Documentation
Hackathon
GEN-511
25th November, 2019

1. Problem Statement

CAPTCHA (Completely Automated Public Turing test to tell Computer and Humans Apart) is a mode to separate genuine users from bots. This is accomplished with the use of distortorted texts written on images so that they are still recognizable to most humans but can become difficult for a computer to recognize. This is one of the most populare practice to test the authenticity of the user and prevent the loss of resources by entertaining the request by computer bots.

Additionaly, many CAPTCHAs also inject noise additional structures such as blobs or lines into the image to further make it difficult to recognize. They are primarily employed as security measure on websites to prevent bots from accessing or performing transactions on the sites.

On the other hand, machine learning (ML) algorithms, in particular deep learning (DL) neural networks have been trained with significant success on similar problems such as handwritten digit recognition. This motivates us to build an ML-based CAPTCHA breaker that maps CAPTCHAs to their solutions.

2. Related Work

As CAPTCHAs are actively used by many websites to protect traffic, major corporations have already invested significant resources in breaking CAPTCHAs to assess the strengths of shortcomings of these data techniques. Google's StreetView team, for example, have used their algorithms for recognizing signs in images on the CAPTCHA problem, achieving 99.8% success on particular types of difficult-to-read CAPTCHAs.

3. Dataset Description

The task of finding a genuine data for targeting such a problem was a difficult. All the datasets present on the internet were either build to target the accuracy of the models built by the user or were very small. Due to unavailibilty of data, we decided to build the data as well.

To complete the task and give our model enough degree of randomness, we employeed various methods to come up with a rather varied and vast dataset. Several PHP and Python scripts were written for CAPTCHA generation and custom impurities such as distortion and noise were introduced. The scripts emplyoyeed usage of several degrees of freedom such as font style , distoration and noise ,which can exploit to increase the diversity of our data and the difficulty of the recognition task.

For the first step, we have created letter string ranging from A to Z and 0 to 1 . Thus, we had a four lettered random text. Now we created a raw image with random color combination of the dimension (72-by-24) pixels. Initially only clean images with varied fonts of character were arranged on the background and later impurities were introduced. Figure 1 shows a collection of data generated by the model. The sample dataset generated illustrates the level of augmentation with distortion and

colors used to make a varied dataset. Thus we move to the next domain of targeting the problem and devising a technique for CAPTCHA recognition. Thus, Finally, we generated a set of 30000 CAPTCHA which further measure how well our algorithm have truly learned the fundamental archetype of each letter based on training data.

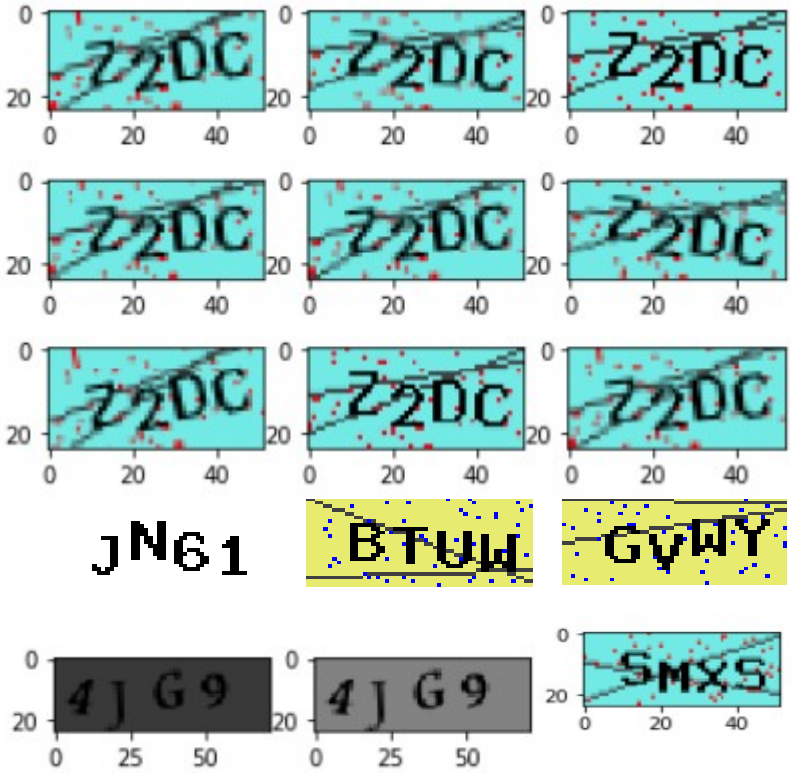

Figure 1: Sample Dataset

4. Feature Analysis:

This problem is based on the selecting the region where the text is present and make a rough rectangular border around it in a way such that each letter forming a part of the CAPTCHA gets cropped separately. Thus, we employeed the techniques of image processing to calculate the region and associating it with the contours and finally bounding it around the desired letter region.

The first step we took to do this was find the threshold of a certain region. For a single image, we passed the image and computed various thresholds using the adaptive and otsu methods. Later, we even employeed the Gaussian blur and then calculated the otsu threshold. Thus, three images, each coming out from a single image and the threshold was being calculated. Another image processing technique which follows is dilation. This morphological operation increases the object area and is used to accentuate features. Similarly, the erosion method erodes the boundaries on the foreground and is used to diminish the feature of the image. On compilation, i.e. computing threshold of an image, the expanding the expectation area by computing dilation and removing the boundaries from the foreground and again expanding the expectation window by computing the dilation of eroded images.

Figure 2 and 3 illustrates the four phases the image went through and after this certain contours are generated for each image. These contours are special as they help by forming a rectangular boundary and give co-ordinates of region potential of containing a letter.

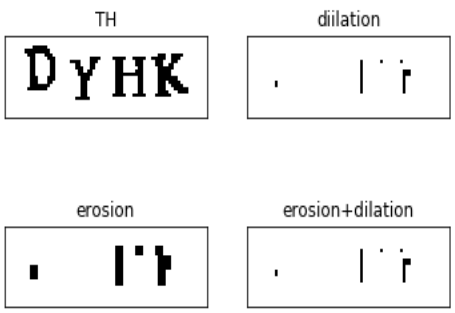

Figure 2: Phases using adaptive threshold

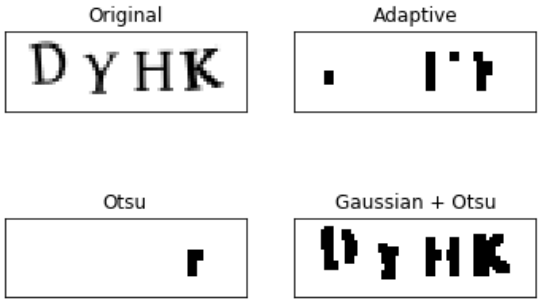

Figure 3: Phases using Gaussian blur with OTSU threshold

5. Method

The methodology discussed in the section 4 gives an outline of image preprocessing and the steps we took to extract possible co-ordinates containing the letters in the given image. We now save each of the letters extracted from the CAPTCHAs and later use them for training the model. Figure 4 shows the letters extracted after the above preprocess.

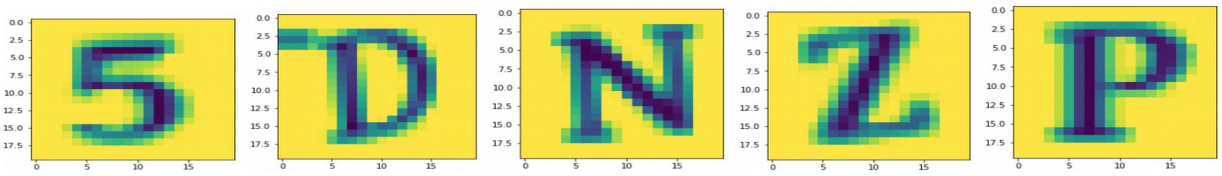

Figure 4: The obtained single letters

Along with these images, we also computed the labels and saved them alongside. This way, the data set was preprocessed and ready for the model to be trained. For training the model, the Keras Python library makes creating deep learning models fast and easy. The sequential API allowed us to create models layer-by-layer for our problems. The simplex layered architecture defined in figure 5 was trained on twenty-six thousand images and the final output was tested against the testing data built. The testing and training data had all unique values. Thus the final dense layer using the softmax activation, had 36 nodes/neurons.

6. Results

While we show single-letter classification results comparable to what is currently state-of-the-art, our attempts to generalize single letter learning to multi-letter CAPTCHA recognition was successful up to 98.733118 per cent. This shows that the problem could be matched and pretty good results could be achieved. Figure 6 shows the performace of model towards a single random captcha. The captcha image is already preprocess before printing.

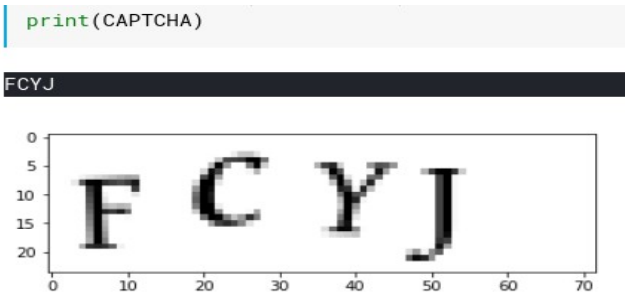

Figure 6: Performance on a single image

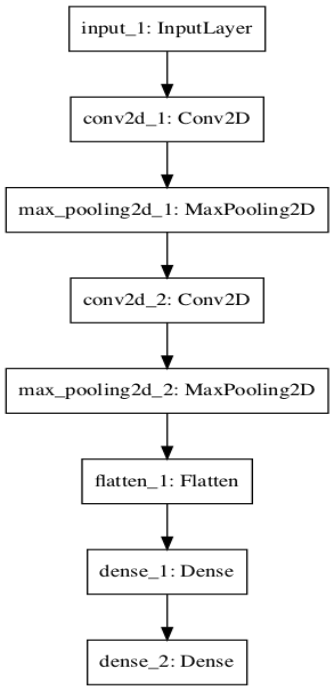

Figure 5: Sequential Artichture used for building model