Project Name: Break Down the CAPTCHA Security with Machine Learning

<u>Author Details</u>

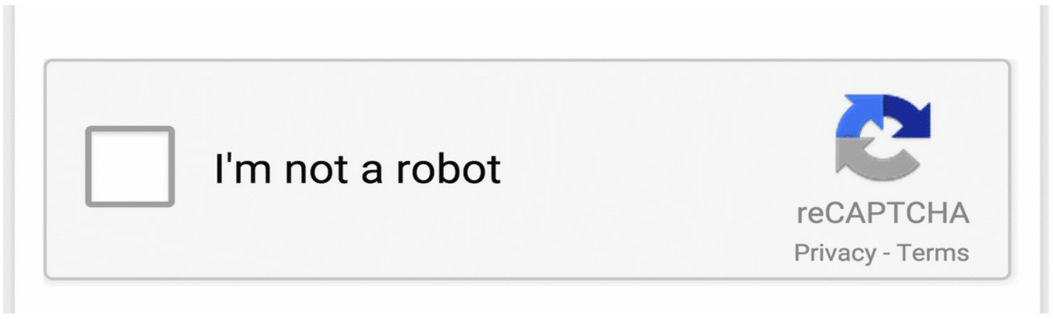
Name	Mail
Sayeed Hossain	me@sayeedhossain.com

Introduction

CAPTCHA is a computer program that distinguishes between humans and machines in order to prevent any form of illegal behaviour on websites. CAPTCHA is based on the notion that only a human could pass this test, while bots and automated scripts would fail. But, after introducing this "Break Down the CAPTCHA Security with Machine Learning" project, I believe such assumption will be incorrect.
This project can circumvent or break the CAPTCHA security. Humans are no longer required to solve the CAPTCHA. That can now be done by a trained machine.

The most significant question at this point is why we even need to break CAPTCHA. People employ automated CAPTCHA solving for a variety of reasons, some of which are unlawful and others which are legal. Spammers employ CAPTCHA solving to harvest user email addresses in order to send as many spam messages as possible. The legitimate examples are situations in which a new client or business partner has joined you and requires access to your Application Programming Interface (API), which is not yet available or cannot be provided owing to security concerns or potential abuse. As a result, the only option is to use automated scripts to get around CAPTCHA.
It will also be useful for penetration testers, military cyber teams, or any other ethical cyber security experts who are attempting to safeguard us from crooks or scammers.

Text-based CAPTCHA, image-based CAPTCHA, reCAPTCHA, and mathematical CAPTCHA are all examples of CAPTCHA. Solving one can be difficult at times, as the technology employed in CAPTCHA and reCAPTCHA is always improving, with new versions arriving at regular intervals.



Problem Domain

The problem domain of this project is Cyber Security, Penetration Testing, Vulnerability, Bug and Time Complexity.

Motivations

I am motivated to develop this type of system for mainly three reasons. The first reason is that I am a penetration tester and I need to solve millions of captchas for penetrating any system and the second reason is that it will be very helpful for the Military Cyber team or any ethical cyber security experts who are trying to protect us from criminals or scammers. And the last thing is that, there has a very important use case for this type of project. Actually we can say that is the legitimate examples for this type of works. The legitimate examples are situations in which a new client or business partner has joined you and requires access to your Application Programming Interface (API), which is not yet available or cannot be provided

owing to security concerns or potential abuse. As a result, the only option is to use automated scripts to get around CAPTCHA. So that's the another motivation for this work.

Aims/Objectives

There are mainly three objectives/aims to develop this type of system. The first one is that I am a penetration tester (as I said before) and that's why I need to solve millions of captchas for penetrating any system and obviously everything should be done automatically in the case of brute forcing. So I must need a system which automatically can solve all the captchas for me. Because it is quite impossible to solve millions of captchas for humans. And the second one is to help the Military Cyber team.
Because sometimes the Military Cyber team's need to access any criminal or scammer system by brute forcing attack and at that time they have to solve the captcha if there is any captcha system enabled. Since this is not possible for humans to solve millions of captchas so that's why this intends me to carry this out. And the last one is that there is a legitimate use case for this works.
Let me explain in details, Suppose there are a situations in which a new client or business partner has joined you and requires access to your Application Programming Interface (API), which is not yet available or cannot be provided owing to security concerns or potential abuse. As a result, the only option is to use automated scripts to get around CAPTCHA. So another objective or aim is to overcome that type of situation through this work.

Tools & Technologies

To develop this system, we need some TOOLS & TECHNOLOGIES. Some of them are mentioned below:

1. Visual Studio Code - Code Editing. Redefined.
3. Python (numpy, imutils, sklearn, tensorflow, keras etc...)
4. Driver for Graphics
5. Ubuntu
6. Linux Kernel
7. Terminal
8. Machine Learning with Env. SetUp
9. WordPress Plugin

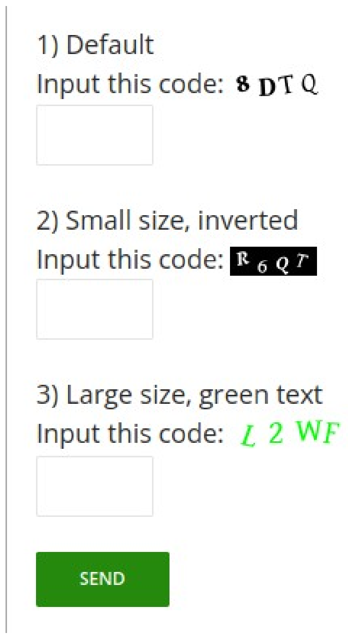
Break Down the CAPTCHA Security with Machine Learning Implementation

Find out Vulnerabilities

To train our system, we must first download hundreds of example photos and manually solve them to break it.
But what if we want to crack a CAPTCHA system that is open source and we have access to the source code?
I looked for "captcha" in the WordPress.org Plugin Registry.
And from there, I choose the greatest CAPTCHA plugin, which, best of all, comes with the source code!
This should be rather simple to crack because we'll have the source code that generates the CAPTCHAs.

Figure Out the Challenges

Let's see what kinds of graphics Really Simple CAPTCHA generates to come up with a strategy. This is what we see on the demo site:


1) Default

Input this code: 8 DTQ

2) Small size, inverted

Input this code: R6QT

3) Large size, green text

Input this code: L 2 WF

The CAPTCHA images appear to be four letters long. Let's look at the PHP source code to be sure:

```
public function __construct() {
                    $this->chars = 'ABCDEFGHJKLMNPQRSTUVWXYZ23456789';
                    $this->char_length = 4;
                    $this->fonts = array(
                              dirname( __FILE__ ) . '/gentium/GenBkBasR.ttf',
                              dirname( __FILE__ ) . '/gentium/GenBkBasI.ttf',
                              dirname( __FILE__ ) . '/gentium/GenBkBasBI.ttf',
                              dirname( __FILE__ ) . '/gentium/GenBkBasB.ttf',
                    );
```

OH! really it’s great. It uses a random combination of four distinct typefaces to generate four-letter CAPTCHAs. To avoid user confusion, it never utilizes the letters "O" or "I" in the codes. That leaves us with a total of 32 letters and digits to remember.

Toolset

Before we diving into more details, let’s explain about the tools and technologies that we’ll use to solve this problem:

Python 3
Python is an enjoyable programming language with excellent machine learning and computer vision libraries.

OpenCV
OpenCV is a prominent computer vision and image processing toolkit. The CAPTCHA images will be

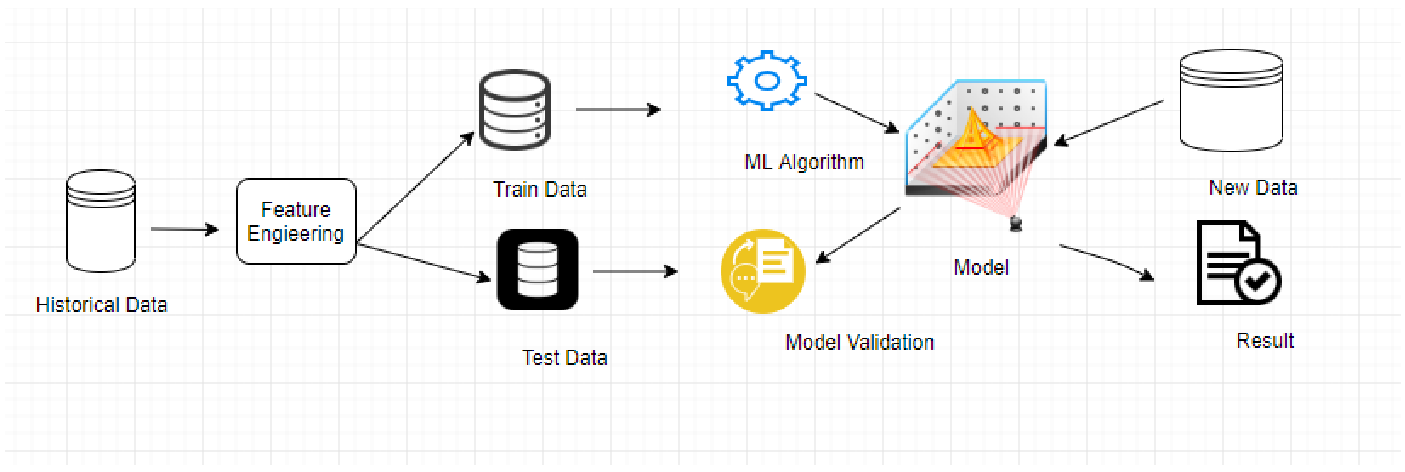
processed with OpenCV. We can use it directly from Python because it offers a Python API.

Keras
Keras is a Python-based deep learning framework. Deep neural networks can be defined, trained, and used with minimum coding.

TensorFlow
TensorFlow is Google's machine learning library. We'll be writing code in Keras, but the neural network logic isn't implemented by Keras. Instead, it does the heavy lifting behind the scenes with Google's TensorFlow library.

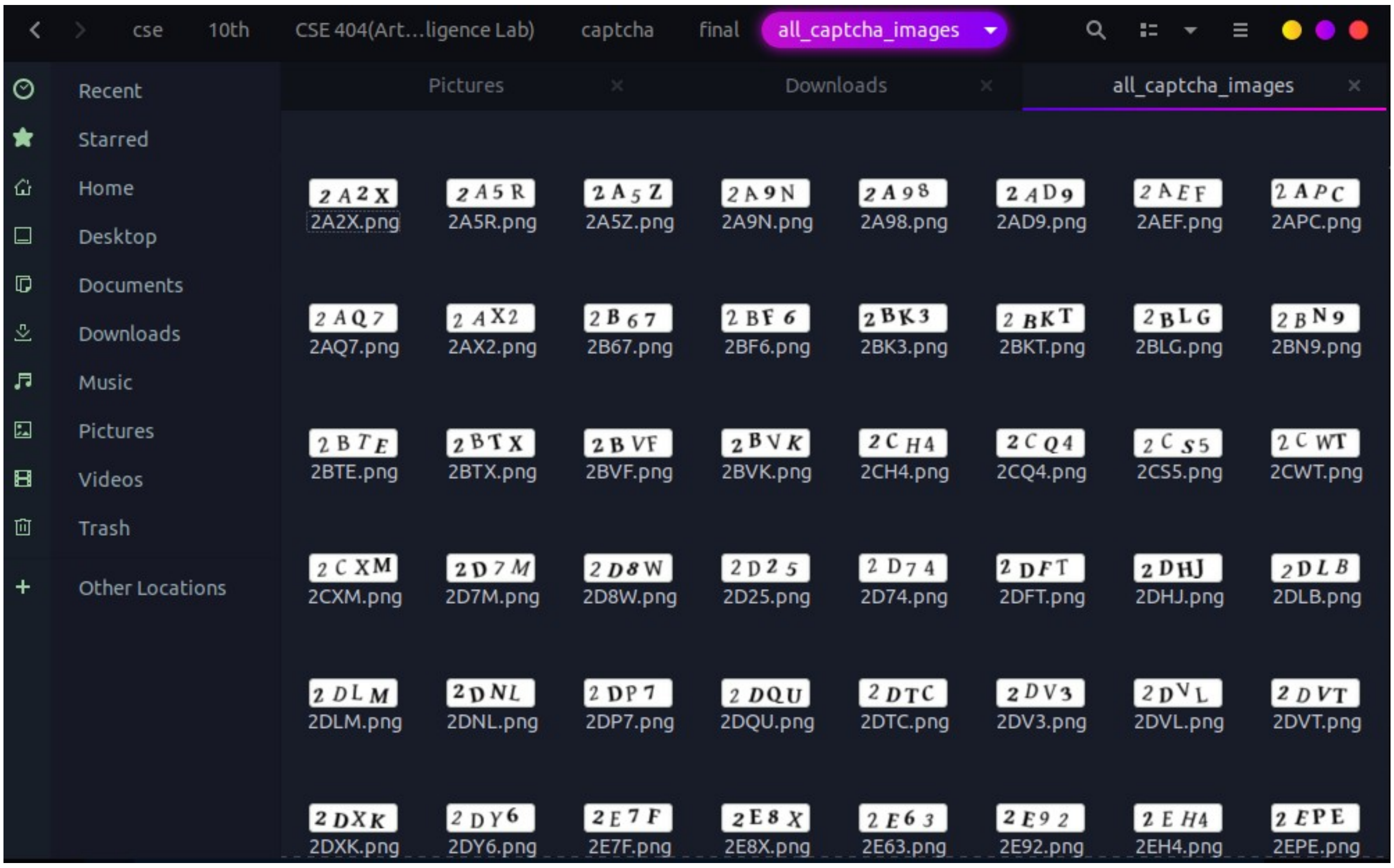
Creating Dataset

We require training data to train any machine learning system. We need training data that looks like this to break a CAPTCHA system:



We may edit the WordPress plug-source in's code to save almost 10,000 CAPTCHA images along with the required response for each image now that we have the source code.

I had a folder with training data — almost 10,000 PNG files with the correct answer for each as the filename — after a few minutes of working on the code and adding a basic 'for' loop.



Simplifying the Problem

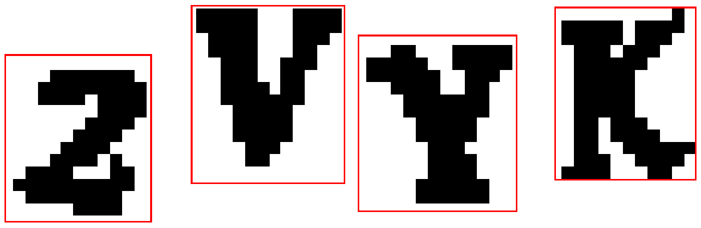
We could utilize our training data to train a neural network directly now that we have it.

This strategy might work with enough training data, but we can make the problem a lot easier to tackle. The less training data and computational power we need to solve a problem, the easier it is.
Fortunately, the CAPTCHA images are always only four letters long. We only need to train the neural network to detect a single letter at a time if we can break the image apart so that each letter is a different image.

I don't have enough time to manually split 10,000 training images into distinct images in Photoshop. That would take days, and I barely have ten minutes to spare. We can't merely divide the photos into four equal-sized parts since the CAPTCHA prevents us from doing so by randomly placing the letters in different horizontal spots.
Fortunately, we can still automate this process. We frequently need to recognize "blobs" of pixels with the same hue in image processing. Contours are the lines that surround such continuous pixels blobs. The findContours() function in OpenCV can be used to detect these continuous regions.

So let's start with a CAPTCHA image in its basic form:

The image will then be converted to pure black and white (a process known as thresholding) so that the continuous portions can be easily identified.
We'll then use OpenCV's findContours() method to find the regions of the image that have continuous blobs of the same color.
After that, it's only a matter of saving each section as its own image file. We can utilize this knowledge to label the letters as we save them because each image should have four letters from left to right. We should store each image letter with the right letter name as long as we save them in that order.

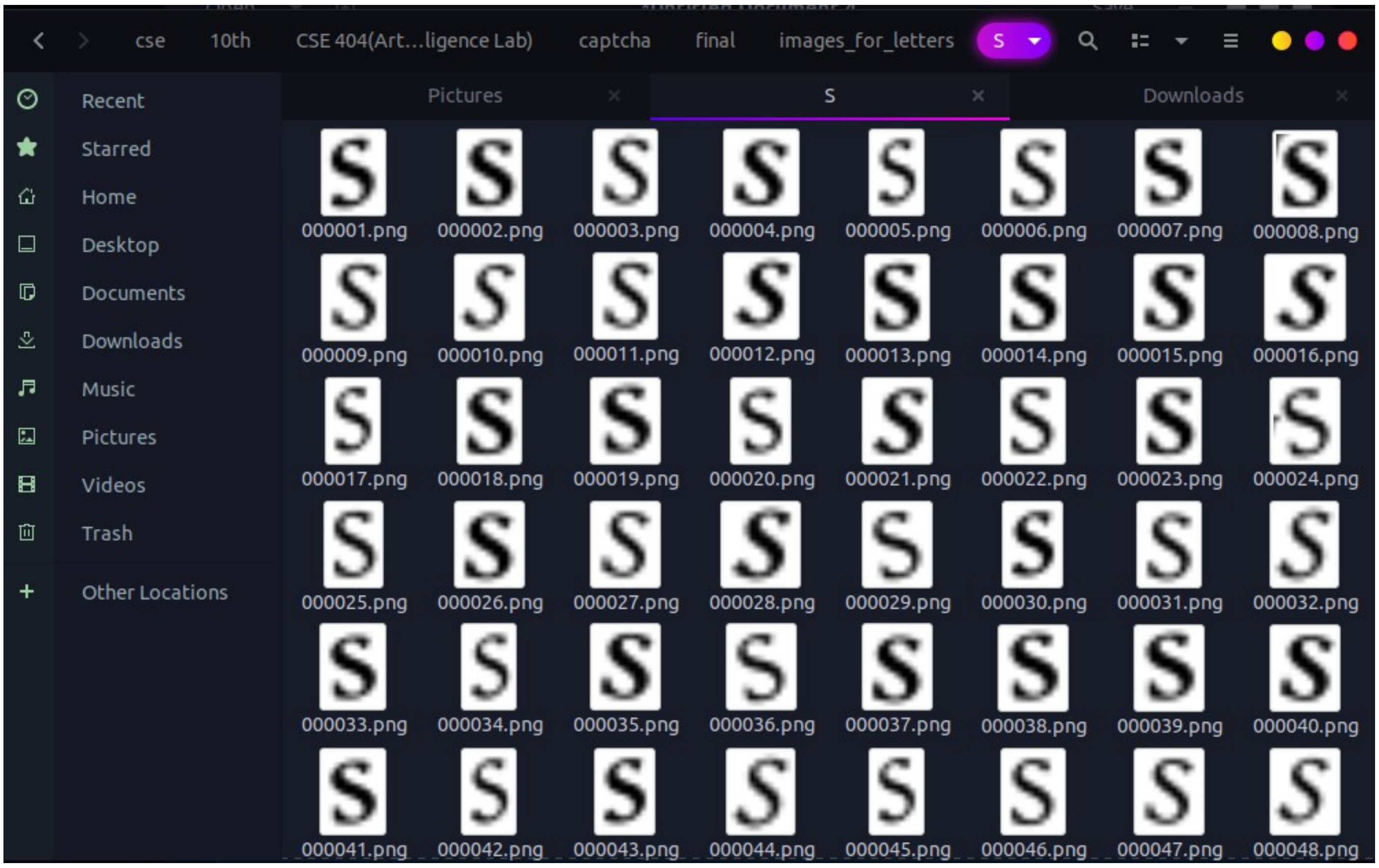


However, there is an intriguing dilemma. CAPTCHAs with overlapping letters appear on occasion.
As a result, we'll end up extracting regions that combine two letters into a single zone.
We'll end up with bad training data if we don't address this issue. We must correct this so that the system does not mistakenly recognize those two squashed-together letters as one.

A simple hack here is to argue that if a single contour region is much broader than tall, we're presumably dealing with two letters squished together. In that instance, we may simply break the conjoined letter in half and treat it as two distinct letters.

Now that we know how to extract individual letters, let's apply it to all of our CAPTCHA images. The objective is to collect as many different permutations of each letter as possible. To keep everything organized, we can save each letter in its own folder.

After I extracted all of the letters, this is what my "S" folder looked like.


cse
10th
CSE 404(Art...ligence Lab)
captcha
final
images_for_letters
S
Pictures
S
Downloads
Recent
Starred
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
000001.png 000002.png 000003.png 000004.png 000005.png 000006.png 000007.png 000008.png
000009.png 000010.png 000011.png 000012.png 000013.png 000014.png 000015.png 000016.png
000017.png 000018.png 000019.png 000020.png 000021.png 000022.png 000023.png 000024.png
000025.png 000026.png 000027.png 000028.png 000029.png 000030.png 000031.png 000032.png
000033.png 000034.png 000035.png 000036.png 000037.png 000038.png 000039.png 000040.png
000041.png 000042.png 000043.png 000044.png 000045.png 000046.png 000047.png 000048.png

Building and Training the Neural Network

We don't require a particularly complicated neural network architecture because we only need to distinguish images of single letters and numerals. Letter recognition is a lot easier than recognizing complex imagery like photographs of animals and dogs.

With two convolutional layers and two fully-connected layers, we'll employ a simple convolutional neural network design.

Using Keras, you can define this neural network design in just a few lines of code:

```
model = Sequential()
model.add(Conv2D(20, (5, 5), padding="same", input_shape=(20, 20, 1), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(50, (5, 5), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Flatten())
model.add(Dense(500, activation="relu"))
model.add(Dense(32, activation="softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Then after that, we can easily train them by using the following code:

```
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), batch_size=32, epochs=10, verbose=1)
```

We got around 100% accuracy after ten cycles through the training data set. So that’s really Amazing !!!

Environment Setup for Simulation

To start the setup, first of all we need the following installed:

1. Python
2. OpenCV
3.numpy
4.imutils
5.sklearn
6.tensorflow
7.keras

Also for easy setup. After installing python. Just type the command "pip3 install -r requirements.txt" to install all the necessary module/packages at a time. Because I listed all the necessary module/packages into the "requirements.txt" file.

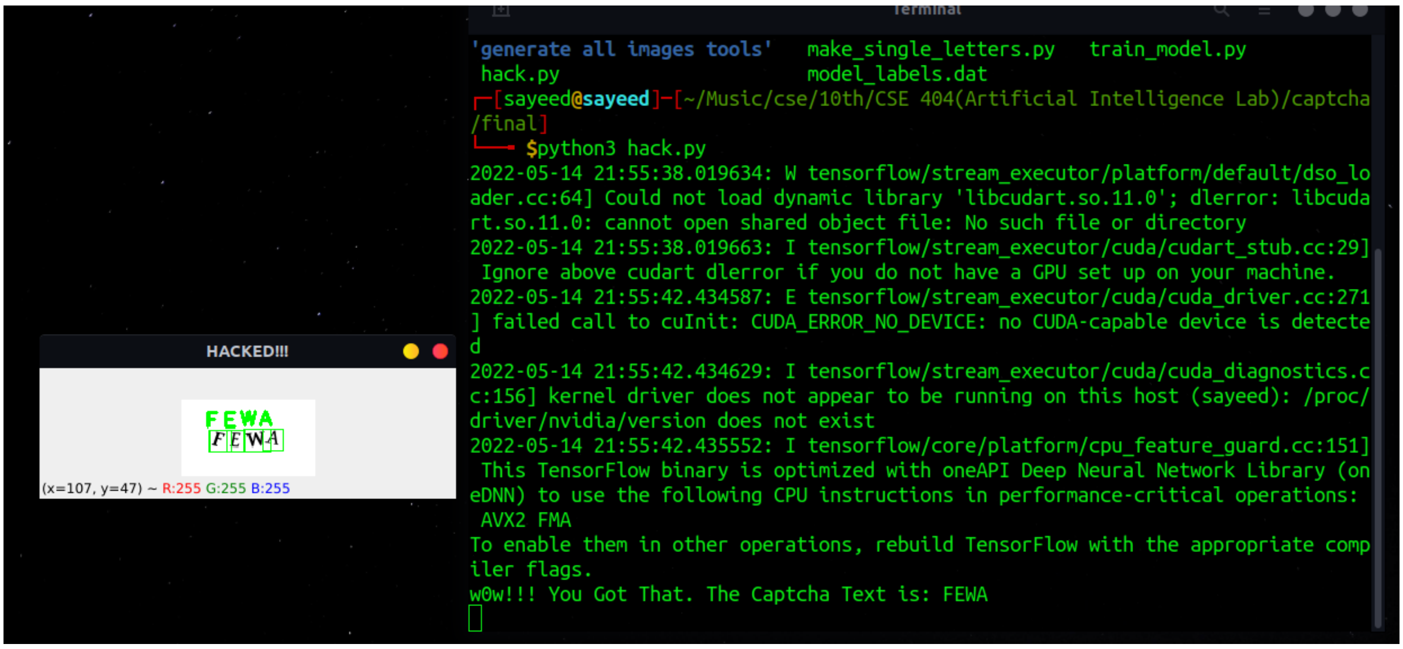
Results and Discussions

First of all, we have to Generate images. To do that, just run “python3 generate_image.py” . Make sure you have font.txt file on the same directory. The results will be saved in the "all_captcha_images" folder.

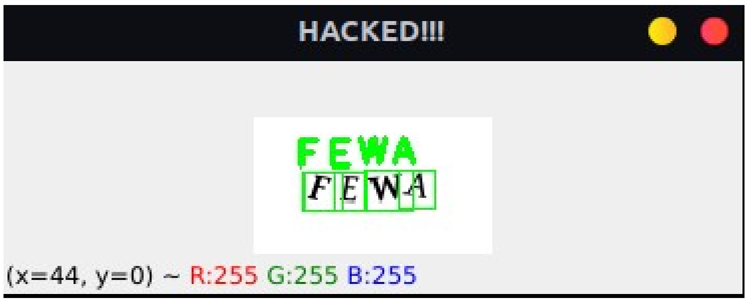
Then it's tiem to Extract single letters from CAPTCHA images .So to do that. Just run the following command.
“python3 make_single_letters.py”
The results will be saved in the "images_for_letters" folder.

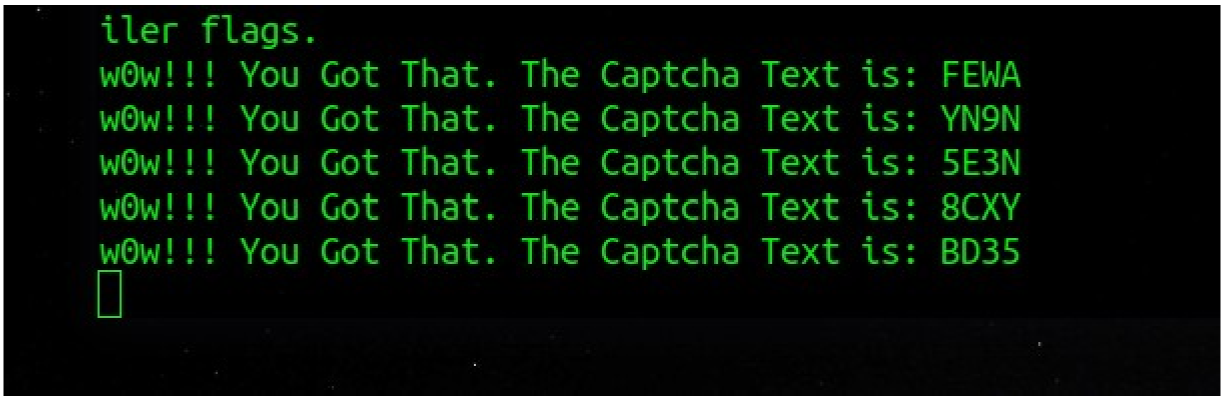
Then we have to train the neural network to recognize single letter to do that just run “python3 train_model.py”. This will write out "captcha_model.hdf5" and "model_labels.dat".

Now finally, it's time to hack the captcha. To got that. Just run the below command.
python3 hack.py
Then the Output or result looks like as below:

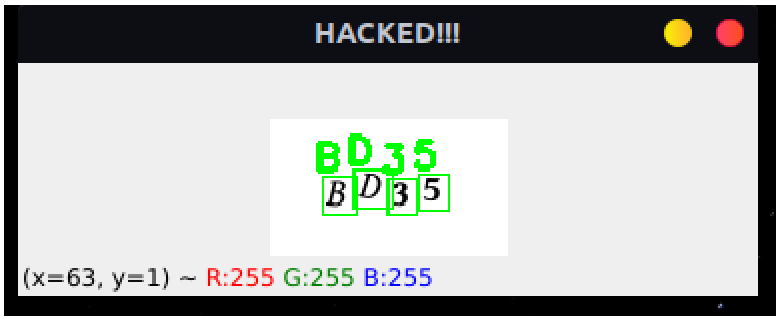


```
'generate all images tools'   make_single_letters.py   train_model.py
 hack.py                      model_labels.dat
┌─[sayeed@sayeed]─[~/Music/cse/10th/CSE 404(Artificial Intelligence Lab)/captcha
/final]
└──╼ $python3 hack.py
2022-05-14 21:55:38.019634: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcuda
rt.so.11.0: cannot open shared object file: No such file or directory
2022-05-14 21:55:38.019663: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
 Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2022-05-14 21:55:42.434587: E tensorflow/stream_executor/cuda/cuda_driver.cc:271
] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detecte
d
2022-05-14 21:55:42.434629: I tensorflow/stream_executor/cuda/cuda_diagnostics.c
c:156] kernel driver does not appear to be running on this host (sayeed): /proc/
driver/nvidia/version does not exist
2022-05-14 21:55:42.435552: I tensorflow/core/platform/cpu_feature_guard.cc:151]
 This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (on
eDNN) to use the following CPU instructions in performance-critical operations:
 AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate comp
iler flags.
w0w!!! You Got That. The Captcha Text is: FEWA
```





```
iler flags.
w0w!!! You Got That. The Captcha Text is: FEWA
w0w!!! You Got That. The Captcha Text is: YN9N
w0w!!! You Got That. The Captcha Text is: 5E3N
w0w!!! You Got That. The Captcha Text is: 8CXY
w0w!!! You Got That. The Captcha Text is: BD35
```



Conclusion

Machine learning is no longer just a word or tech term. CAPTCHA cracking is simply one of many challenges that ML solves. We discovered that there is a lot of room for development in this domain when working on a few problems involving convolutional neural networks. Users have not yet approved the accuracy levels. Web users are no longer unfamiliar with the term CAPTCHA. It's the vexing human validation check that many websites include. Completely Automated Public Turing Test to Tell Computers and Humans Apart is an acronym.

Practical Implications

There are a several Practical Implications is possible for this work. Some of them are mentioned below:

1. It's really effective and useful for penetration tester who solve millions of captchas for penetrating any system.

2. It will be useful for the Military Cyber team. Because sometimes the Military Cyber team's need to access any criminal or scammer system by brute forcing attack and at that time they have to solve the captcha if there is any captcha system enabled. Since this is not possible for humans to solve millions of captchas so that's why this intends me to carry this out.

3. For example, there are a situations in which a new client or business partner has joined you and requires access to your Application Programming Interface (API), which is not yet available or cannot be provided owing to security concerns or potential abuse. As a result, the only option is to use automated scripts to get around CAPTCHA. So in this case, It will be worked like a charm !!!

Scope of Future Work

As I said before, there are a lot of captcha type such as Text-based CAPTCHA, image-based CAPTCHA, reCAPTCHA, mathematical CAPTCHA, Voice CAPTCHA etc. Now, this project is not able to Break Down image-based CAPTCHA, reCAPTCHA, mathematical CAPTCHA and Voice CAPTCHA. For example, this project is not able to break down the google reCAPTCHA. So we can say, this is the limitations of this project.

So, in future, I will try to give more effort on this project to develop more functions so that it can be able to break down the google reCAPTCHA.