Text based CAPTCHA recognition using Deep Learning

Deepthi M Rao
Georgia Institute of Technology
Atlanta, Georgia
drao38@gatech.edu

Harikiran Cherala
Georgia Institute of Technology
Atlanta, Georgia
hcherala3@gatech.edu

Abstract

Advances in deep learning allow us to solve complex AI problems and achieve good results. One such problem is text recognition. In this paper, we focus on finding a way to capture the text from visual text based Completely Automated Public Turing tests to tell Computers and Humans Apart (CAPTCHA) using deep learning algorithms. We propose a CRNN (Convolutional Recurrent Neural Network) based architecture to crack the output of alpha-numeric 5 character visual CAPTCHA. In this paper, we experiment with different CRNN architectures and report results on different evaluation metrics. Our final CRNN network achieves an exact measure accuracy of 98.8% and a cosine similarity of 99.5% on our test data.

1. Introduction

CAPTCHA - Completely Automated Public Turing tests to tell Computers and Humans Apart is a computer test that was designed to tell whether the user is a human or a robot. Hence, it can help users from spam and password decryption by ensuring that its not a machine/robot trying to access the end user's password protected account. Intuitively, CAPTCHA tries to provide problems for end users to quickly solve which would otherwise be difficult for a machine or a robot to solve.

One of the most common types of CAPTCHA used today is a text-based one, where a random sequence alphanumeric characters with random noise and distortions are drawn on an image and shown to the user. The user then must enter the sequence of characters or digits correctly in order to proceed. We have built a deep learning model that can perform text recognition from the visual text-based CAPTCHA images and give the resulting sequence or text as output.

In our literature survey we found a lot of papers that solved the problem of text recognition of text-based CAPTCHA using deep convolutional neural networks (DCNN). RNNs which are mainly used for identification of sequences were not very popular. Deep-Captcha [8] uses 3 convolution and max-pooling layers and 2 fully connected layers. The final layer consists of 5 fully connected layers where each layer consists of total number of units equal to the vocabulary used with a softmax activation function to generate probabilities. They focus on solving a 5 digit text-based CAPTCHA. Stark, et al. [12] uses a similar CNN and focuses on 6 digit CAPTCHA. They use active learning method to add more training data for training. Zhang, et al. [16] perform thresholding operations on the input and feed into a deep CNN to perform the task. They also use an ensemble approach by combining multiple base classifier and using majority voting approach. Huang et al. [7] uses a simple CNN with residual connections and image thresholding operations to perform the task of recognition of a 5 digit CAPTCHA. Osadchy et al. [9] perform adversarial attack by adding noise onto the original image, thus causing the classifier to misclassify while the image still looks the same for humans. Y Hu et al. [5] uses Alex-Net [6] based architecture to perform feature extraction. J Wang et al. [14] uses a modified DenseNet architecture as CNN and a final fully connected classification layer for 4 digit CAPTCHA. A Thobhani et al. [13] create unique binary image for every character in CAPTCHA and attach it to a copy of the input image and use them for training a DCNN. All the models described use DCNN architectures to solve the given problem. One disadvantage of this is that the network has to be changed when the number of digits in input CAPTCHA text labels change and can become more complex. A Priya et al. [10] use Inception net v3 [6] to perform feature extraction and then use an RNN framework to predict the final results. We have used a simple CRNN (Convolutional Recurrent Neural Network) architecture and modified the RNN layers to solve our given problem on a very simple dataset.

Deep Learning models that help solve CAPTCHA might sound counter-intuitive in terms of its main purpose to distinguish humans and machines, but these models can be used to generate harder set of visual text images that cannot be easily recognized by a machine but can be recognized by a human. Hence, different types of CAPTCHA have been evolved such as selecting appropriate images, objects, etc. but we focus mainly on the text-based ones. Also, some websites do not have the audio support for visually impaired people that could help them in solving CAPTCHA and get through to the website. Hence, deep learning models can be used to assist the visually impaired community.

The dataset used for solving this problem was given by Wilhelmy, Rodrigo Rosas, Horacio. (2013). captcha dataset [15]. This dataset consists of 1070 samples of text-based CAPTCHA images. The dimensions of each image is 200 x 50 (width x height) and consists of a random combination of 5 alpha-numeric characters. The labels for each image are present in the name of the image. Figure 1 shows a sample of images with their labels from the training data.

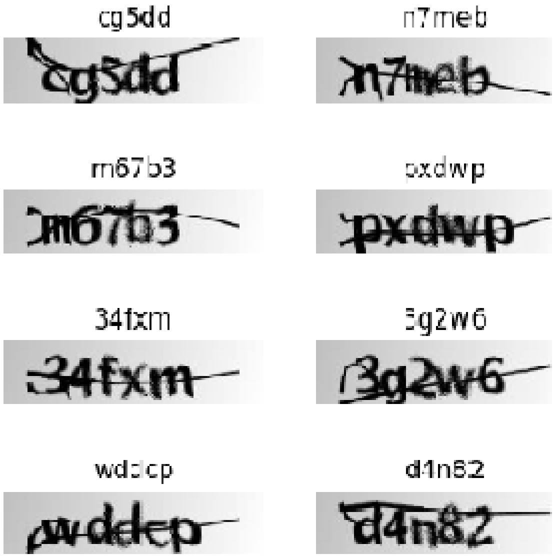


Figure 1. Training samples before pre-processing

2. Approach

We chose the CRNN (Convolutional Recurrent Neural Network) [11] architecture to tackle this problem. RNNs (Recurrent Neural Networks) were designed for identifying sequences. The hidden state present in RNNs helps in remembering some information regarding the sequence. We have experimented with 2 different types of RNN architecures - LSTM (Long Short Term Memory) and GRU (Gated Recurrent Unit). We have experimented with 5 different variations of CRNN architecture and trained our dataset on all the 5 models and reported and compared the results across various evaluation metrics. The basic structure and working of CRNN is described in the Sections 2.1 to 2.5.

2.1. Preprocessing

The set of characters used in our dataset are from the universal set comprising of the alphabet A-Z in both upper case and lower case and the digits 0-9. We sort all of the characters and use this as the vocabulary and hence the total length of the vocabulary is 62. vocab = ”0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz”

We define a layer that maps all characters to integers using Tensorflow keras layer which takes input as the vocabulary. The output will be the current index of the character in the vocabulary plus 1. For example, the mapping of the character ’A’ in the vocabulary will be 11 as it is present at the index 10 in our list. An inversion of this layer is also created which maps the number back to its respective character.

All the images in the dataset are read using the tensorflow’s image decode API. While reading the image, they are converted from their original RGB format to a gray-scale format. In RGB format, the number of channels are 3 while in gray-scale the number of channels will be reduced to 1. There is no loss of information in this colour conversion process as the text required for recognition will not change and colour is not a defining characteristic for text in terms of deep learning models. We then normalize the pixel values that lie in the range of $[0, 255]$ in a gray-scale image to $[0, 1]$. When the image is read, the format of the input dimension is set as HWC (height, width, number of channels). We transpose the normalized image to WHC format before using it for training.

The label for the corresponding image is split into its individual characters and fed into the character to number mapping layer that was previously, which will map each character in the label to its corresponding number in the vocabulary.

2.2. Convolution Neural Network

The convolutional neural network is used to extract sequential feature maps from the input image. These feature maps are produced by a sequence of different Convolutional layers and Max-pooling layers. All the convolutional layers used were 2D convolutional layers with a kernel size of 3 x 3 and ReLu (Rectified Linear Unit) activation function was used for all convolutional layers and a 2 x 2 kernel for max-pooling layers thus downsizing the input by half. In one of our models, we use a max-pooling layer with unequal kernel dimension (1 x 2) in order to only downsize the height dimension and not the width, thus preserving some features of the text in the width dimension.

The feature maps obtained from the CNNs are translation invariant. Hence, each column in the feature vector corresponds to a particular region in the image. This is known as the receptive field of the model which is defined as a region in the input image that the feature map is looking at. Dropout layers have been used in some of the models which random sets the input units to 0 to prevent overfitting. Batch normalization layers have also been used to normalize the inputs to a layer for each mini-batch and thus reducing the number of epochs required for the model to learn.

2.3. Recurrent Neural Network

RNNs are defined for problems that involve sequences and text-based CAPTCHA is one such example that involves a sequence of alpha-numeric characters. RNNs are directional, it uses only past contexts to derive sequences for the future. For image based text recognition, both past and future contexts will be very helpful, hence we use a bidirectional RNN based architecture that provides a look-ahead capability and hence both past and future contexts will be used. By stacking a series of bidirectional RNNs we obtain deep bidirectional RNN. In our models we have experimented with LSTMs and GRUs architectures.

LSTM (Long Short Term Memory) is an RNN architecture that is specifically designed to address the vanishing and exploding gradient problem faced in the traditional RNN architecture. LSTM consists of additional gates such as the forget gate and input gate that allow a better control over the gradient flow and help in preserving long term dependencies.

GRU (Gated Recurrent Unit) is also another type of RNN architecture that is very similar to LSTM, but has less number of parameters as it lacks the output gate. GRUs have been shown to perform better than LSTMs in certain smaller and less frequent datasets. [2] [4]

2.4. Attention Mechanism

In our approach, we have experimented with attention mechanism and achieved better performance. The attention mechanism [1] was introduced to improve the performance of an encoder-decoder architecture. The idea behind this was for the decoder to utilize the most important parts of the encoder's input by performing a weighted combination of the input and the most relevant input gets assigned the highest weights.

Let us assume that x is the input, W is the weights, b is the bias and the current position as t. Then attention mechanism can be defined as follows:

$$e_t = f(W^T x + b)$$
$$a_t = softmax(e)$$
$$output = \sum_{i=0}^{T} x_i a_{t,i}$$

The f(.) function represents any activation function. We used tanh activation function which takes any real value as input and outputs a value between -1 and 1. Tanh is calculated as follows:

$$\text{Tanh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Softmax(.) is also an activation function that converts a vector of numbers into a vector of probabilities that sum up to 1. It is defined as follows:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

The weights vector is randomly initialized with a normal distribution whereas the bias vector is randomly initialized with all zeros. Attention mechanism helps the decoder in finding the parts of the input image that correspond to a character.

The decoder in our case is a simple fully connected layer consisting of 63 neurons (each representing a character in the vocabulary and a blank character) with a softmax activation function. This is also the output layer for the model excluding the loss calculation layer which is added after this layer for computing loss.

2.5. CTC decode and loss

After passing through the CRNN network, (with or without attention) and the final output layer, we get softmax probabilities over the vocabulary. Based on just the mapping from image to text, one should be able to calculate the loss without worrying about the alignment of character to the input image's location. One such loss can be calculated using Connectionist Temporal Classification (CTC) [3].

2.5.1 CTC decode

CTC decode operation consists of two simple operations - merging repetition of characters and removal of blank characters. For example, if after mapping the output, we get a sequence like $--hh-ee-lll-ll-ooo---$ where $-$ represents a blank character, the CTC decode operation will output $hello$.

2.5.2 CTC loss

For explaining CTC loss, lets assume that the vocabulary is $vocab = \{G, T, -\}$. From this we will have 3 predictions (softmax probabilities) over our vocabulary. Lets assume that our ground truth label is GT. Then, loss is calculated as negative log likelihood of getting the ground truth.

CTC loss $= -log(Probabilityofgettingthegroundtruth)$

Hence, in our scenario, the $Probability("GT") = P(GTT) + P(GGT) + P(G-T) + P(-GT) + P(GT-)$.

These probabilities can be calculated from the softmax probabilities obtained. The loss for a perfect match will be 0, whereas the loss for a perfect mismatch will tend to infinity.

Hence, as explained in the previous sections, we built 5 different models following the same CRNN architecture. Initially we built a simpler model with less convolutional and max pooling layers and got the results. We improved these results by following the CRNN architecture with LSTM as explained in [11]. We then modified the RNN architecture by utilizing GRU and adding attention mechanism and improved our results. We believe that the utilization of GRU with attention mechanism has not been explored for recognition of text-based CAPTCHA and it is our novel approach. A basic description of all the models and their computation graphs are shown in the next section.

3. Experiments and Results

We built 5 different models following a CRNN architecture and all the models' computation graphs have been derived using the software Netron and shown in Section 3.1. To evaluate the performance of our models, we have defined several evaluation metrics as described in Section 3.3. We have described the various implementation details for training such as the hyper-parameters, hardware used, code backend and dataset splitting in Section 3.2. Section 3.4 shows the training graphs containing epochs vs loss for all the models and reports their overall performance on the test set.

3.1. Different Model definitions

We started by building a simpler model and then used a similar architecture as explained in the CRNN paper. We then changed the LSTM layers to GRU and added attention mechanism to achieve better performance. The following sections show the computation graphs for all the models. All the models use same CTC layer to compute loss.

3.1.1 Model 1 - Simple CRNN model with LSTM

This model consists of 3 convolutional blocks where each block consists of a 2D convolution layer and max-pooling layer. Each 2D convolution layer uses a kernel size of $(3*3)$ and ReLu activation function. The kernel used in the max-pooling layers is of dimension $(2*2)$ which downsizes the input by half. The number of filters used in the 3 blocks are 32, 64 and 128 respectively. Then a fully connected layer with 128 neurons and ReLu activation function is added and a dropout layer with a dropout rate of 0.2 is used. We then constructed a deep bidirectional LSTM layers by using 3 such layers. Its architecture is shown in Figure 2.

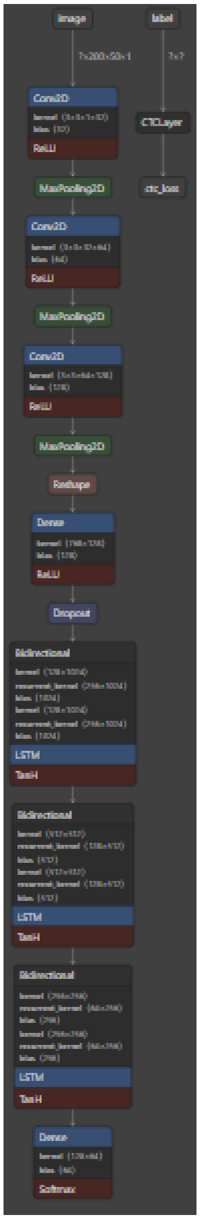


Figure 2. Model 1 architecture

3.1.2 Model 2 - Full CRNN model with LSTM

This model consists of the exact model architecture as mentioned in the CRNN paper [add ref]. It consists of 7 2D convolution layers and batch normalization layers have also been integrated. 2 max-pooling layers with a kernel with odd dimensions $(1*2)$ have been used to preserve the feature maps in the width dimension of the input and downsize it only in the height dimension. This will help in capturing better feature maps. Also, increasing the depth of the CNN compared to Model 1, will lead to better feature representations of the input image, thus leading to a better text recognition. The architecture is shown in Figure 3

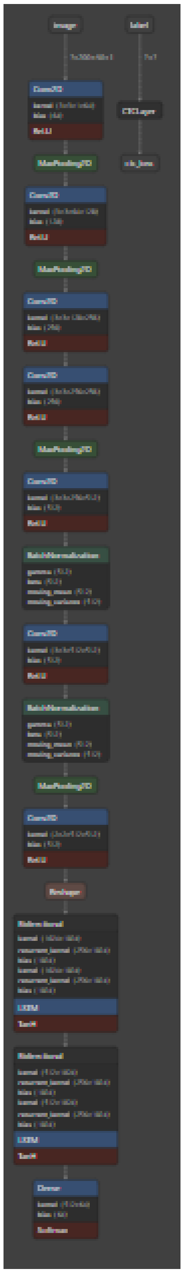


Figure 3. Model 2 architecture

3.1.3 Model 3 - Full CRNN model with GRU

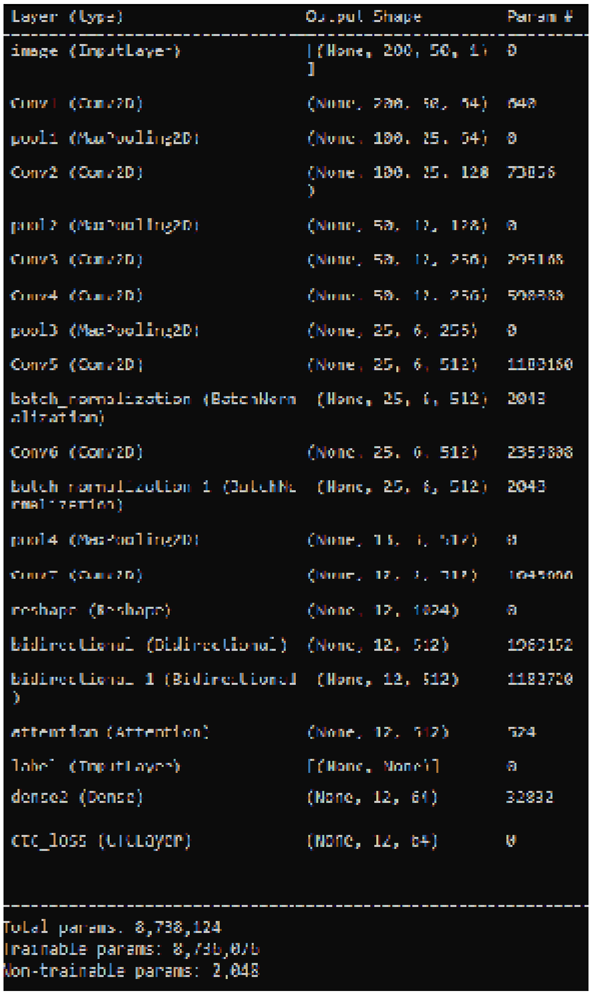
This models adapts the exact same architecture for the CNN part of the Model 2, but the RNN part is changed from LSTM to GRU. We decided to use GRU as it is less complex than an LSTM because it consists of less number of gates than LSTM thus reducing the number of learnable parameters. GRUs are simpler and perform well on small training data with smaller sequences. Since our dataset consists of only 1070 samples and each label has a length of 5, we decided to experiment with GRU.

3.1.4 Model 4 - Full CRNN model with LSTM+Attention

An attention layer is implemented as described in Section 2.4 and added to the model after the deep bidirectional LSTM network and before the output fully connected layer. With attention, the model will be able to focus on the valuable part of the input (which in this case is the output of the bi-directional LSTM layers) and improve the performance. The number of trainable parameters are 9,782,604.

3.1.5 Model 5 - Full CRNN model with GRU+Attention

As explained in Model 4, instead of LSTM layer, GRU layer is used. We built a model combining GRU and attention mechanism to sum up their advantages and achieve much better performance. The total number of trainable parameters for this model are 8,736,076 which is clearly less than that of Model 5. The architecture for model 5 is shown in Figure 4.



```
 Layer (type)                  Output Shape         Param #
===========================================================
 image (InputLayer)            [(None, 200, 50, 1)  0
                               ]

 Conv1 (Conv2D)                (None, 200, 50, 64)  640

 pool1 (MaxPooling2D)          (None, 100, 25, 64)  0

 Conv2 (Conv2D)                (None, 100, 25, 128  73856
                               )

 pool2 (MaxPooling2D)          (None, 50, 12, 128)  0

 Conv3 (Conv2D)                (None, 50, 12, 256)  295168

 Conv4 (Conv2D)                (None, 50, 12, 256)  590080

 pool3 (MaxPooling2D)          (None, 25, 6, 256)   0

 Conv5 (Conv2D)                (None, 25, 6, 512)   1180160

 batch_normalization (BatchNorm  (None, 25, 6, 512)  2048
 alization)

 Conv6 (Conv2D)                (None, 25, 6, 512)   2359808

 batch_normalization_1 (BatchNo  (None, 25, 6, 512)  2048
 rmalization)

 pool4 (MaxPooling2D)          (None, 12, 3, 512)   0

 Conv7 (Conv2D)                (None, 12, 2, 512)   1049088

 reshape (Reshape)             (None, 12, 1024)     0

 bidirectional (Bidirectional)  (None, 12, 512)     1969152

 bidirectional_1 (Bidirectional  (None, 12, 512)    1182720
 )

 attention (Attention)         (None, 12, 512)      524

 label (InputLayer)            [(None, None)]       0

 dense2 (Dense)                (None, 12, 64)       32832

 ctc_loss (CTCLayer)           (None, 12, 64)       0

===========================================================
Total params: 8,738,124
Trainable params: 8,736,076
Non-trainable params: 2,048
```

Figure 4. Model 5 architecture

3.2. Implementation details

The code for all the models has been written using keras with tensorflow as back-end in python. We randomly sampled 100 images (10%) from our dataset of 1070 samples as test set. The remaining images were randomly split into training (90%) and validation (10%). Tensorflow's training and validation dataset was created after preprocessing every image as explained in Section 2.1 and associating its respective label. For each model, we created callbacks that were helpful in storing logs such as training and validation loss values and a checkpoint or weights of the model was saved in the epoch which showed the least validation accuracy. We each shared the training of models amongst ourselves on our systems comprising of RTX 2060 GPU and AMD Ryzen 5 3600 CPU and RTX 2060 and Intel Core i7 10th Gen CPU with the latest CUDA version and Tensorflow 2.8.0.

For all the models, we chose Adam optimizer for performing gradient descent. The hyper-parameters used for training all models are given in the table below:

Hyperparameter	Value
Epochs	200
Learning Rate	0.001
Batch Size	32
Optimizer	Adam
β_1	0.9
β_2	0.999
ϵ	1e-07

Table 1. Hyper-paremeters used for training

All the hyper-parameters were chosen as the default setting given by the underlying libraries used.

3.3. Evaluation Metrics

To evaluate our models and compare them we have defined certain evaluation metrics - jaccard index, levenshtein ratio, cosine similarity and exact match percentage. For all the 5 models, we calculate all the below evaluation metrics and report the average value across 100 test images.

3.3.1 Jaccard Index (JI)

Jaccard similarity index is also defined as the Intersection over Union. It tells us how 2 words are close to each other by calculating how many common characters exist among the predicted label and ground truth over the total number of words. Let A be the set of characters in the ground truth and B be the set of characters in the predicted label, then Jaccard Index is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

The value lies between 0.0 and 1.0 and higher the value, the more similar are the 2 words.

3.3.2 Levenshtein Ratio (LR)

Levenshtein distance is defined as the total number of edits (insertion, deletion or replacement) required to transform the predicted text sequence to the ground truth sequence. Levenshtein ratio is defined as:

$$LevenshteinRatio(A,B) = \frac{total\_len - levenshtein\_dist}{total\_len}$$

where $total\_len$ is defined as the total number of characters present in both the prediction and ground truth. The operation of each replacement has a cost of 2 while calculating the ratio. This ratio lies between 0.0 and 1.0 where 0.0 means perfect mismatch and 1.0 means perfect match.

3.3.3 Cosine Similarity (CS)

Cosine similarity between 2 vectors is defined as:

$$S_c(A,B) = \frac{A.B}{\|A\| \, \|B\|}$$

Each word is converted into a vector by the count of characters present in the word. For example, the text '*aabc*' will be converted to a set consisting of $\{a,b,c\}$ and each element will be mapped to $2, 1, 1$ respectively. Cosine similarity value lies between 0.0 and 1.0 and higher the value, higher the similarity.

3.3.4 Exact Match measure (EM)

This evaluation metric was added specific to our problem as, its important to identify the text correctly with a perfect match in CAPTCHA. This is simply calculated as follows assuming that A is the ground truth:

$$e\_m(x) = \{1 \;\; if \;\; x = True; 0 \;\; if \;\; x = False\}$$

$$ExactMatch(A,B) = \frac{\sum_{i=0}^{|A|} e\_m(A_i == B_i)}{|A|}$$

3.4. Results

This section shows all the training graphs for all the 5 models. The training graph is plotted for Number of epochs (x-axis) vs CTC loss (y-axis). The legend for all graphs shown in the following section is: Blue - Validation and Orange - Training.

3.4.1 Training Graphs

From Figure5, we can observe that it takes more number of epochs than the other models to converge. This can be due to the fact that its a simpler model consisting of smaller number of convolution blocks.

From Figure 8 and Figure 9, we can see that the loss remains steady for approximately 20 epochs in the beginning and then converges compared to the other graphs shown in Figure 6 and 7. This can be due to the application of attention mechanism in these models.

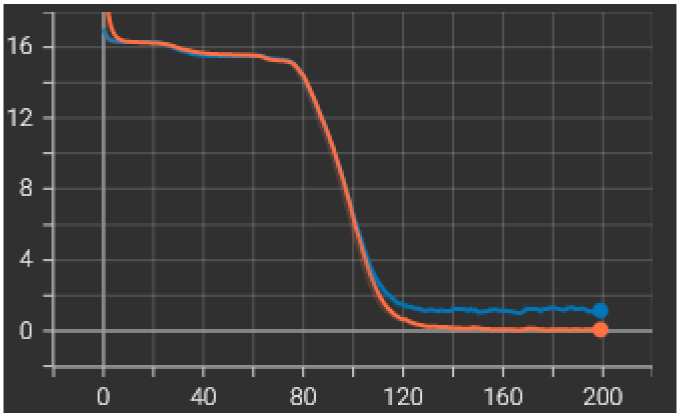


Figure 5. Model 1 - Epochs vs CTC Loss

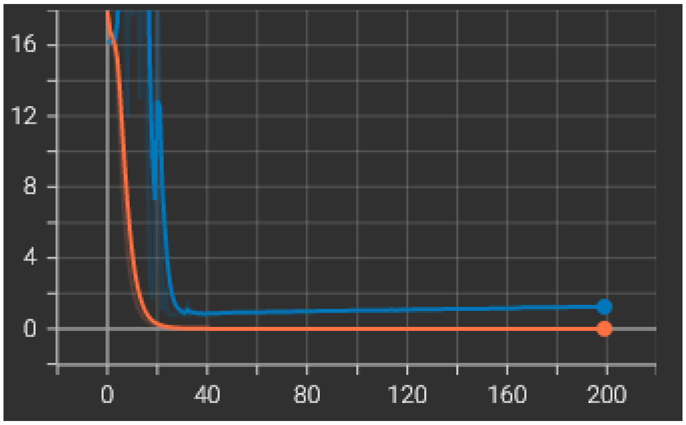


Figure 6. Model 2 - Epochs vs CTC Loss

3.4.2 Models' Test Results

The table 2 shows the test results for all the models across all the evaluation metrics. From the table we can observe that, Model 5 - Full CRNN model with GRU+Attention performs the best with 98.8% exact match score. This is due to the ensemble of the benefits of using both GRU and attention mechanism.

Among the LSTM models, model 4 which is a combination of LSTM with attention performs best with an exact match score of 96.8% again showing the improvement by using attention mechanism.

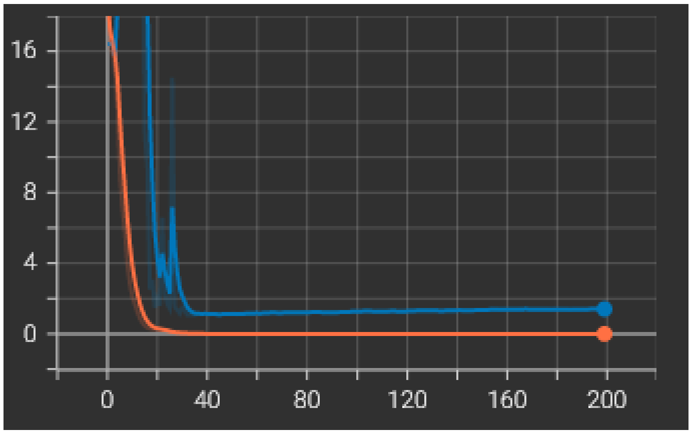


Figure 7. Model 3 - Epochs vs CTC Loss

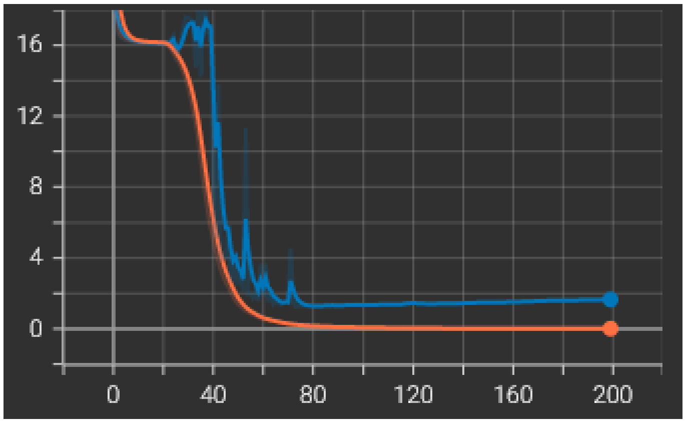


Figure 8. Model 4 - Epochs vs CTC Loss

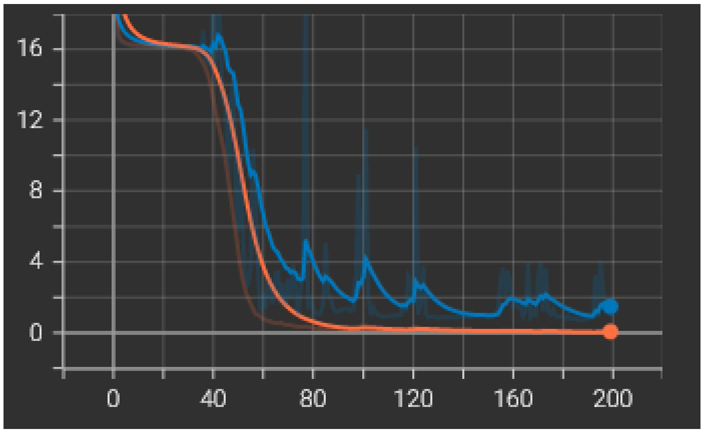


Figure 9. Model 5 - Epochs vs CTC Loss

We can also clear improvement in performance from the simpler model to the full CRNN model architecture because the depth of the CNN increases and better feature representations are learnt.

Model	JI	LR	CS	EM
Model 1	84.0	96.4	97.4	94.2
Model 2	81.0	95.9	96.6	92.8
Model 3	89.0	96.4	98.4	95.6
Model 4	91.0	98.2	98.6	96.8
Model 5	**97.0**	**99.4**	**99.5**	**98.8**

Table 2. Models Results on Test set

4. Conclusion

We were able to achieve cosine similarity of 99.5% and an exact match measure of 98.8% with our CRNN modified model - Model 5. We chose a task of text recognition from text-based visual CAPTCHA using a simple dataset in order to learn about various text recognition models and learn how to pre-process, build, train and infer models using popular deep learning frameworks. Initially we face some challenges learning a new deep learning framework - Tensorflow and several other basic challenges such as setting up of GPU for training. We also faced some challenges while implementation of our models, like deciding layers, number of filters, kernel size, etc. We were able to go through these challenges easily by applying a joint effort and discussions amongst us to learn the various aspects required for this project. From this project, we learnt about various types of architectures used for text-recognition, building an training of models using keras, usage of callback functions to store important results during training, implementation of different layers and loss functions in keras and integrating them into our models and a simple visualization of our models' computation graphs using a third party software. Overall, we were able to learn and implement some of the techniques used in deep learning and was also able to achieve a good result on our test set with a novel approach of using GRU with attention mechanism in the domain of solving text-based CAPTCHA using deep learning.

5. Future Work

We experimented only with CRNN type of architecture and in the future multiple DCNN architectures can be experimented with and the performance can be assessed. We trained our models only on a single dataset, hence in the future multiple datasets can be used to train our models and performance can be reported on all datasets. Our model implementations can easily adapt to any dataset where the dataset is stored in a folder and each image's name corresponds to the label of the image. We were able to train our models with default hyper-parameters and achieved good accuracy. In the future, for all the models, hyper-parameter tuning can be done by varying batch size, learning rate, epochs, optimizer, etc. and performance can be compared. Several synthetic data generations also can be looked at and used for training our models and DCNN models and check the performance.

6. Work Division

The table 3 shows our contributions to the project. We equally contributed to this task and learnt various new things by creating an agenda and holding discussions.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473. 3

[2] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555. 3

[3] A. Graves. Connectionist temporal classification. In *Supervised sequence labelling with recurrent neural networks*, page 61–93. Springer, Berlin, Heidelberg. 3

[4] N. Gruber and A. Jockisch. Are gru cells more specific and lstm cells more sensitive in motive classification of text? *Frontiers in artificial intelligence*, 3:40. 3

[5] Y. Hu, L. Chen, and J. Cheng. A captcha recognition technology based on deep learning. In *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA*, page 617–620. IEEE. 1

[6] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, page 25. 1

[7] S. Lu, K. Huang, T. Meraj, and H.T. Rauf. A novel captcha solver framework using deep skipping convolutional neural networks. *PeerJ Computer Science*, 8:879. 1

[8] Z. Nouri and M. Rezaei. Deep-captcha: a deep learning based captcha solver for vulnerability assessment. Available at SSRN 3633354. 1

[9] M. Osadchy, J. Hernandez-Castro, S. Gibson, O. Dunkelman, and D. Pérez-Cabo. No bot expects the deepcaptcha! introducing immutable adversarial examples, with applications to captcha generation. *IEEE Transactions on Information Forensics and Security*, 12(11):2640–2653. 1

[10] A. Priya, A. Ganesh, R.A. Prasath, and K.J. Pradeepa. February). cracking captchas using deep learning. In *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS*, page 437–443. IEEE. 1

[11] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304. 2, 4

[12] F. Stark, C. Hazırbas, R. Triebel, and D. Cremers. Captcha recognition with active deep learning. In *Workshop new challenges in neural computation (Vol*, page 94. Citeseer. 1

[13] A. Thobhani, M. Gao, A. Hawbani, S.T.M. Ali, and A. Abdussalam. Captcha recognition using deep learning with attached binary images. *Electronics*, 9(9):1522. 1

Student Name	Contributed Aspects	Details
Deepthi M Rao	Literature survey, Model creation, training, inference, evaluation metrics	Performed literature survey - analyzed 7 papers, built and trained Models 1, 2 and 5 and did inference on them, implemented attention mechanism, implemented CTC loss layers in keras by taking assistance from my teammate, implemented the post-processing CTC decode function. Performed code review of my teammate. Contributed to the written part of whatever I have implemented in the report. Analysed and coded evaluation metrics - levenshtein ratio, cosine similarity and assisted in coding of pre-processing.
Harikiran Cherala	Literature survey, Model creation, training, inference, evaluation metrics	Performed literature survey - analyzed 5 papers, performed pre-processing, built and trained models 3 and 4 and did inference on them, assisted in implementation of CTC loss and post-processing. Performed code review of my teammate. Contributed to the written part of whatever I have implemented in the report. Coded the exact measure, jaccard index evaluation metric.

Table 3. Contributions of team members.

[14] J. Wang, J. Qin, X. Xiang, Y. Tan, and N. Pan. Captcha recognition based on deep convolutional neural network. *Math. Biosci. Eng*, 16(5):5851–5861. 1

[15] Horacio. Wilhelmy, Rodrigo Rosas. captcha dataset. 2

[16] X. Zhang, X. Liu, T. Sarkodie-Gyan, and Z. Li. Development of a character captcha recognition system for the visually impaired community using deep learning. *Machine Vision and Applications*, 32(1):1–19. 1