Project Proposal for a CAPTCHA Solver Using Computer Vision and Machine Learning

By Nitish Devrani

11 August 2025

1 Introduction

CAPTCHAs (Completely Automated Public Turing tests to tell Computers and Humans Apart) are widely used to protect online services from automated bots. Recent advances in deep learning have greatly improved the ability of machines to solve text-based CAPTCHAs, raising concerns about their security. The objective of this project is to build an end-to-end model that takes an input image and outputs the predicted text string. The provided training set consists of 60 000 images, the validation set contains 20 000 images, and the test set contains 20 000 images without ground truth. Each image has size 640×160 pixels. For training and validation images, upright bounding boxes around each character are available; the number of characters varies per image. Final evaluation uses the Levenshtein error rate, which averages the normalized Levenshtein distances between predicted and ground-truth sequences.

Traditional CAPTCHA solvers relied on pre-processing, segmentation and hand-crafted features. However, the complex distortions and variability in modern CAPTCHAs make segmentation brittle and limit recognition accuracy. Deep learning methods have emerged as the dominant approach for both CAPTCHA recognition and general scene text recognition because they learn rich features from data and can handle noisy and distorted inputs. This proposal builds on insights from recent research to construct a robust baseline and proposes techniques for addressing further challenges in parts 3 and 4 of the project.

2 Literature Review of Related Work

This section reviews relevant literature on CAPTCHA recognition, scene text recognition, object detection, sequence modelling and data augmentation. We discuss traditional segmentation-based methods, deep convolutional approaches, recurrent sequence models with Connectionist Temporal Classification (CTC), transformer-based architectures and detection–recognition pipelines. Table 1 at the end of the section summarises these approaches and their advantages and disadvantages.

2.1 Segmentation-Based Methods

Early CAPTCHA recognition pipelines followed a two-stage procedure: preprocess the input image to remove noise, then segment the image into individual character regions using morphological operations (e.g., erosion, dilation, connected components) and heuristics such as projection profiles. The segmented character was then classified using a multiclass classifier such as SVM or a simple neural network. Wang et al. surveyed traditional methods and noted that segmentation methods often rely on local minima in projection values or watershed algorithms and suffer from poor accuracy (around 38%–60%) when characters overlap or stick together. Direct segmentation of adhesive characters leads to broken character parts and aggravates the classification task. Another limitation is that the extracted features based on global color or texture descriptors cannot accurately represent characters in complex backgrounds. Due to class imbalance and heuristic parameter tuning, traditional segmentation methods struggled to generalize to diverse CAPTCHA styles.

2.2 Deep Convolutional Classification

With the rise of deep learning, convolutional neural networks (CNNs) replaced hand-crafted features. DenseNet and ResNet variants were applied directly to CAPTCHA images for multi-class classification. Wang *et al.* improved DenseNet by reducing the number of convolutional blocks and constructing a multi-task classifier for different CAPTCHA lengths. Their DFCR network reduced memory consumption and achieved over 99.9% accuracy on noisy Chinese and English CAPTCHAs. However, these approaches treat CAPTCHA recognition as a fixed-length classification problem; each character position corresponds to a separate softmax classifier. Variable-length CAPTCHAs require designing multiple output heads or padding sequences, which complicates training. Moreover, classification models assume a deterministic alignment between image regions and characters, making them less flexible when characters are irregularly spaced or rotated.

2.3 Convolutional Recurrent Networks with CTC

An influential architecture for image-based sequence recognition is the convolutional recurrent neural network (CRNN) introduced by Shi *et al.*, which integrates a CNN for feature extraction, a bi-directional recurrent neural network (RNN) for sequence modelling and a transcription layer based on Connectionist Temporal Classification (CTC). CRNNs are end-to-end trainable and can handle sequences of arbitrary length without explicit segmentation; they map the width dimension of feature maps to a sequence of feature vectors and let CTC learn the alignment between predicted and ground-truth sequences. CTC introduces a blank symbol and uses the forward–backward algorithm to marginalise over all possible alignments. This removes the need to segment characters and allows parallel decoding during training. CRNNs have become the baseline for scene text recognition and are robust to moderate distortions. However, RNNs process sequences sequentially, making inference slower compared to fully convolutional or transformer-based models.

2.4 Attention-Based and Transformer-Based STR

To improve recognition accuracy and capture long-range dependencies, attention-based encoder–decoder models were introduced for scene text recognition. These models consist of a CNN encoder and an autoregressive decoder with cross-attention that aligns visual and linguistic features. Attention-based methods achieve state-of-the-art accuracy but suffer from high computational complexity because each decoding step attends to the entire visual feature map. The complexity grows proportionally with the number of predicted characters and the size of the feature map. To reduce latency, researchers proposed the ViTSTR-Transducer, a cross-attention-free framework inspired by RNN-Transducers. ViTSTR-Transducer uses a vision transformer (ViT) encoder and an autoregressive language model without cross-attention; it achieves competitive recognition accuracy while reducing decoding FLOPs and inference latency. Transformer-based models like ViTSTR and cascaded transformers leverage self-attention to capture long-range dependencies and offer parameter efficiency, but they require substantial GPU memory and training data.

2.5 Object Detection and Two-Stage Pipelines

Some researchers propose to detect each character region using object detection networks and then recognise the characters individually. YOLO (You Only Look Once) is a real-time object detector that applies a single neural network to the entire image and predicts bounding boxes and class probabilities in one pass, processing images at 30 fps and achieving 57.9% mAP on COCO. Recent work extended YOLO to arbitrarily oriented text detection. R-YOLO introduces rotated anchor boxes and a Rotational Distance IoU Non-Maximum Suppression algorithm to detect arbitrarily oriented text. It extracts multi-scale features and significantly improves detection efficiency; on the ICDAR2015 dataset it achieved an F-measure of 82.3% at 62.5 fps. Detection-based pipelines are advantageous when bounding boxes are needed (as in Part 4 of the project). However, they introduce additional complexity: one must cluster detections into sequences and ensure correct left-to-right ordering. Detection also requires handling class imbalance between background and character classes; focal loss down-weights easy examples and focuses learning on hard examples to address this issue.

Table 1: Summary of approaches for CAPTCHA and scene text recognition. The table lists key characteristics, advantages and disadvantages. Each approach is discussed in more detail in the text.

Approach	**Advantages**	**Disadvantages**
Segmentation + classification	Conceptually simple; can leverage classical image processing; bounding boxes are explicit.	Manual tuning required; segmentation errors due to overlapping/distorted characters lead to low recognition rates (38%–60%); sensitive to noise; difficult to handle variable fonts and backgrounds.
Multi-class CNN (e.g., DFCR)	End-to-end training; high accuracy ($>$ 99%) on fixed-length CAPTCHAs; eliminates manual segmentation; DenseNet alleviates vanishing gradients and reduces parameters.	Requires fixed number of output heads; not flexible for variable-length CAPTCHAs; classification of each position assumes deterministic alignment; heavy memory consumption despite improvements.
CRNN with CTC	End-to-end trainable; handles variable-length sequences; no character segmentation; robust to distortions; widely used baseline.	Sequence decoding via RNN is slower than fully convolutional models; CTC can misbehave when characters repeat; struggles with extreme rotations or heavy occlusion.
Attention-based encoder–decoder	Learns an internal language model; aligns visual and linguistic features with cross-attention; achieves state-of-the-art accuracy.	Cross-attention is computationally expensive; inference latency increases with sequence length and feature map size; large models require significant GPU memory.
Transformer (ViTSTR, cascaded)	Captures long-range dependencies with self-attention; parameter-efficient; cascaded models reduce redundant tokens and improve efficiency; cross-attention-free ViTSTR-Transducer achieves competitive accuracy with lower latency.	Requires large training datasets and GPU resources; pre-training beneficial; may underperform on small datasets or extreme noise; still computationally heavier than CNNs.
Detection + recognition pipeline (e.g., YOLO, R-YOLO)	Explicitly localises characters; useful when bounding boxes are needed (parts 3–4); detection networks like YOLO achieve real-time performance; R-YOLO handles arbitrarily oriented text with rotated anchors and improved NMS.	Requires two stages (detection then classification), increasing pipeline complexity; bounding box sorting and association for sequence ordering; class imbalance between background and characters; orientation detection may require additional angle regression.

3 Method Selection and Justification

Given the project requirements and dataset characteristics, we select an end-to-end convolutional recurrent neural network with Connectionist Temporal Classification (CRNN-CTC) as our primary solution. This choice balances recognition accuracy, implementation complexity and computational efficiency. The reasons for this choice are summarized below:

- **Variable-length sequences:** The number of characters per CAPTCHA is variable. CRNN-CTC naturally handles sequences of arbitrary length without requiring multiple output heads or padding, unlike fixed-length CNN classifiers.
- **No explicit segmentation:** CTC removes the need to segment characters; the model learns to align features with character labels implicitly. This is crucial because segmentation of overlapping or distorted characters is prone to error.
- **Efficient inference:** Although RNNs are sequential, CRNNs are significantly lighter than transformer-based models and require less GPU memory. They achieve strong baseline performance on scene text benchmarks.
- **Extensibility:** Bounding box annotations are available in the training set. We can optionally incorporate a detection head to predict bounding boxes (multi-task learning) or use the boxes to crop training patches for auxiliary classification losses.
- **Data availability:** The dataset is moderately large (60k training images), but not millions of images. Transformer-based models generally benefit from very large corpora and heavy augmentation; CRNN is more suitable when data is limited.

The selected approach thus prioritizes practicality and reliability. In later phases we can extend the baseline with transformer decoders or ensemble techniques, but CRNN provides a solid foundation for the primary solution.

4 Data Pipeline and Augmentation Strategies

An effective data pipeline is critical for training robust models. The pipeline for this project includes input pre-processing, optional bounding box cropping, augmentation and batching.

4.1 Pre-processing

- **Grayscale conversion and normalization:** Convert RGB input to grayscale by averaging channels (or using the luminance formula) and scale pixel values to $[0, 1]$. Normalization accelerates convergence and reduces sensitivity to illumination.
- **Optional bounding box cropping:** For each training image, bounding box annotations for characters are provided. These boxes can be used to crop character patches to train auxiliary classification heads. We also compute the tightest horizontal bounding box covering all characters and crop the image accordingly to remove large empty margins. Such cropping helps the network focus on relevant regions.
- **Sequence encoding:** Encode target strings as sequences of integers corresponding to the 36 allowable characters (digits and uppercase letters) plus a blank symbol for CTC.

4.2 Augmentation

Data augmentation expands the training set and helps the model generalise to unseen distortions. Augmentations will be applied on the fly during training. Inspired by common image augmentation techniques, we propose the following transformations:

- **Geometric transformations:**
  - *Random rotation:* Rotate the image by a random angle sampled from $[-15°, 15°]$ to simulate small tilts. Larger rotations will be addressed in part 3.
  - *Random shear and perspective warp:* Apply horizontal/vertical shearing and perspective transformation to mimic characters slanted in different directions.
  - *Scaling and cropping:* Randomly scale the image within a range (e.g., $[0.9, 1.1]$) and crop to the original size, preserving aspect ratio.
  - *Translation:* Shift the image horizontally or vertically by a few pixels with wrap or constant padding.
- **Photometric transformations:**
  - *Brightness and contrast jitter:* Randomly adjust brightness, contrast and saturation within preset ranges.
  - *Color jitter:* Since CAPTCHAs use multiple colors, convert to three channels and apply hue shifts or convert to grayscale with random probability.
- **Noise and blur:**
  - *Gaussian noise:* Add additive Gaussian noise with zero mean and random variance.
  - *Salt-and-pepper noise:* Randomly flip pixel values to simulate pixel corruption.
  - *Motion/gaussian blur:* Apply a small kernel to blur the image, imitating out-of-focus or motion blur.
- **Line distractors and shapes:** Draw random straight lines across the image using random colors, thickness and positions. This simulates the distractor lines in the dataset's training images. Similarly, add random shapes (e.g., circles or ellipses) with low opacity.
- **Background replacement:** Replace the background with a randomly selected texture or noise pattern. To achieve this, extract character masks using thresholding and paste them on random backgrounds. This ensures invariance to complex backgrounds in parts 3–4.
- **Mixup and Cutmix:** With a small probability, blend two training images by weighted averaging (mixup) or cut and paste character regions (cutmix) to increase diversity.

Augmentation policies should be balanced; applying too many transformations per image might produce unrealistic samples. We will adopt a random augmentation pipeline where each transformation is applied with some probability. The pipeline can be tuned on the validation set to maximise performance.

5 Architecture and Training Plan

The overall architecture combines a CNN backbone for feature extraction, a sequence modelling stage via bi-directional LSTMs and a transcription layer using CTC. Figure **??** illustrates the architecture conceptually. The architecture is designed to be fully implemented using only the allowed libraries (`torch`, `torchvision`, `numpy`, etc.).

5.1 CNN Backbone

We select a lightweight CNN to extract features from the input image. The network consists of several convolutional blocks with batch normalization and ReLU activations, followed by max pooling. A possible configuration is:

- Block 1: 3×3 convolution with 64 filters, stride 1, padding 1 $\rightarrow$ batch norm $\rightarrow$ ReLU $\rightarrow$ 2×2 max pooling.

- Block 2: 3×3 conv with 128 filters $\rightarrow$ batch norm $\rightarrow$ ReLU $\rightarrow$ 2×2 max pooling.
- Block 3: 3×3 conv with 256 filters $\rightarrow$ batch norm $\rightarrow$ ReLU $\rightarrow$ 2×1 pooling (only downsample vertically).
- Block 4: 3×3 conv with 256 filters $\rightarrow$ batch norm $\rightarrow$ ReLU $\rightarrow$ 2×1 pooling.
- Block 5: 3×3 conv with 512 filters $\rightarrow$ batch norm $\rightarrow$ ReLU (no pooling).

The input grayscale image of shape $(1, 160, 640)$ is down-sampled in height to a small number of rows (e.g., 1/4 of the original height) while maintaining a relatively large width. The resulting feature map has dimensions (C, H', W') where H' is small. We flatten each vertical column (along height H') into a feature vector of dimension $\mathrm{C} \times H'$; thus the width dimension W' becomes the sequence length. Batch normalization and dropout may be used to regularise the network.

5.2 Recurrent Sequence Model

The sequence of feature vectors is passed to a two-layer bi-directional LSTM. Each LSTM layer has hidden size 256. A bi-directional architecture processes the sequence forwards and backwards, capturing both left-to-right and right-to-left context. The output from the second LSTM layer is projected via a fully connected layer to a tensor of shape (T, N_{classes}), where T is the sequence length and $N_{\text{classes}} = 37$ (26 letters + 10 digits + blank). A softmax over classes at each time step produces probabilities.

5.3 Connectionist Temporal Classification (CTC)

Training uses the CTC loss function. CTC computes the negative log likelihood of the ground-truth label sequence given the predicted distribution over labels and the blank symbol. It efficiently sums over all possible alignments between the predicted sequence of length T and the ground-truth label sequence of length $L \leq T$ using dynamic programming (forward–backward algorithm). During inference, decoding may be performed using simple greedy decoding (taking the most probable class at each time step and collapsing repeats and blanks) or beam search to explore multiple sequences. CTC enables the model to learn alignment without explicit segmentation and imposes a monotonic constraint that characters appear in order. We adopt greedy decoding for efficiency; beam search with width 5 can be employed to improve accuracy if needed.

5.4 Optimizer and Hyperparameters

The network will be trained using the Adam optimizer with an initial learning rate of 1×10^{-3}. We will use a cosine annealing scheduler to reduce the learning rate to 1×10^{-5} over 50 epochs. A batch size of 32 images fits comfortably into a GPU with 8 GB memory. Weight decay (1×10^{-4}) and dropout ($p = 0.2$) help prevent over-fitting. Early stopping on the validation LER will be used. Training will continue for up to 100 epochs or until no improvement on the validation set is observed for 5 consecutive epochs. Data augmentation is applied on the fly. Mixed-precision training (if available) can reduce memory usage and speed up training.

5.5 Multi-Task Extension for Bounding Boxes

While the primary goal is sequence recognition, we propose a multi-task extension to exploit the provided bounding boxes. Parallel to the CTC branch, a detection head predicts upright bounding boxes and class labels for each character. The detection head uses convolutional layers followed by anchor boxes similar to YOLO; focal loss is used to handle class imbalance and emphasise hard examples. The overall loss is a weighted sum of CTC loss for recognition and detection loss (classification + regression). Joint training encourages the network to learn spatially localised features and may improve recognition accuracy. For the oriented bounding box challenge (part 4), the detection head can be adapted to regress rotated bounding boxes using angle parameters, inspired by R-YOLO.

6 Loss Function

6.1 Connectionist Temporal Classification Loss

Let $X \in \mathbb{R}^{T \times N_{\text{classes}}}$ denote the predicted log probabilities (after softmax) at each time step and $Y = (y_1, \ldots, y_L)$ the ground-truth label sequence of length L. CTC defines the set of all valid alignment paths $\Pi(Y)$, where each path is a length-T sequence of labels (including blanks) that, after collapsing repeated labels and removing blanks, yields Y. The CTC loss is

$$\mathcal{L}_{\text{CTC}} = -\ln\left(\sum_{\pi \in \Pi(Y)} \prod_{t=1}^{T} X_{t,\pi_t}\right). \tag{1}$$

The dynamic programming algorithm computes this sum efficiently. During back-propagation, gradients are computed with respect to $X_{t,c}$ for each class c and time step t.

6.2 Detection Loss (Optional)

For the multi-task extension, each anchor predicts a bounding box $(\hat{x}, \hat{y}, \hat{w}, \hat{h})$, an optional rotation angle $\hat{\theta}$ and class probabilities. The loss comprises:

- **Classification loss:** We use the focal loss to address class imbalance between character and background. Focal loss modifies cross entropy by a modulating factor $(1 - p_t)^{\gamma}$, where p_t is the model's estimated probability for the true class and $\gamma > 0$ controls the focusing. It down-weights easy examples and focuses on hard examples.
- **Bounding box regression loss:** We use a smooth L_1 loss between predicted and ground-truth box parameters. For rotated boxes, regression includes the angle difference.
- **Non-maximum suppression:** At inference, Rotational Distance IoU Non-Maximum Suppression (RDIoU-NMS) is applied to remove redundant predictions; this algorithm considers both IoU and angle differences.

The total detection loss is a weighted sum of classification and regression losses. We set weights such that the detection branch does not dominate the recognition branch.

7 Strategies for Parts 3 and 4

7.1 Robustness to Added Degradations (Part 3)

The dataset for part 3 will contain more severe degradations, including larger rotations, random line distractors, noise and complex backgrounds. To handle these:

- **Augmentation:** Intensify augmentation by including larger rotations (up to $\pm 45°$), stronger perspective warps, motion blur, additive noise and random occluders. Generate synthetic CAPTCHAs with additional line and circle distractors to simulate extreme conditions. Data augmentation helps the network learn invariance to these distortions.
- **Rotation-invariant features:** Add spatial transformer networks (STN) or polar transforms at the input of the CNN to normalise rotation. STNs learn to warp the input to an upright orientation, improving robustness to rotations.
- **Test-time augmentation:** At inference, run the model on multiple rotated versions of each test image and combine predictions using majority voting or highest confidence.
- **Uncertainty estimation:** Use model ensembles or Monte-Carlo dropout to estimate prediction confidence. Discard low-confidence predictions or request re-evaluation.

7.2 Challenging Degradations and Oriented Detection (Part 4)

I will choose Option 1: Challenging Degradations. The same strategies as part 3 apply, but we further incorporate adaptive filtering networks. Adaptive CAPTCHA introduces a lightweight CRNN with an adaptive fusion filtering network that reduces parameters and improves accuracy. Incorporating such adaptive filters or attention modules could help focus on relevant character strokes while suppressing distractors. Additionally, using an internal language model (attention-based decoder) can enforce plausible character sequences and reject unrealistic predictions.

References

1. Y. 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*, vol. 39, no. 11, pp. 2298–2304, 2017. The CRNN architecture integrates CNN, RNN and CTC for sequence recognition.

2. Wikipedia contributors, "Connectionist Temporal Classification," *Wikipedia, The Free Encyclopedia*, 2024. CTC adds a blank symbol and computes sequence alignment without segmentation.

3. D. Lin, Y. Liao, X. Wang *et al.*, "Vision Transformer for Fast and Efficient Scene Text Recognition (ViTSTR)," 2021. ViTSTR uses a transformer backbone to achieve efficient recognition.

4. M. Naser and S. Luk, "Efficient and Accurate Scene Text Recognition with Cascaded Transformers," 2025. Cascaded transformers reduce redundant tokens and maintain accuracy.

5. J. Wang *et al.*, "CAPTCHA recognition based on deep convolutional neural network," *Mathematical Biosciences and Engineering*, vol. 16, no. 5, pp. 5851–5861, 2019. DenseNet-based DFCR achieves high accuracy and discusses limitations of traditional segmentation methods.

6. S. Long, X. He and C. Yao, "Scene text detection and recognition: the deep learning era," 2019. Deep learning simplifies the pipeline and yields significant improvements over hand-crafted methods.

7. D. Ren, W. Han *et al.*, "ViTSTR-Transducer: Cross-Attention-Free Vision Transformer Transducer for Scene Text Recognition," 2024. This model eliminates cross attention to reduce latency while maintaining accuracy.

8. C. Sun *et al.*, "Focal Loss for Dense Object Detection," 2017. Focal loss addresses class imbalance by down-weighting well-classified examples.

9. P. P. Belval, "Data augmentation in computer vision: techniques and examples," Lightly AI blog, 2023. Data augmentation improves generalisation by applying geometric and photometric transformations.

10. X. Wang *et al.*, "R-YOLO: A Real-Time Text Detector for Natural Scenes with Arbitrary Rotation," *Sensors*, vol. 21, no. 3, 2021. R-YOLO introduces rotated anchors and RDIoU-NMS to detect arbitrarily oriented text and achieves high accuracy with real-time performance.

11. Y. Tian *et al.*, "Adaptive CAPTCHA recognition using fusion filtering networks," 2024. Adaptive filtering networks and CRNN reduce parameters and improve accuracy.

12. S. Joseph *et al.*, "You Only Look Once: Unified, real-time object detection," 2016. YOLO processes images at 30 fps and predicts bounding boxes and probabilities in one evaluation.