CAPTCHA Decoding: Evaluating Classical and Deep Learning Approaches

Wang Zichen, Zheng Hanmo, Wang Ziwen, Leong See Leng, Qiu Qianhui (Computer Science)

Keywords — CNN, KNN, SVM, TrOCR, GAN, GateCNN

I. Abstract

We compare classical (SVM, KNN) and deep learning (CNN, Gated CNN, CRNN, SAR, TrOCR) methods for CAPTCHA recognition on 10,000 images. Classical models serve as baselines, while CNNs and attention-based models capture complex patterns. SAR uses an RNN-LSTM setup; TrOCR combines ViT and BERT. TrOCR achieved the highest character accuracy (89%), and had the best CAPTCHA-level accuracy (58%).

II. Methodology

II.A. Data Preprocessing

Before we proceeded to developing models, we first *cleaned our dataset* with the following steps.

1. **Invalid data**: 196 invalid CAPTCHAs were manually removed after finding by missing letters, invalid characters (e.g. roman, watermarks) and cutoff letters.

Figure 1: `1vw0zvpw-0.png`

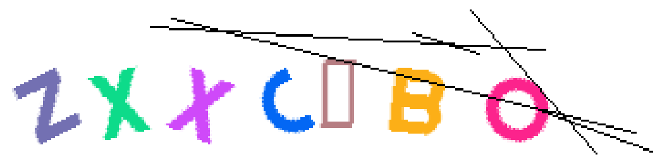
Figure 2: `zxxc1bo-0.png`

2. **Noise removal:** We removed black lines from the image and filled it with the majority neighbour.
3. **Color removal**: We remove color in two steps: firstly, convert the image to grayscale to discard color information; then apply binarization to produce a clear black-and-white image for better training.
4. **Resizing**: For the ease of training models, we resized all valid images with padding to *169x48*, keeping the aspect ratio. We have both colored and black and white images as options for testing. (e.g. folders `train_clean_color_resized`, `test_cleaned_color_resized`)

Figure 3: `z0n69ges-0.png`

Figure 4: after cleaning and resizing

II.B. Model Architectures

II.B.1. Segmentation

For models in Section II.B.2, we first explicitly extract out individual letters in the captcha before identifying them. We leverage the fact that CAPTCHA characters are in different colors.

M1: Flood fill:

1. Apply a Gaussian blur on the image to reduce accuracies from anti-aliasing.
2. Run Flood Fill algorithm, using `np.linalg.norm(np.array(pixel1) - np.array(pixel2)) < atol` to accomodate slight pixel variations due to anti-aliasing.
3. If two regions have the "same" color and one contains the other (in terms of x coordinates), we merge the regions. This is for letters like `i` and `j` to not be separated into two parts.

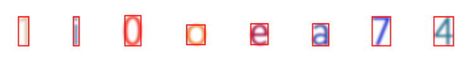
Figure 5: Segmentation example - 1i0oea74-0.png

This method segments **87.5%** of CAPTCHAs into the correct number of letters.

M2: Color-Based Segmentation for CAPTCHA Character Isolation

We first group pixels by color, and each group is copied onto a white background to isolate characters.

Version 1: Assumes the number of characters from filename length. Extracts the most frequent colors and maps according to their horizontal positions.

Version 2: Eliminates reliance on filename assumptions. Instead, it dynamically selects colors by analyzing frequency distributions, with a focus on identifying significant frequency drops to determine character boundaries.

Figure 6: Segmentation example - wtvaivr-0.png

Advantages:

- Overlap Handling: Maintains character integrity even in cases of partial overlaps, preserving overall shapes.
- Positional Accuracy: Ensures correct spatial alignment of characters within the CAPTCHA.

The method achieves a segmentation accuracy of 100% in Version 1 and 83.82% in Version 2.

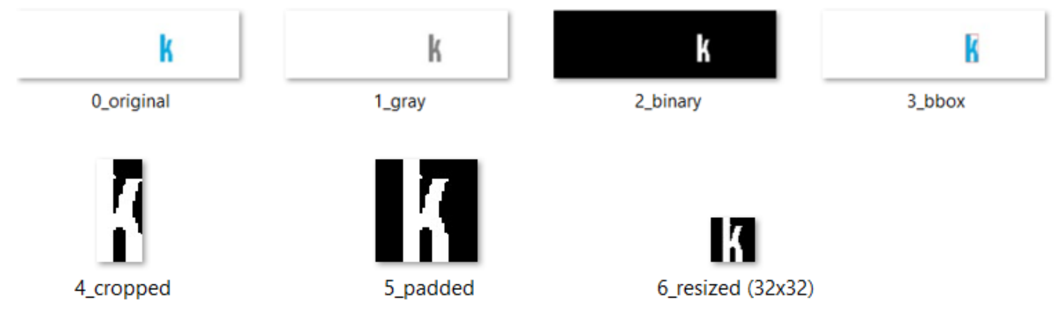

Figure 7: Example for steps after Segmentation

II.B.2. Segmentation-based models

These models are trained on individual characters and combined with Section II.B.1 to extract the full CAPTCHA. We first resize each character to 32x32 before using the models below.

KNN classifies the images into letters based on the majority label of its k nearest neighbors in the feature space, using Euclidean distance. Variations of neighbour counts were tested.

SVM: We employ a multi-class SVM model using a one-vs-one strategy, resulting in 630 binary classifiers trained across 36 character classes. Designed as a discriminative classifier, SVM is favored for its ability to maximize inter-class margins in high-dimensional spaces. It operates on 1024-dimensional handcrafted features, including pixel intensity and spatial coordinates. An RBF kernel with automatic gamma scaling is used to adapt to feature variance and effectively capture non-linear separability. The model comprises approximately 36.2 million parameters, offering a strong non-neural baseline for CAPTCHA recognition.

CNN: Our implementation consists of a lightweight CNN architecture featuring three sequential convolutional blocks (with 3×3 kernels, batch normalization, and max pooling), followed by a fully connected head with dropout regularization. The network flattens the final feature maps and passes them through two dense layers to produce a 36-class output. In total, the model contains 234k parameters, of which 233.7k are trainable. To optimize training, we employed batch processing for efficient gradient updates, validation-based early stopping to prevent overfitting, and continuous validation monitoring to guide generalization and convergence.

II.B.3. End-to-end models

CNN We start with a lightweight **basic CNN**, featuring four sequential convolutional blocks with 3×3 kernels and batch normalization, followed by a 1×1 convolution for classification. We set the hidden dimension to 128 channels, giving approximately 246k trainable parameters in total. This straightforward design is trained with CTC loss and provides a solid baseline for end-to-end CAPTCHA recognition.

GateCNN Building on the basic CNN, we **introduce GateBlocks to enhance feature extraction**. Gating is a multiplicative control strategy that lets a network decide how much of an intermediate representation should flow forward. It enables selective feature control while preserving a residual path for stable optimization. This boosts expressiveness without significantly increasing parameter count. In our design, H(x) is constructed to be antisymmetric and zero-mean, emphasizing high-frequency contrasts (such as edges or stroke crossings). We first replaced the fourth convolutional block with a GateBlock, but saw no improvement. Even a larger GateCNN with 4 million parameters yielded only marginal gains over the baseline.

$$f(\mathbf{x}) = \mathbf{x} + \underbrace{\mathbf{H}(\mathbf{x})}_{\text{candidate}} \odot \underbrace{\mathbf{T}(\mathbf{x})}_{\text{gate}}$$

Figure 8: Gating Mechanism

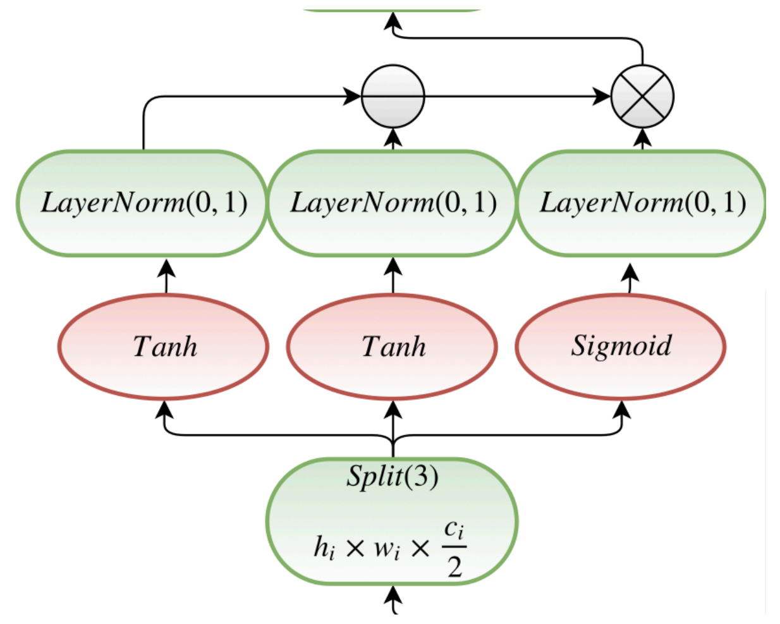

Figure 9: Design of GateBlock

CRNN Noting that both the basic CNN and GateCNN had strong character-level accuracy but struggled to perfectly capture entire CAPTCHAs, we turned to enhancing **sequential processing**. This model still uses the CNN for feature extraction, but it **appends bidirectional LSTM layers** to explicitly model the left-to-right sequence of characters. The recurrent layers maintain contextual information from previous steps, which is critical for recognizing letters in the correct order. Consequently, CRNN substantially boosts CAPTCHA-level accuracy, outperforming earlier purely convolutional models.

SAR To further **improve contextual modeling**, we **replaced** CRNN's sequential **RNN structure with an encoder-decoder** architecture known as Show, Attend, and Read (SAR). Thus the pipeline includes: a CNN feature extractor that converts the input image into a sequence of high-level feature vectors, a bidirectional LSTM encoder that processes these features globally from both directions, and an attention-based decoder using an LSTMCell, which generates one character at a time by selectively focusing on different parts of the encoded features. Additionally, SAR uses a specialized character vocabulary, 128-dimensional embeddings, and a dropout rate of 0.5 for regularization. By leveraging attention at each decoding step, SAR can align more precisely with regions of interest in the input image, leading to notable performance gains over CRNN.

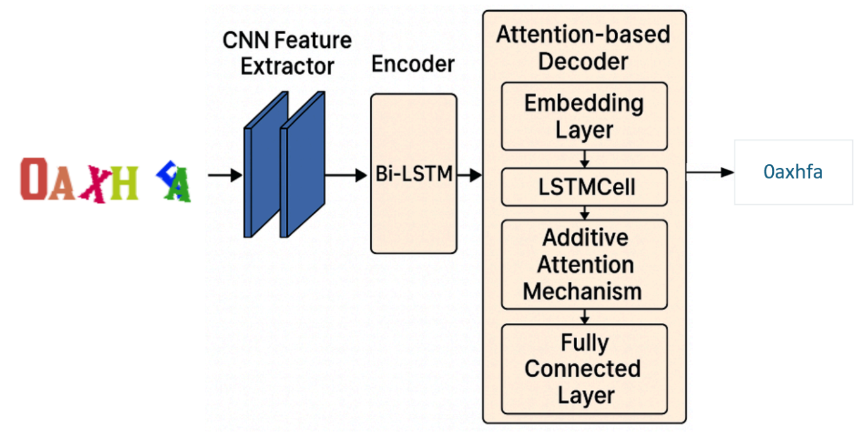

Figure 10: SAR.png

TrOCR Motivated by the strong performance of attention mechanisms, we explored a Transformer-based model, TrOCR, which applies **global self-attention** in both its ViT-based vision encoder and autoregressive decoder. Unlike SAR's localized attention, TrOCR processes the entire image as embedded patches, capturing long-range dependencies more effectively. The causal decoder only attends to previously generated tokens through **masked multi-head self-attention**, and to the encoder's output via **cross-attention**. Fine-tuning the pretrained microsoft/trocr-base-printed model on colored CAPTCHAs (with 20% held out for validation) further improved generalization. After tuning learning rates, epochs, warmup, and beam search, the best results were achieved with AdamW (lr=1e-5, weight_decay=1e-4), 3 epochs, 10% warmup, cosine scheduling, and beam search = 2.

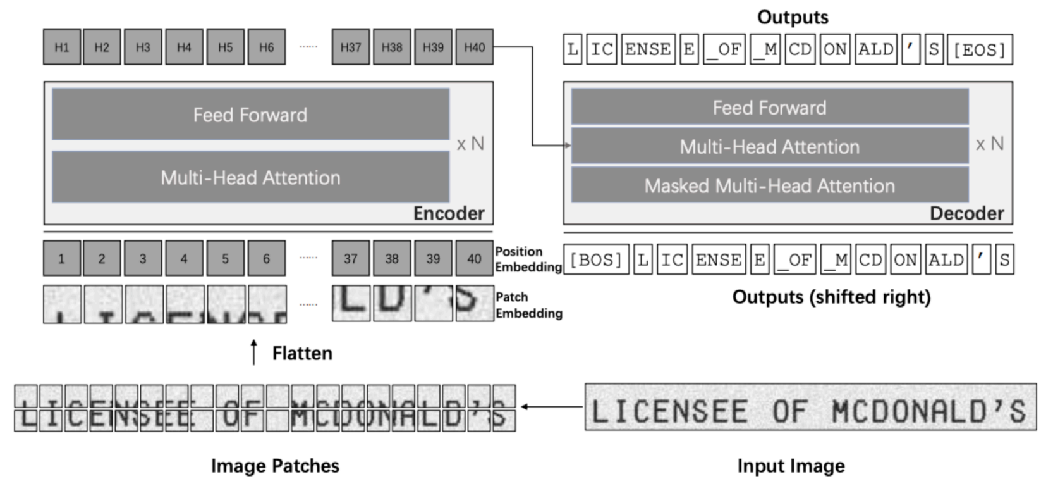

Figure 11: TrOCR Architecture

III. Ablation study

Data Augmentation We apply data augmentation to improve the model's robustness against variations in fonts, noise, and distortions in CAPTCHA images. Techniques such as random rotation, color jitter, and affine transformations have shown to enhance recognition performance.

Beam search In an autoregressive decoder, Beam Search improves the overall quality of sequence generation. Unlike greedy decoding, which selects the most probable token at each step, Beam Search keeps multiple high-probability candidate paths at each step, reducing the risk of early errors that can affect the entire sequence. In CAPTCHA recognition tasks, this helps increase the accuracy of predicting the complete string. Especially in cases of blurred characters or high model uncertainty, Beam Search allows for more stable and reliable generation of the most likely CAPTCHA sequence.

SNAR(Non-Autoregressive SAR) We replaced the original autoregressive decoder with a non-autoregressive structure by introducing learnable positional embeddings and removing the LSTMCell, enabling parallel character prediction. Compared to step-by-step generation in autoregressive decoding, the non-autoregressive design significantly improves inference speed. Although it results in a slight drop in accuracy, the overall recognition performance remains strong, making it suitable for latency-sensitive scenarios.

IV. Results

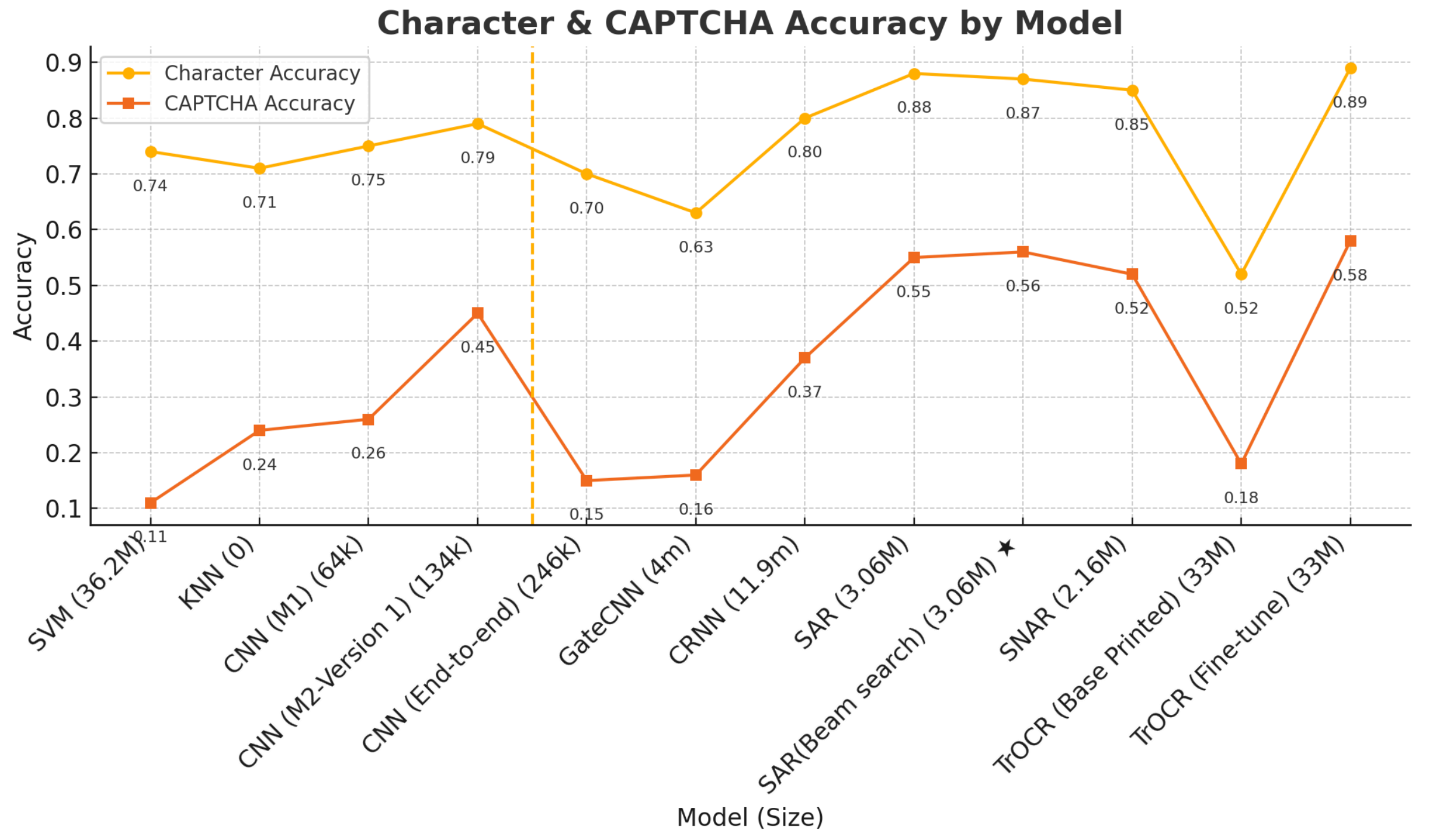

Figure 12: Model Performance

V. Conclusions and Key Insights

Segmentation notably improves performance across SVM, KNN, and CNN models compared to end-to-end approaches like CNN, GateCNN, and CRNN. It simplifies the task into a 36-class classification, where **character boundaries are clearer**. Interestingly, KNN with `neighbors=1` achieved the highest character-level accuracy compared to `neighbors=2,3,4`, suggesting that the segmented characters occupy well-defined regions in feature space — they are not overly varied, and similar instances are close together. This highlights how **segmentation enhances feature separability**, making even simple instance-based models highly effective. However, visually similar classes (e.g., o vs 0, i vs 1) still show lower F1-scores (≈0.5) due to inherent ambiguity, while distinct characters (e.g., m, w) perform well (≈0.9). Segmentation **reduces noise and overlap**, enabling more accurate character recognition, but cannot fully resolve class-level visual similarity.

Inductive-bias mismatch: A CRNN's CNN-plus-Bi-LSTM structure is naturally aligned with how we read—first distilling pixels into glyph descriptors and then scanning them sequentially from left to right—so each recurrent step instantly possesses the entire preceding context, whereas GateCNN, although equipped with more expressive nonlinear gating than a plain CNN, must propagate information across many convolutional hops to reach distant characters, a longer, noisier path that weakens its ability to model complete strings and thus hampers CAPTCHA performance. The lesson here is choosing primitives whose computation graph **resembles the generative process of the data** may be **more important than expressive power itself**.

SAR model's CNN + Bi-LSTM + attention architecture closely mirrors the "extract glyphs first, then read them in sequence" process, allowing each decoding step to leverage full context while precisely focusing on spatial cues. On our small, well-regularized CAPTCHA dataset, applying dropout and L2 regularization effectively curbs overfitting; beam search yields only marginal gains, showing that architectural design is more crucial than decoding strategy. A promising next step is to introduce lightweight data augmentations to diversify fonts and backgrounds, aiming to further boost performance.

TrOCR's early overfitting, with validation loss surpassing training loss by epoch 2/3, possibly stems from the model's large capacity and small dataset. Reducing the learning rate and limiting epochs helped prevent overfitting, while the cosine scheduler improved stability. Minimal improvement from beam search width suggests that model architecture plays a larger role than decoding strategy. More diverse data may be needed to further enhance performance.

VI. Work Division

Leong See Leng – Contributed to data cleaning and segmentation ideation and implementation, worked on KNN and CNN models, participated in poster writing

Qiu Qianhui – Contributed to data cleaning and segmentation ideation and implementation, worked on CNN and SVM models, participated in poster writing

Zheng Hanmo – Worked on the SAR model, participated in poster writing

Wang Ziwen – Worked on CNN based end-to-end (basic CNN, GateCNN and CRNN) models, participated in poster writing

Wang Zichen – Contributed to data cleaning and segmentation ideation, worked on TrOCR model, participated in poster writing

https://github.com/seelengxd/cs4243-captcha