Application for AI-OCR Module: Auto Detection of Emails/Letter Images

Kelsey Fargas, Bingjie Zhou, Elizabeth Staruk, Sheila Tejada
University of Southern California
{ kfargas, binjiez, staruk, stejada } @usc.edu

Abstract

The purpose of this project is to provide instructions for teaching the Artificial Intelligence topic of supervised machine learning for the task of Optical Character Recognition (OCR) at various levels of a student's undergraduate curriculum, such as basic knowledge, novice, and intermediate. The levels vary from beginner with a slight background in computing and computer science to intermediate with a better understanding of computer science fundamentals and algorithms.

Project Description

The Autobots (Tejada, et al. 2016) project uses a library named PIL (Python Imaging Library) and Pytesser (Tesseract 2016) to do a simple text recognition task. The task is to identify a sequence of letters from a given set of images by using pytesser.

This machine learning application will focus on Optical Character Recognition. Optical Character Recognition (OCR) is the mechanical or electronic conversion of images of typewritten or printed text into machine-encoded text. It is widely used as a form of data entry from printed paper data records such as passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation. It is a common method of digitizing printed texts so that it can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as machine translation, text-to- speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Background

There are variety of types of AI, and we will focus on supervised machine learning at this level, and discuss two important categories of supervised learning: classification and regression. Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. Each example is a pair consisting of an input object and a desired output value. A classification task is if the prediction task is to classify the observations in a set of finite labels. An example of this would be the game of chess. Once the computer makes the move, it bases its next move off a finite set of possible moves on the board. It is using the data from the users moving patterns as well as figuring out the best possible move to make given its limited options. It's taking input (user data plus possible options for moves) and spitting the output (finite prediction). A regression task is to predict continuous target variable. Take for example a website that predicts the price of houses in a certain location. It would take the input as the location of the house, and output a price range. Since this is a spectrum of numbers, it is not finite, therefore it is a continuous target variable.

Evaluation. We now know that AI can be used to solve different tasks. But how do we figure out how well our AI solved the task? Does it have to get every answer right? For some tasks, a human being can't even do that! We use different metrics in order to understand the performance of our AI. This is called evaluation. We suggest using evaluation as an optional, advanced way to teach students. This is a very important step to understanding AI, and should at least be broadly touched upon. If the students are experienced programmers (exposure to data structures and algorithms experience), they may be ready to implement their own evaluation metrics. At the beginner level, it is suggested to use the training/test data provided and ask the students to check their results using a provided script. The given Autobots script has restrictions on how the input/output should appear and automatically reports the students' accuracy, precision, recall, and F1 score. This script is implemented in python2.7 and requires no dependencies. All possibilities of these given metrics are provided, and it is encouraged to customize the script according to the students' experience level.

Preprocessing. Scanned documents are not always straight. They can be skewed by the scanner feeder or get inserted in the wrong direction (upside down). For the best possible results, these images need correction before being processed by the OCR engine. Image pre-processing can significantly increase the reliability of an optical inspection. There are many ways to do preprocessing, and we will be using binarization, which is converting a gray-scale document image into a binary document image and accordingly facilitating the ensuing tasks such as document skew estimation and document layout analysis, and noise reduction, the process of removing noise from a signal.

OCR Assignment

Python is a very simple and quick language to learn, therefore it will be the chosen language for this assignment. At the end of this assignment, beginner level students will have exposure to AI and a basic understanding OCR along with utilizing Python. The task is to identify emails given an image that will need to be parsed. PIL is the python imaging library that will help you parse through the image and separate each character.

For privacy issues, the data will be a sequence of letters, but contained in the same format as the original email. The image below shows the original copy from the front page of a student exam (Figure 1).

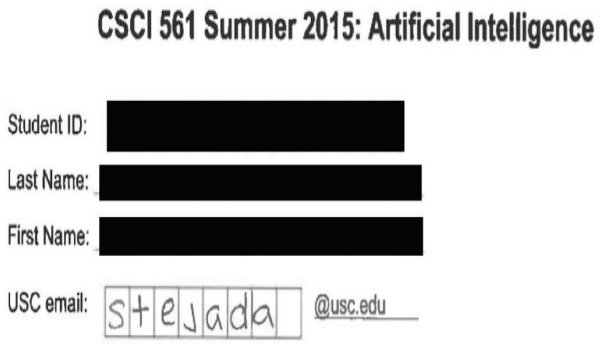

Figure 1. Handwritten email address

Part of the assignment is to crop the necessary section and remove the vertical lines which border each letter. This is done by pre-processing the raw data and the second image demonstrates in Figure 2 the final version of the image after being processed.

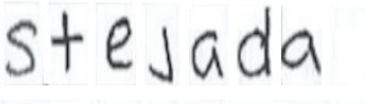

Figure 2. Processed handwritten email address

Input Format: 5 sample input images will be given in the data set. Students will use the library to identify the sequence of letters and verify they are correct. Below in Figure 3 is an example of the image after being parsed.

Figure 3. Example input file

Output Format: Create a file named "output.txt" and write to the file the string that pytesser gives. For example, if the output string from pytesser does not match with any string in given list of email addresses, "None" is written to the output file.

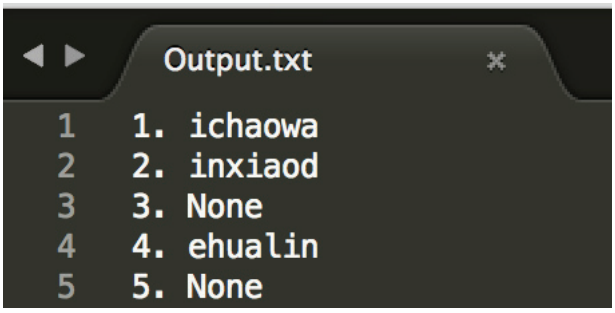

Figure 4. Example output file

The output shows that:

- img1.bmp was matched with ichaowa
- img2.bmp was matched with inxiaod
- img3.bmp was not matched
- img4.bmp was matched with ehualin
- img5.bmp was not matched

Submission: Students submit their source code with the output file and a readme file. The readme file explains the method used to solve this task. The next step is to analyze the results, and explain where/how can the performance be improved.

References

Tejada S., Fargas K., Bingjie Z., Staruk E. 2016. *Autobots*. http://www-scf.usc.edu/~csci561a/Autobots_Assignment.html, Department of Computer Science, University of Southern California, Los Angeles, CA.

Tesseract. 2016. https://github.com/tesseract-ocr/tesseract/wiki