Optical Character Recognition With Pytesseract

Vincent Pangestu
Computer Science
Bina Nusantara University
Tangerang, Indonesia
kevin.tanadi@binus.ac.id

Nabil Ananta Satria Gharu
Computer Science
Bina Nusantara University
Tangerang, Indonesia
nabil.gaharu@binus.ac.id

Jason Christian Wijaya
Computer Science
Bina Nusantara University
Tangerang, Indonesia
jason.wijaya008@binus.ac.id

***Abstract*—Optical Character Recognition (OCR) is a process that converts an image of text into machined-readable text format (soft copy). In this project, we use Pytesseract and CRNN to create Optical Character Recognition (OCR) machine. We use Pytesseract and CRNN because we consider having a great result and easy to use. Pytesseract is an OCR tool for python that also serves as a wrapper for the Tesseract-OCR Engine. While CRNN is OCR tool for python that also serves as a wrapper for the Tesseract-OCR Engine.**

***Keywords*—Optical Character Recognition, OCR, deep learning, CRNN, Convolutional Recurrent Neural Networks**

I. Introduction

Optical character recognition (OCR) is a system that converts input text into machine-encoded format. Nowadays, OCR is helping not only in digitizing the handwritten medieval manuscripts, but also helps in converting physical documents into digital form/soft copy. This has made many people easier as they doesn't have to go through the physical documents and files to search something they want. Many people or organization are satisfying the needs of digital preservation of historic data, law documents, educational persistence, etc. In this paper, we are going to use Python-tesseract and CRNN model.

CRNN is an abbreviation of Convolutional Recurrent Neural Networks. Convolutional Recurrent Neural Networks is the combination of two of the most prominent neural networks. The CRNN (convolutional recurrent neural network) involves CNN(convolutional neural network) followed by the RNN(Recurrent neural networks). CRNN is a model that feeds every window frame by frame into a recurrent layer and use the outputs and hidden states of the recurrent units in each frame for extracting features from the sequential windows.

Tesseract is an open source text recognition (OCR) Engine, available under the Apache 2.0 license. It can be used directly, or (for programmers) using an API to extract printed text from images. Tesseract is compatible with many programming languages and frameworks. It can be used with the existing layout analysis to recognize text within a large document, or it can be used in conjunction with an external text detector to recognize text from an image of a single text line. Tesseract was developed at HP between 1984 and 1994. It appeared from nowhere for the 1995 UNLV Annual Test of OCR Accuracy, shone brightly with its results, and then vanished back under the same cloak of secrecy under which it had been developed. Now for the first time, details of the architecture and algorithms can be revealed.

II. Problem Statement

A. Problem Formulation

1. How does an OCR work?
2. What would you need for an OCR?

B. Purpose of Research

1. Knows how an OCR work
2. Know key problems in a OCR
3. Compare and Pytesseract and from scratch OCR

III. Proposed Method

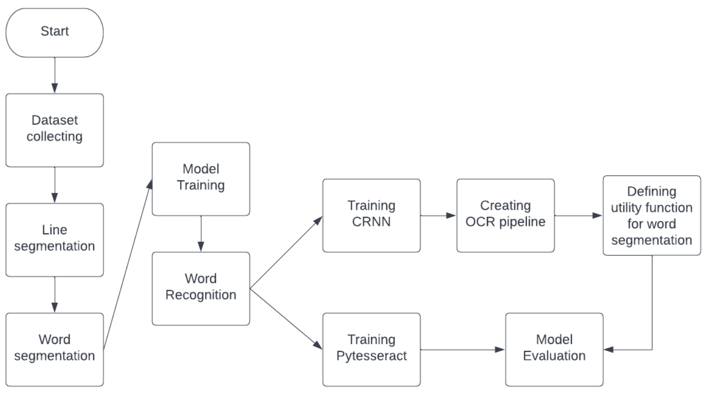


Fig. 1. Overall method

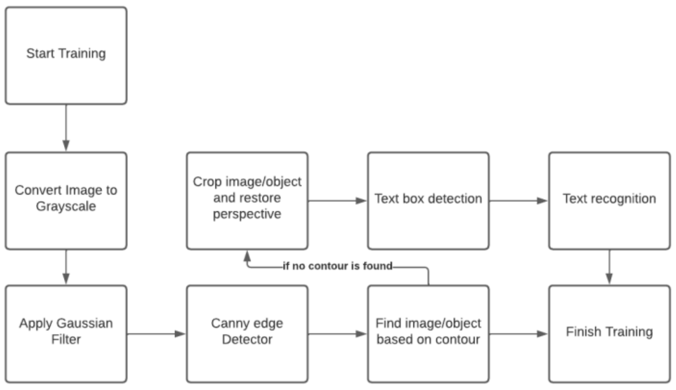


Fig. 2. Model training method

IV. Methodology

1. Dataset

1.1 Text Segmentation

For the Image Segmentation/ text segmentation, we are using 2 kinds of data. The first one we are using cropped images of a text and the second is the masked/annotated version of the first data. With this we can train the machine to detect text inside an image. The data was collected manually and the total amount of data are 300 images of pure text and 300 images of masked text image.

1.2 Word Segmentation

As we already have a model to detect text inside an image, we will use the same method from the text segmentation to create a data for each word by cropping each line from the text recognition data. The total amount of data reach 1.100 for both the text image and the mask image.

1.3 CRNN/ Text Recognition

The data used are generated data from a data generator. The generator will create around 30000 images and text of random text/words

2. Model

We use a certain number of models used differently on each problem in the process. Here are the models used:

2.1 CRNN (Convolutional Recurrent Neural Network)

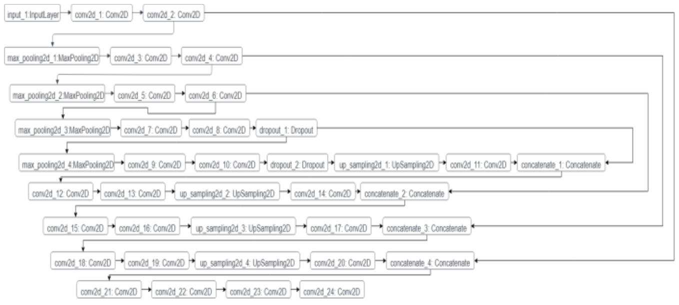
Convolutional Neural Networks are a type of Neural Networks that use the operation of convolution (sliding a filter across an image) in order to extract relevant features. CNN is needed because they perform better on data (rather than using normal dense Neural Networks) in which there is a strong correlation between, for example, pixels because the spatial context is not lost. CNN uses filters in order to extract features. The filters are matrixes that “slide” over the image. They are modified in the training period in order to extract the most relevant features.

On the other hand, a recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate. Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence. While future events would also be helpful in determining the output of a given sequence, unidirectional recurrent neural networks cannot account for these events in their predictions.

The Convolutional Recurrent Neural Networks is the combination of two of the most prominent neural networks. The CRNN (convolutional recurrent neural network) involves CNN (convolutional neural network) followed by the RNN (Recurrent neural networks). The proposed network is like CRNN but generates better or optimal results especially towards audio signal processing. Convolutional Neural Network extracts the features by applying relevant filters and the Recurrent Neural Network analyzes these features, taking into consideration information received from previous time-steps.

2.2 Unet Model

The U-Net is a convolutional neural network architecture that is designed for fast and precise segmentation of images. It has performed extremely well in several challenges and to this day, it is one of the most popular end-to-end architectures in the field of semantic segmentation.



We can split the network into two parts: The encoder path (backbone) and the decoder path. The encoder captures features at different scales of the images by using a traditional stack of convolutional and max pooling layers. Concretely speaking, a block in the encoder consists of the repeated use of two convolutional layers (k=3, s=1), each followed by a non-linearity layer, and a max-pooling layer (k=2, s=2). For every convolution block and its associated max pooling operation, the number of feature maps is doubled to ensure that the network can learn the complex structures effectively.

The decoder path is a symmetric expanding counterpart that uses transposed convolutions. This type of convolutional layer is an up-sampling method with trainable parameters and performs the reverse of (down)pooling layers such as the max pool. Similar to the encoder, each convolution block is followed by such an up-convolutional layer. The number of feature maps is halved in every block. Because recreating a segmentation mask from a small feature map is a rather difficult task for the network, the output after every up-convolutional layer is appended by the feature maps of the corresponding encoder block. The feature maps of the encoder layer are cropped if the dimensions exceed the one of the corresponding decoder layers.

In the end, the output passes another convolution layer (k=1, s=1) with the number of feature maps being equal to the number of defined labels. The result is a u-shaped convolutional network that offers an elegant solution for good localization and use of context. This model performs text segmentation which deals with the correct division of a document into semantically coherent blocks.

2.3 Pytesseract

Pytesseract (Python-tesseract) is an optical character recognition (OCR) tool for python. That is, it will recognize and “read” the text embedded in images. Python-tesseract is a wrapper for Google’s Tesseract-OCR Engine. It is also useful as a stand-alone invocation script to tesseract, as it can read all image types supported by the Python Imaging Library, including jpeg, png, gif, bmp, tiff, and others.

Additionally, if used as a script, Python-tesseract will print the recognized text instead of writing it to a file. We use this model at the final part of our project after we got the scanned version of the image by restoring the perspective. The model applied OCR to find all texts within the image by locating the text boxes on the image followed by extraction of all the texts within the image.

3. Proposed Method

3.1 Preprocessing

We will be adjusting all file sizes and type throught out each process to ensure that all images are the same size. In this preprocessing we resize all images to 512 x 512 by padding each image so that each image can be inserted to the model.

3.2 Text Segmentation Training

Text Segmentation is a process for the machine to find which object is considered a text. In this process. In this process we will use Unet model because of it’s structure. In Unet, the image will be first downsampled and then upsampled to it’s original size. So, the most important part of the image will still be preserved. With this it would help in improving the segmentation. The model of the text segmentation will be saved as a pretrained model and be used in a python file to create the OCR machine.

3.3 Word Segmentation Training

Word Segmentation is a process to segment all the words within the parameters. From text segmentation we will retrieve all the text in the image. From that data, word segmentation will find all available words inside each line of text. In this process we will also be using a Unet model because of the same reason as the text segmentation. The Model of words segmentation will be saved to create the OCR Machine

3.4 Text Recognization Training

Text Recognition is the most important part of the Optical Character Recognition. Text categorization will categorize and know what character it is from the image given. In this Text categorization, we will be using CRNN followed with a CTC layer that will predict each text within the image. The Convolutional layers will extract visual information from the data, the Recurrent Layers will capture all temporal information in the image and then the CTC will predict the character based on the information. The final model of the CRNN will be saved for the OCR machine.

3.5 OCR Machine

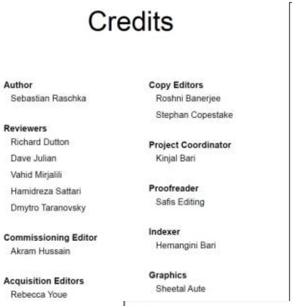
The OCR machine will be comprised of all of the pretrained models obtained from the previous process. The OCR machine will then be given a picture of text to recognize each text in the image.

3.6 Pytesseract

Pytesseract is a well known OCR tool. We are using Pytesseract to compare our OCR and pytesseract. Pytesseract is not very complicated and can be used after a bit of preprocessing to the image. With Pytesseract we will first preprocess the image with gaussian blur. We will highlight all detected text within box and then print them.

V. Results

1. Text Segmentation Training



Credits

Author
Sebastian Raschka

Reviewers
Richard Dutton
Dave Julian
Vahid Mirjalili
Hamidreza Sattari
Dmytro Taranovsky

Commissioning Editor
Akram Hussain

Acquisition Editors
Rebecca Youe

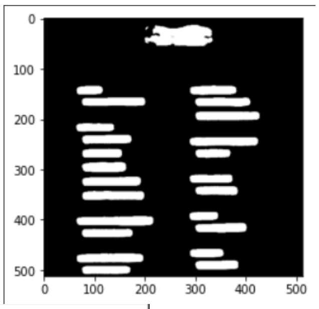
Copy Editors
Roshni Banerjee
Stephan Copestake

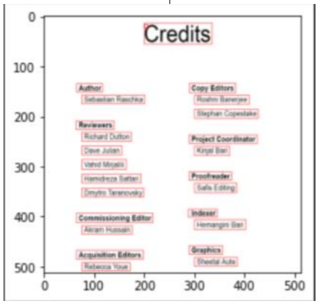
Project Coordinator
Kinjal Bari

Proofreader
Safis Editing

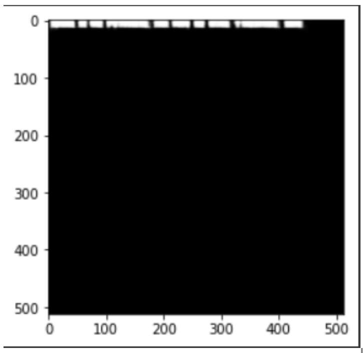
Indexer
Hemangini Bari

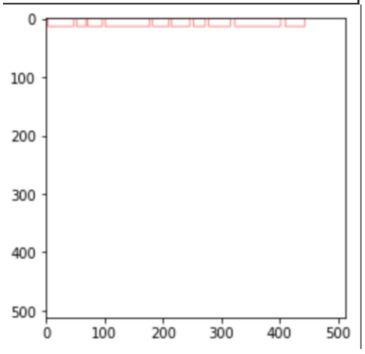
Graphics
Sheetal Aute



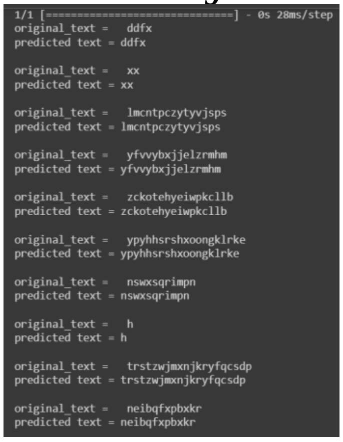


2. Word Segmentation



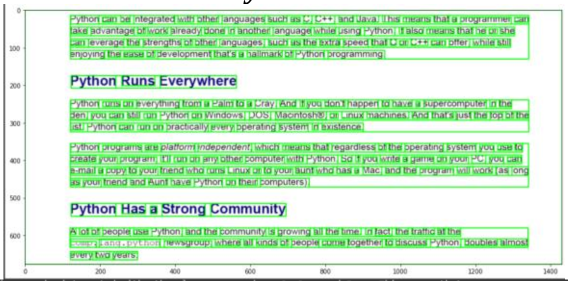


3. Text Recognition



```
1/1 [==============================] - 0s 28ms/step
original_text =   ddfx
predicted text = ddfx

original_text =   xx
predicted text = xx

original_text =   lmcntpczytyvjsps
predicted text = lmcntpczytyvjsps

original_text =   yfvvybxjjelzrmhm
predicted text = yfvvybxjjelzrmhm

original_text =   zckotehyeiwpkcllb
predicted text = zckotehyeiwpkcllb

original_text =   ypyhhsrshxoongklrke
predicted text = ypyhhsrshxoongklrke

original_text =   nswxsqrimpn
predicted text = nswxsqrimpn

original_text =   h
predicted text = h

original_text =   trstzwjmxnjkryfqcsdp
predicted text = trstzwjmxnjkryfqcsdp

original_text =   neibqfxpbxkr
predicted text = neibqfxpbxkr
```

4. Pytesseract



Python can be integrated with other languages such as C, C++, and Java. This means that a programmer can take advantage of work already done in another language while using Python. It also means that he or she can leverage the strengths of other languages, such as the extra speed that C or C++ can offer, while still enjoying the ease of development that's a hallmark of Python programming.

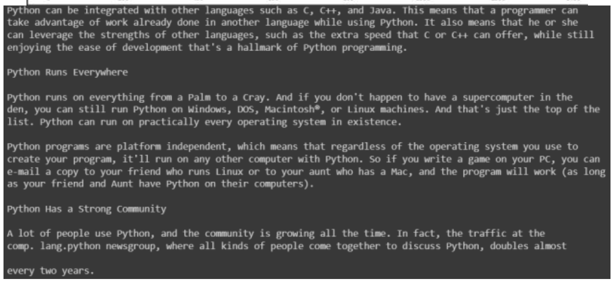
Python Runs Everywhere

Python runs on everything from a Palm to a Cray. And if you don't happen to have a supercomputer in the den, you can still run Python on Windows, DOS, Macintosh®, or Linux machines. And that's just the top of the list. Python can run on practically every operating system in existence.

Python programs are *platform independent*, which means that regardless of the operating system you use to create your program, it'll run on any other computer with Python. So if you write a game on your PC, you can e-mail a copy to your friend who runs Linux or to your aunt who has a Mac, and the program will work (as long as your friend and Aunt have Python on their computers).

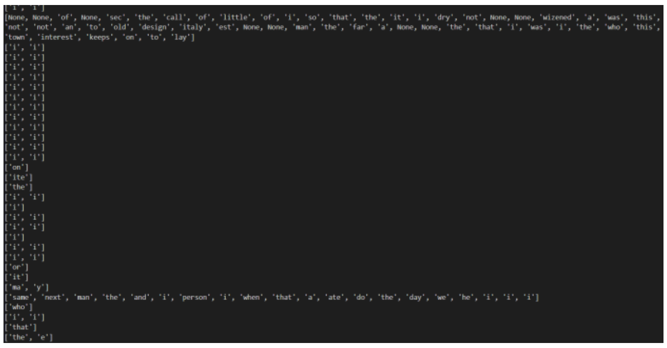
Python Has a Strong Community

A lot of people use Python, and the community is growing all the time. In fact, the traffic at the comp.lang.python newsgroup, where all kinds of people come together to discuss Python, doubles almost every two years.



```
Python can be integrated with other languages such as C, C++, and Java. This means that a programmer can
take advantage of work already done in another language while using Python. It also means that he or she
can leverage the strengths of other languages, such as the extra speed that C or C++ can offer, while still
enjoying the ease of development that's a hallmark of Python programming.

Python Runs Everywhere

Python runs on everything from a Palm to a Cray. And if you don't happen to have a supercomputer in the
den, you can still run Python on Windows, DOS, Macintosh®, or Linux machines. And that's just the top of the
list. Python can run on practically every operating system in existence.

Python programs are platform independent, which means that regardless of the operating system you use to
create your program, it'll run on any other computer with Python. So if you write a game on your PC, you can
e-mail a copy to your friend who runs Linux or to your aunt who has a Mac, and the program will work (as long
as your friend and Aunt have Python on their computers).

Python Has a Strong Community

A lot of people use Python, and the community is growing all the time. In fact, the traffic at the
comp. lang.python newsgroup, where all kinds of people come together to discuss Python, doubles almost

every two years.
```

5. OCR Machine



```
['i', 'i']
[None, None, 'of', None, 'sec', 'the', 'call', 'of', 'little', 'of', 'i', 'so', 'that', 'the', 'it', 'i', 'dry', 'not', None, None, 'wizened', 'a', 'was', 'this',
'not', 'not', 'an', 'to', 'old', 'design', 'italy', 'est', None, None, 'man', 'the', 'far', 'a', None, None, 'the', 'that', 'i', 'was', 'i', 'the', 'who', 'this',
'town', 'interest', 'keeps', 'on', 'to', 'lay']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['i', 'i']
['on']
['ite']
['the']
['i', 'i']
['i']
['i', 'i']
['i', 'i']
['i']
['i', 'i']
['i', 'i']
['or']
['it']
['ma', 'y']
['same', 'next', 'man', 'the', 'and', 'i', 'person', 'i', 'when', 'that', 'a', 'ate', 'do', 'the', 'day', 'we', 'he', 'i', 'i', 'i']
['who']
['i', 'i']
['that']
['the', 'e']
```

VI. CONCLUSION

According to the results shown above, our OCR still needs a lot of improvements and has some errors in the process. As can be seen from the results above, the last OCR machine has failed to recognize some of the text and only outputed some of the characters. We conclude that this error was because of the lack of data from the CRNN model that it could not detect more of its words. Further time will be neeeded to optimize the software. Another test that may help in improving the software are to run more tests. Some of those test that we conclude is: [1] Text Recognition instead can be done for each line instead of every word. [2] increase the number of validations and data to increase accuracy. [3] Increase the number of data trained in the CRNN model

R
eferen
ces

[1] https://medium.com/geekculture/building-a-complete-ocr-engine-from-scratch-in-python-be1fd184753b

[2] https://medium.com/geekculture/detecting-text-lines-in-a-document-image-using-deep-learning-5a21b480bc4c