Segmentation-Free CAPTCHA Recognition System (PoC)

This document outlines a Proof of Concept (PoC) for a comprehensive, segmentation-free CAPTCHA recognition system. It leverages a deep learning architecture to tackle distorted 5-character alphanumeric CAPTCHAs from the Kaggle **"parsasam/captcha-dataset"**.1. Project Objective and Key Innovations

The system's core is an end-to-end learning approach that eliminates traditional character segmentation challenges.

- **Architecture:** Combines Convolutional Neural Networks (CNNs) for spatial feature extraction and Bidirectional LSTMs (BiLSTMs) for sequential modeling.
- **Loss Function:** Uses the **Connectionist Temporal Classification (CTC) loss** for alignment-free training, which handles noise, font variations, and distortions in raw CAPTCHA images without manual feature engineering.
- **Key Innovations:**
  - Custom `DataGenerator` for memory-efficient batching of large datasets (approx. 113k images).
  - Multi-input model design to meet CTC requirements.
  - Greedy decoding strategy for inference.
- **Applications:** Robust recognition suitable for security testing, automated OCR validation, and web vulnerability assessment.

2. Technical Environment Setup

The system is designed for execution in a Python environment (Google Colab/macOS compatible).Dependencies
!pip install numpy pandas matplotlib opencv-python tensorflow scikit-learn
Core Libraries

- **Data Handling:** NumPy, Pandas
- **Image I/O & Preprocessing:** OpenCV (`cv2`)
- **Visualization:** Matplotlib
- **Splitting:** Scikit-learn
- **Modeling:** TensorFlow/Keras
- **Dataset Access:** KaggleHub

Reproducibility and Data Access

Reproducibility is ensured via `np.random.seed(1234)` and `tf.random.set_seed(1234)`. The dataset is downloaded automatically:
path = kagglehub.dataset_download("parsasam/captcha-dataset")
The dataset contains `.jpg` images, with the 5-character alphanumeric label embedded in the filename (e.g., "ABC12.jpg" → "ABC12").3. Dataset Processing PipelineLoading and Metadata Extraction

- **Images:** `images = list(Path(path).glob("*.jpg"))` (approx. 20k+ samples)

- **Labels:** Extracted from filenames (e.g., `img_path.name.split(".jpg")[0]`).
- **Dataset Structure:** A Pandas DataFrame with `img_path` and `label` columns, which is shuffled (`.sample(frac=1)`).
- **Character Set:** The system handles 36 classes (0-9, A-Z).
- **Data Split:** 10% of the data is sampled for rapid iteration, then split 90/10 for training and validation. Labels are validated using an `is_valid_captcha()` function.

Preprocessing (`generate_arrays`)

1. **Grayscale Conversion:** `cv2.cvtColor(BGR2GRAY)`
2. **Resizing:** To `(200, 50)`.
3. **Normalization:** `/255.0`.
4. **Output Shape:** `(N, 50, 200)` arrays.
5. **Transposition:** To `(200, 50, 1)` to align with the CTC sequence convention (time, height, channels).

4. Custom DataGenerator Implementation

The system uses a custom Keras `keras.utils.Sequence` subclass for memory-efficient, infinite batching.Key Components

- **Batching:** `batch_size=16`.
- **Input Dimensions:** `img_width=200`, `img_height=50`, `max_length=5`.
- `__getitem__(idx)`: Creates batched arrays, including:
  - `batch_images` (`(batch_len, 200, 50, 1)`)
  - `batch_labels` (`(batch_len, 5)`)
  - CTC-specific inputs: `input_length` and `label_length`.
- **Output:** Returns a dictionary of batch inputs and dummy outputs (as CTC computes the loss internally).

5. CNN-BiLSTM-CTC Model Architecture

The `build_model()` function constructs a multi-input Keras model.Multi-Input Design
input_img = Input((200,50,1), name='input_data')
labels = Input((5,), name='input_label')
input_length = Input((1,), name='input_length')
label_length = Input((1,), name='input_length')
Model Layers

1. **CNN Backbone (x4 Downsampling):**
   - `Conv2D(32)` → `MaxPool(2,2)`
   - `Conv2D(64)` → `MaxPool(2,2)`
   - Reshape: `(50, 50*64)`
   - `Dense(64, ReLU)` + `Dropout(0.2)`
2. **RNN Sequence Modeling:**

    - `BiLSTM(128, return_sequences=True, dropout=0.2)`
    - `BiLSTM(64, return_sequences=True, dropout=0.25)`
3. **Output Layer:**
    - `Dense(37, softmax)` (36 characters + 1 blank label for CTC).

CTC Layer and Compilation

The model incorporates a custom CTC layer for loss calculation:
self.loss_fn = keras.backend.ctc_batch_cost
loss = self.loss_fn(y_true, y_pred, input_length, label_length)
The model is compiled using `Adam(lr=0.001)`.6. Training Protocol and InferenceTraining

- **Optimizer:** Adam.
- **Early Stopping:** Used to monitor `val_loss` with a `patience=5` to prevent overfitting and restore the best weights.
- **Execution:** `model.fit(train_generator, validation_data=valid_generator, epochs=50, callbacks=[es])`

Inference (Decoding)

The prediction model extracts the output from the `input_data` to the softmax layer. Predictions are decoded using a greedy approach:

- The output is trimmed: `pred = pred[:,:-2]` (to trim padding).
- **Decoding:** `keras.backend.ctc_decode(pred, input_length=np.ones(batch)*T, greedy=True)`
- **Mapping:** The resulting indices are mapped back to the 36 alphanumeric characters.