MACHINE LEARNING – SE204

CLASS PROJECT

PROJECT TITLE:

CNN-based OCR model to break text-captchas

Subject Code: SE204
Subject Name: Machine Learning
Branch: Software Engineering
Year: 2nd Year/04 Semester

Submitted by:
ABHINAV CHAITANYA
2K21/SE/07

Submitted to:
Ms. Shweta Meena
Assistant Professor
Department of Software Engineering
Delhi Technological University

Delhi Technological University
Shahbad Daulatpur, Main Bawana Road, Delhi-110042
April 2023

INTRODUCTION

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a type of security measure known as challenge-response authentication. CAPTCHA helps protect one from spam and password decryption by asking to complete a simple test proving that the user is a human, not a computer trying to break into a password-protected account. A CAPTCHA test is made up of two simple parts: a randomly generated sequence of letters and/or numbers that appear as a distorted image, and a text box. To pass a test and prove one's human identity, a person simply types the characters that he/she sees in the image into the text box.

CAPTCHA could be used to prevent different types of cyber security threats, attacks, and penetrations towards the anonymity of web services, websites, login credentials, or even in semi-autonomous vehicles, and driver assistance systems when a human need to take over control of a machine/system. These attacks often lead to situations when computer programs substitute humans, and it tries to automate services to send a considerable number of unwanted emails or access databases. One of the most common forms of cyber-attacks is the DDOS (a distributed denial-of-service (DDoS) attack, a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic) attack in which the target service is overloaded with unexpected traffic either to find the target credentials or to paralyze the system, temporarily. One of the classic yet very successful. Different types of alphanumerical CAPTCHA sample solutions are utilising a CAPTCHA system in the evolution of cybersecurity systems. Thus, the attacking machines can be distinguished, and the unusual traffics can be banned or ignored to prevent damage. In general, the intuition behind the CAPTCHA is a task that can distinguish humans and machines by offering them problems that humans can quickly answer, but the machines may find them difficult, both due to computation resource requirements and algorithm complexity.

There are several types of captchas that are available to be deployed on a website. These modern-day captchas mainly fall into three major categories these are text-based, image-based, and audio-based captchas. However, text-based captchas are used most widely. They require translation as they are usually displayed in an odd style and the characters are sometimes difficult for even humans to decipher, and set apart the machines. But this effectiveness must not undermine the modern-day technologies that use advanced machine learning and deep learning algorithms and techniques to crack them. Hence, before the deployment of the product, business owners must analyse the effectiveness of their captchas to prevent any sort of bot -attacks that can take place.

As already discussed, we have different types of captchas that have been into use to protect the business logic. However, in this paper, I shall focus on text-based CAPTCHAs as they are still vastly used in high-traffic and dense networks and websites due to their lower computational cost.

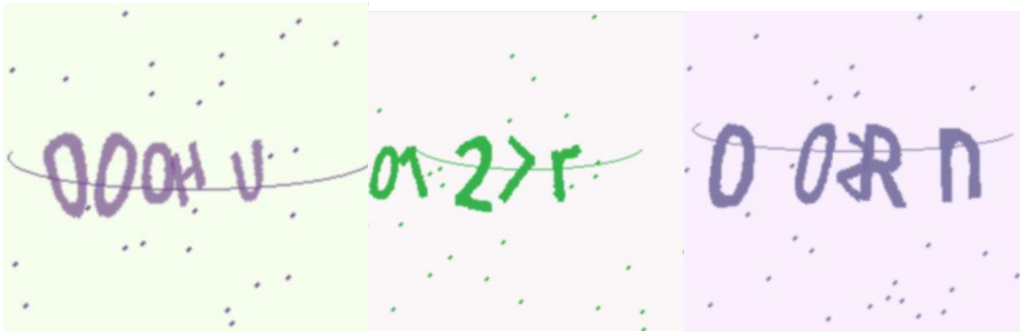
This research tries to solve the CAPTCHA recognition problem to improve the quality of text captchas that are generated and put into use, as the bots and scams are getting more advanced and smarter on a day-to-day basis.

There have been multiple attempts to achieve the aforesaid attempts in the last 15 years by numerous experts in the cybersecurity and machine and deep-learning domains. This project aims to appreciate these efforts and the author's approach to the problem using Convolution Neural Networks using Keras, Tensorflow, and openCV python libraries to try increasing the efficiency of the recognition that may later help in the development of further robust techniques for generation of text-based captchas.

It is worth noting that the author's knowledge of this domain is limited to classwork understanding of the field of machine learning and deep learning as well as just a few weeks of exploration of the topic and its related aspects.

METHODOLOGY

To achieve the aforesaid goal the datasets used are- dataset-1 consisting of 82000+ captcha images and dataset-2 consisting of 1070 captcha images. The images in dataset-1 have been created using the Python PIL library. The datasets have Captcha images named in the format 'captch_word'.png example 0xg12.png for an image containing the captcha 0xg12. Each captcha has 5 characters which could be numbers or alphabets. The captchas differ by color, spacing, orientation of characters, and size of characters. Random Noise and lines have also been introduced into the captcha.



3 example images from dataset-1

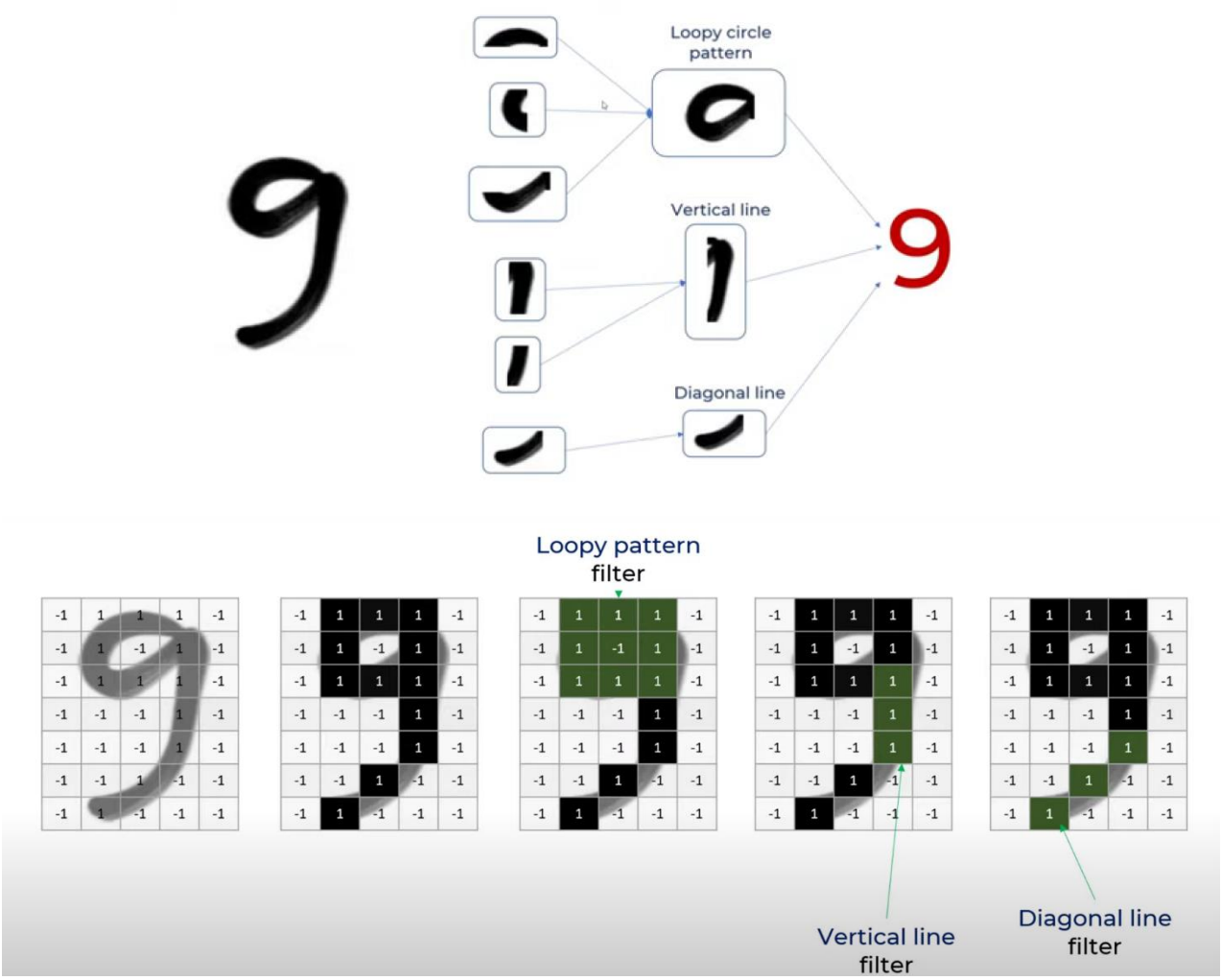
3 example images from dataset-2

Since this classification task is based on images it is quite intuitive to use CNN deep-learning algorithm. It becomes quite important to discuss how CNN works. Discussing broadly it works by identifying small edges across the image through a combination of pixels. Combines them to form several patterns and finally combine them to reach a decision.

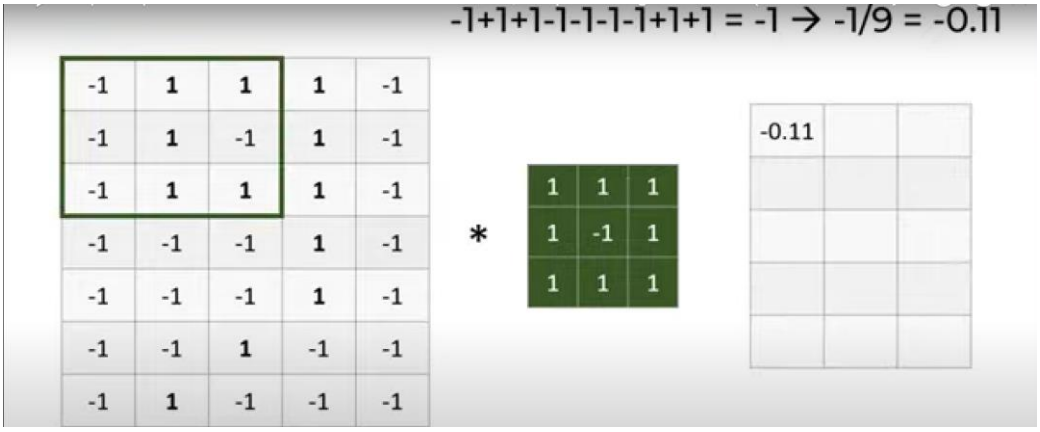
The process is conducted with the use of filters, also known as feature detectors. Actually, an image can be broken into a combination of pixels as grids. Each pixel stores the intensity of color in RGB format. As each RGB pixel has three sets of 8-bit binary numbers I, therefore, has 24 bits of computer information in total. Hence the term '24-bit color'. And as 8 bits equals 1 byte, each RGB pixel, therefore, equals 3 bytes in file size. Since $2^8 = 256$, thus, the values are represented from 0-255 and while training the model we need to scale up these values

between 0-1, which is accomplished by dividing by 255. This operation is clearly facilitated using the NumPy library. We, thus, multiply the original image grids by the grids of filters which are automatically chosen by the CNN algorithm. We add the results of the corresponding cell multiplications, take the average, and thus, are reduced it as a feature map. This helps us detect the features. It is thus, not wise to use ANN as it requires too much computation, treats local pixels the same as pixels far apart, and is a much more sensitive location of an object in the image.

The concept of feature detection and feature maps can be explained by the following examples of handwriting detection. We can clearly see the different features that the algorithm tends to identify on its own using the figures given below for digit 9.



The above picture is representative of the filters the CNN algorithm is expected to detect

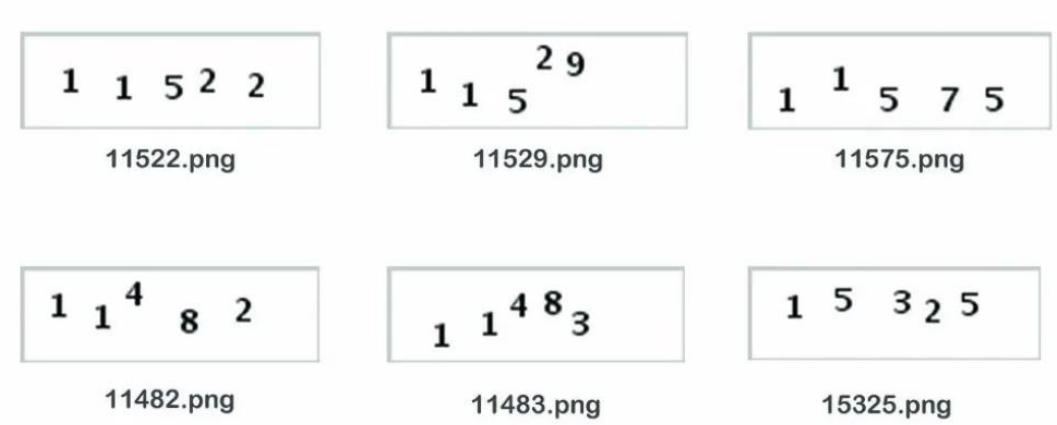


The above picture represents the working of loopy circle filter with stride of one to form a feature map

The completed feature map for the loopy-circle feature

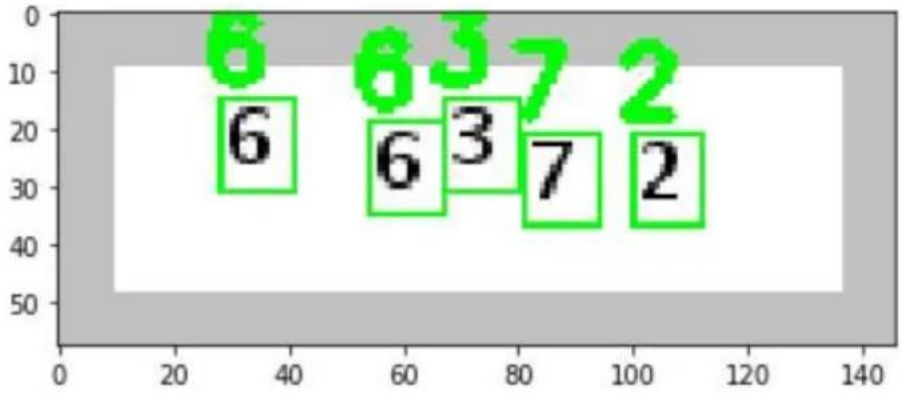
These feature maps are passed via the ReLU activation function to provide non-linearity. Max-pooling strategy is also used for the purpose of position-invariant feature detection, while it also helps in reducing the dimension and computation. It reduces overfitting as a smaller number of parameters are involved. The model is tolerant towards the variations/distortions as this way the noise is filtered as well. Since the captcha has multiple characters and each of them is of interest to us, there is a need for localization/segmentation. We may use approaches like sliding windows, trial, and error or RCNNs, or the most advanced technique of YOLO-v4. To cater to the multiple bounding regions of interest (ROIs) we require to use the approach of Intersection Over Union (IOU). All these techniques grouped together may help us in realizing the aim of this project.

The techniques to be used for developing such a model is quite convincing if the data is like the images given below.



In the above images high degree of accuracy can be developed even by naïve programmers using simpler models. It is quite evident that the segmentation and localization of characters appearing in the following captchas are easy. Moreover, there is no noise in form of randomly scattered lines or dots, making them quite vulnerable to advanced bot attacks.

The individual characters can be easily identified using the OpenCV library using findCountours function with 'THRESH_BINARY_INV' threshold to grab the counters and the coordinates for the bounding rectangles. The final output shall be something like this.

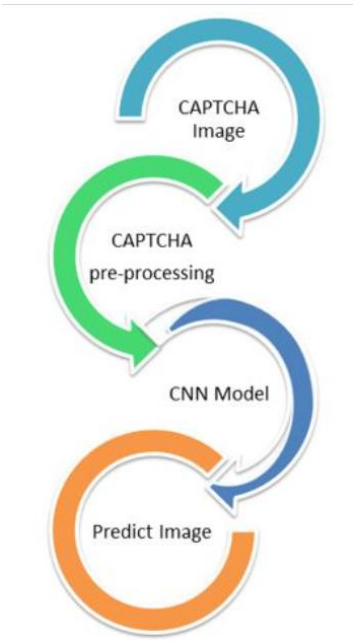


In dataset-2 we clearly see that the captcha images are comparatively complex and have noise in form of random lines as well. Hence, these captchas are better.

The author trained his model on dataset-2 and achieved an accuracy of 97.826% in training. While the 214 captcha images for testing were recognized with an accuracy of 69.626% (i.e., successfully cracked 149 of the 214 test images). Dataset-2 had just 1070 images which may owe to a high degree of accuracy; however, the model is expected to perform equally well with dataset-1 having around 82300+ images. The author could not run the code for the model for dataset-1 due to machine and cloud constraints on part of the author.

The methodology followed by the author is as follows-

The model is developed using Convolutional Neural Network for detecting the Captcha present in the image. For this purpose, the training sample dataset is first pre-processed and then the model is developed consisting of 3 convolution layers with a max-pooling strategy and the same padding for each layer. Next, the author used a hidden layer corresponding to each character in the captcha.



This is the flowchart for the methodology followed.

Data preprocessing-

The dataset consists of images of size 50 in height and 200 in width. These images first need to be pre-processed before developing the model. For the purpose of training, the images in the dataset have a filename the same as the CAPTCHA possessed by the image. The images are first pre-processed by reading them in grayscale. This helps us to remove the noise to some extent as the images get invariant with the background color. Simultaneously, the file name is also stored in a string. Each grayscale image is then scaled and reshaped. Then we create an array of dimensions 5*36, used to store the character present at each position in the CAPTCHA. At each position, through the filename, we find which characters are present at each position of the CAPTCHA and update the corresponding location to one in this array. This array will thus be used for training the model. Thus, after pre-processing we obtain all the grayscale images and the target array containing information regarding the characters present in each CAPTCHA image.

Model Training –

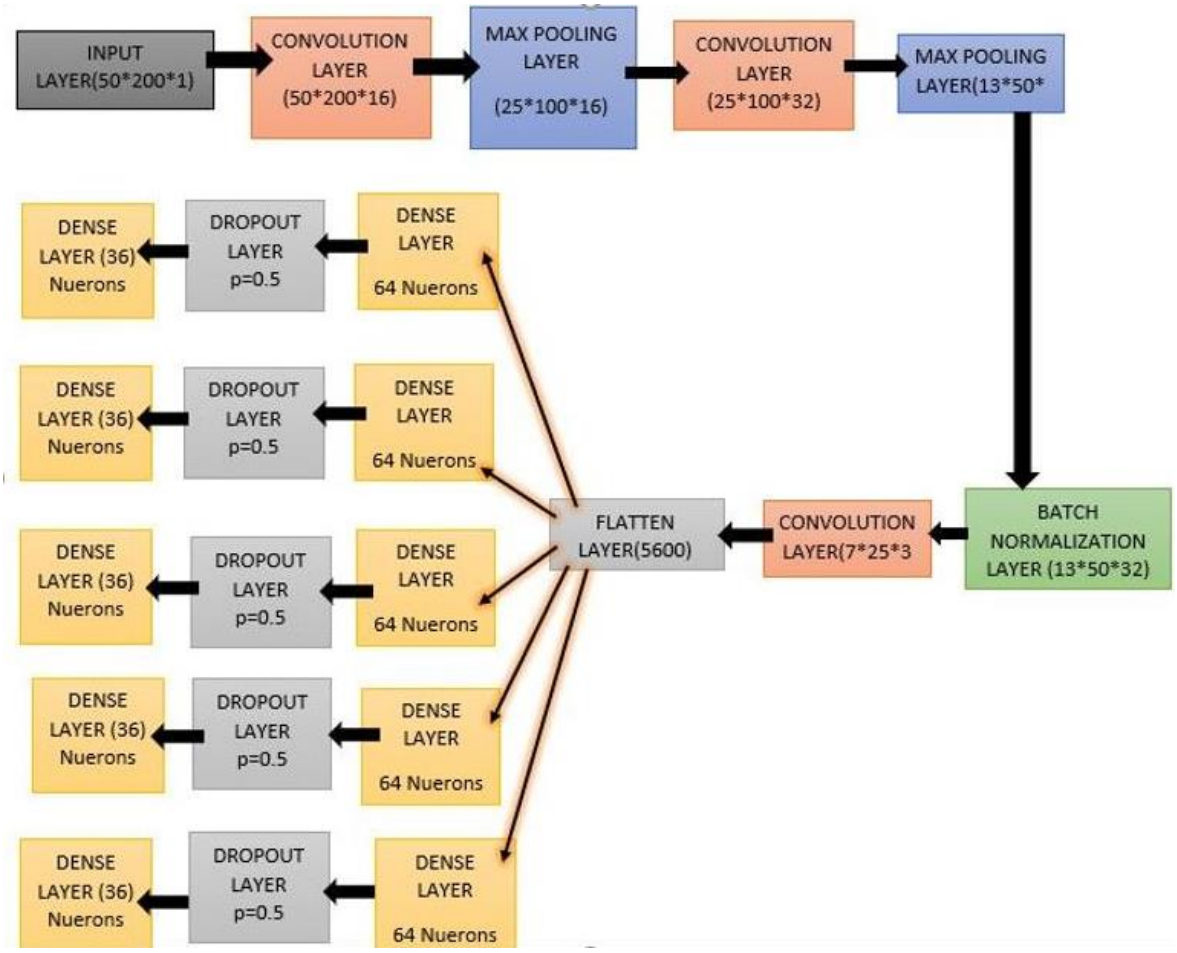
The code goes as follows-

```
#create model
def createmodel():
    img = layers.Input(shape=imgshape) # Get image as an input of size 50,200,1
    # convolution layers
    conv1 = layers.Conv2D(filters = 16, kernel_size = (3, 3), padding='same', activation='relu')(img) #50*200
    mp1 = layers.MaxPooling2D(padding='same')(conv1)  # 25*100
    conv2 = layers.Conv2D(filters = 32, kernel_size = (3, 3), padding='same', activation='relu')(mp1)
    mp2 = layers.MaxPooling2D(padding='same')(conv2)  # 13*50
    conv3 = layers.Conv2D(filters = 32, kernel_size = (3, 3), padding='same', activation='relu')(mp2)
    bn = layers.BatchNormalization()(conv3) #to improve the stability of model
    mp3 = layers.MaxPooling2D(padding='same')(bn)  # 7*25

    flat = layers.Flatten()(mp3) #convert the layer into 1-D

    # hidden and output layers for the 5 outputs
    outs = []
    for _ in range(5): #for 5 letters of captcha
        dens1 = layers.Dense(64, activation='relu')(flat)
        drop = layers.Dropout(0.5)(dens1) #drops 0.5 fraction of nodes
        res = layers.Dense(nchar, activation='softmax')(drop)

        outs.append(res) #result of layers

    # Compile model and return it
    model = Model(img, outs) #create model
    model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=["accuracy"])
    return model
```

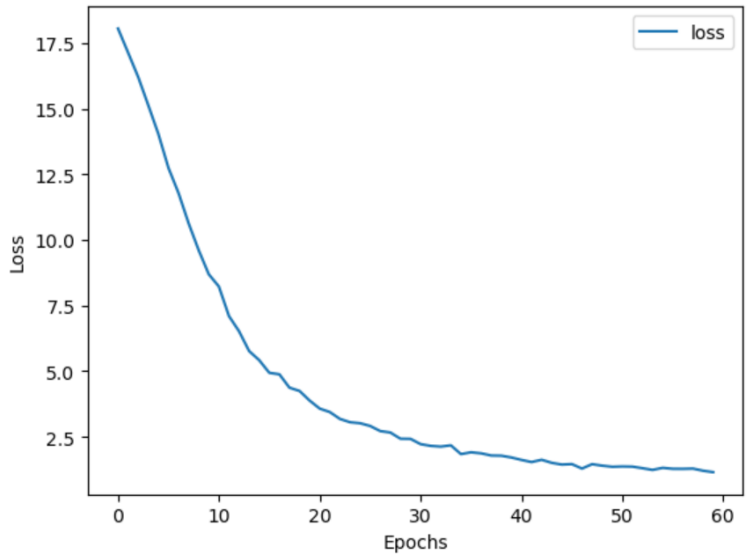
The model developed for this CAPTCHA dataset uses CNN. It consists of a total 24 layers comprising the input layer, convolutional layers, max-pooling layers, dense layers, flatten layers, and dropout layers. The total number of parameters is 1,818,196 where 1,818,132 parameters are trainable and 64 are non-trainable parameters. A brief architecture of the layers is depicted in the figure. The first layer is the input layer which takes the image as input. Then we have convolutional layers and the max pooling layers which extract the most prominent

features from the images. The next layer is the batch normalization layer, used to improve the stability of the model. Then we have a flattened layer that converts the input from the max pool layer to a long vector of desired dimensions. This is done so that the further neural network is easily processed and the backpropagation is carried out easily. Further, there are five dense layers in this neural network each of which is connected to the flattened layer. Each of these dense layers deploys the activation function 'ReLU' to train the parameters. Further to each of these dense layers is connected a dropout layer used for regularisation. Following the dropout, the layer is again a dense layer that uses the activation function 'softmax.' Earlier the sigmoid function was used but the softmax activation function only benefitted the model.

Thus, the model uses an image of dimension (50, 200, 1) as input and gets output from 5 layers each having dimension 36. The model uses the optimizer 'Adam'. The model uses the loss function as 'categorical cross entropy'.



Results –



The model was trained for 60 epochs. The following graph was obtained for loss with respect to the number of epochs. We see that as the number of epochs increases, the loss decreases exponentially. The loss obtained on the training set is 0.6046 while the loss on the test set is 0.5392.
As discussed earlier, the test captchas were recognized with an accuracy of 69.626%. The model had an accuracy of 97.826% on training images.
Visit the GitHub repository - https://github.com/AbhinavChaitanya01/captcha-breaker for detailed results and actual code.

Conclusions and Learning Outcome-
The model helped us realize the potential of CNN in the field of cyber security, particularly in captcha detection. The model may prove to be of some help in generating better and tough-to-crack CAPTCHAs and to assess the current effectiveness of a captcha in a deployed website.
The model helped us understand how CNN works, its underlying details, and its effectiveness.
The future scope of this work lies to expand this CAPTCHA recognition system for larger & noisier CAPTCHA containing all the possible symbols. The results of the model on our dataset-1 are also awaited, subject to machine constraints.

References-

- Hands-On Machine Learning with Scikit-Learn and TensorFlow – Aurleien Geron
- Noury, Z., & Rezaei, M. (2020). Deep-CAPTCHA: a deep learning-based CAPTCHA solver for vulnerability assessment. *arXiv preprint arXiv:2006.08296*.
- https://www.youtube.com/playlist?list=PLeo1K3hjS3uu7CxAacxVndI4bE_o3BDtO
- https://shubhamchauhan125.medium.com/cnncaptchasolver-5625b189a14f
- https://www.youtube.com/watch?v=XF_3fuPp8Wk