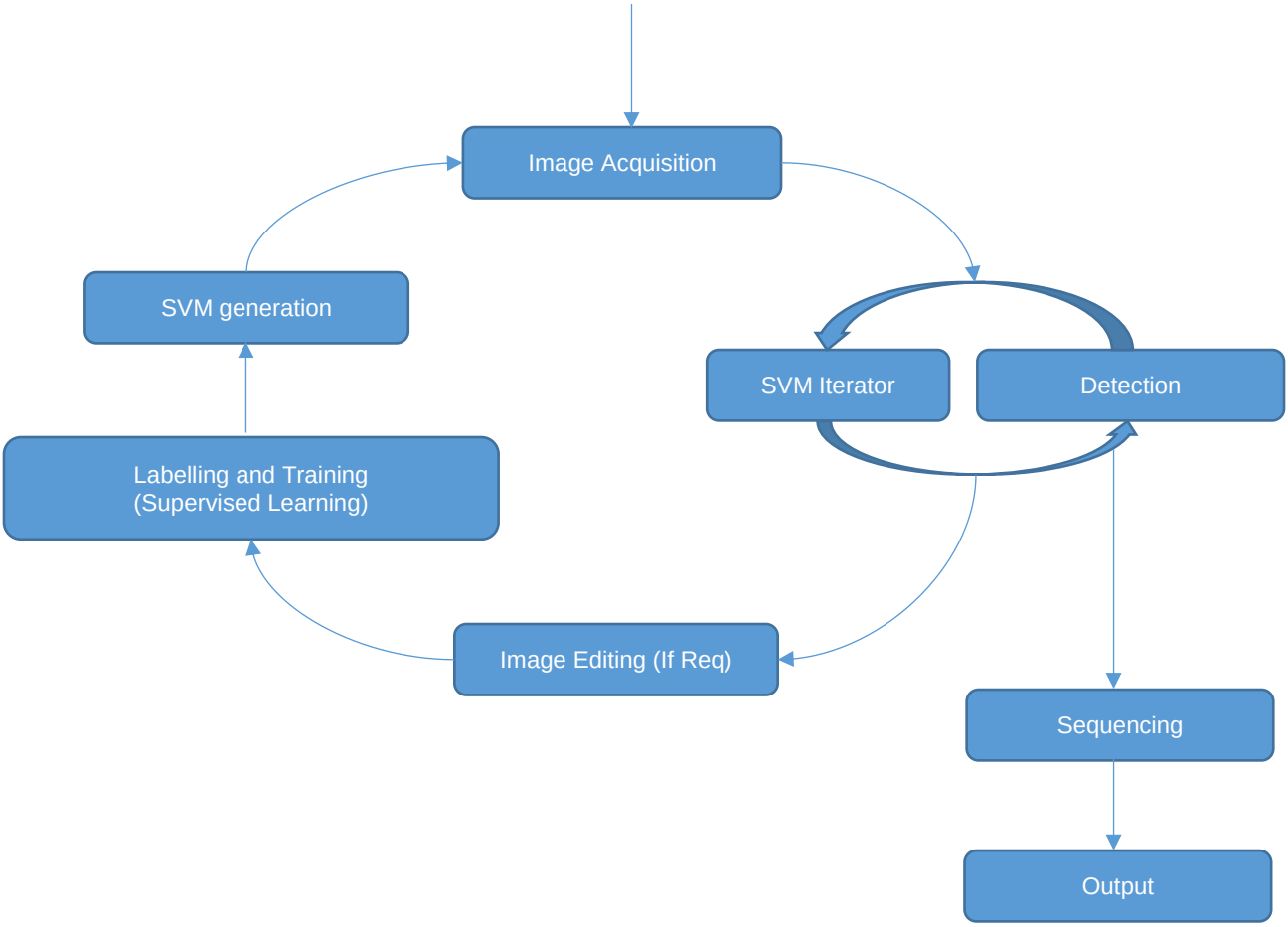

FIGURE 3. Life-Cycle of deCAPTCHA

3. IMAGE EDITING TOOL

The utility is packed with a minimal image editing tool that can be used to edit the CAPTCHA images for training. Generally CAPTCHAs have cross lines, dots, net lines and small circles to increase complexity. For training purpose this can be excluded to increase the chance of the objects to be detected. The problem in Object Detection is that it will not differentiate between an actual character and one with a cross line on it. If training is done with a CAPTCHA having a cross line, the training algorithm will assume the whole thing as an object. During the detection it expects the same kind of line on the character. The utility requires far less number of training samples if we avoid this. For this, a few images might need some tedious editing using the image editing tool.

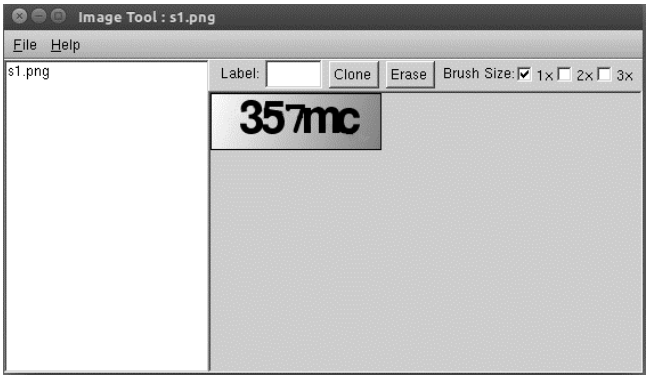

FIGURE 4. Image editing tool

3. SVM MANAGEMENT

SVM is an essential part of the any machine learning program because it holds all the training data. During the detection process the SVM will be de-serialized again to detect the objects. All the SVM files will be saved in the neural_data folder where the deCAPTCHA is running. Respective image files that has been copied to training_data folder can be used to regenerate the SVM. Each and every character will have its own SVM file for both lower and upper case. From the learnings of various images and various representation of characters, a single SVM file will be generated by the training algorithm.

VI. SEQUENCE PROBLEM IN IDENTIFIED CAPTCHA

After successful detection of CAPTCHA text, the challenging problem is to sequence it in the correct order. The detectors iterate randomly to detect objects. During the detection the objects can be found anywhere in the image. So we cannot place the CAPTCHA characters in the order of identification as it will be incorrect.

To overcome the problem I've used traditional bubble sort method to form the CAPTCHA text order like in the image. The identified objects will be returned with the x and y coordinates. To form the order it needs to be sorted based on the x position using the sort algorithm.

```
If ( Obj[i].x > Obj[j].x )
temp_letter = captcha_letters[i]
captcha_letters[i] = captcha_letters[j]
captcha_letters[j] = temp_letter
```

VII. EXPERIMENTS AND RESULTS

The results are very promising when compared to other CAPTCHA solvers. deCAPTCHA delivers best results with less number of training data sets.

USAGE

deCAPTCHA was designed as a command-line based utility, to ensure that it will run on servers that doesn't have UI support. When required it will try to use a UI for operations like labelling and editing image. The options are

- c <arg>	Set captcha size <arg>, to avoid the SVM iteration after the valid text identified (default: 5)
-d <arg>	Detect the captcha text from the image.
-h	Display this help message.
-i <arg>	Open the Image tool.
-n <arg>	Use <arg> threads for training (default: 10).
-s <arg>	Set size of the sliding window to about <arg> pixels in area (default: 80*80).
-t <arg>	Train an object detector from XML file and save the detector to disk with filename.
-u <arg>	Upsample each input image <arg> times. Each upsampling quadruples the number of pixels in the image (default: 0).

TABLE 1. Command line options

TESTING

The test has been conducted with an average CAPTCHA that is available on the internet and data set trained with only one image.

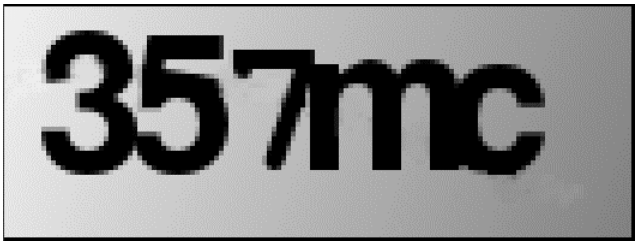

FIGURE 5. A plain CAPTCHA image that was used for training

The above image was the only image used to train the deCAPTCHA yet the results were very good. It is able to identify the same CAPTCHA with cross line or dots. It know the difference between the object and noisy pixels. The below detection is a success case since it identified all characters even though there's a difference in the character representations compared to the trained image in the above.

FIGURE 6. A success case, identified letters are highlighted by deCAPTCHA.

A failure case will look like below. After the detection is completed it will ask the user to teach it about the undetected characters. Users can label the letters through Image Tool that is packed with deCAPTCHA.

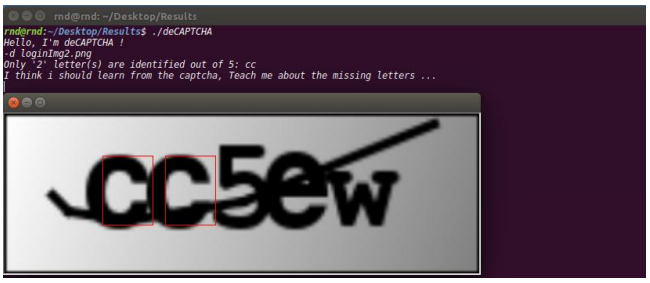

FIGURE 7. A failure case, deCAPTCHA only identified two letters.

The image will be forwarded to the image tool for training, since it failed to identify all the letters. The user can teach about each letter by labelling them in the Image tool. After labelling it will be forwarded to training algorithm, after which it will generate the SVM detectors for individual letters.

FIGURE 8. Teaching deCAPTCHA about missing letters by labelling letters.

During the training phase each letter is processed separately and trained in various aspects nearly 40 to 120 iterations. Based on the iteration's learning, the detector data will be serialized and saved.

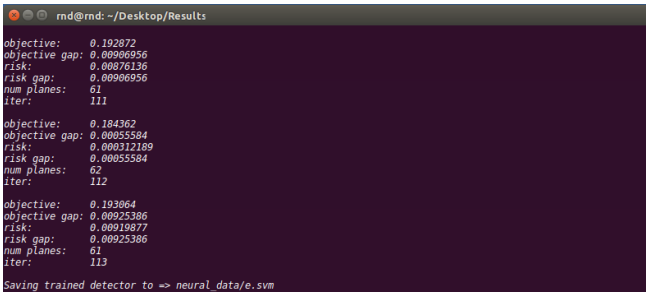

FIGURE 9. A screenshot taken when deCAPTCHA is learning about the letter 'e' with nearly 110 iterations and finally the detector saved in the local disk.

The same procedure will be followed for rest of the unidentified letters. deCAPTCHA is capable of giving better results with fewer image data sets when compared to other CAPTCHA solvers which requires more number of images, which is clearly an advantage.

deCAPTCHA vs GOOGLE CLOUD VISION

The same CAPTCHA images tested with deCAPTCHA has also been tested with Google's Cloud Vision (A Computer Vision Platform). Surprisingly the results are not good as deCAPTCHA. There might be many reasons, one among is that it's built for general purpose and capable of identifying all kinds of real world objects, whereas deCAPTCHA is focused on CAPTCHA related characters. Even though google vision uses object detection, it's primarily using OCR to identify text.

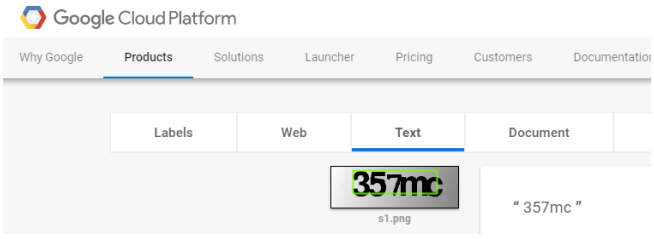

FIGURE 10. A success case, Cloud Vision identified all letters.

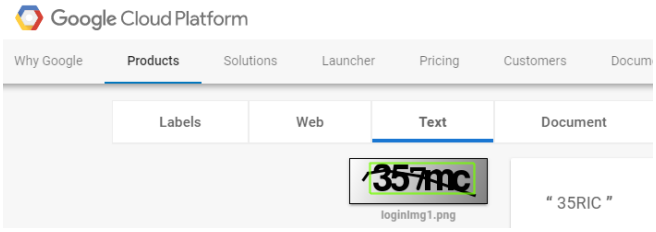

FIGURE 11. A failure case, Cloud Vision identified the letters as '35RIC' instead of '357mc'. The same was successfully identified with deCAPTCHA.

VIII. FUTURE WORKS

CONVOLUTIONAL NEURAL NETWORK INTEGRATION

CNN or Convolutional Neural Network known for its performance and lesser weight on the network layer, can be implemented in the existing system to increase the performance and improve decision making. The existing system doesn't have separate neural network, instead it independently uses SVM detectors to identify objects. Promisingly the existing system runs with performance and speed but this can be increased much more by implementing CNN. All the detectors can be connected and used to produce better results. Using the combined knowledge of connected SVMs, new objects can be identified correctly.

SOLVING IMAGE BASED CAPTCHAs

In AI, Object Detector is primarily used to detect all real world objects. deCAPTCHA uses object detection to identify the CAPTCHAs. It will definitely work on solving image based CAPTCHAs with some changes made to the existing system. On the other side this requires immense amount of images to train on to produce good results. In alphanumeric CAPTCHAs there might be different types of character representation of 26 characters and 10 digits, so the image requirement for this is very much less. Image CAPTCHA's are generated based on real world images and you might need millions of images to train on. Open-source communities like ImageNet provides the image sources to train the machines on real world images. So it is possible to solve image based CAPTCHAs in the future.

[4] https://cloud.google.com/vision/

[5] http://www.image-net.org/

VI. CONCLUSION

This paper presents a solution and a toolset for automating CAPTCHA in the process of Automation or Performance Testing. The utility has been designed for dynamic usage. It is able to solve all kind of alphanumeric CAPTCHAs with little effort of training. The requirement of very less data set makes the utility occupy very less disk space. The system can be hosted as a separate service to solve the CAPTCHAs or it can efficiently run on any less configured system without much special requirements other than standard libraries libpng, libjpeg and libX11.

REFERENCES

[1] Davis E King, ``Dlib: a modern C++ toolkit containing machine learning algorithms and tools, '' version 19.4. 7 Mar. 2017.

[2] Davis E King, ``Make your own Object Detector,'' Posted on 3 FEB 2014. [Online]. Available: http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html

[3] Davis E king. ``Max-Margin Object Detection,'' arXiv:1502.00046v1 [cs.CV] 31 Jan. 2015. [Online]. Available: https://arxiv.org/pdf/1502.00046.pdf

[4] Felzenszwalb's., ``Pattern Analysis and Machine Intelligence,'' *IEEE Transactions on 32.9 (2010): 1627-1645.*

[5] PACE, ``Machine learning for identifying captcha text,''. COGNIZANT, [Online]. Available: https://blogs.cognizant.com/PACE/2017/03/15/machine-learning-for-identifying-captcha-text/

[6] Fei Fei Li, ``How we teach computers to understand pictures,'' *TED, YOUTUBE*, 23 Mar 2015. [Online]. Available: https://www.youtube.com/watch?v=40riCqvRoMs

[7] Benny Thörnberg, ``Introductory lecture in Machine vision,'' *YOUTUBE*, 13 Sep 2013. [Online]. Available: https://www.youtube.com/watch?v=GrG9nGAQ6Jw

[8] [8] Benny Thörnberg, ``Lecture in Machine Vision: OCR - Original Character Recognition,'' *YOUTUBE*, 20 Jan 2014. [Online]. Available: https://www.youtube.com/watch?v=yPlwMUy2y2U