CS4243 Mini Project CAPTCHAS

Team 18: Lo Yong Zhe, Teng Hui Xin Alicia, Chan Wei Hao, Kam Jia Yue, Sivakumar Karthikraj

Deployed CRNN model: https://huggingface.co/spaces/TechRaj/cs4243-miniproject-captcha-recognition

Deployed GAN model: https://huggingface.co/spaces/loyongzhe/CaptchaGANModel

1. Introduction

We developed an end-to-end CAPTCHA recognition framework based on a Convolutional Recurrent Neural Network (CRNN) with Connectionist Temporal Classification (CTC) decoding. The model processes unsegmented CAPTCHA images as continuous sequences, removing the need for explicit character segmentation. A ResNet-based CNN extracts spatial features, which are modeled temporally by a bidirectional LSTM. The CTC layer enables alignment-free training and decoding. This integrated design effectively handles variable-length and distorted CAPTCHAs, achieving high recognition accuracy and strong generalization to noisy inputs.

2. Data Preprocessing

Before training the models, we first cleaned the dataset with the following alterations:

1) **Invalid Data:** 224 invalid images were removed from the dataset. These included images containing non-alphanumeric symbols, watermarks, cut-off characters, and other irregularities.

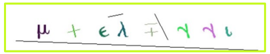


Figure 1. m1el7ggi-0.png

2) **Noise Removal:** Black lines were removed using a color-based masking and inpainting approach. Images were converted to the HSV color space to isolate dark regions, which were then inpainted with OpenCV's algorithm to reconstruct affected areas.

3) **Colour Removal:** To eliminate redundant color information and enhance visual clarity, all images were converted to grayscale and contrast was increased using histogram equalization.

4) **Resizing:** All images were resized and padded to a uniform resolution of 80 × 280 pixels to maintain consistent input dimensions across the dataset and ensure compatibility with the model architecture.

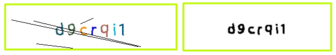
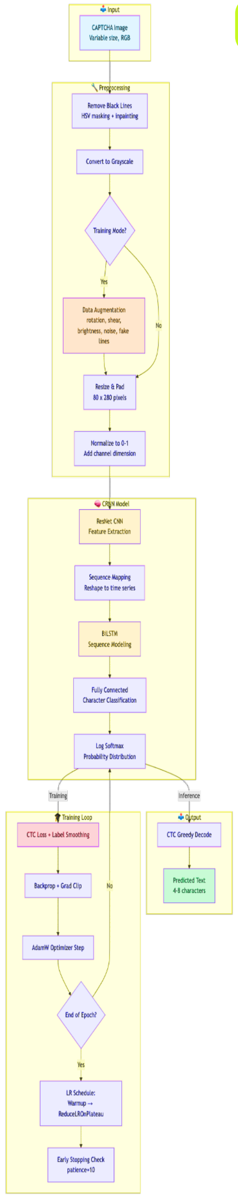


Figure 2. d9crqi1-0.png (Before & After)

5) **Data Augmentation:** The following changes were applied during training to prevent overfitting.

- Random rotation (±5°)
- Random translation (±3, ±2 pixels)
- Brightness jitter (0.8-1.2×)
- Gaussian noise (σ=0.01)
- Black line simulation (1-3px, mimics real CAPTCHA noise)

This improved generalisation by **8-10%.**



Input
CAPTCHA Image Variable size, RGB
Preprocessing
Remove Black Lines HSV masking + inpainting
Convert to Grayscale
Training Mode?
Yes
No
Data Augmentation rotation, shear, brightness, noise, fake lines
Resize & Pad 80 x 280 pixels
Normalize to 0-1 Add channel dimension
CRNN Model
ResNet CNN Feature Extraction
Sequence Mapping Reshape to time series
BiLSTM Sequence Modeling
Fully Connected Character Classification
Log Softmax Probability Distribution
Training
Inference
Training Loop
CTC Loss + Label Smoothing
Backprop + Grad Clip
AdamW Optimizer Step
End of Epoch?
LR Scheduler: Warmup → ReduceLROnPlateau
Early Stopping Check patience=10
Output
CTC Greedy Decode
Predicted Text 4-8 characters

Figure 3. Overview of CRNN–CTC Pipeline

4. Model Architecture

The architecture integrates three main components: a **ResNet convolutional** backbone, a **bidirectional recurrent layer**, and a **CTC classification head** (Figure 4).

Convolutional Feature Extractor (ResNet)

The convolutional front-end uses ResNet-style residual blocks (not pretrained) with channel sizes of 64, 128, 256, and 512, enabling deeper networks without vanishing gradients. This setup captures features at multiple levels while maintaining efficient gradient flow during training. The feature maps are down sampled to a height of **5 pixels** and approximately **70 time steps** in width, producing per-timestep feature vectors of around **2560 dimensions.**

Bidirectional Recurrent Encoder (BiLSTM)

The features are passed into a **two-layer Bidirectional LSTM** with a **hidden size of 384 per direction**, generating a **70 × 768 output tensor.** This allows the model to capture contextual information from both past and future positions in the sequence.

Regularisation Strategy

- **Spatial Dropout** (0.3) after final CNN layer
- **LSTM Dropout** (0.5) between recurrent layers
- **Weight Decay** (1e-4) to prevent overfitting
- **Label Smoothing** (0.1) for CTC loss stability

CTC Output Head

The recurrent outputs are projected to **37 logits**, representing the character set (a–z, 0–9) plus a blank token. A **log-softmax operation** is applied for numerical stability.

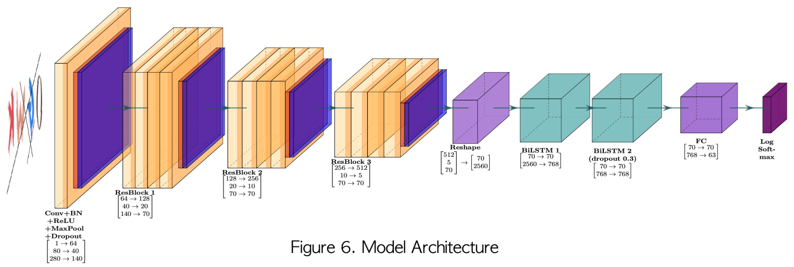


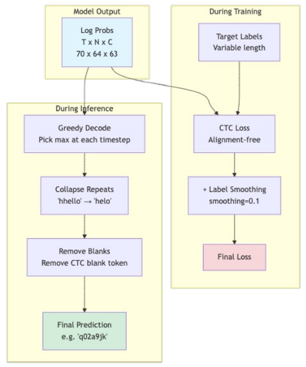
Figure 6. Model Architecture

5. Training & Decoding

The model is trained using **CTC loss** with **label smoothing (0.1)** to improve generalisation. During training, **greedy decoding** (argmax at each timestep) is used for efficiency. We explored beam search but found marginal gains (~0.5%) did not justify the 3x slower inference.

Training Strategy (100 epochs)

- **Optimizer**: AdamW (lr=0.001, weight_decay=1e-4)
- **LR Schedule**: Warmup (5 epochs→ReduceLROnPlateau
- **Early stopping**: patience =10 epochs
- **Best checkpoint**: Epoch 55-60



Model Output
Log Probs T x N x C 70 x 64 x 63
During Training
Target Labels Variable length
During Inference
Greedy Decode Pick max at each timestep
CTC Loss Alignment-free
Collapse Repeats 'hhello' → 'helo'
+ Label Smoothing smoothing=0.1
Remove Blanks Remove CTC blank token
Final Loss
Final Prediction e.g. 'q02a9jk'

Figure 7: CTC Training Pipeline

6. Results

55.6% Sequence Accuracy	85.8% Character Accuracy

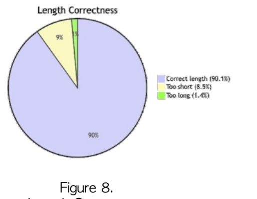


Figure 8. Length Correctness

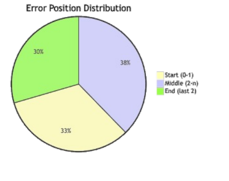


Figure 9. Error Position Distribution

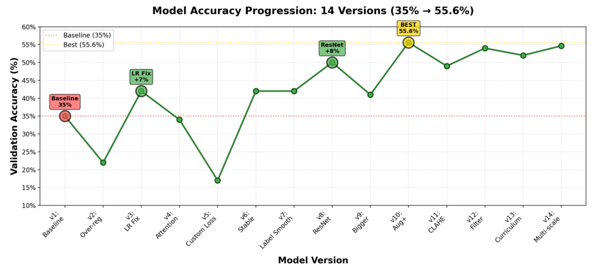


Figure 10. Model Accuracy Progression

3. Secondary Approach

Segmentation

This method uses cv2.RETR_EXTERNAL contour detection to find outer shapes, sorts contours from left to right based on position, merges vertically close bounding boxes to handle connected parts, and specially disconnected characters like 'i' an 'j', and finally crops each merged region to produce individual character images from the captcha.

Figure 4. x7ru6bb.png

Segmentation (enhancement)

The previous algorithm relies on a fixed rule approach to denoise the images and determine the contours. We made an enhancement by training a U-Net model to isolate the characters from the background and noise using the output of data preprocessing pipeline as ground truth. The cleaned image from U-Net model is segmented with the same algorithm.

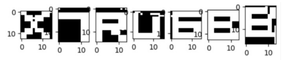


Figure 5. Segmentation of x7ru6bb.png

Classification

We trained a simple CNN classifier on the segmented captcha dataset by resizing each crop, labelling it with each ground-truth character, and feeding batches. The network learns to map each segmented character image into its correct class label.

50.82% Character Accuracy	17.45% Sequence Accuracy

7. Generative Adversarial Networks (GANs)

Data & Conditioning

We train a text-conditioned GAN on a variety of data from the given training set and Kaggle. Each image is cleaned and converted into a one-hot condition vector shared by both the generator and the discriminator.

Generator (G)

The generator maps random noise plus a learned text embedding through a stack of upsampling convolutional blocks to produce 64 × 256 grayscale CAPTCHA images.

Discriminator (D)

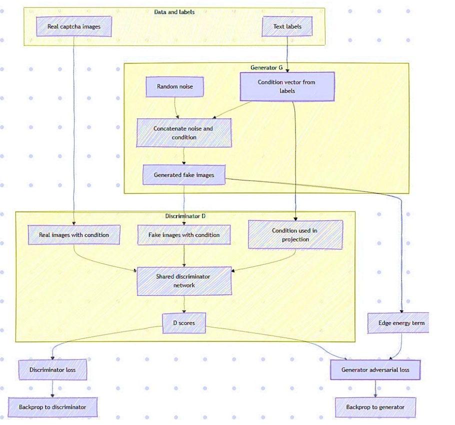
The discriminator uses a projection architecture that combines image features with the same text condition, so it judges not only realism but also whether an image matches its label.

Training & Stabilisation

Training uses hinge losses with R1 regularization, together with DiffAug, instance noise, and a small edge-energy loss that encourages sharp character strokes.

Sampling

An exponential-moving-average (EMA) copy of the generator is used for sampling, giving cleaner and more stable synthetic CAPTCHA images.



Data and labels
Real captcha images
Text labels
Generator G
Random noise
Condition vector from labels
Concatenate noise and condition
Generated fake images
Discriminator D
Real images with condition
Fake images with condition
Condition used in projection
Shared discriminator network
D scores
Edge energy term
Discriminator loss
Generator adversarial loss
Backprop to discriminator
Backprop to generator

Figure 11. Overview of GAN Model

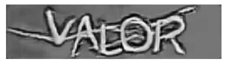


Figure 12. "valor" generated by GAN model

8. Team Contributions

All: Research and ideation of approaches, weekly meeting participation
Lo Yong Zhe: GAN model development
Teng Hui Xin Alicia: Data preprocessing and cleaning, poster design and writing
Chan Wei Hao: Image Segmentation (Enhancement)
Kam Jia Yue: Image Segmentation
Sivakumar Karthikraj: CRNN model development and deployment