HANDWRITING TEXT DETECTION

Using CNN, RNN, BiLSTM and CTC

Abstract

To build a deep learning model that recognizes handwritten names from grayscale images.

By:

Kirtan Shah

Objective:

To build a deep learning model that recognizes handwritten names from grayscale images.

Model Architecture:

Convolutional Neural Network (CNN)

CNN is a deep learning architecture particularly effective for image processing tasks. In this handwriting recognition model, the CNN component is designed to extract spatial features from grayscale input images. It comprises three convolutional blocks. Each block includes:

- Conv2D layer with increasing filter sizes (32, 64, and 128)
- BatchNormalization to stabilize training
- ReLU activation for non-linearity
- MaxPooling2D layers to reduce spatial dimensions
- Dropout layers to prevent overfitting in later blocks

These extracted spatial features are then reshaped and passed to the sequential layers for further processing.

1.Conv2D Layers

The CNN part of the model is structured as follows:

- Conv Block 1
    - Conv2D: 32 filters, 3×3 kernel, ReLU activation
    - BatchNormalization
    - MaxPooling2D: 2×2 pool size

- Conv Block 2
    - Conv2D: 64 filters, 3×3 kernel, ReLU activation
    - BatchNormalization
    - MaxPooling2D: 2×2 pool size
    - Dropout: 0.3

- Conv Block 3
    - Conv2D: 128 filters, 3×3 kernel, ReLU activation
    - BatchNormalization

    - MaxPooling2D: 1×2 pool size (to reduce width)
    - Dropout: 0.3

1.2 MaxPooling:

- Downsamples the feature maps to reduce spatial dimensions.
- Retains the most important features in each region (max value).
- Reduces computational load and helps in faster training.
- Provides spatial invariance, making the model robust to small translations of features.

1.3 BatchNormalization:

- Normalizes the activations after each convolutional layer.
- Improves training stability and speeds up convergence.
- Reduces internal covariate shift by standardizing the inputs to each layer.
- Helps in using higher learning rates and reduces overfitting.

1.4 ReLU (Rectified Linear Unit)

- ReLU is a widely used activation function in neural networks. It outputs the input directly if it's positive; otherwise, it returns zero.
- It is applied in the Conv2D layers of your CNN to introduce non-linearity into the model, which enables it to learn complex patterns in the data. Without ReLU or a similar activation function, the network would just be a series of linear transformations, limiting its ability to learn the complex features of handwritten text.
- It helps the model to recognize and learn non-linear relationships in the input data, which is crucial for accurately recognizing handwritten characters.

1.5 Dropout

- Dropout is a regularization technique where randomly selected neurons are ignored during training, helping to prevent overfitting.
- It is applied in the second and third convolutional blocks (with a dropout rate of 0.3) to reduce overfitting. It helps the model to generalize better, especially when the dataset is small or the model is too complex.
- It improves the model's ability to generalize, ensuring it performs well on new, unseen handwritten text, not just the training data.

2. Reshape Layer:

It reorganizes the 3D feature maps (height, width, channels) into a 2D sequence format, where each time step represents a portion of the input image. This allows sequential models, like LSTM, to process each step independently, maintaining temporal dependencies.

In this project, the Reshape Layer is used after the CNN layers to convert the extracted spatial features into a sequence format suitable for the Bidirectional LSTM to process.

3. (Recurrent Neural Network - RNN)

RNN stands for Recurrent Neural Network. It is a type of neural network specially designed to handle sequential data — data where the order matters, like text, speech, or handwriting.

Unlike normal neural networks, RNNs have a “memory” that allows them to use previous information to understand the current input.

3.1. Bidirectional LSTM (Long Short-Term Memory)

The Bidirectional LSTM is a type of recurrent neural network that reads input sequences in both forward and backward directions. This allows the model to understand context from both past and future time steps, improving accuracy in sequence prediction.

In this project, it is used after the Reshape Layer to process the sequential data generated from the CNN features, helping the model understand the full context of handwritten text from left to right and right to left.

4. CTC (Connectionist Temporal Classification) Loss

CTC (Connectionist Temporal Classification) is a loss function used for tasks where the input and output sequences are not aligned. It’s useful in handwriting recognition because the positions of characters in an image aren’t fixed. CTC allows the model to predict a sequence (e.g., text) from an image, even when the characters are irregularly spaced or connected.

In this project, CTC loss is used to train the model to predict a sequence of characters from handwritten images, without requiring the characters to be precisely aligned in the image. This is crucial because:

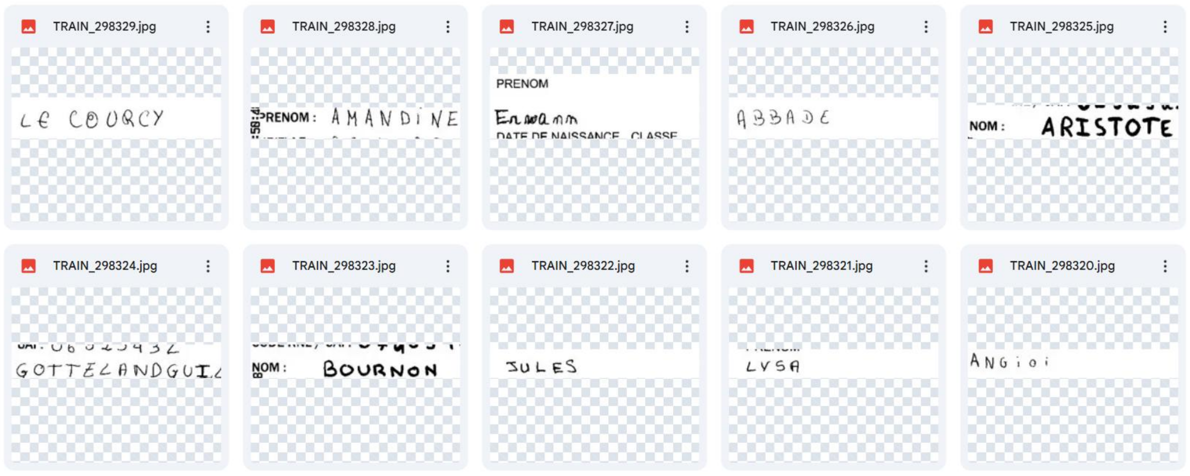
- Handwritten images have varying lengths, and characters can overlap or be spaced irregularly.
- CTC enables training without needing to specify the exact position of each character in the image, making it ideal for this kind of unsegmented sequence prediction.

2) Dataset:

Source: https://www.kaggle.com/datasets/landlord/handwriting-recognition

Files Used:

- written_name_train_v2.csv
- written_name_validation_v2.csv
- Image directories: train_v2/train/, validation_v2/validation/, etc.



3) Data Preprocessing:

- Dropped some data to speed up training (used 21,000 training + 2100 validation samples).
- Removed rows with missing labels or labeled as "UNREADABLE".
- Converted all labels to uppercase for consistency.
- Images were resized to 256×64, padded and rotated for uniformity.
- Normalized pixel values to range [0, 1].

4) Character Encoding:

- Alphabets Used: A-Z, ', -, space
- Max String Length: 24 characters
- Label Processing:
  - Labels padded with -1 for CTC.
  - Input and label lengths tracked separately.

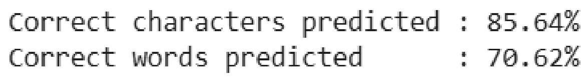
5) Training Configuration:

- Epochs: 60
- Batch Size: 128
- Optimizer: Adam

- Learning Rate: 0.0001
- Loss Function: CTC Loss

6) Evaluation Metrics:

- Character-Level Accuracy: Measures % of correctly predicted characters.
- Word-Level Accuracy: Measures exact match rate of predicted vs. actual words.



```
Correct characters predicted : 85.64%
Correct words predicted      : 70.62%
```

7) Code Explanation

1. Kaggle Dataset Setup

```
!mkdir -p ~/.kaggle

!cp /content/kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json


!kaggle datasets download -d landlord/handwriting-recognition

!unzip handwriting-recognition.zip -d handwriting_data
```

- **Creates Kaggle credentials folder** and copies the kaggle.json file for authentication.
- **Downloads the handwriting recognition dataset** from Kaggle and unzips it into the handwriting_data directory.

2. Importing Libraries

```
import os

import cv2

import random

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow.keras import backend as K

from tensorflow.keras.models import Model

from tensorflow.keras.layers import (

    Input, Conv2D, MaxPooling2D, Reshape, Bidirectional,

    LSTM, Dense, Lambda, Activation, BatchNormalization, Dropout

)

from tensorflow.keras.optimizers import Adam
```

- Loads necessary libraries: image processing (cv2), data handling (pandas, numpy), visualization (matplotlib), and deep learning (TensorFlow/Keras).

3. Data Loading and Visualization

```
train = pd.read_csv('/content/handwriting_data/written_name_train_v2.csv')

valid = pd.read_csv('/content/handwriting_data/written_name_validation_v2.csv')

train

valid

train.info()

valid.info()
```

- **Reads training and validation metadata** (CSV files contain image file names and corresponding labels).

Image Visualization

```
plt.figure(figsize=(15, 10))

for i in range(6):

    ax = plt.subplot(2, 3, i+1)

    img_dir = '/content/handwriting_data/train_v2/train/'+train.loc[i, 'FILENAME']

    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)

    plt.imshow(image, cmap = 'gray')

    plt.title(train.loc[i, 'IDENTITY'], fontsize=12)
    plt.axis('off')
plt.subplots_adjust(wspace=0.2, hspace=-0.8)
```

- **Displays 6 sample handwriting images** with their corresponding labels (IDENTITY).

4. Data Cleaning and Preprocessing

Missing and Unreadable Labels

```
print("Number of NaNs in train set      : ", train['IDENTITY'].isnull().sum())
print("Number of NaNs in validation set : ", valid['IDENTITY'].isnull().sum())
train.dropna(axis=0, inplace=True)
valid.dropna(axis=0, inplace=True)
unreadable = train[train['IDENTITY'] == 'UNREADABLE']
unreadable.reset_index(inplace = True, drop=True)
plt.figure(figsize=(15, 10))
for i in range(6):
    ax = plt.subplot(2, 3, i+1)
    img_dir = '/content/handwriting_data/train_v2/train/'+unreadable.loc[i, 'FILENAME']
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    plt.imshow(image, cmap = 'gray')
    plt.title(unreadable.loc[i, 'IDENTITY'], fontsize=12)
    plt.axis('off')
plt.subplots_adjust(wspace=0.2, hspace=-0.8)
train = train[train['IDENTITY'] != 'UNREADABLE']
valid = valid[valid['IDENTITY'] != 'UNREADABLE']
```

- **Removes rows** with missing or "UNREADABLE" labels — such data is not useful for training a supervised model.

Standardization

```
train.loc[:, 'IDENTITY'] = train['IDENTITY'].str.upper()
```

```
valid.loc[:, 'IDENTITY'] = valid['IDENTITY'].str.upper()
train.reset_index(inplace = True, drop=True)
valid.reset_index(inplace = True, drop=True)
```

- Converts labels to uppercase to ensure consistency.

5. Image Preprocessing

```
def preprocess(img):
  (h, w) = img.shape
  final_img = np.ones([64, 256])*255
  # crop
  if w > 256:
    img = img[:, :256]

  if h > 64:
    img = img[:64, :]
  final_img[:h, :w] = img
  return cv2.rotate(final_img, cv2.ROTATE_90_CLOCKWISE)
```

- Resizes/crops each image to **64×256**, rotates to match orientation needed by the model, and normalizes pixel values to 0–1.

6. Data Preparation

```
train_size = 21000
valid_size= 2100
train_x = []
for i in range(train_size):
  img_dir = '/content/handwriting_data/train_v2/train/'+train.loc[i, 'FILENAME']
  image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
```

```
    image = preprocess(image)
    image = image/255.
    train_x.append(image)
valid_x = []
for i in range(valid_size):
    img_dir = '/content/handwriting_data/validation_v2/validation/'+valid.loc[i, 'FILENAME']
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    image = preprocess(image)
    image = image/255.
    valid_x.append(image)
train_x = np.array(train_x).reshape(-1, 256, 64, 1)
valid_x = np.array(valid_x).reshape(-1, 256, 64, 1)
```

- Loads, preprocesses, and stores all training and validation images into arrays.
- Shapes them into **(num_samples, 256, 64, 1)** — the format expected by the CNN.

7. Label Encoding

```
alphabets = u"ABCDEFGHIJKLMNOPQRSTUVWXYZ-' "
max_str_len = 24 # max length of input labels
num_of_characters = len(alphabets) + 1 # +1 for ctc pseudo blank
num_of_timestamps = 64 # max length of predicted labels
def label_to_num(label):
    label_num = []
    for ch in label:
        label_num.append(alphabets.find(ch))
    return np.array(label_num)
def num_to_label(num):
    ret = ""
    for ch in num:
        if ch == -1:  # CTC Blank
```

```
            break
        else:
            ret+=alphabets[ch]
    return ret
```

- Defines the **character set**.
- Converts each label (name) into a **numeric sequence** corresponding to character indices.
- Uses -1 to indicate padding (for CTC loss).
- train_label_len: Actual lengths of labels.
- train_input_len: Fixed input length for model (62).
- Same steps repeated for validation data.
- Final print checks one sample's original and encoded label.

8. Model Architecture

Input Layer

Input(shape=(256, 64, 1))

- Takes grayscale images of size **256×64×1** (height, width, channels).

CNN Blocks (Feature Extraction)

Conv Block 1

Conv2D(32) → BatchNorm → ReLU → MaxPooling(2x2)

- Extracts low-level features.
- Reduces spatial size.

Conv Block 2

Conv2D(64) → BatchNorm → ReLU → MaxPooling(2x2) → Dropout(0.3)

- Learns more complex features.
- Adds regularization.

Conv Block 3

Conv2D(128) → BatchNorm → ReLU → MaxPooling(1x2) → Dropout(0.3)

- Captures fine details with less vertical reduction.

Reshape + Dense (RNN Prep)

Reshape((64, 1024)) → Dense(64)

- Converts CNN output to **sequence format** for RNN.
- Each of 64 time steps has 64 features.

BiLSTM Layers (Sequence Learning)

Bidirectional(LSTM(256)) × 2

- Learns **temporal dependencies** from both directions.
- Helps in recognizing character sequences better.

Output Layer (Character Prediction)

Dense(num_of_characters) → Softmax

- Outputs probability distribution for each character (including CTC blank).

Model Summary

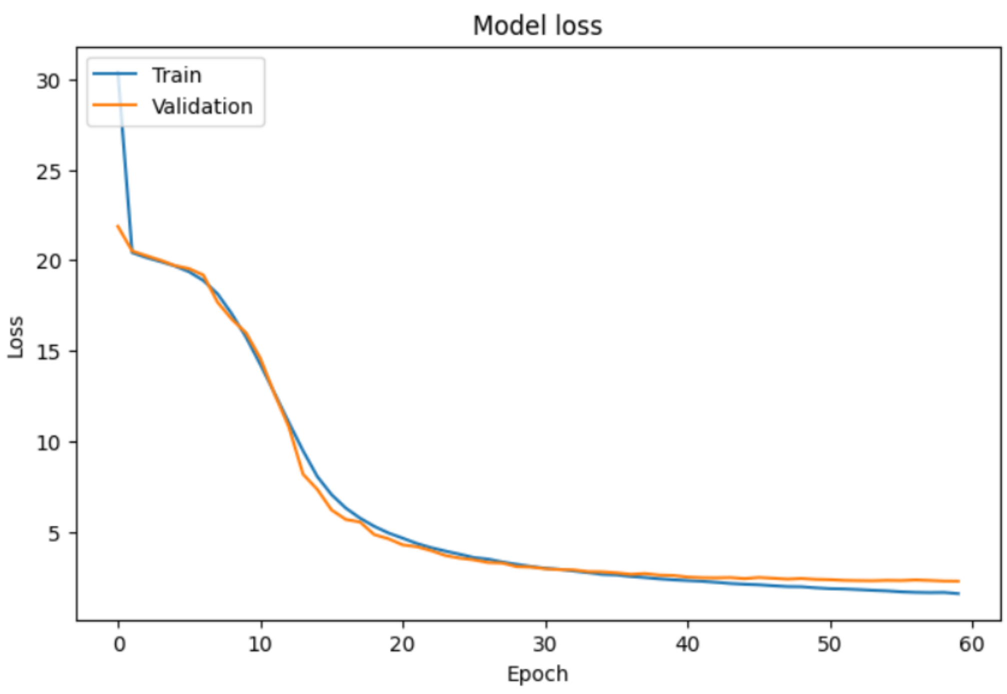
- Combines CNN (for features) and RNN (for sequence modeling).
- Ready for **CTC loss-based handwriting recognition**.

10. Model Compilation and Training

```
def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    # the 2 is critical here since the first couple outputs of the RNN
    # tend to be garbage
    y_pred = y_pred[:, 2:, :]
    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)


# Inputs
labels = Input(name='gtruth_labels', shape=[max_str_len], dtype='float32')
input_length = Input(name='input_length', shape=[1], dtype='int64')
label_length = Input(name='label_length', shape=[1], dtype='int64')


# Move slicing outside the ctc_lambda_func to avoid tf.function error
y_pred_sliced = Lambda(lambda x: x[:, 2:, :], name='y_pred_sliced')(y_pred)
```

```
# Define the CTC loss function without slicing inside
def ctc_lambda_func(args):
    y_pred_sliced, labels, input_length, label_length = args
    return K.ctc_batch_cost(labels, y_pred_sliced, input_length, label_length)

# CTC loss layer
ctc_loss = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')(
    [y_pred_sliced, labels, input_length, label_length]
)

# Final model that includes inputs for data and CTC loss calculation
model_final = Model(inputs=[input_data, labels, input_length, label_length], outputs=ctc_loss)

from tensorflow.keras.optimizers import Adam

# the loss calculation occurs elsewhere, so we use a dummy lambda function for the loss
model_final.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=Adam(learning_rate=0.0001))

history = model_final.fit(x=[train_x, train_y, train_input_len, train_label_len], y=train_output,
                validation_data=([valid_x, valid_y, valid_input_len, valid_label_len], valid_output),
                epochs=60, batch_size=128)
```

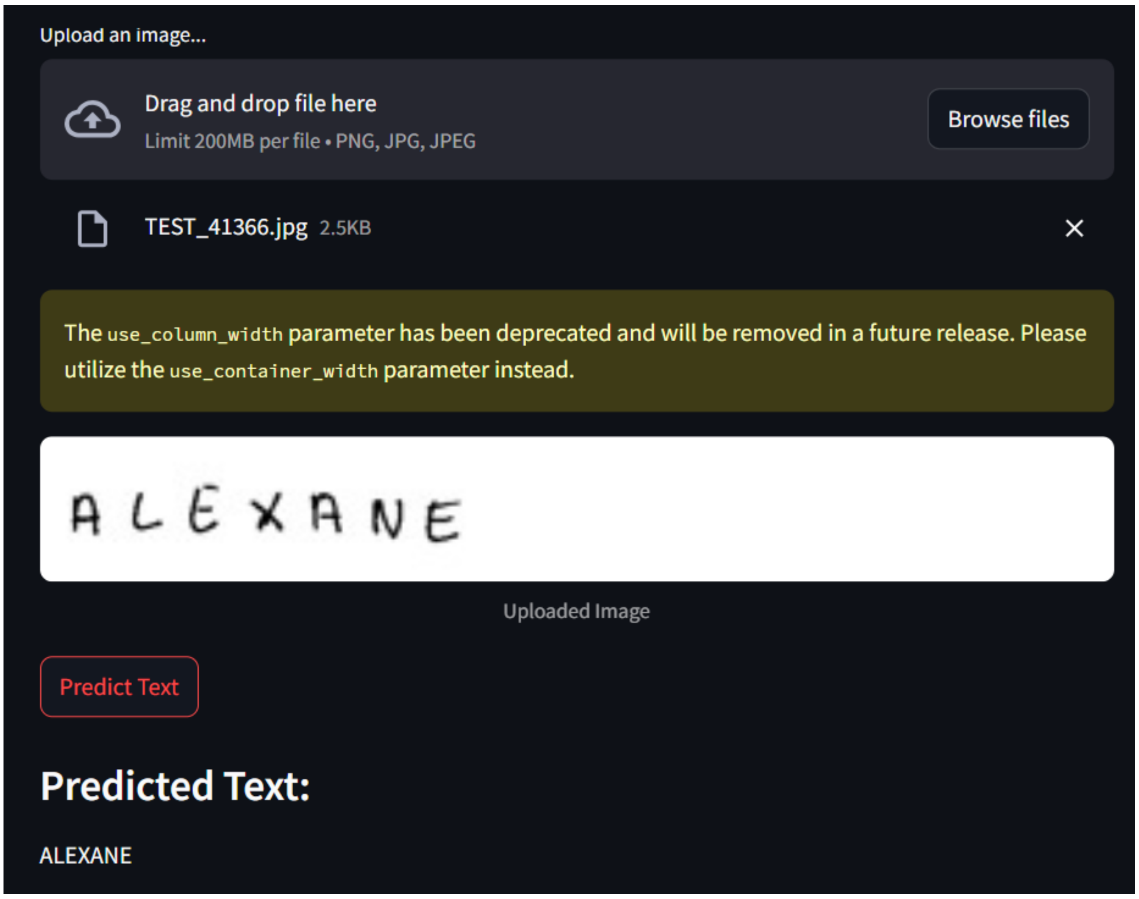
- Compiles the full model using a dummy lambda loss since CTC loss is computed inside the model.
- Trains the model for 60 epochs, using the image and label arrays.



11. Saving Model to Google Drive

```
model_final.save('model_final.h5')


# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')


# Define the path to save the model in Google Drive
model_save_path = '/content/drive/My Drive/handwriting_recognition_model/model_final.h5'


# Save the trained model to Google Drive
model_final.save(model_save_path)


print(f"Model saved to: {model_save_path}")
```

- Mounts Google Drive and saves the trained model (.h5) to a specified directory for later use.



Upload an image...

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

Browse files

TEST_41366.jpg 2.5KB

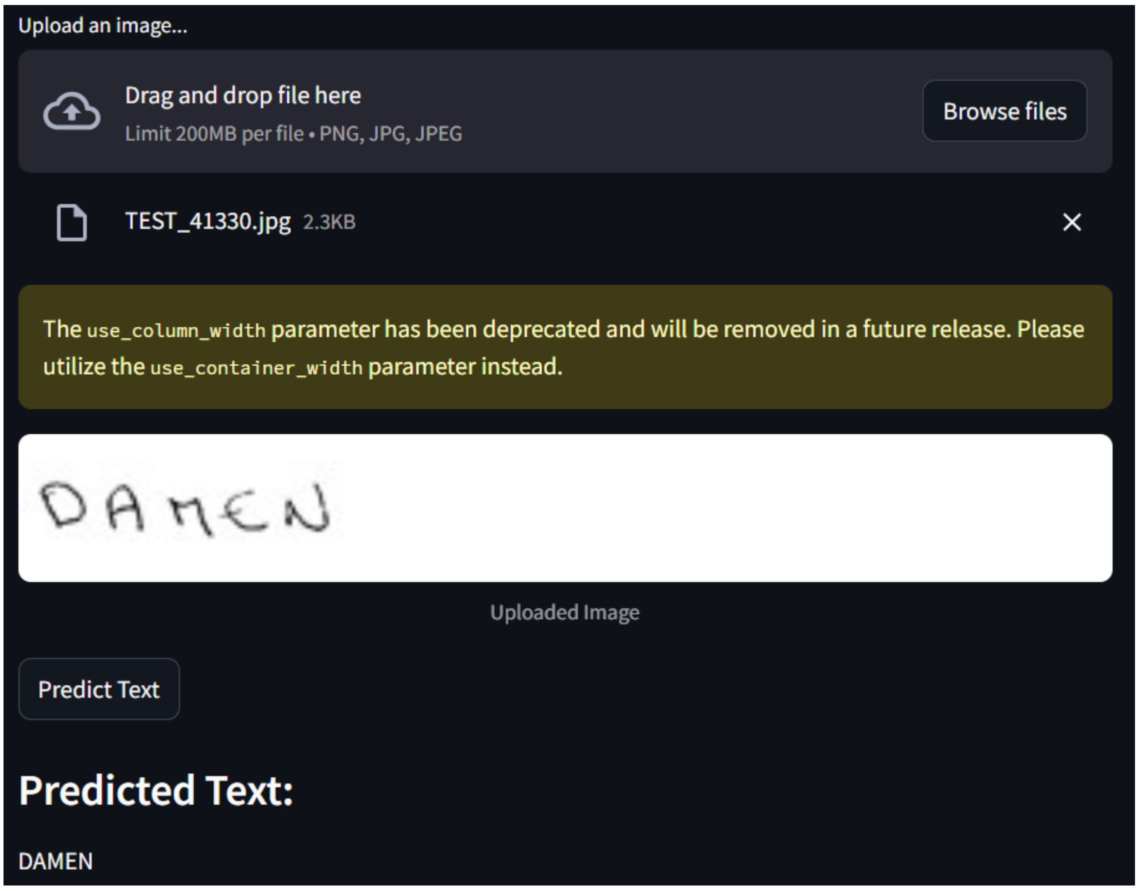
The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Uploaded Image

Predict Text

Predicted Text:

ALEXANE



Upload an image...

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

Browse files

TEST_41330.jpg 2.3KB

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Uploaded Image

Predict Text

Predicted Text:

DAMEN

Plotting Training History:

- The plot_training_history() function visualizes the model's loss over epochs for both training and validation datasets. The plot helps evaluate the model's convergence and the potential overfitting or underfitting during training.

Model Prediction and Evaluation:

- After training, the model makes predictions on the validation data (valid_x), and the CTC loss decoder (K.ctc_decode) is used to decode the predicted sequence into labels. The predictions are compared with the ground truth labels (y_true), and character-level and word-level accuracy are computed. The results are printed as percentages.

Visualizing Predictions on Test Data:

- For a selected range of test images, the code reads and preprocesses each image, feeds it into the model, and decodes the predicted labels using the same CTC decoder. The predictions are then displayed alongside the corresponding images for visual inspection.

Testing with a Single Image:

- A single image (TEST_0005.jpg) is loaded, preprocessed, and passed through the model to make a prediction. The predicted label is decoded and printed.

CTC Loss Customization:

- A custom CTC loss function (ctc_lambda_func) is registered, where the first two time steps in the predicted sequence are skipped, and then the CTC loss is calculated using K.ctc_batch_cost. This is necessary to avoid unstable predictions in the early time steps.

Description of the Model

The model is a **hybrid deep learning architecture** that combines **Convolutional Neural Networks (CNNs)**, **Bidirectional Long Short-Term Memory (BiLSTM) networks**, and the **Connectionist Temporal Classification (CTC) loss function** for handwritten text recognition.

- **CNN layers** extract spatial and visual features from input grayscale images. They effectively capture local patterns like edges and strokes which are essential in recognizing character shapes.
- **BiLSTM layers** process the extracted features as sequences, capturing contextual information from both past (left context) and future (right context) directions. This is crucial because handwriting recognition depends on understanding character sequences and their relationships.

- **CTC loss** enables the model to learn sequence-to-sequence mapping without requiring pre-segmented labels, handling the variable length and alignment between input images and corresponding text sequences.

Why This Model?

- **End-to-end learning:** The CNN-BiLSTM-CTC framework allows direct mapping from images to text sequences without manual segmentation, simplifying training and improving robustness.
- **Handles variable-length sequences:** CTC loss aligns predicted sequences with varying lengths and unsegmented inputs, perfect for handwritten text where character spacing varies.
- **Contextual understanding:** BiLSTMs capture dependencies in both directions, improving recognition accuracy, especially for ambiguous or connected handwriting.
- **Proven success:** This architecture is widely used in handwriting recognition and speech recognition tasks due to its flexibility and strong performance.

Limitations of This Model

- **Computationally intensive:** The combination of CNN and BiLSTM layers results in a large number of parameters, requiring significant computational resources and longer training times.
- **Data dependency:** Requires a large, diverse labeled dataset for effective training to generalize well across different handwriting styles.
- **Difficulty with extremely noisy or degraded images:** Performance may degrade if input images have severe noise, distortions, or low resolution.
- **Limited explicit language understanding:** The model recognizes character sequences but does not inherently understand language semantics, which can lead to errors in ambiguous contexts.

How to Overcome These Limitations

- **Model optimization:** Use model pruning, quantization, or knowledge distillation to reduce model size and speed up inference.
- **Data augmentation:** Employ techniques such as rotation, scaling, noise addition, and elastic distortions to artificially increase dataset diversity and robustness.
- **Preprocessing:** Apply image enhancement and denoising methods to improve input quality.
- **Integrate language models:** Combine with statistical or neural language models (like an RNN or Transformer-based language model) to provide contextual post-processing and correct recognition errors based on language context.
- **Transfer learning:** Use pretrained CNN backbones or train on large handwriting datasets and fine-tune on domain-specific data to improve generalization.

Conclusion

The **CNN + BiLSTM + CTC** model provides a powerful, end-to-end solution for handwritten text recognition by combining effective spatial feature extraction, sequence modeling, and flexible alignment loss. While computational and data challenges exist, these can be mitigated through model optimization, data augmentation, and language modeling integration. This architecture remains a state-of-the-art approach, balancing accuracy and practicality for diverse handwriting recognition applications.