Optical Character Recognition Libraries and Trained Keras Model On Written Text: supplemental report

TYSON GRANT AND ADAM KOLODZIEJCZAK

April 7, 2025

1. INTRODUCTION

This document is designed to assist in understanding the report on Optical Character Recognition Libraries and a Trained Keras Model on Written Text.

2. LEVENSHTEIN DISTANCE

The Levenshtein distance, also known as edit distance, is a quantifiable metric that calculates the minimum number of character edits - insertions, deletions, or substitutions - between two strings. It calculates the number of changes required to move from one string to another [1].

$$D(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+\text{cost} \end{cases} & \text{otherwise} \end{cases} \tag{S1}$$

Where $D(i,j)$ is the edit distance between the first i characters of String A, and the first j characters of String B. Eq. (S1) contains a recursive function where the base case is if the string is empty, and the recursive case that compares the characters at each position and applies the necessary operation [1].

As seen in the Eq. (S2), if the characters at the given index are equal then the cost will be zero and will not be added to the substitution in Eq. (S1).

$$\text{cost} = \begin{cases} 0 & \text{if } A_i = B_j \\ 1 & \text{if } A_i \neq B_j \end{cases} \tag{S2}$$

3. CONNECTIONIST TEMPORAL CLASSIFICATION LOSS

Connectionist Temporal Classifcation (CTC) loss is used to compute the loss between unaligned sequences, producing an output that is differentiable for each input [2]. Regarding OCR, sequence recognition problems are those that the text with images can vary in width and spacing, with the goal of producing a string of the text. Due to these issues, the CTC loss function is used for input sequences longer and may not be perfectly aligned compared to the output [2]. Allowing for characters to be matched from one input to an output, dealing with the issues mentioned in OCR. CTC also contains a blank token to combine repeated predictions into a single character for efficiency.

$$k.ctc\_batch\_cost(labels, y\_pred, input\_length, label\_length) \tag{S3}$$

As seen in Eq. (S3) above, the Keras built-in loss function takes inputs of the ground truth labels as encoded integers, the predicted character probability distribution over time, the length of each predicted sequence and the length of each encoded label sequence. With these inputs, CTC loss computes the negative log-likelihoods of the correct output sequences over time with model predictions [3]. This is done by summing all possible valid alignments, noting that CTC utilizes a blank token and automatically collapses repeated predictions into one character [3].

4. OPTICAL CHARACTER RECOGNITION LIBRARIES

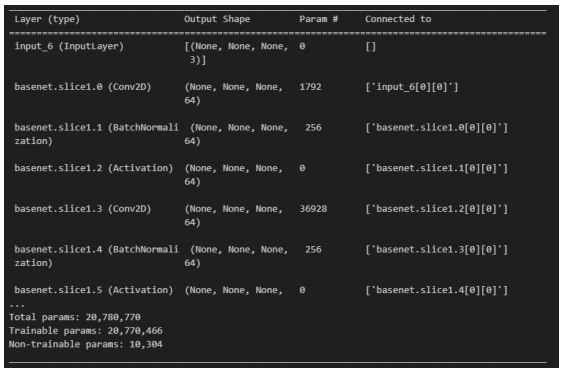
A. PyTesseract

PyTesseract is Python function encapsulator (wrapper) for Google's Tesseract-OCR Engine, replicating its behaviour [4]. The model is built using rule-based Template Matching, which combines template matching with defined rules to discover patterns. The model also uses LSTM Neural Networks to allow for increased recognition accuracy for text of varying size [4]. PyTesseract is primarily trained on typed text of different fonts, so it is not as accurate for different variations of text such as handwritten, causing it to be most accurate when reading printed documents. The `image_to_string` function apart of the library uses minimal image cleaning, so the model quickly processes images. Due to the library being a function encapsulator from the Tesseract Engine, it is only limited to what the Engine can produce since it only makes calls to the engine [4].

Note: PyTesseract does not have a model summary as it strictly creates requests to Google's Tesseract Engine

B. KerasOCR

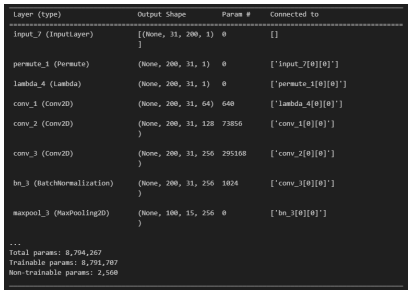
KerasOCR is an OCR library built off of the Keras deep learning framework. It is designed to use deep learning techniques to extract text from noisy image data. The model consists of a detector which finds a bounding box around the text using EAST or CRAFT (both methods of calculating region scores), and a recognizer which recognizes the individual characters within an image using Convoluutional Recurrent Neural Networks (CRNN) [5]. Keras is fast for clean data, but slow for noisy data since it has to leverage the deep networks to compute the image processing. In addition, this pretrained model may be outperformed by those such as PyTesseract or EasyOCR, but due to the use of deep learning networks KerasOCR is highly customizable, allowing for extensive fine-tuning (such as architecture swap) and additional training to increase the recognition accuracy on a specific dataset [5].



Layer (type)	Output Shape	Param #	Connected to
input_6 (InputLayer)	[(None, None, None, 3)]	0	[]
basenet.slice1.0 (Conv2D)	(None, None, None, 64)	1792	['input_6[0][0]']
basenet.slice1.1 (BatchNormalization)	(None, None, None, 64)	256	['basenet.slice1.0[0][0]']
basenet.slice1.2 (Activation)	(None, None, None, 64)	0	['basenet.slice1.1[0][0]']
basenet.slice1.3 (Conv2D)	(None, None, None, 64)	36928	['basenet.slice1.2[0][0]']
basenet.slice1.4 (BatchNormalization)	(None, None, None, 64)	256	['basenet.slice1.3[0][0]']
basenet.slice1.5 (Activation)	(None, None, None,	0	['basenet.slice1.4[0][0]']

...
Total params: 20,780,770
Trainable params: 20,770,466
Non-trainable params: 10,304

Fig. S1. KerasOCR Detector Model Summary.



Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	[(None, 31, 200, 1)]	0	[]
permute_1 (Permute)	(None, 200, 31, 1)	0	['input_7[0][0]']
lambda_4 (Lambda)	(None, 200, 31, 1)	0	['permute_1[0][0]']
conv_1 (Conv2D)	(None, 200, 31, 64)	640	['lambda_4[0][0]']
conv_2 (Conv2D)	(None, 200, 31, 128)	73856	['conv_1[0][0]']
conv_3 (Conv2D)	(None, 200, 31, 256)	295168	['conv_2[0][0]']
bn_3 (BatchNormalization)	(None, 200, 31, 256)	1024	['conv_3[0][0]']
maxpool_3 (MaxPooling2D)	(None, 100, 15, 256)	0	['bn_3[0][0]']

...
Total params: 8,704,267
Trainable params: 8,701,707
Non-trainable params: 2,560

Fig. S2. KerasOCR Recognizer Model Summary.

C. EasyOCR

EasyOCR is another deep-learning based library, but it uses a hybridization of deep learning models and PyTorch. EasyOCR can also be implemented to read up to 80 different languages, allowing multiple languages to be read in the same image. It leverages Convolutional Neural Networks to detect text, and Recurrent Neural Networks to recognize text [6]. This model is efficient in recognizing complex text, but noisy text or unusual writing styles may affect the accuracy of the model. It is also not accurate for poor-quality images since it does not allow for image processing within the deep learning network, which can be avoided by applying pre-processing techniques [6]. EasyOCR is most slightly customizable using PyTorch deep learning techniques, but does not allow for customization to the loss function or image pre-processing within the model pipeline [6]. The EasyOCR model is best used suited for multi-lingual OCR tasks, or those with complex formats such as those with varying direction of text [6].

```
CRAFT(
  (basenet): vgg16_bn(
    (slice1): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU(inplace=True)
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (9): ReLU(inplace=True)
      (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (slice2): Sequential(
      (12): ReLU(inplace=True)
      (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (16): ReLU(inplace=True)
      (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
...
    (7): ReLU(inplace=True)
    (8): Conv2d(16, 2, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

Fig. S3. EasyOCR Detector Model Summary.

```
Model(
  (FeatureExtraction): VGG_FeatureExtractor(
    (ConvNet): Sequential(
      (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): ReLU(inplace=True)
      (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (7): ReLU(inplace=True)
      (8): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (9): ReLU(inplace=True)
      (10): MaxPool2d(kernel_size=(2, 1), stride=(2, 1), padding=0, dilation=1, ceil_mode=False)
      (11): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (13): ReLU(inplace=True)
      (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (16): ReLU(inplace=True)
      (17): MaxPool2d(kernel_size=(2, 1), stride=(2, 1), padding=0, dilation=1, ceil_mode=False)
      (18): Conv2d(256, 256, kernel_size=(2, 2), stride=(1, 1))
      (19): ReLU(inplace=True)
    )
  )
  (AdaptiveAvgPool): AdaptiveAvgPool2d(output_size=(None, 1))
  (SequenceModeling): Sequential(
    (0): BidirectionalLSTM(
      (rnn): DynamicQuantizedLSTM(256, 256, batch_first=True, bidirectional=True)
      (linear): DynamicQuantizedLinear(in_features=512, out_features=256, dtype=torch.qint8, qscheme=torch.per_tensor_affine)
```

Fig. S4. EasyOCR Recognizer Model Summary.

REFERENCES

1. Paperspace, *Measuring Text Similarity Using Levenshtein Distance*, Available at: https://blog.paperspace.com/measuring-text-similarity-using-levenshtein-distance/.
2. GeeksforGeeks, *Connectionist Temporal Classification (CTC) for Sequence Modelling*, Available at: https://www.geeksforgeeks.org/connectionist-temporal-classification/.
3. PyTorch Documentation, *torch.nn.CTCLoss*, Available at: https://pytorch.org/docs/stable/generated/torch.nn.CTCLoss.html.
4. Samuel Hoffstaetter. *Python Tesseract - A Python wrapper for Google Tesseract-OCR*. Available at: https://github.com/madmaze/pytesseract. Accessed: March 31, 2025.
5. Fausto Morales. *keras-ocr: A packaged OCR pipeline using Keras and TensorFlow*. Available at: https://github.com/faustomorales/keras-ocr. Accessed: March 31, 2025.
6. JaidedAI. *EasyOCR: Ready-to-use OCR with 80+ supported languages*. Available at: https://github.com/JaidedAI/EasyOCR. Accessed: March 31, 2025.