De-CAPTCHA : Numerical Captcha Solver using Deep Learning Models

Divya NA [1]

[1]

Abstract

De-CAPTCHA is a deep-learning based CAPTCHA solver which is a human-centered test to distinguish humans from bots. CAPTCHA (Completely Automated Public Turing Test to tell Computers and Humans Apart) is the most popular way to prevent bots from entering the unauthorized systems. For an efficient model, the automatically generated image dataset of 10,000 images with obfuscations has been used. The aim of the project is to apply deep learning models like VGG11 and custom Convolutional Nueral Networks on the images generated in the program to correctly detect the text present in noisy data. As the outcome of the project, various observations related to small CNN and large CNN models and comparison of models has been done. The projects also analyses performance based on accuracy of models on different datasets.

1. Introduction

CAPTCHA (Completely Automated Public Truing Test to Tell Computers and Humans Apart) is human-centric turing test to distinguish between humans and bots. It is commonly used for prevention of bots from entering the unauthorized and confidential systems. Also, it is to verify the identity of user who is logging in the secured system. The techniques to use Captcha has been very famous since years. There are various benefits of using CAPTCHA on system which requires security as it helps prevent cyber attacks, Crucial Information Leakage, penetrations attacks and unidentified automated registrations. It ensures the user is human in risky situations like autonomous vehicles as presented in (Rezaei & Klette, 2014).

In particular, these attacks often lead to stressful conditions where mass spamming, unauthorized access to database gives hold of the personal information to the illegal user. To provide a stronger layer of protection to such delicate systems and databases, use of CAPTCHA has been very useful.

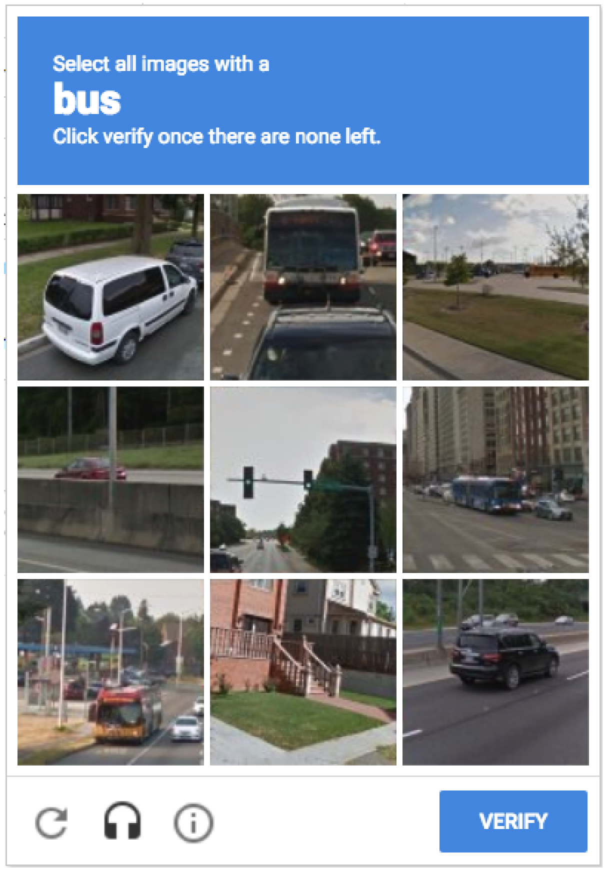


Figure 1. Image CAPTCHA generated by google which asks the user to choose images of bus amongst 9 different images.

The basic intuition behind creating a layer of CAPTCHA is security of the system where the system/database can ask the user about their identity. If it's a human, he/she will successfully decode the CAPTCHA whereas it is a harder task for bots to do.

CAPTCHAs are created in different ways including image CAPTCHA, Numerical CAPTCHA and Alphanumeric CAPTCHA. Image CAPTCHA as in Figure 1 asks user to choose some images related to the thing mentioned in the

[1]Department of Electrical and Computer Engineering, University of Waterloo, Ontario, Canada. Correspondence to: Divya NA <d3na@uwaterloo.ca>.

question which can be choosing the pictures of Roads , bus , Traffic Lights etc. Similarly, Alphanumeric CAPTCHAs include the code which has numbers as well as Alphabets present in it. The Aplhanumeric code is a combination of random alphabets and Numbers present in obfuscated image. Similarly, Numerical CAPTCHAs include a numeric code in the distorted image which makes it an arduous task for bots to clearly find the code present. Both Alphanumeric and Numeric CAPTCHAs require users to manually type the code present in the image and submit it further.

CAPTCHAs are generated by adding noises which are of different types. Noise can be addition of crossing horizontal lines across code, various usage of font sizes , rotation of numbers and letters, adding background noises like dots, contours etc. The lines must go across the code as to avoid line segmentation algorithm which can detect the line and uncover the code. Sometimes, addition of noise in the data can ruin the code completely. So, one needs to take care of the level of distortions in the image so that even after substantial amount of noise, the code can be human-readable. For Alphanumeric codes, some numbers and letters are better to avoid to go together for example 'O' and '0' which after distortions can look very similar to (Zahra Noury, 2021). Similarly, 'I' and '1' cannot be coded together to reduce confusions.

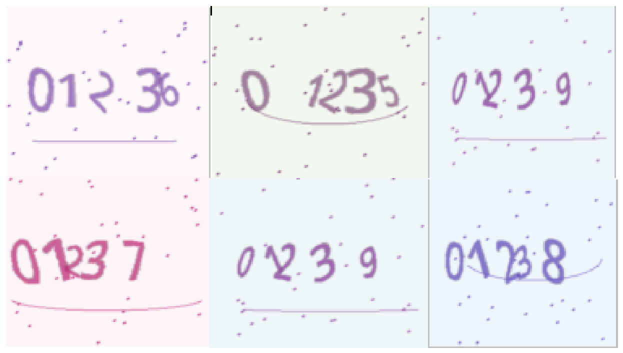


Figure 2. CAPTCHA Images generated using automated CAPTCHA Generation Library in Python with some distortions to make it harder for bots to predict. Distortions include background noise and lines across the images.

Furthermore, one of the applications of CAPTCHAs is in OCR(Optical Character Recognition) systems. The OCR algorithms are very robust, though there are some areas where it fails in recognizing the handwritten or complex texts. Nowadays, the technique of re-CAPTCHA which fine tunes the OCR algorithms. In this project, I have worked on two Numerical Datasets which are automatically generated through python library as in Figure 2 as well as ready dataset from Kaggle shown in Figure 3. So, the project will focus on text-based numeric codes of particular length. The downloaded dataset[2] has 5 levels of images with increased distortions and complications in each progressing levels. My project used Level 5 images which involved the highest distortions.

The main aim of the project is to Decode the numerical CAPTCHA coded present in the images (De-CAPTCHA). For that, different deel learning models like VGG11, Resnet and customCNN models have been applied. The results of each algorithm is analyzed and comparison is done among all.

The rest of the report is organised as follows: Section 2 includes Related Work which discusses the similar work done in past by other authors and their results. In Section 3, named Methods, the implementational details related to dataset , preprocessing and models are mentioned. Section 4 analyses the results after implementation of mentioned algorithms. Finally, Conclusion and References are presented in Section 5 and Section 6 respectively.

2. Related Work

This section briefly discusses the recent work by authors on CAPTCHA related datasets.

(Garg & Pollett, 2016) has implemented a short model on their dataset which involves two Convolutional Maxpool layers followed by a dense and a softmax layer. The implementation of model included use of SGD optimizer with Nesterov momentum and used Recurrent layers for testing phase with simple Dense layers. They proved in their research that use of Dense layers has aided in the accuracy of model detection.

(Stark & Cremers, 2015) has also implemented CNN on CAPTCHA dataset to recognize the code present in the image. In their model, they implemented using three Convolutional layers, two Dense layers and worked on solving six-digit CAPTCHAs.

In paper by (Sivakorn & Keromytis.), they have worked on creation of web-based project to solve CAPTCHAs and used GRIS (Google Reverse Image Search).

In paper done by (Bostik & Klecka, 2018) and (Yousef & Mohammed, 2018), authors talked about adding noise to dataset like line-noise or point-based noise to increase the complexity for security layer for systems.

Furthermore, the authors in (George, 2017) and (Ye & Wang, 2018) discussed the results of GAN (Generative Adversarial Networks and Generative Models to train better model on data.

[2]https://www.kaggle.com/datasets/tomtillo

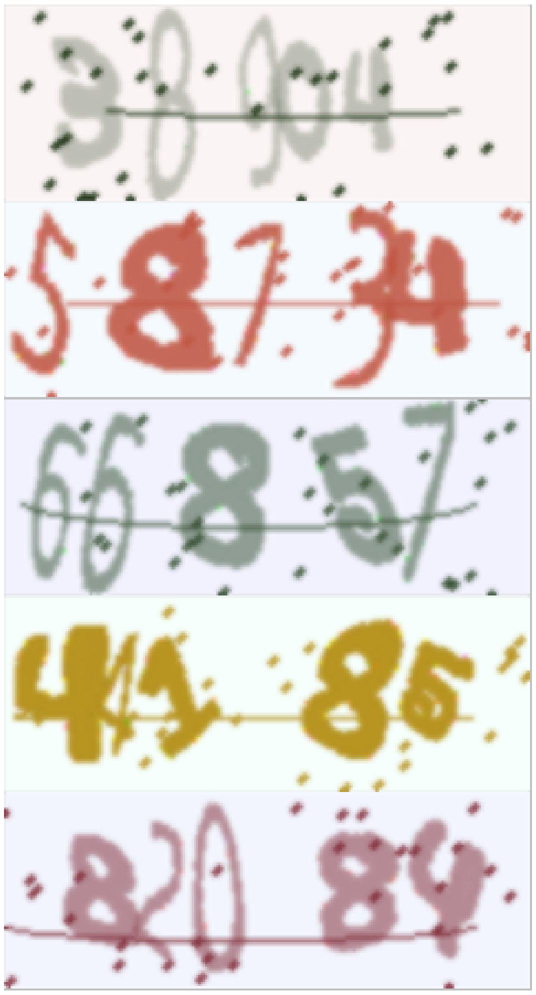


Figure 3. Some of the Numerical CAPTCHA Images from Kaggle CAPTCHA. The pictures are from Level 5 of dataset which includes highest level of distortions in the images like lines, dots in the background and distorted numbers to make it more noisy.

Many researches have worked on Adversarial networks and one of them by (Osadchy & Pérez-Cabo) talked about making more noisy data and viewing them as human-readable for humans even after such obfuscations.

In work by (Kwon & Park, 2019) , (Karthik & Recasens, 2017) , (Zhao & Jiang, 2017) on CAPTCHA dataset have used CNN models and cracking the code in images.

3. Methods

The downloaded captcha dataset was trained using two models - VGG11 model with some additional layers and the Resnet model. Both the models were implemented using the Pytorch library.Further, the auto-generated dataset was trained using Custom CNN.

3.1. Dataset

In this project , we have used two datasets for evaluation of models. The ready-dataset from Kaggle was used to implement VGG11 and Resnet Models. All the datasets used in the project includes code of fixed digit i.e 5 digits code. The dataset has 10,000 images each 60 x 160 pixel with high level of distortions as shown in Figure 3. Highly Complex and deep neural networks like VGG11 and Resnet was implemented on this data. Also, the auto-generated data as shown in figure 2 has 30,000 images which are orginally, 135 x 50 pixels in size. For application of ML model, the images are preprocessed further.

3.2. Preprocessing

Applying some pre-processing operations such as image size reduction, colour space conversion, and noise reduction filtering can have a tremendous overall increase on the network performance.

3.2.1. VGG11 Model and RESNET

The original size of images in the dataset used for VGG11 is 60 x 160 x 3 pixels. As per the model requirements, VGG11 required 32 x 32 pixel image. So, further resizing the image into 32 x 32 pixel solved the input image issues as it is the required size of VGG11 model. About VGG11 model, it includes various convolutional , batch norma, maxpool, Linear layers and has worked well for MNIST dataset. So, the size reduction helped in training in the better and efficient way.

Further, each image in the dataset has 3 channels as it is RGB image, converting each image to Grayscale images is another preprocessing method used in the model. This will reduce the complexity in the image and ease the training and prediction process. As such , the results from converting the number of channels from 3 (RGB) to 1 (GRAY scale) does not have any impact on the performance.

Additionally, the RESNET model which is also implemented in the project requires an image of size 32 x 32 and have gone under the same preprocessing methods as VGG11 images.

The last preprocessing method of removing unneccessary noise from the image. This step helped a lot in improving the accuracy. The process of removing the noise filtering using Median Filter is similar as explained by (Zahra Noury, 2021).The Medina filter algorithm used is shown in Algorithm 1.

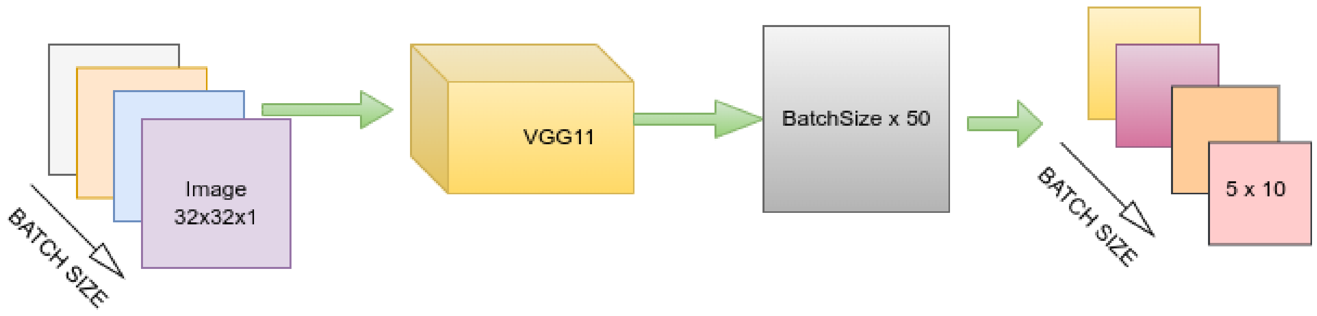


Figure 4. Architectural aproach of using VGG11 on image dataset to correctly predict the code in image.

Algorithm 1: Median filter noise filtering

```
Input: image, window size
Output: resultImage
for x from 0 to image width do
    for y from 0 to image height do
        i = 0 for fx from 0 to window width do
            for fy from 0 to window height do
                window[i] := image[x + fx][y + fy];
                i := i + 1;
            end
        end
        sort entries in window;
        resultImage[x][y] := window[window width *
         window height / 2];
    end
end
```

3.2.2. Custom CNN

The original size of images in the auto-generated dataset is 135 x 50 x 3 pixels. As per the model requirements, it required 67×25 pixel image. So, further resizing the image into 67 × 25 pixel solved the input image issues as it is the required size for custom model.

3.2.3. Custom CNN

3.3. Architecture

RNNS's(Recurrent Neural Networks) are one of the great options for predicting CAPTCHA characters, this project concentrated on sequential models because they are faster than RNNs and can produce very accurate results if the model is adequately developed.

3.3.1. VGG11 Model

The structure of VGG11 has been shown below in figure 4. The input to this model is a batch of images , each of size 32 x 32 x 1. Further, they are given to the VGG11 Model consists of 8 convolution and batch normalization layers connected sequentially one after the other. The output from these goes through 3 linear layers, after which a dropout layer with 30 percent dropout is added. The final output goes through the softmax layer which gives as output a tensor of size Batch size x 50, changing the output in the 5 x 10 vector denoting a one-hot encoded vector for each of the CAPTCHA digits.

The output in the form of 5 x 10 would depict each image having a tensor of 5 x 10 digits. In each tensor , each row suggests the probabilities of each digit from 0 to 9 for each place in code. Also, the 5 suggests here the length of code and 10 are the possible digits in that place i.e 0 to 9. This becomes the output of the model which includes probabilities for each digit in particular place.

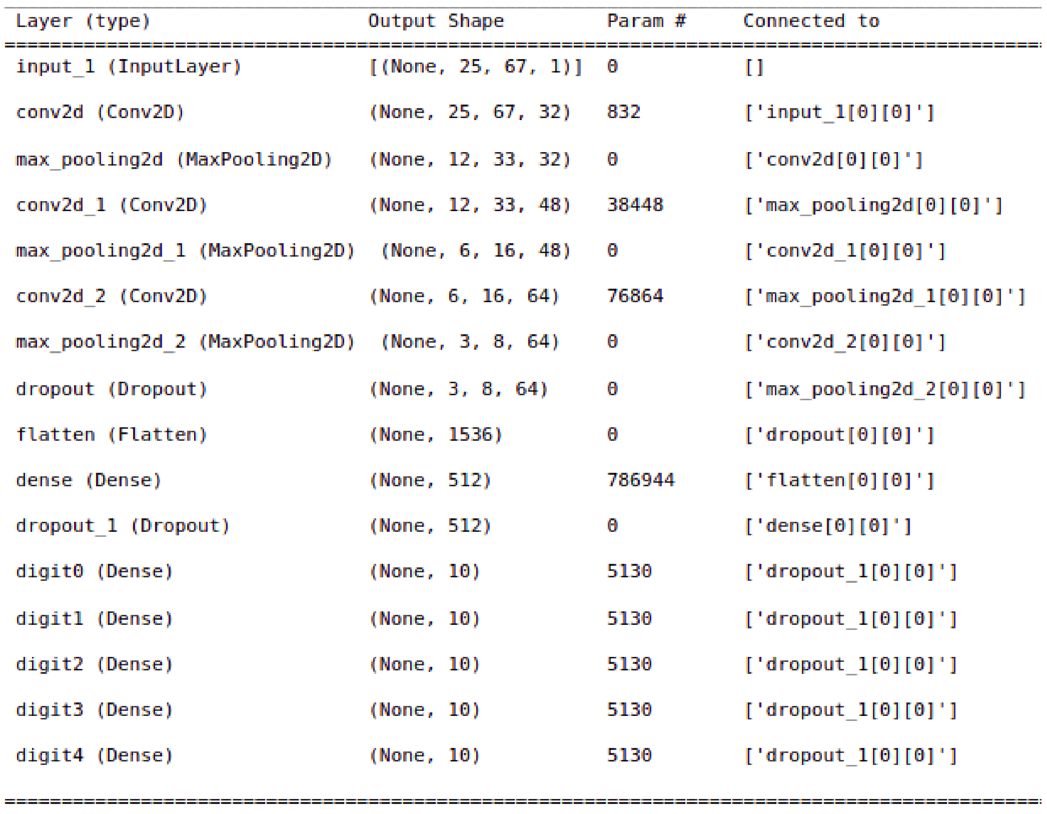
The loss function used is Binary Cross Entropy loss and optimizer used is SGD (stochastic Gradient descent).

3.3.2. ResNet Model

The ResNet model was implemented using the Pytorch library. The architectural implementation is shown in Figure 6. The input of the dimension (3, 32, 32) is given to the first layer which applies convolution, batch normalization, and relu activation function. The output from the initial layer goes through a total of 5 residual layers. Each residual layer applies convolution and bath normalization twice. The output layer consists of a linear layer, after which the softmax function is applied to produce the output of the form of a 5 x 10 one hot encoded vector, similar to the output of the VGG11 model. The architecture is and shapes of input is shown in the figure.

3.3.3. Custom CNN

The network starts with a Convolutional layer with 32 input neurons, the ReLU activation function, and 5 × 5 Kernels. A 2×2 Max-Pooling layer follows this layer. Then, we have



```
Layer (type)                    Output Shape          Param #     Connected to
==========================================================================================
input_1 (InputLayer)            [(None, 25, 67, 1)]   0           []

conv2d (Conv2D)                 (None, 25, 67, 32)    832         ['input_1[0][0]']

max_pooling2d (MaxPooling2D)    (None, 12, 33, 32)    0           ['conv2d[0][0]']

conv2d_1 (Conv2D)               (None, 12, 33, 48)    38448       ['max_pooling2d[0][0]']

max_pooling2d_1 (MaxPooling2D)  (None, 6, 16, 48)     0           ['conv2d_1[0][0]']

conv2d_2 (Conv2D)               (None, 6, 16, 64)     76864       ['max_pooling2d_1[0][0]']

max_pooling2d_2 (MaxPooling2D)  (None, 3, 8, 64)      0           ['conv2d_2[0][0]']

dropout (Dropout)               (None, 3, 8, 64)      0           ['max_pooling2d_2[0][0]']

flatten (Flatten)               (None, 1536)          0           ['dropout[0][0]']

dense (Dense)                   (None, 512)           786944      ['flatten[0][0]']

dropout_1 (Dropout)             (None, 512)           0           ['dense[0][0]']

digit0 (Dense)                  (None, 10)            5130        ['dropout_1[0][0]']

digit1 (Dense)                  (None, 10)            5130        ['dropout_1[0][0]']

digit2 (Dense)                  (None, 10)            5130        ['dropout_1[0][0]']

digit3 (Dense)                  (None, 10)            5130        ['dropout_1[0][0]']

digit4 (Dense)                  (None, 10)            5130        ['dropout_1[0][0]']

==========================================================================================
```

Figure 5. Custom CNN model Summary

two sets of these Convolutional-MaxPooling pairs with the same parameters except for the number of neurons, which are set to 48 and 64, respectively. I have to note that all of the Convolutional layers have the “same” padding parameter.After the Convolutional layers, there is a 512 dense layer with the ReLU activation function and a 0.3 drop-out rate. The model summar can be seen Figure 5. Finally, L separate Softmax layers, where L is the number of expected characters in the CAPTCHA image. The optimizer used in Custom CNN is Adam and loss is calculated using Cross Entropy loss.

The intuition behind using Adam optimiser is its capability in training the network in a reasonable time. This can be easily inferred from results in next section in which the Adam optimiser achieves better results in comparison with Stochastic Gradient Descent (SGD) which was used din VGG11 and RESNET, but with a much faster convergence.

4. Results

4.1. Experimental Results

After developing the models successfully, the model was tested on different datasets. VGG11 and RESNET were tested on 10,000 images whereas customCNN was tested on 30,000 images.

4.1.1. VGG11

The experimental results on applying VGG11 model on the image dataset which has 10,000 images are shown in Figure 7 which shows the Training Accuracy of each digit , Figure 8 shows the Testing accuracy of each digit and Figure 9 shows the Training loss during the epochs.

4.1.2. RESNET

The experimental results on applying RESNET model on the image dataset which has 10,000 images are shown in Figure 10 which shows the Training Accuracy of each digit , Figure 11 shows the Testing accuracy of each digit and Figure 12 shows the Training loss during the epochs.

4.1.3. CustomCNN

The customCNN model implemented on auomatically generated . The code first randomly generates a code with permutations to avoid repetitions and more complexity using Python ImageCaptcha Library. The experimental results

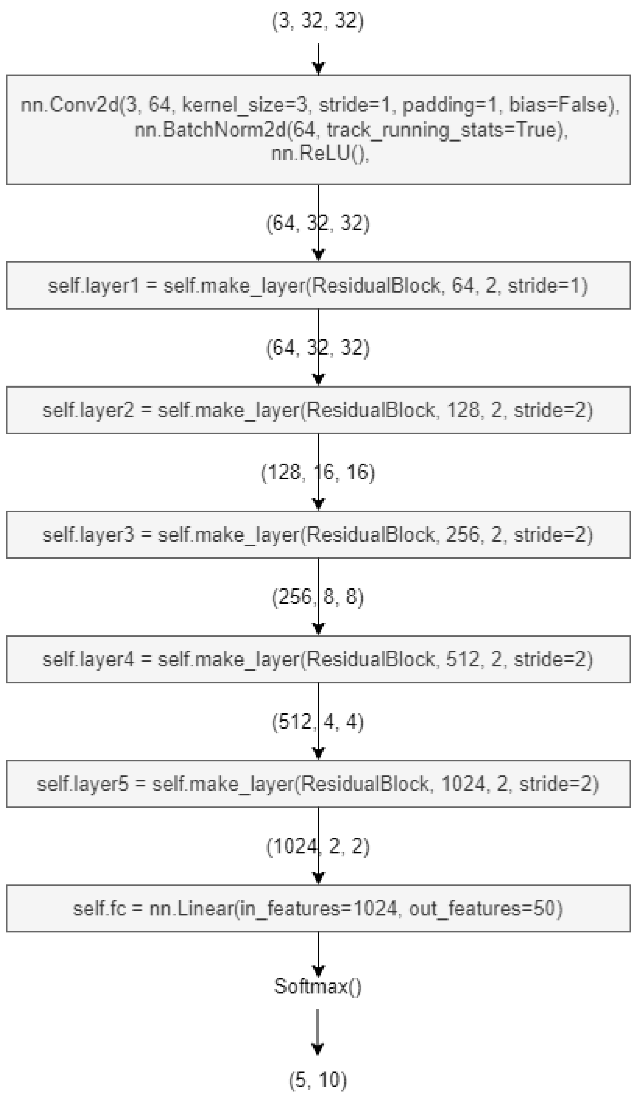

(3, 32, 32)
nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
nn.BatchNorm2d(64, track_running_stats=True),
nn.ReLU(),
(64, 32, 32)
self.layer1 = self.make_layer(ResidualBlock, 64, 2, stride=1)
(64, 32, 32)
self.layer2 = self.make_layer(ResidualBlock, 128, 2, stride=2)
(128, 16, 16)
self.layer3 = self.make_layer(ResidualBlock, 256, 2, stride=2)
(256, 8, 8)
self.layer4 = self.make_layer(ResidualBlock, 512, 2, stride=2)
(512, 4, 4)
self.layer5 = self.make_layer(ResidualBlock, 1024, 2, stride=2)
(1024, 2, 2)
self.fc = nn.Linear(in_features=1024, out_features=50)
Softmax()
(5, 10)

Figure 6. Architectural representation of RESNET model

are as below where Training accuracy is shown in Figure 13, Validation accuracy in Figure 14 and Training loss in figure 15.

4.2. Performance Analysis

After implementing the models, there has been a large distinction between the performance of models on the similar type of images.

After applying VGG11 which is a large deep neural network , the training accuracy went as high as 63.7% for Digit 0 as shown in Figure 16. Similarly, 59.62 % for Digit 1 , 60% for Digit 2 , 59.9% on Digit 3 and 59.02% on Digit 4. Basically, it is same every digit averaging out to almost 60% for each

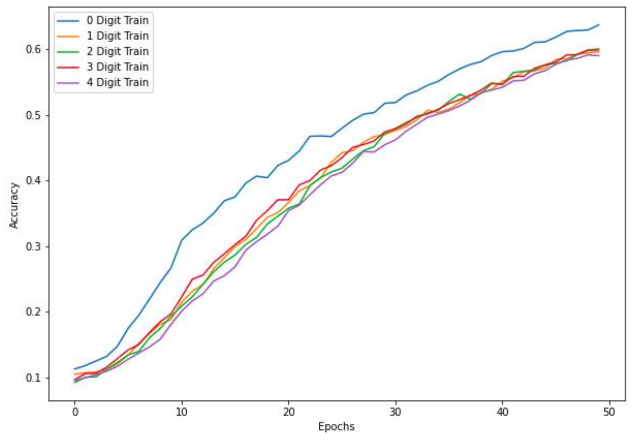


Figure 7. VGG11 training accuracy

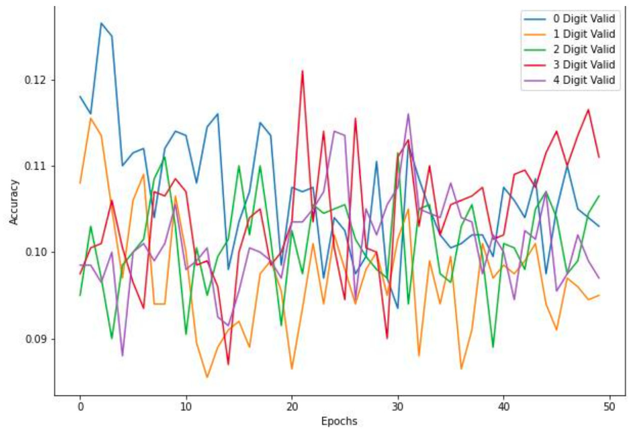


Figure 8. VGG11 Validation accuracy

digit.

The testing results on VGG11 were not impressive as it gave only 10% average accuracy for each digit which show the model did not work well for CAPTCHA dataset.

Similarly analysing the results on RESNET dataset, the training accuracy went upto around 50 % average accuracy for each digit. The architecture of resnet model assures good results on image dataset as it is contains deep layers of networks. The model is too large for this dataset.

The training accuracies of both the models are shown in the Table 1 for each digit where both models did almost equally well. Both the models trained the data well and gave fair results on seen dataset whereas testing accuracy went as low as 10% on both the models. The reason for their bad performance can be different.

Further, when applied a short custom CNN model on auto

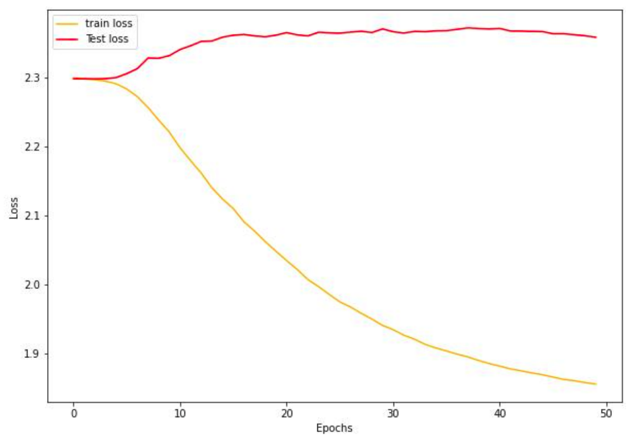


Figure 9. VGG11 training LOSS

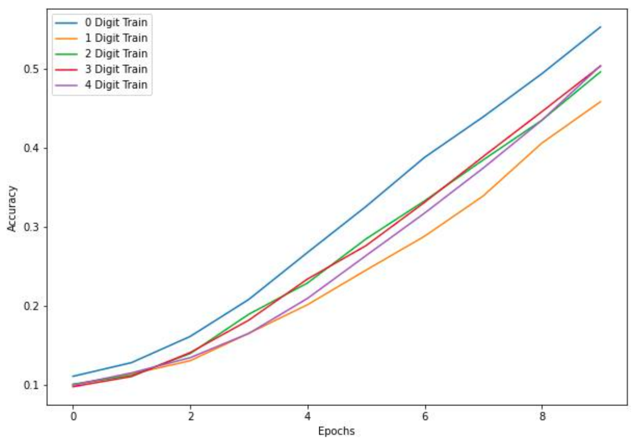


Figure 10. RESNET training accuracy

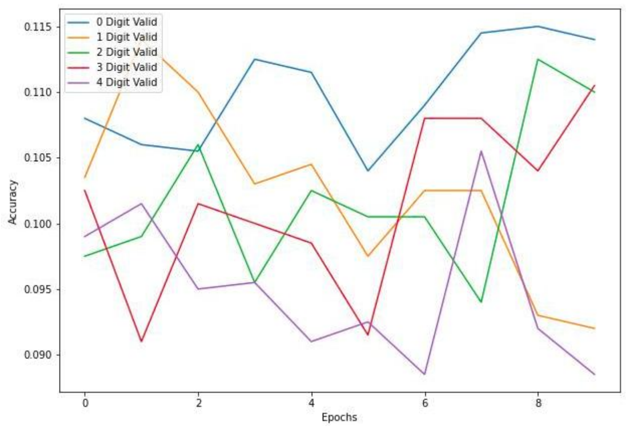


Figure 11. RESNET validation Accuracy

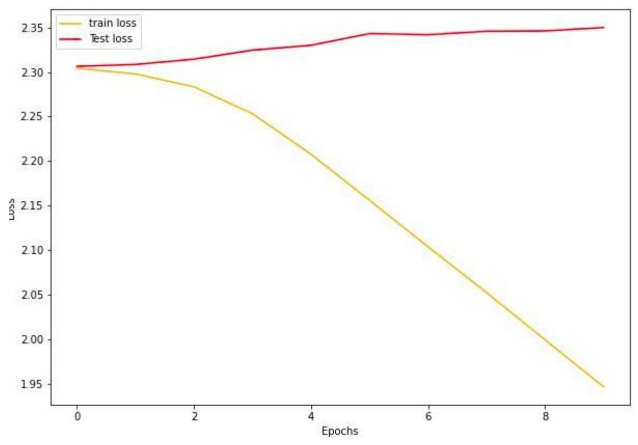


Figure 12. RESNET loss

generated images, the model worked the best on images for identifying the correct code in the image. The Train accuracy reported is 99.93 % whereas the testing accuracy is 93.67 % which shows the model did well in decoding the CAPTCHA code inside the obfuscated images. The length of dataset used is three times the dataset used in previous models.

As discussed above in the architecture section, the VGG11 model included Conv2d layers followed by Batch norm and max pool layer. This combination is repeated over 8 layers and then flattened thereafter. Three linear layers along with the layer of dropouts of 0.3 value has been applied. Further, the model is reshaped to get the output in 5 x 10 format. The model itself is very complex and deep. Alongside, the images are not that clear and gray scale which made VGG11 more complex and hard to decode the value of CAPTCHA inside images.

One of the reasons of bad performance of VGG11 can be the quality of images which needs more pre-processing before feeding into the network as mentioned in (Wei & Wang, 2019) which did image segementation before applying VGG16 and predicting the output digit by digit.

Other reason for relatively bad performance is the length of dataset. The dataset used in VGG11 and RESNET is 10,000 images long whch is less than Custom CNN.

Comparing the Custom CNN model with already existing model gives the project a broader aspect to analyse the performance. This model really outperformed other models which involvedd similar proprocessing and same parameters. So, for this type of CAPTCHA dataset which involves noisy data, we can affirm that smaller models can work well for that dataset.

Digit	VGG11	ResNet	CustomCNN
1	63.71	55.30	99.99
2	59.62	45.83	99.99
3	60.05	49.62	99.98
4	59.91	50.35	99.98
5	59.02	50.40	99.99

Table 1. Training accuracies for VGG11 and ResNet models for different digits of captcha

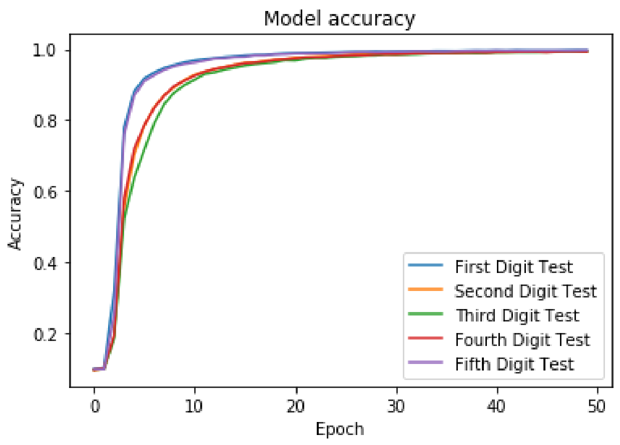


Figure 13. CUSTOM CNN TRAINING ACCURACY

4.3. Conclusion

To show the strengths and shortcomings of common CAPTCHA generators, the project aims at creating, customising, and optimising a CNN-based deep neural network for numerical based CAPTCHA detection. The use of a sequence of paralleled Softmax layers improved detection significantly.

In comparison to the prior 90.04 percent accuracy rate in the same network,imrpovement in the model got accuracy up to 93.67 percent accuracy using only the Sigmoid layer in the architecture of model. The observation made using the results of VGG11 and RESNET is that using large and deep networks is not always a good choice. Sometimes, small and intelligent models predicts accurately.

Although the algorithm was relatively accurate in CAPTCHAs that were somewhat random, several conditions made it incredibly difficult for Custom CNN to crack them. I believe that considering the difficulties raised can aid in the creation of more trustworthy and robust CAPTCHA samples, making them more complicated and less likely to be cracked.

I propose solving CAPTCHAs with variable character length as a feasible avenue for future work, which is not

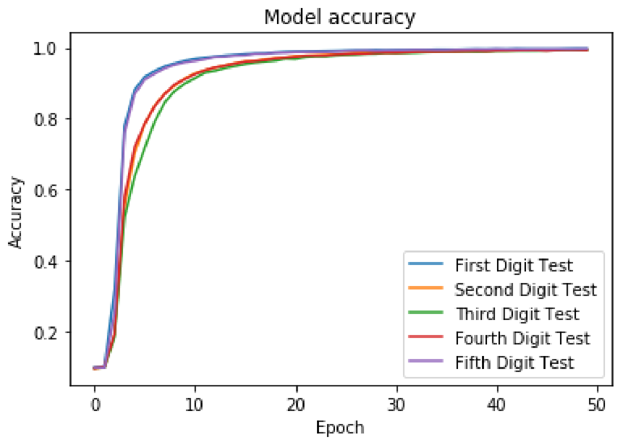


Figure 14. CUSTOM CNN VALIDATION ACCURACY

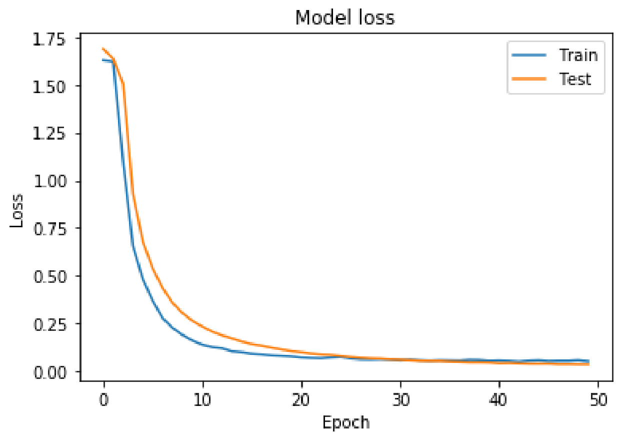
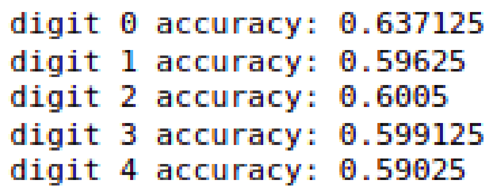


Figure 15. CUSTOM CNN LOSS

limited to numerical characters but also applies to mixed problematic alpha-numerical characters, as stated in section 4. Further research on the application of Recurrent Neural Networks as well as traditional image processing approaches to extract and recognise the CAPTCHA characters, individually, is also recommended

I looked at (Wang & Pan, 2019) research, which included evaluations of the following approaches: DenseNet-121 and ResNet-50 are fine-tuned versions of the original DenseNet and ResNet networks for solving CAPTCHAs, respectively, as well as DFCR, an optimised approach based on the DenseNet network. The DFCR claims to have a 99.96 percent accuracy rate, which is the highest among other approaches. This model, on the other hand, has only been trained on a few thousand samples and solely on four-digit CAPTCHA images. Although the (Wang & Pan, 2019) 's



```
digit 0 accuracy: 0.637125
digit 1 accuracy: 0.59625
digit 2 accuracy: 0.6005
digit 3 accuracy: 0.599125
digit 4 accuracy: 0.59025
```

Figure 16. VGG11 training accuracy values

proposed model outperforms all suggested techniques in the quantitative comparison in Table 1, the method's validity cannot be proven on larger datasets or complex alphanumerical CAPTCHAs.

Therefore, the main focus comes upon the quality of preprocessing done on the images before feeding in to the network.

Finally, both numerical CAPTCHAs appear to yield relatively satisfactory results using Custom CNN. With a simple network architecture, we can more easily use this network for different reasons. Furthermore, compared to state-of-the-art, having an automated CAPTCHA generating technique allows us to train our network with greater accuracy while preserving detection of more complex and comprehensive CAPTCHAs.

References

Bostik, O. and Klecka, J. *Recognition of CAPTCHA characters by supervised machine learning algorithms.* IFAC-PapersOnLine, 6 edition, 2018.

Garg, G. and Pollett, C. Neural network captcha crackers. Technical report, Future Technologies Conference (FTC), 2016.

George, Dileep, W. L. K. K. M. L. G. C. L. B. M. X. L. e. a. A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science 358, no. 6368*, 2017.

Karthik, C.-P. and Recasens, R. A. Breaking microsoft's captcha. Technical report, Technical report, 2017.

Kwon, Hyun, H. Y. and Park, K.-W. Captcha image generation using style transfer learning in deep neural network. Technical report, International Workshop on Information Security Applications pp. 234–246. Springer, Cham, 2019.

Osadchy, Margarita, J. H.-C. S. G. O. D.-m. and Pérez-Cabo, D. No bot expects the deepcaptcha! introducing immutable adversarial examples, with applications to captcha generation. Technical report.

Rezaei, M. and Klette, R. Look at the driver, look at the road: No distraction! no accident! Technical report, CVF Computer Vision and Pattern Recognition, 2014.

Sivakorn, Suphannee, I. P. and Keromytis., A. D. (eds.).

Stark, Fabian, C. H.-R. T. and Cremers, D. *Captcha recognition with active deep learning.* PhD thesis, Workshop newchallenges in neural computation, 2015.

Wang, Jing, J. H. Q. X. Y. X. Y. T. and Pan, N. Captcha recognition based on deep convolutional neural network. Technical report, Math. Biosci. Eng. 16, no. 5, 2019.

Wei, Li, X. L. T. C. Q. Z. L. Z. and Wang, W. Research on optimization of captcha recognition algorithm based on svm. Technical report, In Proceedings of the 2019 11th International Conference on Machine Learning and Computing, 2019.

Ye, Guixin, Z. T. D. F. Z. Z. Y. F. P. X. X. C. and Wang, Z. Yet another text captcha solver: A generative adversarial network based approach. Technical report, In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 332-348., 2018.

Yousef, Mohamed, K. F. H. and Mohammed, U. S. Accurate, data-efficient, unconstrained text recognition with convolutional neural networks. arXiv preprint, 2018.

Zahra Noury, M. R. Deep-captcha: A deep learning based captcha solver for vulnerability assessment. Technical report, Faculty of Computer and Electrical Engineering, Qazvin Azad University Faculty of Environment, Institute for Transport Studies, The University of Leeds, 2021.

Zhao, Nathan, Y. L. and Jiang, Y. Captcha breaking with deep learning. Technical report, 2017.