Image Classification and Text Detection with reCAPTCHA and CAPTCHA

Kenny Li KENNY.LI@YALE.EDU
Yale University

Jason Zheng JASON.ZHENG@YALE.EDU
Yale University

Abstract

This project explores the application of deep learning techniques for the classification and text detection in CAPTCHA and reCAPTCHA images, two prevalent forms of image-based security measures. We introduce two distinct neural network architectures tailored to these challenges. For CAPTCHA images, an Optical Character Recognition (OCR) model employing a Convolutional Recurrent Neural Network (CRNN) with Long Short-Term Memory (LSTM) cells and Connectionist Temporal Classification (CTC) is presented. It demonstrated high accuracy, exceeding 90% on diverse datasets, with a notable accuracy of over 98% on a smaller dataset (1,070 images) and more than 92% on a larger one (82,300 images). The approach to reCAPTCHA images involved constructing a deep Convolutional Neural Network (CNN). We initially employed the Resnet50 architecture, later transitioning to InceptionV3, both augmented with custom layers for enhanced performance. This adaptation was key in achieving consistent accuracy levels above 80% on a dataset of 11,730 images. Moreover, the model's training on an extensive dataset of over 200,000 images demonstrates its scalability and adaptability. These results illustrate the effectiveness of the proposed models in tackling complex image-based security systems and contribute to the broader field of applying advanced neural network techniques in enhancing digital security mechanisms.

Keywords: Optical Character Recognition, Convolutional Neural Network, Recurrent Neural Network, Connectionist Temporal Classification Loss, Long Short-Term Memory

Introduction

The advent of the internet brought with it a myriad of opportunities and challenges, one of which is the distinction between human and automated access. To address this, CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) was introduced. Initially patented in 1997, CAPTCHA's primary function was to prevent automated bots from exploiting web services. The first commercial use of CAPTCHA was the Gausebeck–Levchin test in 2000, used by idrive.com and PayPal to prevent fraud. The classic CAPTCHA involved distorted text that bots found challenging to decipher, effectively blocking their interaction with websites.

Over time, various types of CAPTCHAs evolved, including text-based, image-based, and audio CAPTCHAs, each catering to different aspects of bot prevention. Text-based CAPTCHAs,

the oldest form, presented known words or random warped texts in a visually distorted form. Image-based CAPTCHAs required users to identify specific features or elements in a set of pictures, while audio CAPTCHAs, developed for visually impaired users, played a series of letters and numbers for the user to input. Despite their effectiveness, these methods faced challenges with accessibility for visually impaired individuals and were susceptible to advances in bot capabilities.

The introduction of reCAPTCHA by Google marked a significant evolution in this technology. Developed at Carnegie Mellon University and acquired by Google in 2009, reCAPTCHA utilized real-world images and text, such as street addresses and excerpts from printed books, making it more complex for bots to solve. The initial versions of reCAPTCHA helped digitize text from sources like The New York Times archives and Google Books, leveraging human input for a dual purpose: verification and digitization. The NoCAPTCHA reCAPTCHA, a more recent advancement, uses sophisticated methods like tracking the user's cursor movements and assessing browser cookies and device history to determine if a user is a bot.

The effectiveness of CAPTCHA and reCAPTCHA systems as a security measure has been a subject of debate, given their susceptibility to advanced bots and the potential intrusion of user privacy. Thus, the motivation behind our project is to develop models that can accurately classify and detect text in CAPTCHA and reCAPTCHA images, pushing the boundaries of what is possible in machine learning and artificial intelligence.

Initially, our motivation was to build an automatic bot that would scrape the CAPTCHA and reCAPTCHA data live from a website, run it through our machine learning models, and then auto-populate the correct result, but we quickly realized that this was against many websites' terms of service. Thus, we primarily focused on developing the machine learning models.

For handling text-based CAPTCHAs, the project employs an Optical Character Recognition (OCR) framework, underpinned by a sophisticated Convolutional Recurrent Neural Network (CRNN). This network integrates a Connectionist Temporal Classification (CTC) output layer, crucial for decoding sequences without explicit segmentation. This configuration is designed to handle the varying distortions and styles inherent in CAPTCHA texts, thereby enhancing the model's adaptability and accuracy.

In tackling the more complex image-based reCAPTCHAs, the project leverages a CNN that incorporates transfer learning, utilizing Google's InceptionV3 model. This approach is selected for its proficiency in image analysis and object detection, essential for discerning the nuanced differences in reCAPTCHA images.

Initially, the project faced challenges with a large dataset intended for reCAPTCHA training, particularly due to its size and class imbalance (predominantly 'car' images). To mitigate this, we first opted for a smaller, more balanced dataset to avoid model bias and ensure efficient training. Moreover, to stay within the computational constraints of plat-

forms like Google Colab and Kaggle Notebooks, we were challenged with identifying more computationally efficient architectures that still produced the desired accuracy.

Adjustments were made to address overfitting and runtime efficiency. Callback functions and data augmentation techniques were implemented to facilitate model convergence and prevent overfitting. These modifications were critical in optimizing the model's training within the available computational resources and time constraints.

Data

- **Small CAPTCHA Dataset:** Sourced from `https://www.kaggle.com/datasets/fournierp/captcha-version-2-images`, this dataset provides a collection of 1070 CAPTCHA images. Each image contains a sequence of 5 characters, limited to lowercase letters or numbers. They are 200 x 50 grayscale PNGs.
- **Large CAPTCHA Dataset:** Available at `https://www.kaggle.com/datasets/akashguna/large-captcha-dataset/data`, this extensive dataset offers a diverse range of CAPTCHA images, allowing for the development of more robust text recognition models. Each image again contains a sequence of 5 characters, although the limitation to lowercase letters or numbers has been relaxed to just numbers and letters. The images are 256 x 256 RGB PNGs, and thus had to be pre-processed before being input into our model.
- **reCAPTCHA Dataset:** The dataset for reCAPTCHA, found at `https://www.kaggle.com/datasets/mikhailma/test-dataset`, contains 11,730 images across 12 classes. These images are representative of those used in Google's reCAPTCHA V2, ideal for training image-based recognition models.

Methodology

CAPTCHA OCR

A CRNN model integrating LSTM cells and a CTC layer was developed, following the architecture in Shi et al. (2017). The model initiates with convolutional layers (Conv2D) using SELU activation and same padding for primary feature extraction from images. This is followed by Max pooling (MaxPool2D) to reduce feature dimensionality. Batch normalization (BatchNormalization) is included after specific convolutional layers to enhance stabilization and training efficiency. For sequence processing, the model utilizes bidirectional LSTM layers (Bidirectional, CuDNNLSTM), critical for capturing sequential dependencies within character series. The architecture progresses to a Map-to-Sequence stage and then to a dense layer (Dense) with SELU activation, facilitating the transition from feature maps to sequence predictions. This is followed by a second batch normalization layer for further stabilization. The final classification of characters is performed by a softmax layer (Dense with softmax activation), determining the probability distribution over the character set. The model employs the Adam optimizer for effective training. A custom CTC loss layer (CTCLayer) is integrated, crucial for sequence alignment; it computes the probability of the target sequence given the model's inputs, enabling the handling of variable-length

sequences and alignment between predictions and ground-truth labels without predefined segmentation, a concept inspired by Graves et al. (2006).

Layer Type	**Specifications**	**Layer Name**
Input	Shape: (256, 80, 1) (transposed)	Image
Conv2D	64 filters, 3x3, SELU, same padding	Conv1
MaxPool2D	2x2, stride 2	MaxPool1
Conv2D	128 filters, 3x3, SELU, same padding	Conv2
MaxPool2D	2x2, stride 2	MaxPool2
Conv2D	256 filters, 3x3, SELU, same padding	Conv3
Conv2D	256 filters, 3x3, SELU, same padding	Conv4
MaxPool2D	2x1, stride 2	MaxPool3
Conv2D	512 filters, 3x3, SELU, same padding	Conv5
BatchNormalization		BatchNormalization1
Conv2D	512 filters, 3x3, SELU, same padding	Conv6
BatchNormalization		BatchNormalization2
MaxPool2D	2x1, stride 2	MaxPool4
Map-to-Sequence and Sequence Processing Begins		
Conv2D	512 filters, 2x2, SELU, valid padding	Conv7
Reshape	Squeeze to new dimensions	Reshape
Dense	512 units, SELU	Dense1
BatchNormalization		BatchNormalization3
Bidirectional LSTM	256 units, return sequences	Bi-LSTM1
Bidirectional LSTM	256 units, return sequences	Bi-LSTM2
Dense	Softmax activation	Dense2
CTCLayer		CTC_Loss

Table 1: Layers of the CRNN Model

reCAPTCHA CNN

For the reCAPTCHA model, we decided to build and implement a Convolutional Neural Network. We built a class, InceptionV3Model, that took advantage of the pre-existing InceptionV3 model within Keras. This model was built based off of Szegedy et al's 'Rethinking the inception architecture for computer vision'. Additionally, we added a few custom layers on top of the pre-existing architecture of the Inceptionv3 model. These custom layers included a GlobalAveragePooling2D layer, 2 Dropout layers, and 2 Dense layers. To further enhance our model, we unfreeze more layers of the Inceptionv3 model after 10 epochs are ran. The optimizer used in our model was Stochastic Gradient Descent.

Layer Type	Specifications	Layer Name
InceptionV3	Input Shape: (150, 150, 3)	BaseModel
GlobalAveragePooling2D		GlobalPool
Dropout	Rate: 0.7	Dropout1
Dense	256 Units, RELU	Dense1
Dropout	Rate: 0.5	Dropout2
Dense	Softmax Activation	Dense2

Table 2: Layers of the CNN Model

Implementation Details

CAPTCHA OCR

For the CAPTCHA OCR, a CRNN model was implemented using TensorFlow and Keras, with specific configurations for training and preprocessing. The key steps in the implementation are as follows:

1. **Hyperparameters and Callbacks:** The model was trained with a batch size of 128, and an Adam optimizer was used with an initial learning rate of 0.001, beta1 set to 0.9, beta2 set to 0.999, and a clip norm of 1.0. Training was conducted for a maximum of 100 epochs. To prevent overfitting and ensure efficient training, an early stopping callback with a patience of 10 epochs and a ReduceLROnPlateau callback to adjust the learning rate based on the validation loss were employed. ReduceLROnPlateau monitored the val_loss and reduced the learning rate by a factor of 0.1 to a minimum of 1×10^{-5} with a patience of 5 epochs.

2. **Image Preprocessing:** The images were resized to fit the input requirements of the model, with a width of 256 and a height of 80 pixels. This preprocessing involved either cropping or padding the images to maintain their aspect ratio. Additionally, images were converted to grayscale and their data type was changed to float32. Prior to being input into the image, the pixels were also transposed.

3. **Dataset Preparation:** The dataset was split into training and testing sets with a ratio of 80:20. For each image in the dataset, the `encode_single_sample` function was applied, which handled the image resizing and label encoding. Labels were encoded using a StringLookup layer, converting each character in the label to its corresponding integer representation.

4. **Data Augmentation:** The training dataset was augmented to include a variety of transformations to improve the robustness of the model. The augmentation included random adjustments to the image brightness, contrast, and orientation.

5. **Model Training:** The training and validation datasets were fed into the CRNN model for training. The custom CTCLayer calculated the CTC loss between the predicted output and the true labels during training.

6. **Model Evaluation:** Post-training, the model's performance was evaluated on the test dataset to assess its accuracy and generalization capabilities.

reCAPTCHA CNN

The reCAPTCHA CNN model was also implemented using TensorFlow and Keras, with specific configurations for training and preprocessing. The key steps in the implementations are as follows:

1. **Hyperparameters and Callbacks:** The model was trained with a batch size of 64, and a Stochastic Gradient Descent optimizer was used with a learning rate of 0.001. Training was conducted for a maximum of 100 epochs. To prevent overfitting and ensure efficient training, an early stopping callback with a patience of 10 epochs and a ReduceLROnPlateau callback to adjust the learning rate based on the validation loss were employed. After 10 epochs, the model is stopped and unfreezes some more layers, to allow for more finite tuning. The model is then restarted on the remainder of the 100 epochs with a new learning rate of 0.0001.

2. **Image Preprocessing:** The images were resized to fit the input requirements of the model, with a width of 150 and a height of 150 pixels.

3. **Dataset Preparation:** The dataset was split into training and validation sets with a ratio of 80:20.

4. **Data Augmentation:** The training dataset was augmented to include a variety of transformations to improve the robustness of the model. The augmentation included random adjustments to the image brightness, contrast, and orientation via random rotations, zooms, and shifts.

5. **Model Training:** The training and validation datasets were fed into the CNN model for training.

6. **Model Evaluation:** Post-training, the model's performance was evaluated on the validation dataset to assess its accuracy and generalization capabilities.

Results

We successfully met our primary goal of achieving an accuracy rate of 75% or higher across all models and datasets.

For the large CAPTCHA dataset, our CAPTCHA OCR model demonstrated a remarkable accuracy of 95.53% on the test data. This level of accuracy is particularly noteworthy given the inherent difficulty in differentiating between visually similar characters such as "I", "1", "7", or "o", "O", "0". The model's ability to discern these subtle differences, which can even challenge human perception, underscores its robustness and precision in handling complex CAPTCHA images.

On the small CAPTCHA dataset, the model achieved an even higher accuracy of 98.13% on the test data, further solidifying the effectiveness of our approach in dealing with varied CAPTCHA challenges, albeit on a much smaller dataset.

The reCAPTCHA model, designed to tackle more complex image-based recognition tasks, attained a commendable accuracy of 81.60% on the validation dataset. This performance is particularly significant considering the diverse and intricate nature of reCAPTCHA images, which are designed to be challenging for automated systems.

References

A. Graves, S. Fernández F. Gomez, and J. Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *ICML 2006 - Proceedings of the 23rd International Conference on Machine Learning*, pages 369–376. ACM, 2006.

B. Shi, X. Bai, and C. Yao. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 39:462–467, 2017.

C. Szegedy, V. Vanhoucke, and J. Shlens. Rethinking the inception architecture for computer vision. 2016.

Szegedy et al. (2016) Shi et al. (2017) Graves et al. (2006)