Project: Creating OCR software for image recognition

Building Model 1

Our goal with this project was to create an OCR web app that takes in a handwritten document and predicts the output of the document. In order to accomplish this task, we started working on the first model and we trained an LSTM model on data we found online. We made a start by importing all the necessary packages and importing the py file which we define all our functions and classes.

```
import myProject # Import the package that contains my py files for
functions and classes
from myProject import model1

# Import all necessary libraries
import os
import jax
import jax.numpy as jnp
import numpy as np
import optax
import string
import random
import editdistance
import orbax.checkpoint
from flax import linen as nn
from PIL import Image, UnidentifiedImageError
```

Then, we can start working on model1. First, we created a character mapping between characters and numerical indices for a custom character encoding system.

```
# Define the character set that includes lowercase English letters (a-
z), digits (0-9) and common
# punctuation marks and special characters
CHARS = string.ascii_lowercase + string.digits + ".,;:'\"!?()&- "
# Create a mapping from each character to a unique index (starting
from 1)
char_to_index = {char: i + 1 for i, char in enumerate(CHARS)}
# Create a reverse mapping from index to character
index_to_char = {i: char for char, i in char_to_index.items()}
```

In order to train this model, we needed to find a large dataset of words. Thankfully, we were able to use the IAM words dataset. Once we downloaded this dataset, we found out that it contained over 100k+ images for us to use. However, the way the data was formatted proved to be problematic. The dataset was structured as a bunch of nested folders in which the images were contained in. To extract the images, we needed to create a class to help with extracting the images.

```python
from myProject.model1 import IAMDataset
```

This **init** method initializes an instance of the IAMDataset class, which is used for loading handwritten text images and their corresponding labels. The method takes three parameters: img_root, which specifies the directory containing the images; label_file, which points to a text file that maps image filenames to their transcriptions; and img_size, a tuple that defines the target size for resizing images, defaulting to (32, 128).

Inside the constructor, self.img_root stores the image directory path, and self.img_size keeps the desired image dimensions for consistency in model input. The self.valid_samples list is initialized as an empty list and will later be populated with valid (image_path, encoded_label) pairs after filtering out missing or corrupted images. This setup prepares the dataset class for further processing, ensuring that only properly formatted and accessible images are included during training and evaluation.

This portion of the **init** method is responsible for reading the label file, extracting image-text pairs, and filtering out invalid samples before storing them in self.valid_samples. It begins by opening the label file in read mode with UTF-8 encoding to handle special characters properly. The file contains mappings between image filenames and their corresponding transcriptions, which are processed line by line.

Each line is checked to see if it starts with a # symbol, which typically indicates a comment or metadata. These lines are ignored to ensure only relevant data is processed. The remaining lines are split into separate components using spaces as delimiters. The first part of the line is the image filename, while the actual text transcription is found from index 8 onward. These parts are joined together into a complete sentence and converted to lowercase for consistency.

To prepare the text for machine learning, each character is mapped to a numerical index using a predefined dictionary (char_to_index). This ensures that the model receives structured numerical input instead of raw text. Any characters not found in the predefined set are ignored.

The method then reconstructs the full image file path by extracting the folder names from the filename. The IAM dataset follows a structured directory format, where images are stored in subfolders based on the first two segments of the filename. Using os.path.join(), the method dynamically builds the correct file path, ensuring compatibility across different operating systems with os.path.normpath().

Finally, the method verifies whether the image file exists and is valid using the _is_valid_image() function. If the image passes both checks, its path and corresponding numerical label are added to self.valid_samples, ensuring that only usable image-text pairs are included in the dataset. This filtering step is crucial for preventing errors during training and ensuring that the dataset is properly structured for handwriting recognition.

The **len** method defines the length behavior of the IAMDataset class, allowing it to work with Python's built-in len() function.

The **getitem** and _is_valid_image methods play a crucial role in retrieving and validating dataset samples.

The **getitem** method is responsible for retrieving and preprocessing an image-label pair from the dataset when accessed using an index. It first extracts the corresponding image path and label from self.valid_samples. The image is then opened using the Pillow (PIL) library and converted to grayscale ("L") to simplify the input for the model. To ensure uniformity, the image is resized to (128, 32) pixels using bilinear interpolation.

After resizing, the image is normalized by converting it into a NumPy array and scaling its pixel values to the range [0,1] by dividing by 255.0. Since deep learning models typically expect images with a specific number of channels, the method expands the array's dimensions to include a channel axis (axis=-1). The processed image is then returned alongside its corresponding label, which is converted into a JAX NumPy (jnp) array for compatibility with JAX-based models. If an image is corrupted or cannot be loaded, the method gracefully handles the exception and returns None, preventing potential errors during training.

The is_valid_image method serves as a safeguard against corrupt or unreadable images. It attempts to open an image file and verify its integrity using img.verify(). If an image is not a valid file or is damaged, PIL raises an UnidentifiedImageError or OSError, which is caught by the method. In such cases, is_valid_image returns False, indicating that the image should be excluded from the dataset. This method is particularly useful in the dataset initialization process (**init**), where it ensures that only valid images are added to self.valid_samples.

The next important function to go over is the jax_dataloader function. The purpose of this function is to create mini-batch data loader for the IAMDataset, ensuring efficient training and evaluation in a JAX-based machine learning pipeline.

```python
from myProject.model1 import jax_dataloader
```

The function first generates a list of indices representing the dataset samples. If shuffle is set to True, these indices are randomly shuffled to ensure different batch compositions across epochs, improving the generalization of the model. The dataset is then processed in batches of batch_size (default: 32), iterating through the shuffled (or sequential) indices.

For each batch, the function retrieves the corresponding dataset samples using dataset[i], filtering out any None values to handle potential errors from **getitem**. It then stacks the images into a single JAX array using jnp.stack(images), ensuring that all images in the batch have the same shape.

Since text labels vary in length, the function determines the longest label in the batch and applies padding to all shorter labels using jnp.pad(). Padding ensures that all label sequences have the same length, with zero (0) used as the padding value to represent blank tokens. The function finally yields three outputs per batch: the stacked images, the padded labels, and an array indicating the original lengths of each label before padding.

This data loader is essential for training neural networks efficiently in JAX, ensuring that input batches are dynamically created while handling variable-length text sequences. It also optimizes the learning process by shuffling data and properly structuring inputs for the model.

Now that we have discussed how the data handling works, we can now talk about the model. The model we chose to use was the LSTM (Long Short-Term Memory) and a CNN (Convolution Nerual Network). LSTM is a type of recurrent neural network that is designed to process sequential data. This is important to us because images with words on them are just sequences of characters. Unlike traditional RNNs, which suffer from the vanishing gradient problem, LSTMs use gates (input, forget, and output) to selectively retain and discard information, allowing them to remember important patterns in handwriting across longer sequences. Here is the code for our model. Since handwriting varies over time, a simple CNN cannot fully recognize sequential patterns. That's why we use LSTMs to model temporal dependencies.

```
from myProject.model1 import CRNN
```

The setup method in this CRNN (Convolutional Recurrent Neural Network) model initializes the layers used for handwritten text recognition, combining convolutional layers for feature extraction with bidirectional LSTMs for sequence modeling.

The first part of the method defines five convolutional layers with increasing numbers of filters. These layers extract spatial features from the input images, such as stroke patterns and character shapes. Each convolutional layer uses a 3×3 kernel with a stride of 1 and 'SAME' padding, ensuring that the spatial dimensions remain consistent throughout the network. The number of filters starts at 64 and gradually increases to 512, allowing the network to learn progressively complex patterns in handwriting.

After the convolutional layers, a fully connected (dense) layer is defined. This layer maps the extracted high-level features into a set of character predictions, with the number of output neurons corresponding to the total number of possible characters in the dataset (including a blank token for CTC loss). This step is crucial for translating image features into meaningful text.

To process the extracted features sequentially, the method initializes bidirectional LSTMs. LSTMs are particularly well-suited for handwritten text recognition because they can capture long-term dependencies between characters. Instead of processing each character independently, LSTMs recognize how letters flow together in a sequence. Using bidirectional LSTMs, which process the text from both left-to-right and right-to-left, helps improve accuracy, as context is essential for predicting characters correctly—especially when dealing with messy or ambiguous handwriting.

By combining CNNs and LSTMs, this model effectively learns both spatial and sequential patterns in handwriting. The CNN layers extract meaningful features from image inputs, while the LSTM layers ensure that character dependencies are considered, leading to a more accurate handwritten text recognition system.

The **call** method in this CRNN (Convolutional Recurrent Neural Network) model defines the forward pass, applying convolutional layers for feature extraction and bidirectional LSTMs for sequential character modeling. This method is decorated with @nn.compact, meaning all layers are instantiated dynamically within the method rather than in setup().

The first part of the method processes the input x through five convolutional layers, each followed by batch normalization, dropout, and max pooling (except the last two layers, which skip max pooling). ReLU activation is applied after each convolution, introducing non-linearity. Batch normalization stabilizes training by normalizing activations, and dropout (set to 30%)

helps prevent overfitting by randomly deactivating neurons during training. The max-pooling layers progressively reduce the spatial dimensions, ensuring that the model focuses on the most important features.

After feature extraction, the tensor x is reshaped from a grid-like structure into a sequence. Since LSTMs process sequential data, the image features need to be arranged in a way that the model can interpret them as a series of character representations rather than a static image.

The model then initializes bidirectional LSTMs, using initialize_carry() to set up their hidden states. These LSTMs process the sequence in both forward and backward directions, capturing dependencies between characters. The forward LSTM processes text left to right, while the backward LSTM processes it in reverse, helping the model understand context more effectively.

For each time step in the sequence, both forward and backward LSTMs process the input, and their outputs are concatenated. This ensures that each character representation benefits from information about surrounding characters. The output is then stacked into a tensor, which is passed through a fully connected layer (self.fc) to generate predictions for each time step.

Finally, the method applies log softmax activation to the output, converting the raw predictions into log-probabilities for each character class. This step is essential for CTC (Connectionist Temporal Classification) loss, which is commonly used for handwriting recognition since it allows the model to learn without requiring pre-segmented characters.

Overall, this forward pass leverages CNNs for spatial feature extraction, bidirectional LSTMs for sequence modeling, and a dense layer for character classification, making it highly effective for handwritten text recognition tasks.

Now, we will describe how our loss function works:

```
from myProject.model1 import loss_fn_with_batch_stats
```

The loss_fn_with_batch_stats function computes the loss for training the handwriting recognition model while also updating batch normalization statistics. It plays a crucial role in optimizing the model by applying CTC (Connectionist Temporal Classification) loss, which is well-suited for sequence-based tasks like handwritten text recognition.

The function begins by passing the input images through the model.apply() function, which runs the forward pass using the given model parameters (params) and current batch normalization statistics (batch_stats). Since the model uses dropout for regularization, it also requires a random number generator (rng) to control dropout behavior during training. The mutable=['batch_stats'] argument ensures that batch statistics are updated during training, which is essential for proper batch normalization behavior.

To handle the alignment issues that arise when dealing with variable-length sequences, the function creates padding masks:

```
logit_paddings: A zero matrix that acts as a placeholder for missing
logits in the output.
label_paddings: A mask where zero-valued entries in labels (padding
tokens) are marked as 1, indicating which parts of the sequence should
be ignored in loss computation.
```

The function then computes the CTC loss using optax.ctc_loss(), which measures how well the predicted sequences align with the ground truth text. This loss function is specifically designed for sequence-to-sequence tasks without explicit alignment (e.g., converting an image of text into character sequences without knowing exact character positions). Finally, the function returns the mean CTC loss and the updated batch normalization statistics (new_model_state['batch_stats']), ensuring that the model continues to improve its normalization parameters during training.

By incorporating both loss computation and batch statistics updates, this function helps train a robust handwriting recognition model, optimizing it for better accuracy in recognizing variable-length handwritten text sequences.

In order to train the model, we implemented the following code:

```
from myProject.model1 import train_step
```

The train_step function performs a single training iteration for the handwriting recognition model, optimizing parameters using JAX's Just-In-Time (@jax.jit) compilation for efficiency. It begins by computing the Connectionist Temporal Classification (CTC) loss, which aligns predicted sequences with ground truth labels while updating batch normalization statistics. Using jax.value_and_grad(), it calculates both the loss and gradients, ensuring that the model learns effectively. To prevent instability, it applies gradient clipping, restricting values between -1.0 and 1.0, which is particularly useful for RNN-based models like LSTMs. The function then updates model parameters using an Adam optimizer from Optax, applying computed weight adjustments while maintaining the optimizer's internal state. Finally, it returns the updated parameters, batch statistics, optimizer state, and the loss, ensuring that training progresses smoothly. By leveraging JIT compilation, efficient loss computation, and stability mechanisms, this function optimizes model training for large-scale handwritten text recognition tasks.

In order to evaluate the performance of the model, we calculate a metric called Character Error Rate(CER). CER measures how many character-level errors occur in the model’s predictions compared to the ground truth labels, providing insight into the model's accuracy. Here is our implementation:

```
from myProject.model1 import compute_cer
```

The function initializes counters for total characters and total errors, then iterates through predicted and target sequences, converting them from numerical indices to text using index_to_char. It updates the total character count and calculates the Levenshtein distance between predictions and ground truth using editdistance.eval(), which measures character-level differences. Finally, it computes CER as the ratio of errors to total characters, returning float("inf") if no characters are present. This metric provides a clear assessment of the model’s character-level accuracy in handwritten text recognition.

Since our model outputs a sequence of numbers representing mapped characters, we need a way to convert these predictions back into readable text. To do this, we implemented a greedy_decode function converts predicted character indices into a readable text string using greedy decoding, a simple and efficient method for CTC-based handwriting recognition. Here is our implementation:

```
from myProject.model1 import greedy_decode
```

The function processes each index in preds, applying two key CTC rules: (1) skipping blank tokens (0) and (2) collapsing consecutive repeated characters to avoid duplicates.

The function maintains a prev_char variable to track the last processed character, ensuring that repeated characters are only appended once. If a character index is not found in index_to_char, it defaults to "?". After processing, the function returns the decoded text as a cleaned string, removing any leading or trailing spaces. This method ensures accurate sequence decoding while handling character alignment errors common in handwriting recognition models trained with CTC loss.

Because we stored our predictions in an array, we need to convert them back into readable text using our character mapping (index_to_char). In order to do this, we implemented the following function:

```
from myProject.model1 import evaluate_model
```

The evaluate_model function assesses the performance of the handwriting recognition model by computing the average loss and Character Error Rate (CER) over the given dataset. It processes the dataset in mini-batches, evaluates the model's predictions, and compares them to the ground truth labels.

The function starts by initializing counters for total loss and batch count, as well as lists to store all predicted and actual character sequences. It then iterates over batches of images and labels using jax_dataloader, ensuring shuffling is disabled to maintain consistency in evaluation. For each batch, it calculates the CTC loss using loss_fn_with_batch_stats, accumulating it for later averaging. The model's predictions are obtained by applying a forward pass and selecting the most likely character at each time step using argmax.

After collecting all predictions and targets, the function decodes and prints a few sample predictions using greedy_decode, comparing them against their respective ground truth labels. This provides insight into how well the model is recognizing handwritten text. Finally, it computes the CER (Character Error Rate) using compute_cer, which quantifies the proportion of incorrect characters in the predictions. The function returns the average loss and CER, serving as key metrics to evaluate the model's accuracy and efficiency in recognizing handwritten text.

Finally, we created a function to train our model:

```
from myProject.model1 import train_model
```

The train_model function trains the handwriting recognition model for a specified number of epochs while evaluating its performance after each epoch.

It starts by defining global variables (params, batch_stats, opt_state) to ensure updates persist across function calls. The function then iterates through the specified number of epochs, tracking the training loss and batch count for averaging. Within each epoch, it processes mini-batches from train_dataset using jax_dataloader, calling train_step to update model parameters and compute the loss. The total loss is accumulated and averaged over the batch count to obtain the epoch's average training loss.

After completing one epoch of training, the model is evaluated on the validation dataset using evaluate_model, which computes the validation loss and Character Error Rate (CER). Finally, it prints a summary of the epoch's training loss, validation loss, and CER, providing key insights into the model's progress and performance. This function ensures structured model training, evaluation, and performance monitoring over multiple epochs.

Here is our training configurations:

```
# Define the directory paths for images and labels
img_dir =
"/content/handwriting_dataset_1/handwriting_dataset_1.zip/iam_words/
words/"
label_file =
"/content/handwriting_dataset_1/handwriting_dataset_1.zip/iam_words/
words.txt"

# Initialize the dataset
dataset = IAMDataset(img_dir, label_file)

# Determine the split index for an 80-20 train-validation split
split_idx = int(0.8 * len(dataset))

# Create separate train and validation datasets
train_dataset = IAMDataset(img_dir, label_file)
val_dataset = IAMDataset(img_dir, label_file)

# Assign data samples: first 80% for training, last 20% for validation
train_dataset.valid_samples = dataset.valid_samples[:split_idx]
val_dataset.valid_samples = dataset.valid_samples[split_idx:]

# Define the number of classes
num_classes = len(CHARS) + 1

# Initialize the CRNN model with given parameters
model = CRNN(img_height=32, num_classes=num_classes,
lstm_hidden_size=512)

# Create a JAX PRNG key for random number generation
rng = jax.random.PRNGKey(0)

# Generate a dummy input tensor to initialize the model
dummy_input = jnp.ones((1, 32, 128, 1))
print("Dummy input shape:", dummy_input.shape)  # Verify input shape

# Initialize model parameters and batch normalization statistics
variables = model.init({'params': rng, 'dropout':
jax.random.PRNGKey(1)}, dummy_input, train=True)
params = variables['params']  # Extract model parameters
batch_stats = variables['batch_stats']  # Extract batch normalization
statistics
```

```python
# Define an exponential learning rate decay schedule
schedule = optax.exponential_decay(
    init_value=5e-4,
    transition_steps=500,
    decay_rate=0.95
)

# Initialize Adam optimizer with the learning rate schedule
optimizer = optax.adam(schedule)

# Initialize optimizer state using model parameters
opt_state = optimizer.init(params)
```

First, it specifies the dataset paths, pointing to the directory containing images (img_dir) and the label file (label_file). The IAMDataset class is instantiated to load and preprocess the dataset, and an 80-20 split is applied to separate training and validation datasets.

Next, the model configuration is defined. The number of character classes (num_classes) is set based on the predefined character set (CHARS). The CRNN model is instantiated with a height of 32 pixels and an LSTM hidden size of 512, ensuring a balance between model complexity and efficiency. A random number generator (rng) is initialized for JAX operations, and a dummy input tensor is created to verify the expected input shape. The model is then initialized, extracting key parameters (params) and batch normalization statistics (batch_stats).

Finally, the learning rate schedule is defined using exponential decay, starting with a relatively small initial value (5e-4) and gradually decreasing every 500 steps to maintain stability. The Adam optimizer from Optax is set up with this learning rate schedule, and the optimizer state (opt_state) is initialized. This configuration ensures a structured and efficient training process, preparing the model for optimization and evaluation.

Now, we trained the model for 50 epochs and monitored the progress. After the model was trained, we created a dictionary to save the final state of the model and we exported the model as a checkpoint.

```python
train_model(train_dataset, val_dataset, epochs=50)

# Create a dictionary containing the final state of the model.
trained_state = {
    'params': params,
    'batch_stats': batch_stats
}

# Define a directory and file path for the checkpoint.
checkpoint_dir = "/content/crnn_checkpoint"
os.makedirs(checkpoint_dir, exist_ok=True)
checkpoint_path = os.path.join(checkpoint_dir, "crnn_model")

# Initialize the Orbax PyTreeCheckpointer and save the checkpoint.
checkpointer = orbax.checkpoint.PyTreeCheckpointer()
checkpointer.save(checkpoint_path, trained_state)
```

```python
# Confirm that it is actually saved
print(f"Trained model saved to {checkpoint_path}")
```

```
Dummy input shape: (1, 32, 128, 1)
Decoded Target: "
Decoded Prediction: .
========================================
Decoded Target: my
Decoded Prediction: .
========================================
Decoded Target: background
Decoded Prediction: .
========================================
Decoded Target: is
Decoded Prediction: .
========================================
Decoded Target: a
Decoded Prediction: .
========================================
Decoded Target: doctor
Decoded Prediction: .
========================================
Decoded Target: of
Decoded Prediction: .
========================================
Decoded Target: 68
Decoded Prediction: .
========================================
Decoded Target: ,
Decoded Prediction: .
========================================
Decoded Target: who
Decoded Prediction: .
========================================
Epoch [1/50], Loss: 72.4179, Val Loss: 12.6410, CER: 1.1052
Decoded Target: "
Decoded Prediction:
========================================
Decoded Target: my
Decoded Prediction:
========================================
Decoded Target: background
Decoded Prediction:
========================================
Decoded Target: is
Decoded Prediction:
========================================
Decoded Target: a
Decoded Prediction:
```

```
========================================
Decoded Target: doctor
Decoded Prediction:
========================================
Decoded Target: of
Decoded Prediction:
========================================
Decoded Target: 68
Decoded Prediction:
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction:
========================================
Epoch [2/50], Loss: 69.9969, Val Loss: 11.1862, CER: 1.0000
Decoded Target: "
Decoded Prediction:
========================================
Decoded Target: my
Decoded Prediction:
========================================
Decoded Target: background
Decoded Prediction:
========================================
Decoded Target: is
Decoded Prediction:
========================================
Decoded Target: a
Decoded Prediction:
========================================
Decoded Target: doctor
Decoded Prediction:
========================================
Decoded Target: of
Decoded Prediction:
========================================
Decoded Target: 68
Decoded Prediction:
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction:
========================================
Epoch [3/50], Loss: 68.6648, Val Loss: 10.1007, CER: 0.9992
Decoded Target: "
Decoded Prediction: .
========================================
Decoded Target: my
Decoded Prediction:
========================================
Decoded Target: background
Decoded Prediction:
========================================
Decoded Target: is
Decoded Prediction:
========================================
Decoded Target: a
Decoded Prediction:
========================================
Decoded Target: doctor
Decoded Prediction:
========================================
Decoded Target: of
Decoded Prediction:
========================================
Decoded Target: 68
Decoded Prediction:
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction:
========================================
Epoch [4/50], Loss: 67.4182, Val Loss: 9.3368, CER: 0.9855
Decoded Target: "
Decoded Prediction: .
========================================
Decoded Target: my
Decoded Prediction:
========================================
Decoded Target: background
Decoded Prediction: ep
========================================
Decoded Target: is
Decoded Prediction:
========================================
Decoded Target: a
Decoded Prediction:
========================================
Decoded Target: doctor
Decoded Prediction:
========================================
Decoded Target: of
Decoded Prediction: o
========================================
Decoded Target: 68
Decoded Prediction: 
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: .
========================================
Epoch [5/50], Loss: 66.3580, Val Loss: 9.0563, CER: 0.9488
Decoded Target: "
Decoded Prediction: .
========================================
Decoded Target: my
Decoded Prediction: 
========================================
Decoded Target: background
Decoded Prediction: om
========================================
Decoded Target: is
Decoded Prediction: 
========================================
Decoded Target: a
Decoded Prediction: 
========================================
Decoded Target: doctor
Decoded Prediction: 
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: .
========================================
Epoch [6/50], Loss: 65.2357, Val Loss: 8.9982, CER: 0.8917
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: 
========================================
Decoded Target: background
Decoded Prediction: geopment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tn
========================================
Decoded Target: of
Decoded Prediction: o
========================================
Decoded Target: 68
Decoded Prediction: b
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [7/50], Loss: 64.2414, Val Loss: 8.7661, CER: 0.7787
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: a
========================================
Decoded Target: background
Decoded Prediction: nonpnment
========================================
Decoded Target: is
Decoded Prediction: is
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: nan
========================================
Decoded Target: of
Decoded Prediction: o
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: 
========================================
Epoch [8/50], Loss: 63.1684, Val Loss: 8.4051, CER: 0.7138
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: ponvmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tasn
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: 
========================================
Epoch [9/50], Loss: 62.1816, Val Loss: 8.4177, CER: 0.7317
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: a
========================================
Decoded Target: background
Decoded Prediction: popement
========================================
Decoded Target: is
Decoded Prediction: i
```

```
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: nateo
========================================
Decoded Target: of
Decoded Prediction: ,
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: 
========================================
Epoch [10/50], Loss: 61.2144, Val Loss: 8.1258, CER: 0.6970
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: 
========================================
Decoded Target: background
Decoded Prediction: nonens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tosn
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: "
```

```
========================================
Epoch [11/50], Loss: 60.4291, Val Loss: 8.1764, CER: 0.7222
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: a
========================================
Decoded Target: background
Decoded Prediction: gonvnmene
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: nosen
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: o
========================================
Epoch [12/50], Loss: 59.7968, Val Loss: 8.3250, CER: 0.6838
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: genpsene
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: taten
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [13/50], Loss: 59.3131, Val Loss: 8.3827, CER: 0.6438
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: ponpmen
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: taen
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: 
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [14/50], Loss: 58.8172, Val Loss: 8.5367, CER: 0.7037
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: ianpment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toeo
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: 
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: o
========================================
Epoch [15/50], Loss: 58.5011, Val Loss: 8.5610, CER: 0.6939
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: genvens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: doser
========================================
Decoded Target: of
Decoded Prediction: o
========================================
Decoded Target: 68
Decoded Prediction: b
========================================
Decoded Target: ,
Decoded Prediction: "
========================================
Decoded Target: who
Decoded Prediction: 
========================================
Epoch [16/50], Loss: 58.2651, Val Loss: 8.7925, CER: 0.6572
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: ponpnment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: touon
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: m
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: 
========================================
Epoch [17/50], Loss: 58.1101, Val Loss: 8.8149, CER: 0.6508
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: ganvrnmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tases
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: m
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: 
========================================
Epoch [18/50], Loss: 58.0194, Val Loss: 8.9014, CER: 0.6956
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonpenmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tater
========================================
Decoded Target: of
Decoded Prediction: 
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [19/50], Loss: 57.8849, Val Loss: 9.2027, CER: 0.6511
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: ganpment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tasers
========================================
Decoded Target: of
Decoded Prediction:
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [20/50], Loss: 58.0812, Val Loss: 9.2066, CER: 0.6494
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonpment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
```

```
========================================
Decoded Target: doctor
Decoded Prediction: tatos
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction:
========================================
Epoch [21/50], Loss: 57.7730, Val Loss: 9.3553, CER: 0.6290
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonvnment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: totos
========================================
Decoded Target: of
Decoded Prediction:
========================================
Decoded Target: 68
Decoded Prediction: b
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [22/50], Loss: 57.6705, Val Loss: 9.5896, CER: 0.6430
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: to
========================================
Decoded Target: background
Decoded Prediction: genpmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tosos
========================================
Decoded Target: of
Decoded Prediction:
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [23/50], Loss: 57.6267, Val Loss: 9.4236, CER: 0.6752
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: a
========================================
Decoded Target: background
Decoded Prediction: gonvenment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tateo
========================================
Decoded Target: of
Decoded Prediction:
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction:
========================================
Epoch [24/50], Loss: 57.6500, Val Loss: 9.4550, CER: 0.6626
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: to
========================================
Decoded Target: background
Decoded Prediction: ganvrnment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toteo
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction:
========================================
Epoch [25/50], Loss: 57.7179, Val Loss: 9.5433, CER: 0.6259
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: to
========================================
Decoded Target: background
Decoded Prediction: ganpnment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toleon
========================================
Decoded Target: of
Decoded Prediction: af
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction:
========================================
Epoch [26/50], Loss: 57.5912, Val Loss: 9.5341, CER: 0.6214
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: ganpment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toles
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
==========================================
Decoded Target: who
Decoded Prediction: "
==========================================
Epoch [27/50], Loss: 57.5631, Val Loss: 9.6475, CER: 0.6570
Decoded Target: "
Decoded Prediction: "
==========================================
Decoded Target: my
Decoded Prediction: to
==========================================
Decoded Target: background
Decoded Prediction: gonpnment
==========================================
Decoded Target: is
Decoded Prediction: i
==========================================
Decoded Target: a
Decoded Prediction: a
==========================================
Decoded Target: doctor
Decoded Prediction: toeo
==========================================
Decoded Target: of
Decoded Prediction: 
==========================================
Decoded Target: 68
Decoded Prediction: a
==========================================
Decoded Target: ,
Decoded Prediction: 
==========================================
Decoded Target: who
Decoded Prediction: "
==========================================
Epoch [28/50], Loss: 57.5217, Val Loss: 9.7135, CER: 0.6589
Decoded Target: "
Decoded Prediction: "
==========================================
Decoded Target: my
Decoded Prediction: t
==========================================
Decoded Target: background
Decoded Prediction: gonvmens
==========================================
Decoded Target: is
Decoded Prediction: i
```

```
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toters
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [29/50], Loss: 57.5284, Val Loss: 9.8206, CER: 0.6379
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: genvnment
========================================
Decoded Target: is
Decoded Prediction: is
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: later
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
```

```
========================================
Epoch [30/50], Loss: 57.4780, Val Loss: 10.0468, CER: 0.6433
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonpnmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toteos
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [31/50], Loss: 57.4640, Val Loss: 10.1125, CER: 0.6307
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: to
========================================
Decoded Target: background
Decoded Prediction: gonpnmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toen
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [32/50], Loss: 57.4984, Val Loss: 10.2704, CER: 0.6533
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonvment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tolon
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: b
========================================
Epoch [33/50], Loss: 334.7333, Val Loss: 10.2477, CER: 0.6130
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: to
========================================
Decoded Target: background
Decoded Prediction: gonvmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tolos
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [34/50], Loss: 57.4903, Val Loss: 10.2652, CER: 0.6304
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: ganvnment
========================================
Decoded Target: is
Decoded Prediction: is
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toten
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: t
========================================
Epoch [35/50], Loss: 57.5084, Val Loss: 10.3233, CER: 0.6253
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonvemens
========================================
Decoded Target: is
Decoded Prediction: is
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toson
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: t
========================================
Epoch [36/50], Loss: 57.4718, Val Loss: 10.3024, CER: 0.6198
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonvmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: totes
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: b
========================================
Epoch [37/50], Loss: 57.5108, Val Loss: 10.3709, CER: 0.6391
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonvement
========================================
Decoded Target: is
Decoded Prediction: is
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toses
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [38/50], Loss: 57.4705, Val Loss: 10.5518, CER: 0.6514
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: a
========================================
Decoded Target: background
Decoded Prediction: gonvmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toses
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [39/50], Loss: 57.4405, Val Loss: 10.5348, CER: 0.6654
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: 1o
========================================
Decoded Target: background
Decoded Prediction: ionpmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
```

```
========================================
Decoded Target: doctor
Decoded Prediction: touen
========================================
Decoded Target: of
Decoded Prediction: at
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: "
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [40/50], Loss: 57.4455, Val Loss: 10.5926, CER: 0.6323
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: a
========================================
Decoded Target: background
Decoded Prediction: gonpment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: jolen
========================================
Decoded Target: of
Decoded Prediction: at
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [41/50], Loss: 57.4206, Val Loss: 10.5359, CER: 0.6142
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: a
========================================
Decoded Target: background
Decoded Prediction: gonvomens
========================================
Decoded Target: is
Decoded Prediction: is
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: toteo
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: t
========================================
Epoch [42/50], Loss: 57.4477, Val Loss: 10.9607, CER: 0.6262
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: 1
========================================
Decoded Target: background
Decoded Prediction: gonpment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: joleos
========================================
Decoded Target: of
Decoded Prediction: at
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [43/50], Loss: 57.4169, Val Loss: 10.7501, CER: 0.6077
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: m
========================================
Decoded Target: background
Decoded Prediction: gonpment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: joteos
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "o
========================================
Epoch [44/50], Loss: 57.4202, Val Loss: 11.0116, CER: 0.6365
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonprment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: joweos
========================================
Decoded Target: of
Decoded Prediction: at
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [45/50], Loss: 57.4164, Val Loss: 10.7709, CER: 0.6346
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonvnment
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: jaseos
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: -o
========================================
Epoch [46/50], Loss: 57.4587, Val Loss: 10.8878, CER: 0.6419
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: 1
========================================
Decoded Target: background
Decoded Prediction: gonvement
========================================
Decoded Target: is
Decoded Prediction: is
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: joses
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: b
========================================
Decoded Target: ,
Decoded Prediction: 
========================================
Decoded Target: who
Decoded Prediction: t
========================================
Epoch [47/50], Loss: 57.4390, Val Loss: 11.1575, CER: 0.6063
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: 1
========================================
Decoded Target: background
Decoded Prediction: ganvrmens
========================================
Decoded Target: is
Decoded Prediction: i
```

```
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: jaseon
========================================
Decoded Target: of
Decoded Prediction: a
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [48/50], Loss: 57.4483, Val Loss: 11.3931, CER: 0.6564
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonvrnmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: joseor
========================================
Decoded Target: of
Decoded Prediction: at
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
```

```
========================================
Epoch [49/50], Loss: 57.4547, Val Loss: 11.4287, CER: 0.6220
Decoded Target: "
Decoded Prediction: "
========================================
Decoded Target: my
Decoded Prediction: t
========================================
Decoded Target: background
Decoded Prediction: gonvnmens
========================================
Decoded Target: is
Decoded Prediction: i
========================================
Decoded Target: a
Decoded Prediction: a
========================================
Decoded Target: doctor
Decoded Prediction: tasor
========================================
Decoded Target: of
Decoded Prediction: at
========================================
Decoded Target: 68
Decoded Prediction: a
========================================
Decoded Target: ,
Decoded Prediction:
========================================
Decoded Target: who
Decoded Prediction: "
========================================
Epoch [50/50], Loss: 57.4145, Val Loss: 11.1003, CER: 0.6102
Trained model saved to /content/crnn_checkpoint/crnn_model
```

Critical Analysis of Model 1

The model has been trained for 50 epochs, showing steady learning progress with a final validation loss of 11.1003 and a Character Error Rate (CER) of 0.6102. The CER steadily decreased over training, indicating that the model was learning meaningful patterns. The transition from CER = 0.6626 to 0.6102 suggests that the model successfully reduces errors, though a lower CER would be preferable. Also, from the decoded prediction being printed, we can see a lot of successful prediction, especially for words with fewer letters. Despite initial misclassifications, the model learns to filter out extraneous information and improves prediction accuracy. The exponential learning rate decay strategy ensures that the training remains stable while gradually fine-tuning the model.

However, there are still many clear drawbacks of the model. Some predictions exhibit frequent misclassifications of characters, such as: "background" being predicted as "ganvrnment”, “doctor" being predicted as "toteo". The model occasionally omits characters, which suggests that it struggles with character spacing or specific handwriting styles. More importantly, the model is currently not user interactive. One of our key goals of this project is creating an user interactive dash app which the user can input an handwritten image and the correct output will be spit out by the website. This lead us to building model 2.

Building Model 2

Train the model

Now we will see how to train the model that recognizes the word in images.

First, we will import needed functions from py file.

```
'''
import os
from keras import utils
import tensorflow_datasets as tfds
import keras
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.image as mpimg
'''

import myProject # Import the package that contains my py files for
functions and classes
from myProject import model2
```

Strategy for Model

Our strategy is to load dataset we got (IAM_words), which is made of images of the words and the labels of them. Then, we will resize the images, divide them into multiple batches, and train the model.

Take a look at dataset

Let's print few examples of dataset, images and the labels of them.

```
# Example usage
from myProject.model2 import parse_labels
label_file = '/content/word_dataset_new_mar20/words.txt'
image_paths, labels = parse_labels(label_file)

from myProject.dataset import IAMDataset
# Let's print a few examples
for path, label in zip(image_paths[:5], labels[:5]):
    print(f"{path} => {label}")
    img = mpimg.imread(path)
    i = 1
    ax = plt.subplot(1,5,i)
    i += 1

    plt.imshow(img)
    plt.axis('off')
    plt.show()


# image_paths and labels are ready to be used in your training
pipeline


### **Explanation:**
"""
- **Reads each line** of your labels file and skips comments.
- Extracts the relevant information from each line.
- Constructs the correct image path based on your described directory
structure.
- Associates each path correctly with its label.

This resulting list of paths and labels can now easily feed into
TensorFlow or other ML pipelines.
"""
```

```
/content/word_dataset_new_mar20/words/e01/e01-014/e01-014-00-00.png =>
We
```

```
/content/word_dataset_new_mar20/words/e01/e01-014/e01-014-00-01.png =>
know
```

```
/content/word_dataset_new_mar20/words/e01/e01-014/e01-014-00-02.png =>
from
```

```
/content/word_dataset_new_mar20/words/e01/e01-014/e01-014-00-03.png =>
the
```

```
/content/word_dataset_new_mar20/words/e01/e01-014/e01-014-00-04.png =>
diaries
```

```
{"type":"string"}
```

Vectorization

We will develop a way to convert alphabetical words into numerical representation of them, for easier calculation. The words will be represented as vectors of numbers.

The char_to_num will convert characters into numbers, and num_to_char will do the inverse of it.

```
# Create a set of unique characters
characters = sorted(set(''.join(labels)))


# Proper definition: need to make 0 a padding character, not normal
character
char_to_num = tf.keras.layers.StringLookup(
    vocabulary=characters,
    num_oov_indices=1,
    mask_token=None,  # explicitly reserve 0 for padding/blank
)

num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(),
    invert=True,
    num_oov_indices=1,
    mask_token=None
)

# Ensure blank token is correctly mapped
blank_index = len(char_to_num.get_vocabulary())  # Last index is blank
print(f"Blank token index: {blank_index}")
```

```
Blank token index: 71
```

```
print(characters)
```

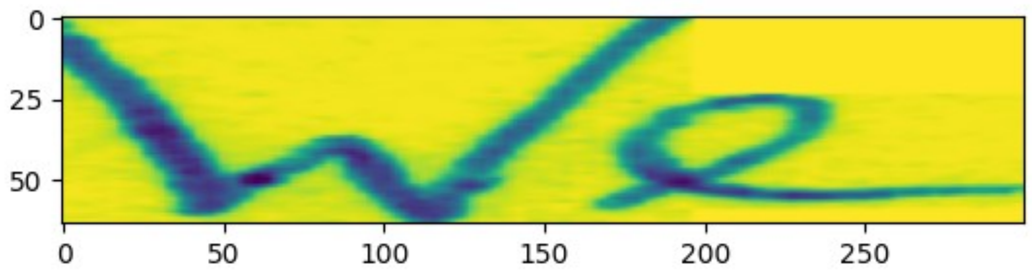
```
['!', '"', "'", '(', ')', ',', '-', '.', '/', '0', '1', '2', '3', '4',
'5', '6', '8', '9', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'R', 'S', 'T', 'U', 'V',
'W', 'Y', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

```
IMAGE_HEIGHT, IMAGE_WIDTH = 64, 300  # Unifying size of images - Why
64, 300?
# Average length of english word is 4.7 characters (Googled)
# Assuming a character is square, it means word width is 4.7 times
word height
# 64*4.7 = 300

# maybe split the image into single characters?
# ML models only accept unified dimensions
```

This is an example of how vectorization works, as you see word "We" is represented as tensor of 43, 49.

```
from myProject.model2 import encode_label
from myProject.model2 import load_and_preprocess_image

# Testing load_and_preprocess_image and encode_label functions
# Should print processed image and numerical tensor-representation of
word label
plt.imshow(load_and_preprocess_image(image_paths[0]))
print(encode_label(labels[0]), char_to_num)
```

```
tf.Tensor([43 49], shape=(2,), dtype=int64) <StringLookup
name=string_lookup_2, built=True>
```



Pre-processing the dataset

Then, we will prepare dataset by processing images, such as unifying the dimensions of images and converting them into grayscale. It also includes process of vectorizing the label words. Then, the dataset is prepared by using the preprocess_dataset, which does all the things mentioned above.

```
BATCH_SIZE = 32

dataset = tf.data.Dataset.from_tensor_slices((image_paths, labels))


from myProject.model2 import preprocess_dataset


# Apply padding during batching
dataset = dataset.map(preprocess_dataset,
num_parallel_calls=tf.data.AUTOTUNE)

dataset = dataset.padded_batch(
    BATCH_SIZE,
    padded_shapes=(
        {"image": [IMAGE_HEIGHT, IMAGE_WIDTH, 1]},
        [None]
    ),
    padding_values=(
        {"image": 0.0},
        np.int64(0)
    )
).prefetch(tf.data.AUTOTUNE)



# Get dataset size to confirm dataset is well established
size = dataset.cardinality().numpy()
print("Dataset size:", size)
```

```
Dataset size: 182
```

Build model

The most important part--Model--is built using CNN and RNN layers. CNN is used to extract the features of images, and RNN is used to decode the sequence of characters- since it's best at recognizing patterns of sequences. The RNN layers use LSTM, which is passed to dense output.

The goal of the model is to minimize the CTC loss, which is special loss function for OCR. The CTC returns probabilities of each characters in the image, which is used to "build" words off of predicted characters.

```python
from myProject.model2 import ctc_loss

input_img = tf.keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH,
1), name='image')

# CNN layers
x = tf.keras.layers.Conv2D(32, (3,3), activation='relu',
padding='same')(input_img)
x = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(x)

x = tf.keras.layers.Conv2D(64, (3,3), activation='relu',
padding='same')(x)
x = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(x)

# Prepare data for RNN
new_shape = (IMAGE_WIDTH // 4, (IMAGE_HEIGHT // 4) * 64)
x = tf.keras.layers.Reshape(target_shape=new_shape)(x)

# RNN layers
x = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,
return_sequences=True))(x)
x = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,
return_sequences=True))(x)

# Output layer
output = tf.keras.layers.Dense(len(characters) + 2,
activation='softmax')(x)

model = tf.keras.Model(inputs=input_img, outputs=output)

model.compile(optimizer='adam', loss=ctc_loss)
```

Training

We will train the model. It is found that around 40 epochs lead to best result, with CTC loss of around 2. 10 epoch is used in notebook for training demonstration.

```python
EPOCHS = 10

model.fit(dataset, epochs=EPOCHS)
```

```
Epoch 1/10

/usr/local/lib/python3.11/dist-packages/keras/src/models/
functional.py:237: UserWarning: The structure of `inputs` doesn't
match the expected structure.
Expected: image
```

```
Received: inputs=['Tensor(shape=(None, 64, 300, 1))']
  warnings.warn(msg)

182/182 ━━━━━━━━━━━━━━━━━━━━ 19s 63ms/step - loss: 45.3523
Epoch 2/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 11s 61ms/step - loss: 18.2344
Epoch 3/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 21s 62ms/step - loss: 17.7731
Epoch 4/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 11s 61ms/step - loss: 17.3574
Epoch 5/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 11s 62ms/step - loss: 17.0125
Epoch 6/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 20s 60ms/step - loss: 16.6352
Epoch 7/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 11s 62ms/step - loss: 16.2679
Epoch 8/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 11s 61ms/step - loss: 15.9318
Epoch 9/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 11s 62ms/step - loss: 15.6318
Epoch 10/10
182/182 ━━━━━━━━━━━━━━━━━━━━ 11s 62ms/step - loss: 15.3502

<keras.src.callbacks.history.History at 0x7955d2d4c510>
```

Using the model

Finally, we will try to use the model to recognize words in images.

The predict_text function will get the input image, and use the model to get vector result of predicted word. Then the decode_ctc will convert the numeric vector into word, using num_to_char function.

The result of predict_text is displayed below. It has problem of empty space displayed as UNK, but we will fiter those out in final dash webapp.

```python
from myProject.model2 import decode_ctc
from myProject.model2 import preprocess_single_image
from myProject.model2 import predict_text


# Example Usage:
image_path = '/content/word_dataset_new_mar20/words/e01/e01-014/e01-
014-00-00.png'
predicted_word = predict_text(model, image_path)
print(f"The image says: {predicted_word}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 399ms/step
Predicted indices: [[64 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71
71 71 71 71 71 71 71 71
  71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71
71
  71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71 71
71
  71 71 71]]
The image says: t[UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK]
[UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK]
[UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK]
[UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK]
[UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK]
[UNK][UNK][UNK][UNK][UNK][UNK][UNK][UNK]
```

Final Webapp

This is the final dash webapp using the second model, using CNN, RNN, and CTC Loss.

Warning: This webapp requires loading model with "ocr_model_8.keras" filename. This is the trained model.

The preprocess_image and decode_ctc functions are used to do similar things in the codes in notebook previously- process input image into processable form, and decode numeric representation of words into readable alphabetical words. Then, the predict_text1 function will predict the word using model and these functions.

First, the trained model is loaded-which is trained 40 epochs until there is no more loss decrease. Then there would be a button to upload an image file, which is put into input argument of handle_image, which does all the processing and predicting internal works.

The website is designed with the goal of giving Matrix computer console vibe, so that it feels futuristic and tech-y.

In [1]:
```
import dash
from dash import html, dcc, Output, Input
import base64
import tensorflow as tf
import numpy as np
import io
from PIL import Image
import socket

import myProject # Import the package that contains my py files for functior
from myProject import model2
from myProject.model2 import preprocess_image, predict_text1, find_free_port

# Load model
model = tf.keras.models.load_model("ocr_model_8.keras", compile=False)

# Image dimensions from training
IMAGE_HEIGHT, IMAGE_WIDTH = 64, 300

# Character decoding function
characters = ['!', '"', "'", '(', ')', ',', '-', '.', '/', '0', '1', '2', '3

char_to_num = tf.keras.layers.StringLookup(vocabulary=characters, num_oov_ir
num_to_char = tf.keras.layers.StringLookup(vocabulary=char_to_num.get_vocabu

# Build the Dash app
```

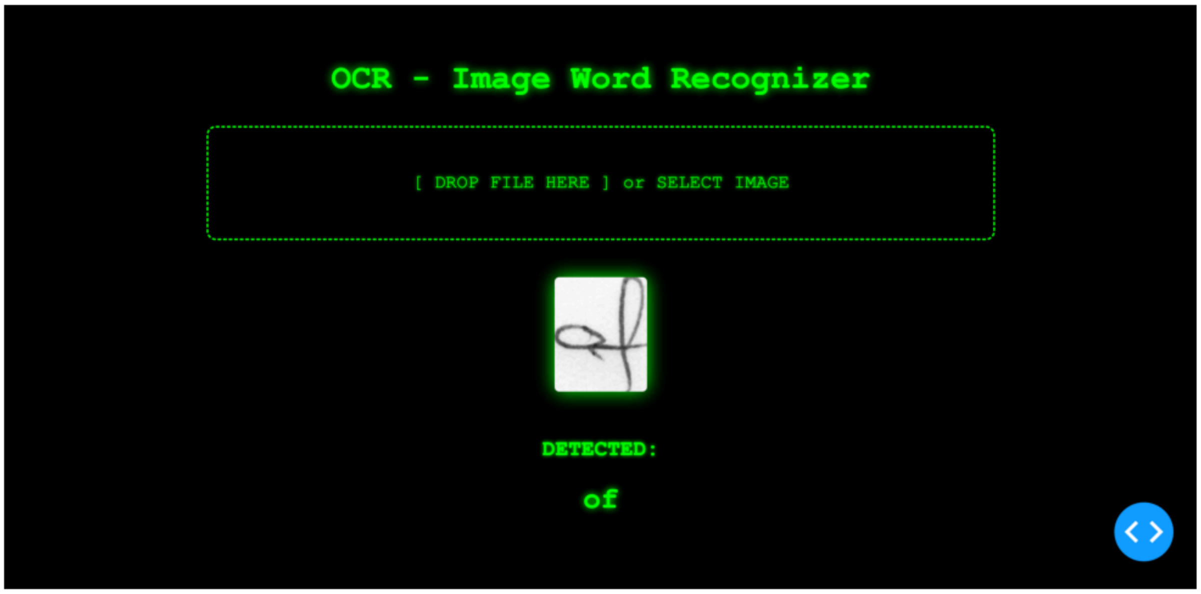
```python
app = dash.Dash(__name__)
app.title = "OCR Console"

app.layout = html.Div([
    html.H1("OCR - Image Word Recognizer", style={
        'textAlign': 'center',
        'color': '#00FF00',
        'fontFamily': 'Courier New',
        'fontSize': '36px',
        'marginBottom': '30px',
        'textShadow': '0 0 10px #00FF00'
    }),

    dcc.Upload(
        id='upload-image',
        children=html.Div(['[ DROP FILE HERE ] or ', html.A('SELECT IMAGE')]
        style={
            'width': '70%',
            'height': '120px',
            'lineHeight': '120px',
            'borderWidth': '2px',
            'borderStyle': 'dashed',
            'borderRadius': '10px',
            'textAlign': 'center',
            'margin': 'auto',
            'backgroundColor': '#000000',
            'color': '#00FF00',
            'fontSize': '20px',
            'fontFamily': 'Courier New',
            'transition': '0.3s ease',
            'textShadow': '0 0 5px #00FF00'
        },
        multiple=False
    ),

    html.Div(id='output', style={
        'textAlign': 'center',
        'marginTop': '40px',
        'color': '#00FF00',
        'fontFamily': 'Courier New, monospace',
        'fontSize': '20px'
    }),
], style={
    'backgroundColor': '#000000',
    'height': '100vh',
    'padding': '40px'
})

@app.callback(
    Output('output', 'children'),
    Input('upload-image', 'contents')
)

if __name__ == '__main__':
    port = find_free_port()
    print(f"Running on http://127.0.0.1:{port}")
    app.run_server(debug=True, port=port)
```

```
2025-03-21 05:18:44.147409: I tensorflow/core/platform/cpu_feature_guard.cc:
210] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
Running on http://127.0.0.1:61283
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 687ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
```

In [4]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```python
fig, axes = plt.subplots(3, 1, figsize=(15, 15))  # 3 rows, 1 column

img_paths = ['success_1.png', 'success_2.png', 'success_3.png']

for ax, path in zip(axes, img_paths):
    img = mpimg.imread(path)
    ax.imshow(img)
    ax.axis('off')

plt.tight_layout()
plt.show()
```

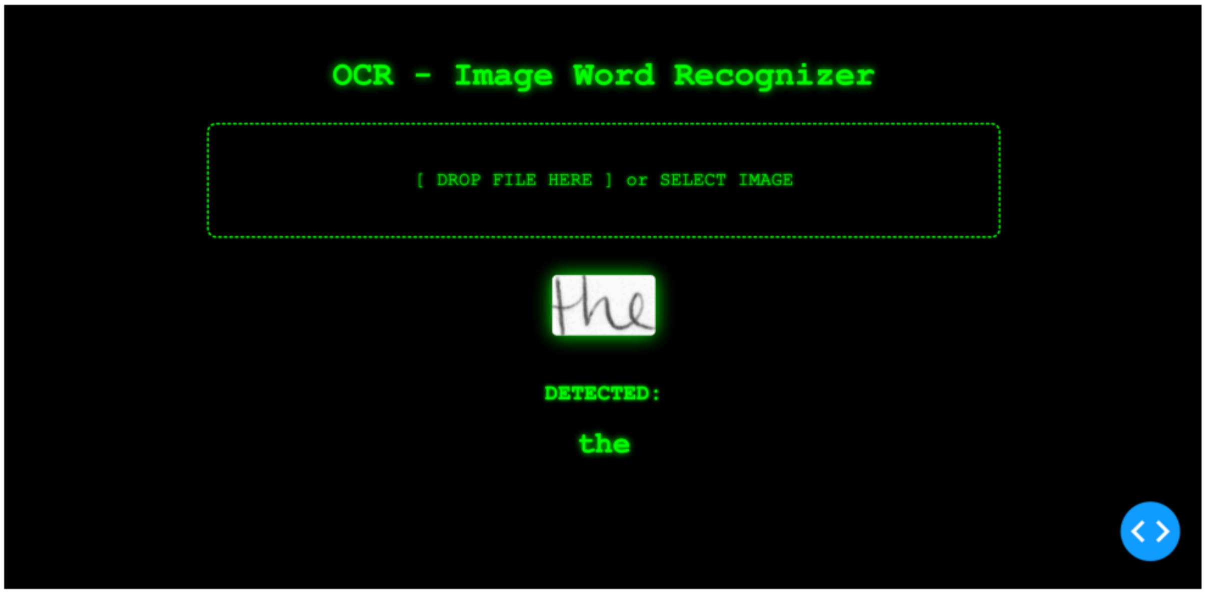


OCR - Image Word Recognizer

[DROP FILE HERE] or SELECT IMAGE

DETECTED:

of

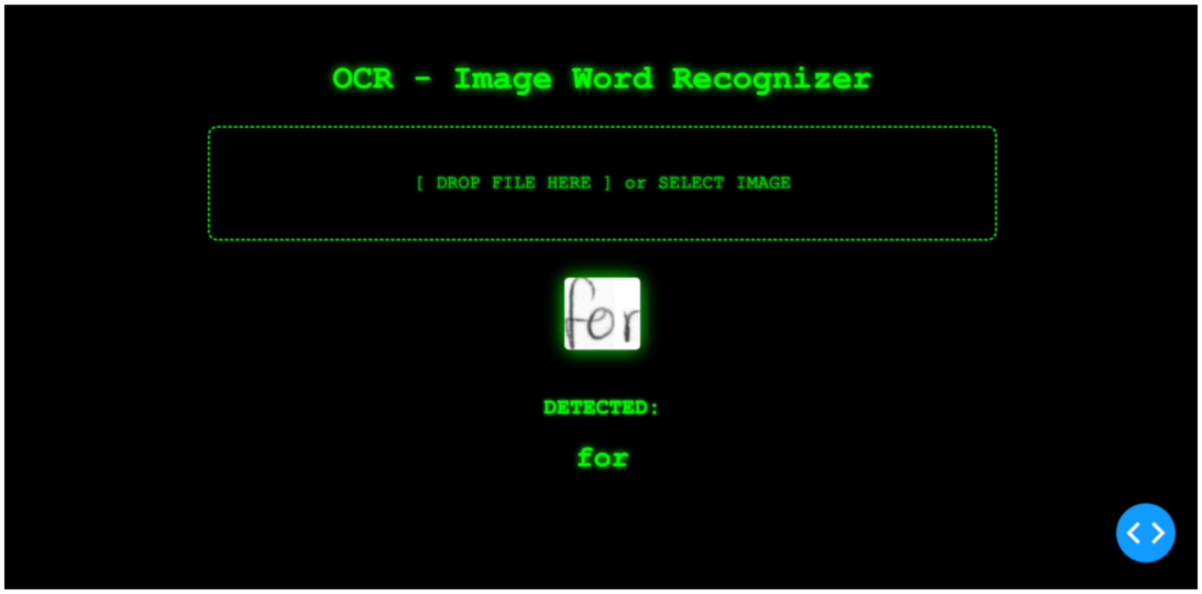


OCR - Image Word Recognizer

[DROP FILE HERE] or SELECT IMAGE

DETECTED:

the



OCR - Image Word Recognizer

[DROP FILE HERE] or SELECT IMAGE

DETECTED:

for

```
In [ ]:
```