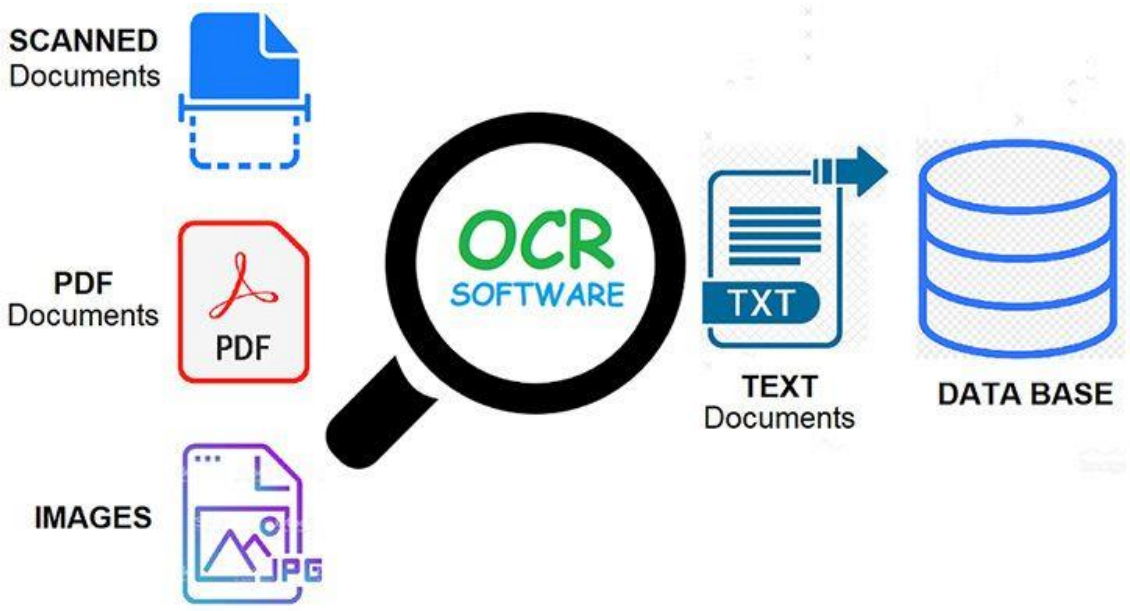
OCR Custom Model - Crash Course

Introduction of OCR:

OCR, or Optical Character Recognition, is a process of recognizing text inside images and converting it into an electronic form. These images could be of handwritten text, printed text like documents, receipts, name cards, etc., or even a natural scene photograph.

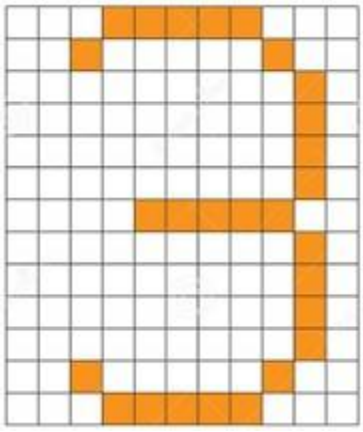


Motivation:

Now that you understand what an OCR is, you must be wondering “That is all well and good, but why do we have the need of one?”

Well, our computers are frankly very dumb, and cannot simply read text wherever they see it. Data is fed to a computer in various formats like .txt, .pdf, .docx, .ppt, .jpg and many more extensions. The point is, while few of these formats (such as txt or docx) allow the computer to detect text, others (such as jpg) do not. In this case, the computer is confused!

Take this picture for example:



What we see in this image is the number 3. But that is not what the computer sees. It sees an image that has certain pixels white, while some others orange. But what it does not see, is the number 3.

So **how** are we going to make an OCR?

We are basically trying to **teach** the computer, like we do in all machine learning algorithms, how to read these patterns in pixels and relate it to the numeric value: 3.

We need to use a program that will allow us to teach a computer, which, you guessed it! We need to use Neural Networks; more specifically, Convolutional Neural Networks (CNNs).

Now assuming you are familiar with the idea of CNN, we can move on with OCR. Since we are to teach the program to understand various characters by the patterns of pixels, let us use one of the most basic and yet powerful applications of CNNs: Image Classifier. This little program allows the computer to differentiate two images based on the difference in their detected features after convolution.

When we have a CNN Model (Classifier Model) ready, we need some images to train the model on. I searched and experimented a lot. I eventually found the NIST dataset to be the most useful and produced accurate results (around 94.8% accurate).

To give you a very vague idea of what needs to be done, here is the list

- Image Thresholding
- Contour formation
- Detecting individual characters and forming a Region of Image (ROI) around detected characters.
- Passing the ROIs through our trained model and finding out the predicted character
- Saving/Appending this character in a string, which is basically going to be your output.

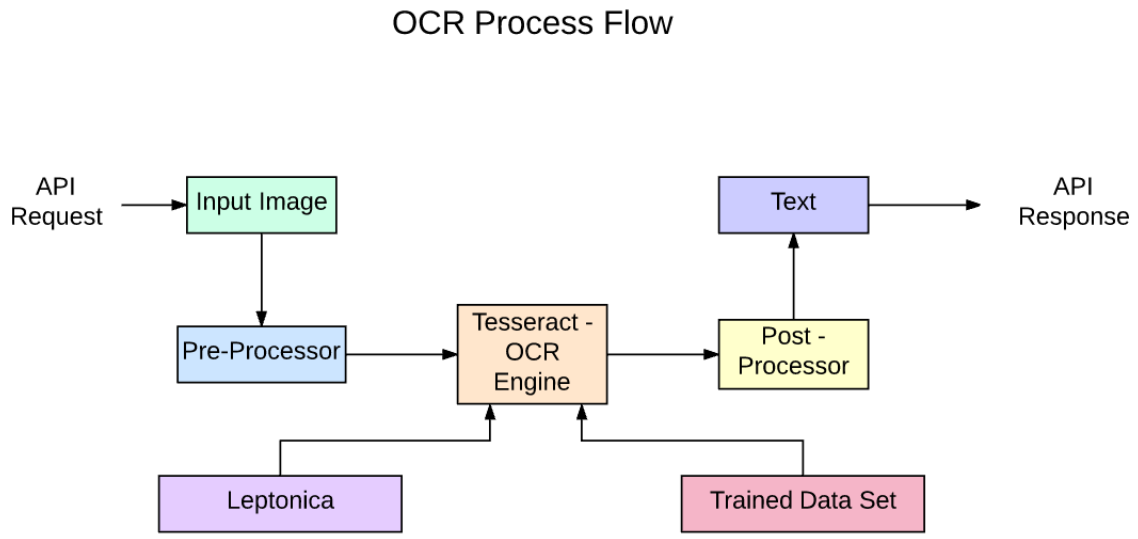
If you manage to do all of this, you have successfully made an OCR right from scratch.

Concepts:

Tesseract — is an optical character recognition engine with open-source code, this is the most popular and qualitative OCR-library.

Tesseract is finding templates in pixels, letters, words, and sentences. It uses two-step approach that calls adaptive recognition. It requires one data stage for character recognition, then the second stage to fulfil any letters, it was not insured in, by letters that can match the word or sentence context

If we look at a Tesseract OCR engine, here is the diagram of how it works



.

Deep Learning Techniques

EAST

EAST (Efficient accurate scene text detector) is a simple yet powerful approach for text detection. Using a specialized network.

Unlike the other methods, this is limited only to text detection (not actual recognition) however its robustness makes it worth mentioning.

Another advantage is that it was also added to open-CV library (from version 4)

The network is actually a version of the well-known U-Net, which good for detecting features that may vary in size. The underlying feed forward "stem" of this network may very—PVANet is used, however OpenCV implementation use Resnet. Obviously, it can be also pre-trained (with ImageNet etc.) . As in U-Net, features are extracted from different levels in the network.

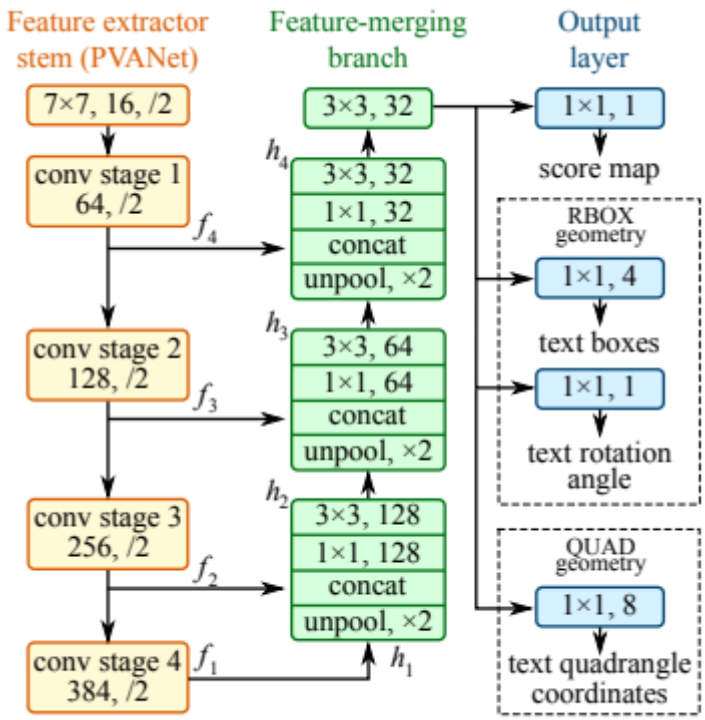


Figure 3. Structure of our text detection FCN.

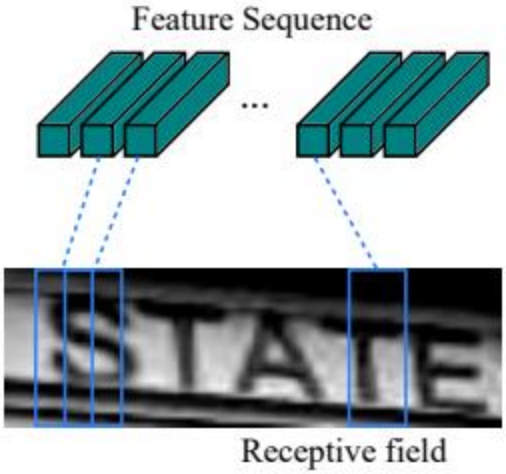
Finally, the network allows two types of outputs rotated bounding boxes: either a standard bounding box with a rotation angle (2X2+1 parameters) or "quadrangle" which is merely a rotated bounding box with coordinates of all vertices.

If real life results will be as in the above images, recognizing the texts will not take much of an effort. However, real life results are not perfect.

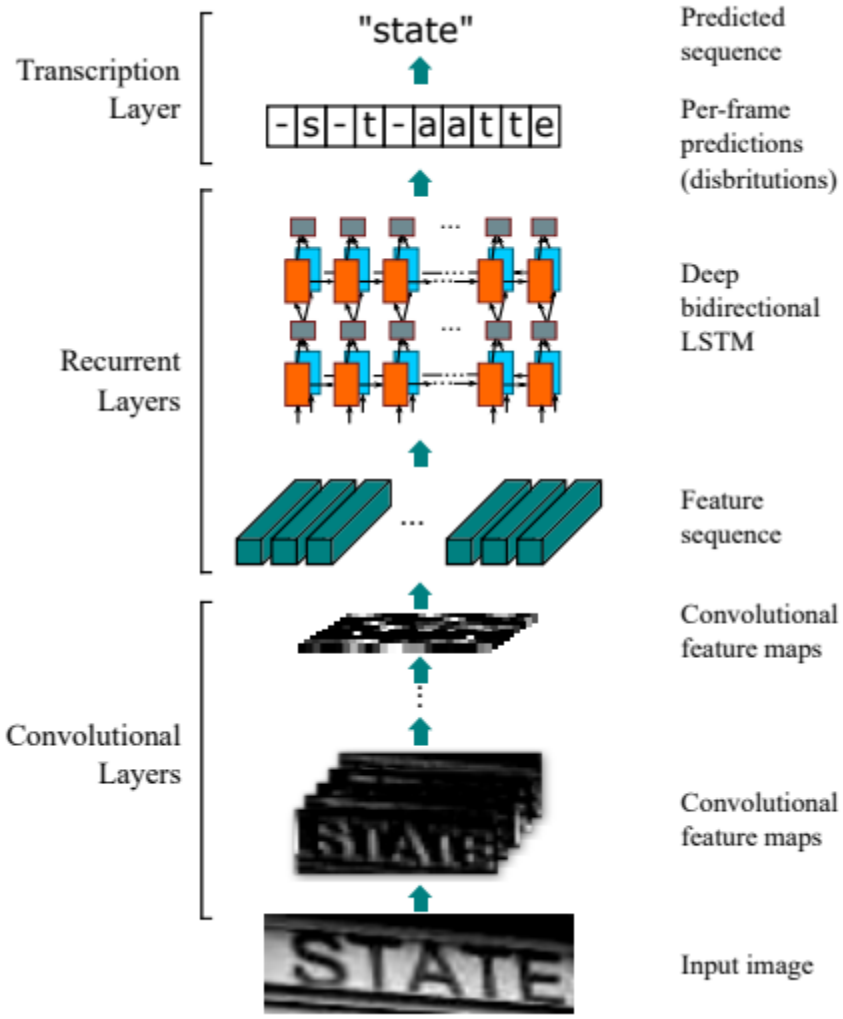
CRNN

Convolutional-recurrent neural network suggests a hybrid end to end architecture, that is intended to capture words, in a three step approach.

The idea goes as follows: the first level is a standard fully convolutional network. The last layer of the net is defined as feature layer and divided into “feature columns”. See in the image below how every such feature column is intended to represent a certain section in the text.



Afterwards, the feature columns are fed into a deep-bidirectional LSTM which outputs a sequence and is intended for finding relations between the characters.



Finally, the third part is a transcription layer. Its goal is to take the messy character sequence, in which some characters are redundant, and others are blank, and use probabilistic method to unify and make sense out of it.

This method is called CTC loss, and can be read about here. This layer can be used with/without predefined lexicon, which may facilitate predictions of words.

This paper reaches high (>95%) rates of accuracy with fixed text lexicon, and varying rates of success without it.

Let us start making our own:

1. Image Acquisition

Like any machine learning technique, the very first step is to collect the data for the font you are trying to read from image, pdf or any extension through OCR.

Suppose you are going to make a custom OCR for number plates or the following specific font of number plates. We will start by gathering all the data we can get. More the data, higher the accuracy (in most cases)

2. Pre-Processing

After you have gathered all the data, next step in to crop the part of the ROI, region of interest, which is in this case, the number plate or any text on the image.

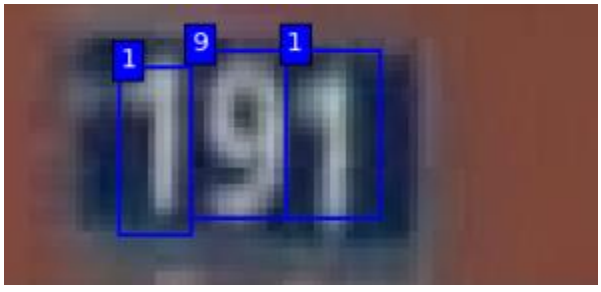
Pre-processing is as important as any other step as we are the ones who will tell the machine that which letter is in the picture or what number is at what position.

3. Segmentation

The next still will be to annotate the images so that our model can know where the target object is and learn itself, much like in neural networks or deep learning.

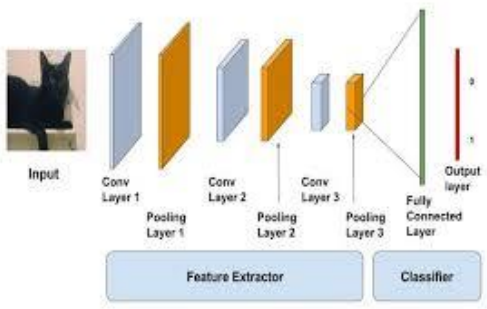
As mentioned above, if we make mistakes in this step, the machine will not be able to correct itself automatically.

For example, take the following example, if we tell the machine that number ‘8’ is actually number ‘3’ and train the machine such that every number ‘8’ is marked as ‘3’ by us, our final model will always classify ‘8’ as ‘3’, considering you manage to get the accuracy high.



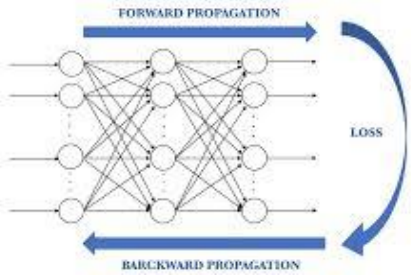
4. Pre-trained Model

After the data has been annotated, you will proceed with choosing any pretrained model of your choosing, few of them are mentioned above, but there are many variant pre-trained models and we can choose any of them depending on the trade-off between performance and speed.



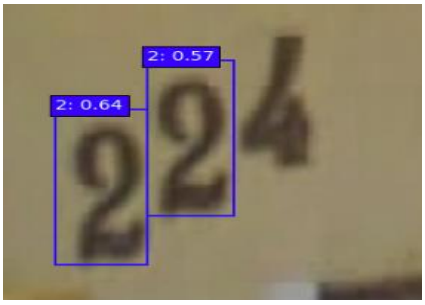
5. Training

Of course, when everything is set, then we tune the parameters a little and start the training process and go for a long break as this will take a lot of time.



6. Classification and Recognition

The last but not the least, testing your new custom-made OCR model on cropped images to find the accuracy of the model.




ANY
QUESTIONS???