CAPTCHA Sequence Recognition using CNN-RNN with CTC Loss

Nick Cheliotis
cheliotisnick@gmail.com

Introduction

CAPTCHA systems consist of distorted alphanumeric strings embedded in noisy images, designed to be difficult for machines to interpret.With the rise of deep learning these systems are increasingly vulnerable to neural networks.

The task of recognizing CAPTCHA sequences poses many challenges.Unlike typical classification,CAPTCHA decoding is a sequence recognition problem.Each image contains a variable set of spatially dependent characters that must be predicted in order, without any explicit segmentation or character alignment.

To address these challenges, this project implements a deep learning architecture combining Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Connectionist Temporal Classification (CTC) loss.The CNN is responsible for extracting visual features from input images, while the RNN captures temporal dependencies across the feature sequence. The CTC loss enables training without requiring character-level alignment making it perfect for unsegmented sequence prediction.

This neural network was trained on a CAPTCHA dataset consisting of 100.000 greyscale images, each consisting of 5 alphanumerical values.Special metrics were used for this kind of project such as character accuracy and the average edit distance.The final model achieved 53.06% exact accuracy, 79.74% character-level accuracy and an average edit distance of 0.76.This report contains the whole pipeline and the experimental process.

Technical Details

The project was implemented in Python 3.11 using PyTorch within the PyCharm IDE for code development and debugging. To accelerate training and handle experiments efficiently, Google Colab was used for running the deep learning models.

During experimentation a T4 GPU was used for initial training runs, hyperparameter tuning, and debugging. For the final training phase and evaluation the model was executed on an A100 GPU.Mixed precision training (AMP) was enabled to maximize GPU utilization and allowing larger batch sizes and faster training without sacrificing model accuracy.

Architecture

The model consists of three main components: a Convolutional Neural Network (CNN), a Recurrent Neural Network (RNN) module using LSTM layers, and a fully connected (FC) output layer trained with Connectionist Temporal Classification (CTC) loss.

1.CNN: The CNN part gets as input a 150x40 greyscale image and outputs a 4D feature map.It consists of 3 convolution blocks, each with 2D convolution layer, batch normalization ,ReLU activation and 2x2 max pooling.After all three blocks, the final feature map has shape (**Batch_size**, **Channels**=256, **Height**=5, **Width**=18).

2.RNN: The CNN output is reshaped and fed into a 2-layer bidirectional LSTM.Each sequence consists of 18 timesteps.The RNN outputs a sequence of 18 vectors, each of size 256.

3.Output layer and CTC:RNN’s vectors are passed through a linear layer producing a final shape of (Time_steps=18, Batch_size, Number_of_classes).A manual bias trick is used to penalize over-prediction of the blank token early in training.The Linear layer converts RNN output to logits for each character class.CTC matches unaligned sequences during training.

Its important to add that during the final training, gradient clipping was used for safety as well as a learning rate scheduler (with patience=3), in case **Loss** becomes stagnant.

Data preprocessing

Before feeding images to our NN, the following preprocessing steps were applied:

1. Grayscale conversion: All images were converted into grayscale to eliminate irrelevant color information.This ensures the CNN focuses on structural character features (edges, curves, shapes).

2. Pixel Normalization: Pixel values were normalized to the [0, 1] range by dividing by 255.0.This prevents exploding gradients in the CNN and RNN layers.

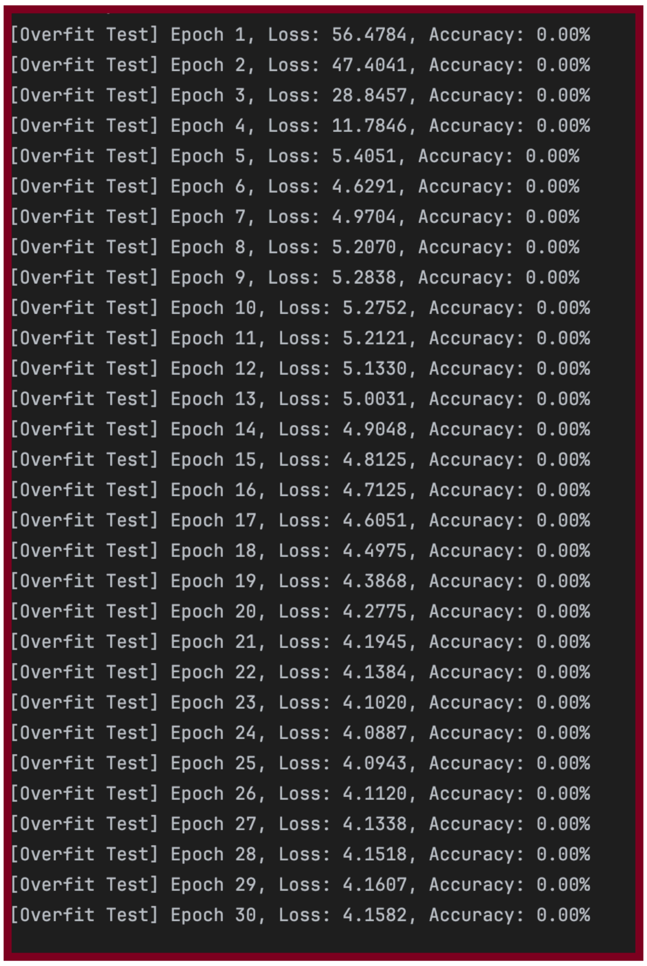
3. Labels and Image Size Validation: Every image was checked to be 150x40. All labels were verified to be exactly 5 characters long.

Experimental Process

This section outlines the experimental process and the key steps taken during model development, from early trials to the final trained system.

First phase - overfitting tests

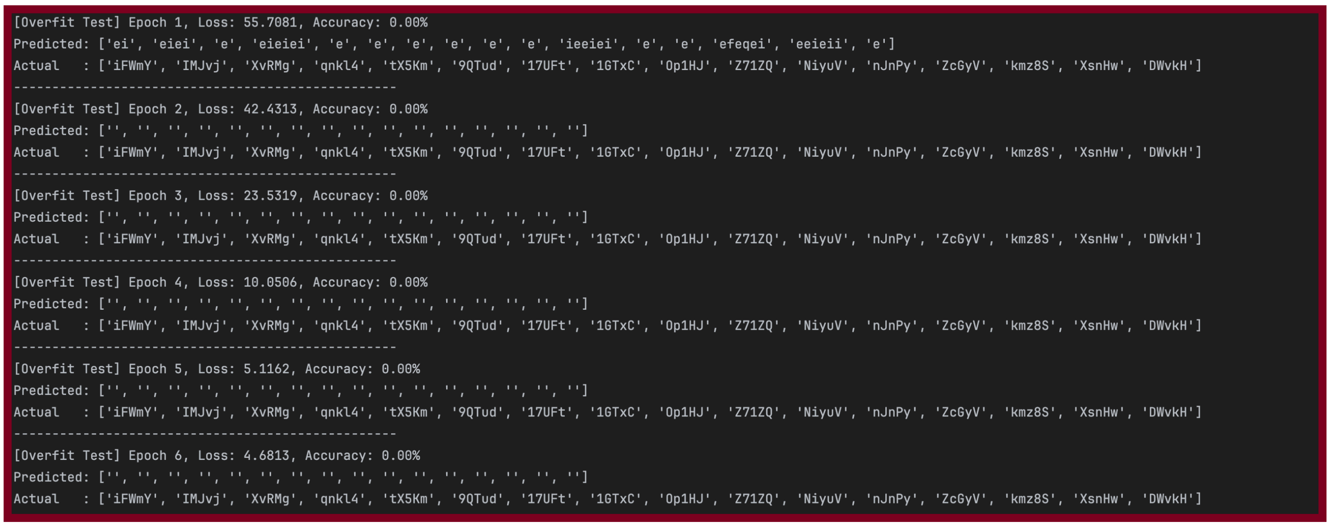
The results of the first run are depicted in Figure 1.We run an overfit test with 16 samples and 30 epochs to figure out if our model is capable of learning.Loss was dropping but Accuracy was not increasing.



```
[Overfit Test] Epoch 1, Loss: 56.4784, Accuracy: 0.00%
[Overfit Test] Epoch 2, Loss: 47.4041, Accuracy: 0.00%
[Overfit Test] Epoch 3, Loss: 28.8457, Accuracy: 0.00%
[Overfit Test] Epoch 4, Loss: 11.7846, Accuracy: 0.00%
[Overfit Test] Epoch 5, Loss: 5.4051, Accuracy: 0.00%
[Overfit Test] Epoch 6, Loss: 4.6291, Accuracy: 0.00%
[Overfit Test] Epoch 7, Loss: 4.9704, Accuracy: 0.00%
[Overfit Test] Epoch 8, Loss: 5.2070, Accuracy: 0.00%
[Overfit Test] Epoch 9, Loss: 5.2838, Accuracy: 0.00%
[Overfit Test] Epoch 10, Loss: 5.2752, Accuracy: 0.00%
[Overfit Test] Epoch 11, Loss: 5.2121, Accuracy: 0.00%
[Overfit Test] Epoch 12, Loss: 5.1330, Accuracy: 0.00%
[Overfit Test] Epoch 13, Loss: 5.0031, Accuracy: 0.00%
[Overfit Test] Epoch 14, Loss: 4.9048, Accuracy: 0.00%
[Overfit Test] Epoch 15, Loss: 4.8125, Accuracy: 0.00%
[Overfit Test] Epoch 16, Loss: 4.7125, Accuracy: 0.00%
[Overfit Test] Epoch 17, Loss: 4.6051, Accuracy: 0.00%
[Overfit Test] Epoch 18, Loss: 4.4975, Accuracy: 0.00%
[Overfit Test] Epoch 19, Loss: 4.3868, Accuracy: 0.00%
[Overfit Test] Epoch 20, Loss: 4.2775, Accuracy: 0.00%
[Overfit Test] Epoch 21, Loss: 4.1945, Accuracy: 0.00%
[Overfit Test] Epoch 22, Loss: 4.1384, Accuracy: 0.00%
[Overfit Test] Epoch 23, Loss: 4.1020, Accuracy: 0.00%
[Overfit Test] Epoch 24, Loss: 4.0887, Accuracy: 0.00%
[Overfit Test] Epoch 25, Loss: 4.0943, Accuracy: 0.00%
[Overfit Test] Epoch 26, Loss: 4.1120, Accuracy: 0.00%
[Overfit Test] Epoch 27, Loss: 4.1338, Accuracy: 0.00%
[Overfit Test] Epoch 28, Loss: 4.1518, Accuracy: 0.00%
[Overfit Test] Epoch 29, Loss: 4.1607, Accuracy: 0.00%
[Overfit Test] Epoch 30, Loss: 4.1582, Accuracy: 0.00%
```

Figure 1.

This was to be expected, CTC requires many epochs.Further testings were made to see the predictions the model was making.What we saw, is that the model always predicted the blank token (Figure 2.).This issue had to be addressed so in future testings the blank token got a high penalty (-20).In each test, we lowered the penalty to find the optimum balance between blank predictions and char predictions.



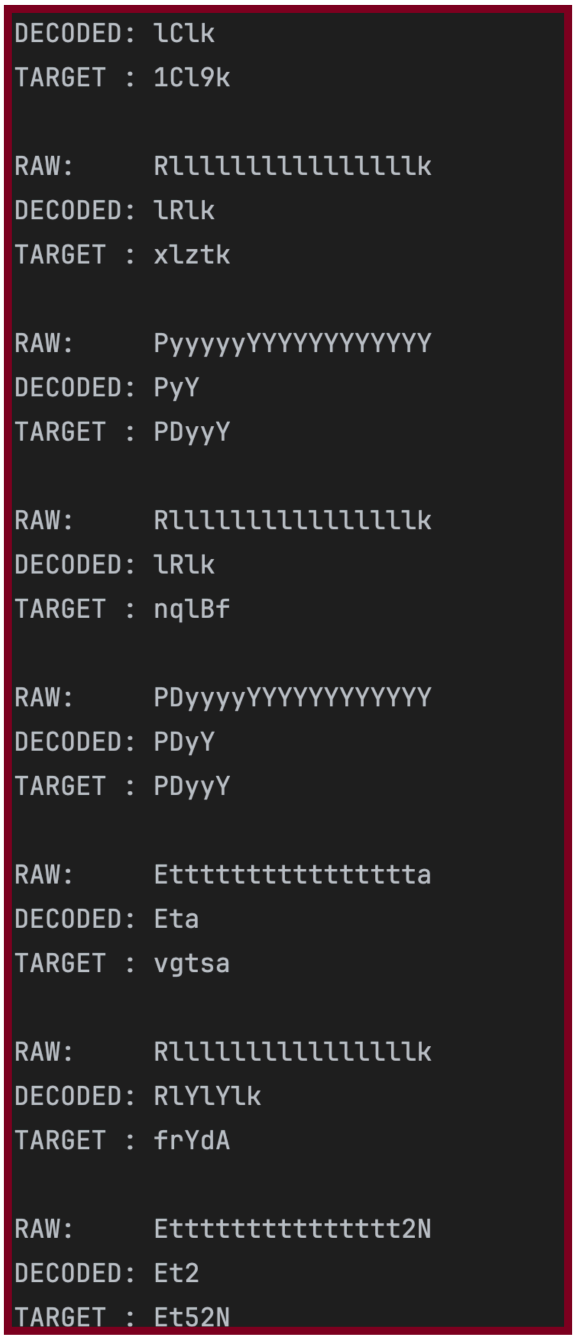
```
[Overfit Test] Epoch 1, Loss: 55.7081, Accuracy: 0.00%
Predicted: ['ei', 'eiei', 'e', 'eieiei', 'e', 'e', 'e', 'e', 'e', 'e', 'ieeiei', 'e', 'e', 'efeqei', 'eeieii', 'e']
Actual   : ['iFWmY', 'IMJvj', 'XvRMg', 'qnkl4', 'tX5Km', '9QTud', '17UFt', '1GTxC', 'Op1HJ', 'Z71ZQ', 'NiyuV', 'nJnPy', 'ZcGyV', 'kmz8S', 'XsnHw', 'DWvkH']
--------------------------------------------------
[Overfit Test] Epoch 2, Loss: 42.4313, Accuracy: 0.00%
Predicted: ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
Actual   : ['iFWmY', 'IMJvj', 'XvRMg', 'qnkl4', 'tX5Km', '9QTud', '17UFt', '1GTxC', 'Op1HJ', 'Z71ZQ', 'NiyuV', 'nJnPy', 'ZcGyV', 'kmz8S', 'XsnHw', 'DWvkH']
--------------------------------------------------
[Overfit Test] Epoch 3, Loss: 23.5319, Accuracy: 0.00%
Predicted: ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
Actual   : ['iFWmY', 'IMJvj', 'XvRMg', 'qnkl4', 'tX5Km', '9QTud', '17UFt', '1GTxC', 'Op1HJ', 'Z71ZQ', 'NiyuV', 'nJnPy', 'ZcGyV', 'kmz8S', 'XsnHw', 'DWvkH']
--------------------------------------------------
[Overfit Test] Epoch 4, Loss: 10.0506, Accuracy: 0.00%
Predicted: ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
Actual   : ['iFWmY', 'IMJvj', 'XvRMg', 'qnkl4', 'tX5Km', '9QTud', '17UFt', '1GTxC', 'Op1HJ', 'Z71ZQ', 'NiyuV', 'nJnPy', 'ZcGyV', 'kmz8S', 'XsnHw', 'DWvkH']
--------------------------------------------------
[Overfit Test] Epoch 5, Loss: 5.1162, Accuracy: 0.00%
Predicted: ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
Actual   : ['iFWmY', 'IMJvj', 'XvRMg', 'qnkl4', 'tX5Km', '9QTud', '17UFt', '1GTxC', 'Op1HJ', 'Z71ZQ', 'NiyuV', 'nJnPy', 'ZcGyV', 'kmz8S', 'XsnHw', 'DWvkH']
--------------------------------------------------
[Overfit Test] Epoch 6, Loss: 4.6813, Accuracy: 0.00%
Predicted: ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
Actual   : ['iFWmY', 'IMJvj', 'XvRMg', 'qnkl4', 'tX5Km', '9QTud', '17UFt', '1GTxC', 'Op1HJ', 'Z71ZQ', 'NiyuV', 'nJnPy', 'ZcGyV', 'kmz8S', 'XsnHw', 'DWvkH']
```

Figure 2.

Second phase - overfitting tests

During the second phase several changes were made.We increased the epochs from 30 to 200.We added 1 more layer to the CNN to extract more meaningful features.We added 1 more layer to the LSTM and made it bidirectional.Greedy decode was replaced with beam search(k=5).Despite being more computational expensive, its more well suited in our project.

The model was starting to make actual predictions (Figure 3.).Accuracy wasn’t rising, but it was a positive step.



```
DECODED: lClk
TARGET : 1Cl9k

RAW:     Rllllllllllllllllk
DECODED: lRlk
TARGET : xlztk

RAW:     PyyyyyYYYYYYYYYYYY
DECODED: PyY
TARGET : PDyyY

RAW:     Rllllllllllllllllk
DECODED: lRlk
TARGET : nqlBf

RAW:     PDyyyyYYYYYYYYYYYY
DECODED: PDyY
TARGET : PDyyY

RAW:     Etttttttttttttttta
DECODED: Eta
TARGET : vgtsa

RAW:     Rllllllllllllllllk
DECODED: RlYlYlk
TARGET : frYdA

RAW:     Ettttttttttttttt2N
DECODED: Et2
TARGET : Et52N
```

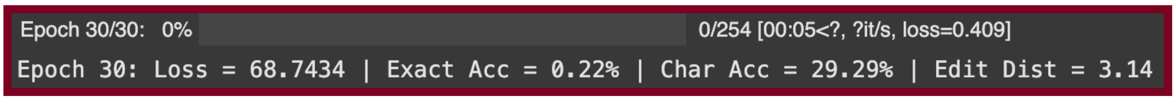
Figure 3.

Tests on Colab

We increased the sample size to 10.000 and set the batch size equal to 32.Accuracy was a strict metric for our task, so to evaluate our model we introduced 2 new metrics, character accuracy and average edit distance.The first depicts the percentage of characters we correctly predicted in a CAPTCHA sequence.The second one depicts the average number of mistakes our models makes per sequence.To execute these tests, a T4 GPU was used.

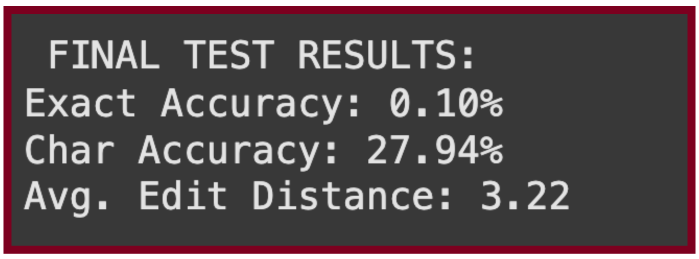
The results were hopeful.Our model at 30 epochs was able to achieve during evaluation (Figure 4.) a character accuracy of 29.29%, an average edit distance of 3.14 and a total accuracy 0.22% (less that 1 precent).On the test dataset (Figure 5.) it achieved a char accuracy of 27.94%, an average edit distance of 3.22 and a total accuracy of 0.10%.

Although the overall accuracy was low, the steady decrease in loss over 30 training epochs -despite CTC typically requiring many more- indicates strong learning potential and significant room for improvement.



```
Epoch 30/30:  0%                                0/254 [00:05<?, ?it/s, loss=0.409]
Epoch 30: Loss = 68.7434 | Exact Acc = 0.22% | Char Acc = 29.29% | Edit Dist = 3.14
```

Figure 4.



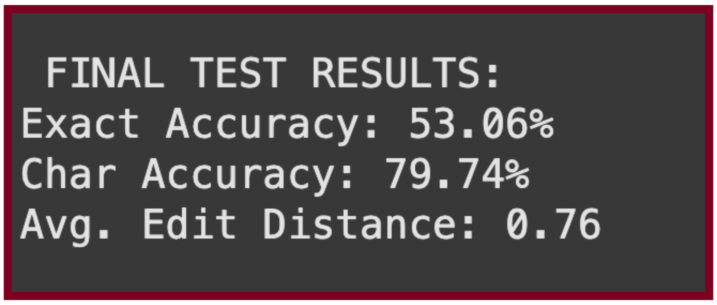
```
 FINAL TEST RESULTS:
Exact Accuracy: 0.10%
Char Accuracy: 27.94%
Avg. Edit Distance: 3.22
```

Figure 5.

Final tests

For the final tests, we used the whole dataset (100.000 samples).We set the batch size equal to 64 and epochs equal to 50.We introduced gradient clipping (to deal with vanishing/exploding gradients) and a learning rate scheduler (with patience equal to 3, tied to loss).We also used AMP (Automatic Mixed Precision) to speed up training and save on computational power.The tests were ran with an A100 for speed.

On the test dataset, our model achieved (Figure 6.) an exact accuracy of 53.06%, a character accuracy of 79.74% and an average edit distance of 0.76.



```
 FINAL TEST RESULTS:
Exact Accuracy: 53.06%
Char Accuracy: 79.74%
Avg. Edit Distance: 0.76
```

Figure 6.

Results

Despite achieving a relatively low accuracy of ~53%, the model showed consistent loss reduction across epochs, suggesting that the architecture was learning effectively. Since CTC-based training typically requires many more epochs for convergence, these results highlight substantial room for improvement with further tuning.

Other studies using the same CNN + BiLSTM + CTC architecture have achieved significantly higher accuracies, even exceeding 95–99%, but they often:

1. Trained for 100+ epochs
2. Used large datasets with careful data balancing
3. Applied extensive hyperparameter tuning.

Conclusion

This project demonstrated the effectiveness of combining CNNs, BiLSTMs, and CTC Loss for sequence-based image tasks like CAPTCHA recognition. While our results were modest, the consistent loss reduction and architectural behavior validated the learning pipeline.

With more training epochs and deeper tuning, this architecture has the potential to reach state-of-the-art performance, as shown in related work.

References

1.Shi et al. (2019), End-to-End CAPTCHA Recognition Using Deep CNN-RNN Network

2.Yu et al. (2021), Adaptive CAPTCHA: A CRNN-Based Text CAPTCHA Solver with Adaptive Fusion Filter Networks

3.Graves et al. (2006), Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks