Project Proposal: A CRNN (CNN + BiLSTM + CTC) CAPTCHA Solver

Computer Vision (Winter 2024/2025)

Muhid Abid Siddiqui

August 11, 2025

1 Literature review of related methods

CTC and alignment-free learning. Connectionist Temporal Classification (CTC) lets us train without character-level alignment by summing over all valid monotonic paths with a special *blank* symbol [2]. Together with Long Short-Term Memory (LSTM) units [3], this has been very effective for speech, handwriting, and short text strings where timing is uncertain [4].

CRNN as a practical baseline. Shi *et al.* proposed the CNN→RNN→CTC pipeline (CRNN) that reads text directly from images without explicit segmentation; it is simple and fast to implement [1]. For CAPTCHA-style OCR (short strings, moderate warps and thin lines), CRNN usually gives reliable accuracy with modest compute.

Geometry rectification. Spatial Transformer Networks (STN) learn small geometric warps that can reduce rotation, shear, and perspective before recognition [5]. I keep the baseline simple, but a light STN/TPS module is an easy add-on if slant or curvature hurts results.

Attention/Transformers. Transformer decoders bring stronger language modeling and help when characters overlap or layouts are more complex. TrOCR is a well-known example with strong results, but it is heavier to train [6]. Under our constraints, CRNN+CTC remains the main choice, with attention models as optional extension.

Takeaway: CRNN with CTC is a good fit for this project: it is simple, efficient, and proven for short OCR strings.

2 Method selection

I choose a **CRNN** approach: a compact CNN turns the image into a left-to-right sequence; a two-layer **BiLSTM** models context; and **CTC** decodes the final string.

Why this method for our data? The course dataset has 640×160 images and strings over {0–9, A–Z}, with some rotation, shear, multi-color backgrounds, and noise. CRNN handles variable length and mild misalignment naturally, and it does not require character boxes during training. It also fits the allowed libraries (PyTorch, NumPy, Pillow, torchvision, matplotlib, scikit-learn).

Pros. Easy to implement and tune; no explicit detection; fast to train on a single mid-range GPU.

Cons and fallback. With very strong overlaps/occlusions, CTC may struggle. As backup, I will (i) try a small beam search and (ii) optionally add a light STN if validation errors show heavy slant cases.

3 Data pipeline including augmentation strategies

Preprocessing. Convert to grayscale, normalize to $[0, 1]$, resize short side to $H \in \{64, 96, 128\}$, then pad/crop width to $W \approx 512$ so the time length T after the CNN is stable.

Label encoding. Map {0–9, A–Z} to IDs {0..35}; reserve index 36 for the CTC *blank*. For each sample I store input length T and target length L for `CTCLoss`.

Train-time augmentations (CAPTCHA-style, moderate).

- **Geometric:** small rotation ($\pm 12°$), shear (± 0.15), mild perspective.
- **Photometric:** Gaussian noise ($\sigma \leq 0.05$), Gaussian blur ($\sigma \leq 1.0$), contrast jitter.
- **Distractors:** thin random lines/arcs (1–3 px), small blobs, and gentle textures to imitate real CAPTCHAs.
- **Fonts/spacing:** optional kerning/weight jitter in synthetic "hard examples".

Validation. Only deterministic resize/pad + normalization (no randomness). I will report **Levenshtein Error Rate (LER)** on the val set and keep the best model by lowest LER (as defined by the assignment).

Notes on splits and boxes. Train/val/test are 60k/20k/20k images. Boxes are provided for train/val but are not needed for the baseline; I may use them later if I try the detection-based bonus.

4 Training including architecture and optimizer

Architecture.

- **CNN:** small VGG/ResNet-18 style with BatchNorm+ReLU. Keep width stride at 4–$8\times$ (not $16\times$) to preserve time steps.
- **Sequence formation:** average-pool features over height to get a left-to-right sequence $\mathbf{F} \in \mathbb{R}^{T\times C}$.
- **BiLSTM head:** 2 layers, hidden 256–512 (bidirectional).
- **Classifier:** Linear to $|\mathcal{A}| = 37$ (36 chars + blank).

Optimization and schedule. AdamW (lr 3×10^{-4}, weight decay 1×10^{-4}), cosine decay with 5-epoch warmup [7, 8], batch ≈ 128, 80–150 epochs depending on convergence. Mixed precision (AMP) and gradient clipping (1.0).

Regularization. Dropout 0.3 in BiLSTM; optional small label smoothing (≤ 0.1) before CTC.

Decoding. Greedy CTC during training/validation; try beam width 5 for the final report.

Diagnostics and ablations. I will check character confusions and common edit types (ins/del/-sub), then run small ablations: stride $4\times$ vs. $8\times$; BiLSTM depth 1 vs. 2; hidden 256 vs. 512; greedy vs. beam; light vs. moderate augmentation.

Compute and reproducibility. One RTX 3060-class GPU; about 6–10 hours per full run; 10–20 GB disk. I will fix seeds, save configs, and log loss/LER; the best checkpoint is picked by lowest val LER.

Minimal PyTorch skeleton (for clarity).

Listing 1: CRNN + CTC training loop (high level).

```
class CRNN(nn.Module):
    def __init__(self, num_classes=37, hidden=256, layers=2):
        super().__init__()
        self.cnn = make_cnn_backbone(stride_w=4)   # keep width
           stride small (4-8x)
        C_out = 256
        self.proj = nn.Conv2d(self.cnn.out_channels, C_out, 1)
        self.rnn = nn.LSTM(C_out, hidden, num_layers=layers,
                           bidirectional=True, batch_first=False,
                              dropout=0.3)
        self.fc  = nn.Linear(2*hidden, num_classes)

    def forward(self, x):                  # x: (B,1,H,W) if grayscale
        f = self.proj(self.cnn(x))         # (B,C',H',W')
```

```
        f = f.mean(dim=2)                    # pool over height -> (B,C
           ',W')
        f = f.permute(2,0,1)                 # (T,B,C')
        y = self.fc(self.rnn(f)[0])          # (T,B,num_classes)
        return y

ctc = nn.CTCLoss(blank=36, zero_infinity=True)
for X, labels, T_in, L_tgt in train_loader:
    X = augment(X)
    logp = F.log_softmax(model(X), dim=-1)
    loss = ctc(logp, labels, T_in, L_tgt)
```

5 Detailed description of the loss function(s)

Let logits be $\mathbf{Z} \in \mathbb{R}^{T\times|\mathcal{A}|}$ and $\mathbf{p}_t = \mathrm{softmax}(\mathbf{Z}_t)$. For target $\mathbf{y} = (y_1, \ldots, y_L)$ over the 36 symbols (plus blank $\varnothing$), CTC defines

$$P(\mathbf{y} \mid \mathbf{X}) = \sum_{\pi\in\mathcal{B}^{-1}(\mathbf{y})} \prod_{t=1}^{T} p_{t,\pi_t},$$

where π is a path over $\mathcal{A}$ including blanks and repeats, and $\mathcal{B}$ collapses repeats and removes blanks. We minimize $\mathcal{L}_{\text{CTC}} = -\log P(\mathbf{y} \mid \mathbf{X})$ via `torch.nn.CTCLoss`. In practice I feed *log-softmax*, set `blank=36`, and pass the correct input/label lengths. I will report the official **Levenshtein Error Rate (LER)** on validation as the selection metric.

6 Part 3 and part 4 strategies

Part 3 (Added degradations). No extra training set is provided, so I focus on robustness without new labels:

1. **Stronger but bounded augs:** heavier line distractors/occlusions and higher-variance noise.
2. **EMA self-ensembling:** keep an exponential moving average teacher to stabilize predictions [9].
3. **TTA + confidence switch:** average a few test-time crops/scales; if mean frame entropy is high, fall back to beam decoding.

Part 4 (choose one).

- **Option 1: Challenging degradations and distractors.** A short curriculum from clean to harder: circular distractors, non-ASCII distractors, harder fonts, more blur, and partial character overlap. I will stage these over epochs to avoid early collapse.

- **Option 2: Oriented Bounding Box Implementation.** An anchor-free rotated-box head predicting $(\Delta x, \Delta y, w, h, \theta)$ and a 36-way class. Losses: focal (classification), Smooth-ℓ_1 for (x, y, w, h), and angle loss on $(\sin\theta, \cos\theta)$. Report mAP@0.5:0.95 and optionally feed detected crops to the recognizer for a detect-then-recognize variant.

References

[1] B. Shi, X. Bai, C. Yao, "An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition," *IEEE TPAMI*, 2017. arXiv:1507.05717.

[2] A. Graves, S. Fernandez, F. Gomez, J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," *ICML*, 2006. PDF link.

[3] S. Hochreiter, J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, 1997. doi:10.1162/neco.1997.9.8.1735.

[4] A. Graves, N. Jaitly, "Towards End-to-End Speech Recognition with Recurrent Neural Networks," *ICML*, 2014. arXiv:1412.5567.

[5] M. Jaderberg, K. Simonyan, A. Zisserman, K. Kavukcuoglu, "Spatial Transformer Networks," *NIPS*, 2015. arXiv:1506.02025.

[6] M. Li, C.-C. Lin, T. Chen, et al., "TrOCR: Transformer-based Optical Character Recognition with Pretrained Models," *NeurIPS*, 2021. arXiv:2109.10282.

[7] I. Loshchilov, F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," *ICLR*, 2017 (workshop). arXiv:1608.03983.

[8] I. Loshchilov, F. Hutter, "Decoupled Weight Decay Regularization," *ICLR*, 2019. arXiv:1711.05101.

[9] A. Tarvainen, H. Valpola, "Mean Teachers are Better Role Models: Weight-Averaged Consistency Targets Improve Semi-Supervised Deep Learning Results," *NIPS*, 2017. arXiv:1703.01780.