**PROJECT LEVEL 2 (SOME CODING REQUIRED)**

**HIGH SCHOOL TRANSCRIPT READER**

**PROBLEM:**

**Background:**

A high school student’s transcript is one of the many items a university uses for its admissions criteria. Having this information in our app is needed for many purposes:

* To provide better college recommendations
* To further personalize your admissions predictor calculation for each college
* To surface other students similar to you and the colleges they are looking at

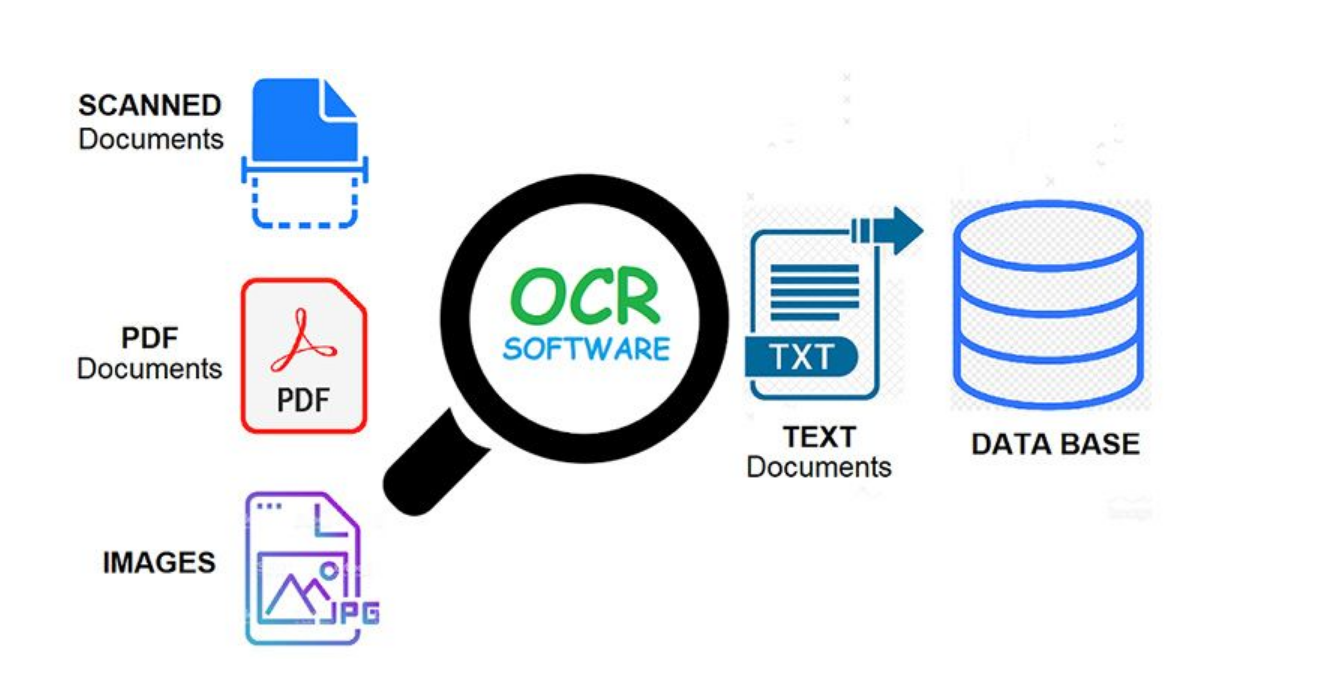
**Challenges:**

* Need a faster and easier way to read the transcript information from high school students
* The input format of transcripts is not standard.
* The transcript’s contents are also different among the high schools. Below is an example of a transcript.

**DELIVERABLE:**

The transcript will be provided as an image or as a pdf and we need a way to ‘read’ the text from the file and save it in some sort of data structure (an array, a json blob, etc.). Data results should be organized, by semester/year, subject and grade. Some possible avenues would be:

* Use an existing library that already reads text like python’s [tesseract](https://github.com/tesseract-ocr/tesseract)
* Use machine learning to create a model that recognizes the text and dumps it on a data structure



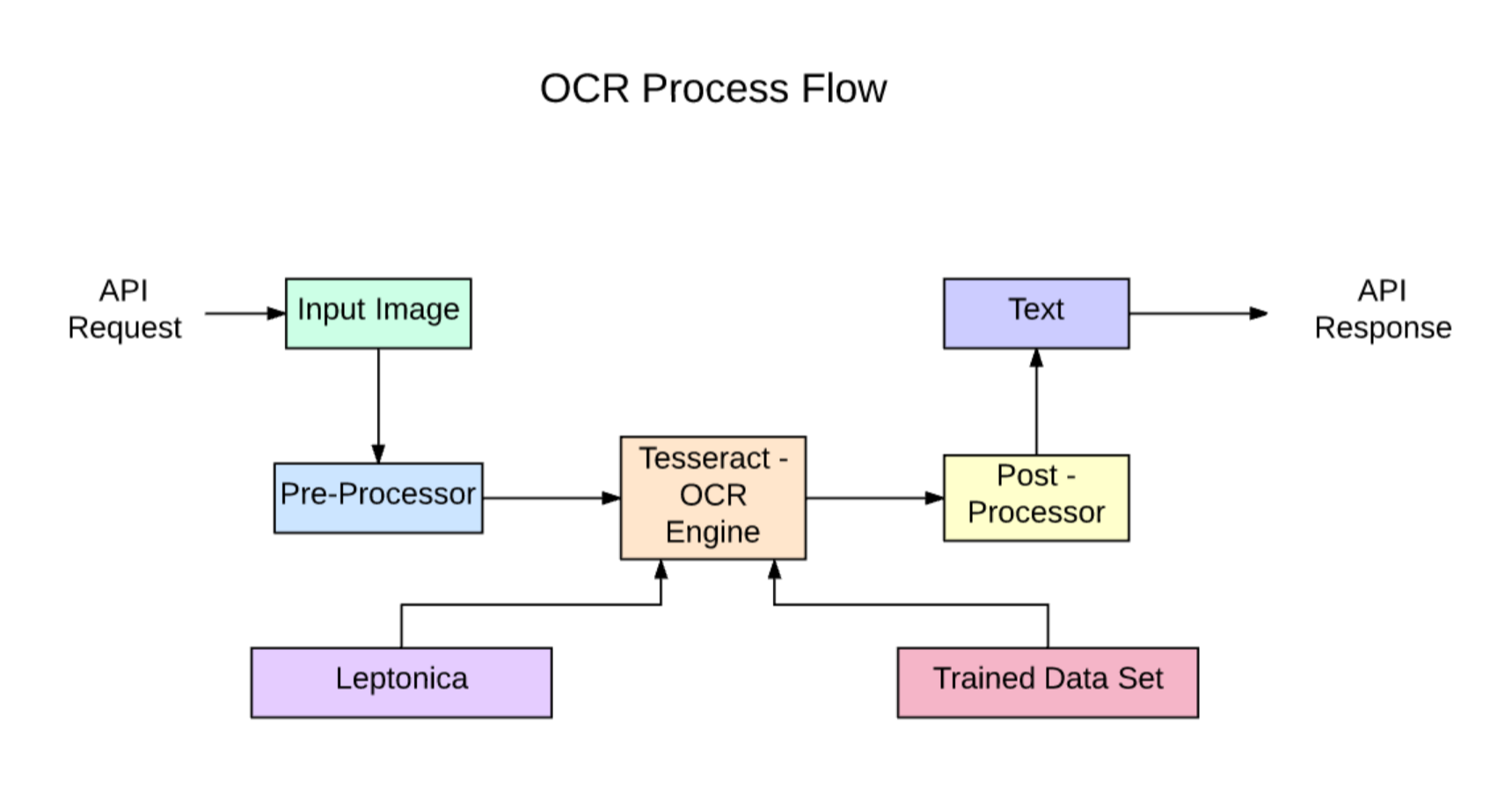
OCR = Optical Character Recognition. In other words, OCR systems transform a two-dimensional image of text, that could contain machine printed or handwritten text from its image representation into machine-readable text. OCR as a process generally consists of several sub-processes to perform as accurately as possible. The subprocesses are:

* Preprocessing of the Image
* Text Localization
* Character Segmentation
* Character Recognition
* Post Processing

Conventional OCR vs. Next-generational OCR

Next-generation OCR engines deal with these problems mentioned above really good by utilizing the latest research in the area of deep learning. By leveraging the combination of deep models and huge datasets publicly available, models achieve state-of-the-art accuracies on given tasks. Nowadays it is also possible to [generate synthetic data](https://github.com/Belval/TextRecognitionDataGenerator) with different fonts using generative adversarial networks and few other generative approaches.

How to implement OCR in python using the Tesseract engine.



Tesseract 4.00 includes a new neural network subsystem configured as a text line recognizer. It has its origins in [OCRopus' Python-based LSTM](https://github.com/tmbdev/ocropy" \o "Opens in a new window" \t "_blank) implementation but has been redesigned for Tesseract in C++.

The neural network system in Tesseract pre-dates TensorFlow but is compatible with it, as there is a network description language called Variable Graph Specification Language (VGSL), that is also available for TensorFlow.

To recognize an image containing a single character, we typically use a Convolutional Neural Network (CNN). Text of arbitrary length is a sequence of characters, and such problems are solved using RNNs and LSTM is a popular form of RNN. Read this post to learn more about [LSTM](http://colah.github.io/posts/2015-08-Understanding-LSTMs/).

We will be walking through the following modules:

* Tesseract OCR Features
* Preprocessing for OCR using OpenCV
* Running Tesseract with CLI and Python
* Limitations of Tesseract engine

Tesseract works best when there is a clean segmentation of the foreground text from the background. In practice, it can be extremely challenging to guarantee these types of setup. There are a variety of reasons you might not get good quality output from Tesseract like if the image has noise on the background. The better the image quality (size, contrast, lightning) the better the recognition result. It requires a bit of preprocessing to improve the OCR results, images need to be scaled appropriately, have as much image contrast as possible, and the text must be horizontally aligned. Tesseract OCR is quite powerful but does have the following limitations.

**Tesseract limitations summed in the list.**

* The OCR is not as accurate as some commercial solutions available to us.
* Doesn't do well with images affected by artifacts including partial occlusion, distorted perspective, and complex background.
* It is not capable of recognizing handwriting.
* It may find gibberish and report this as OCR output.
* If a document contains languages outside of those given in the -l LANG arguments, results may be poor.
* It is not always good at analyzing the natural reading order of documents. For example, it may fail to recognize that a document contains two columns, and may try to join text across columns.
* Poor quality scans may produce poor quality OCR.
* It does not expose information about what font family text belongs to.

<https://en.wikipedia.org/wiki/Comparison_of_optical_character_recognition_software>

<https://en.wikipedia.org/wiki/Tesseract_(software)>

Tesseract was considered one of the most accurate open-source OCR engines then available.

Tesseract's output will have very poor quality if the input images are not preprocessed to suit it.

Version 4 adds [LSTM](https://en.wikipedia.org/wiki/Long_short-term_memory) based OCR engine and models for many additional languages and scripts, bringing the total to 116 languages.

**LSTM:**

**Long short-term memory** (**LSTM**) is an artificial [recurrent neural network](https://en.wikipedia.org/wiki/Recurrent_neural_network) (RNN) architecture[[1]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-lstm1997-1) used in the field of [deep learning](https://en.wikipedia.org/wiki/Deep_learning). Unlike standard [feedforward neural networks](https://en.wikipedia.org/wiki/Feedforward_neural_network), LSTM has feedback connections.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

<https://en.wikipedia.org/wiki/Long_short-term_memory>

<https://github.com/tesseract-ocr>

**Technology - How it works**

LSTMs are great at learning sequences but slow down a lot when the number of states is too large. There are empirical results that suggest it is better to ask an LSTM to learn a long sequence than a short sequence of many classes.

Installation of Tesseract

<https://digi.bib.uni-mannheim.de/tesseract/>

**OCR with Pytesseract and OpenCV**

Pytesseract is a wrapper for 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 Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, and others.

<https://github.com/NanoNets/ocr-with-tesseract>

There is also one more important argument, OCR engine mode (oem). Tesseract 4 has two OCR engines — Legacy Tesseract engine and LSTM engine. There are four modes of operation chosen using the --oem option.  
0    Legacy engine only.  
1    Neural nets LSTM engine only.  
2    Legacy + LSTM engines.  
3    Default, based on what is available.

import pdf2image

try:

from PIL import Image

except ImportError:

import Image

import pytesseract

def pdf\_to\_img(pdf\_file):

return pdf2image.convert\_from\_path(pdf\_file)

def ocr\_core(file):

text = pytesseract.image\_to\_string(file)

return text

def print\_pages(pdf\_file):

images = pdf\_to\_img(pdf\_file)

for pg, img in enumerate(images):

print(ocr\_core(img))

print\_pages('sample.pdf')

<https://app.nanonets.com/#/keys?&utm_source=nanonets.com%2Fblog%2F&utm_medium=blog&utm_content=%5BTutorial%5D%20OCR%20in%20Python%20with%20Tesseract,%20OpenCV%20and%20Pytesseract>