Task 1

Optical Character Recognition

Optical Character Recognition involves the detection of text content on images and translation of the images to encoded text that the computer can easily understand. An image containing text is scanned and analyzed in order to identify the characters in it. Upon identification, the character is converted to machine-encoded text. Text on an image is easily discernible and we are able to detect characters and read the text, but to a computer, it is all a series of dots.
The image is first scanned and the text and graphics elements are converted into a bitmap, which is essentially a matrix of black and white dots. The image is then pre-processed where the brightness and contrast are adjusted to enhance the accuracy of the process.
The image is now split into zones identifying the areas of interest such as where the images or text are and this helps kickoff the extraction process. The areas containing text can now be broken down further into lines and words and characters and now the software is able to match the characters through the comparison and various detection algorithms. The final result is the text in the image that we're given.

Easy OCR

Easy OCR is used to complete this task
EasyOCR is built with Python having a GPU could speed up the whole process of detection. The detection part is using the CRAFT algorithm and the Recognition model is CRNN. It is composed of 3 main components, feature extraction (we are currently using Resnet), sequence labelling (LSTM) and decoding (CTC). EasyOCR doesn't have much software dependencies, it can directly be used with its API.

Installation

Pip install easy ocr

Text Detection in Images with EasyOCR

EasyOCR can process multiple languages at the same time provided they are compatible with each other.

The Reader class is the base class for EasyOCR which contains a list of language codes and other parameters such as GPU that is by default set to True. This needs to run only once to load the necessary models. Model weights are automatically downloaded or can be manually downloaded as well.

Result

Input image:

```
Image("imageocr1.jpg")
```

Output:

```
: output
```

```
: [([[386, 145], [496, 145], [496, 171], [386, 171]],
   'DL-3C',
   0.5926580165261918),
  ([[382, 172], [438, 172], [438, 198], [382, 198]], 'AR-', 0.9999472110163546),
  ([[456, 172], [534, 172], [534, 202], [456, 202]],
   '0200',
   0.9957411289215088)]
```

<matplotlib.image.AxesImage at 0x23385cac400>

Text extracted is enclosed within rectangle

TASK 2

For this OCR project, the *PyTesseract* is used, a library which is a wrapper for Google Tesseract-OCR engine

For creating a web server Flask is used in this task. Flask application would make it more user-friendly and versatile. For instance, we can upload photos via the website and get the extracted text displayed on the website or we can capture photos via the web camera and perform character recognition on them.

First, we'll add functionality to upload images to our Flask app and pass them to the ocr_core function that we wrote above. We will then render the image beside the extracted text on our web app as a result:

As we can see in the upload_page() function, we will receive the image via POST and render the upload HTML if the request is GET.
We check whether the user has really uploaded a file and use the function allowed_file() to check if the file is of an acceptable type.
Upon verifying that the image is of the required type, we then pass it to the character recognition script we created earlier.
The function detects the text in the image and returns it. Finally, as a response to the image upload, we render the detected text alongside the image for the user to see the results.

The upload.html file will handle the posting of the image and rendering of the result by the help of the jinja templating engine which ships with Flask by default.