Enhancing TEXT

Enriching LIVES

TEAM-3

Index

Solution Overview

With DigiCon, we're giving a soltution to this age old problem by providing a web and mobile interface that both pharmacists and patients can easily use to read doctor prescriptions.

Our solution works **for any image, in any orientation,** scanned even with a mobile camera. We **detect all bounding boxes** surrounding only the text in the image and satisfactorily return the parsed text that is **spell corrected with both English and medical vocabulary.**

Our system can **extract information regarding the medical aspects** mentioned in the text, the **exact dosages of the drugs prescribed, the names of the doctor, the hospital, date of prescription, along with other named entities** in the text. We let the end user view and download the corrected text in both image and pdf formats, **with the text either smartly overlaid on the original handwriting or on a fresh sheet** entirely.

DEMO : http://159.89.169.76:3000/

Abstract

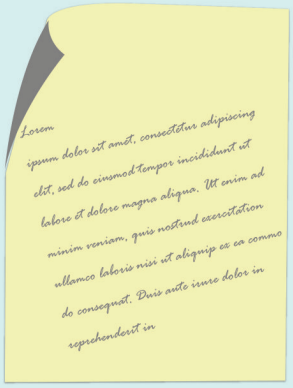
Medicines have become ubiquitous and happen to be one of the most important parts of our life. We get our medicines through doctors and most of the time, medicines are beneficial. But on occasions, they do harm the person taking them if wrong medicine or correct medicine in wrong dosage is taken.

From a recent study by the Institute of Medicine (IOM), sloppy handwritings of a doctor was found to be a major concern that causes the death of approximately 7000 people yearly in the US. It is a shocking statistic.

Many such errors result from unclear abbreviations, dosage indications and illegible writing on some of the 3.2 billion prescriptions in US per annum.

With the above-mentioned statistics at hand, we can remark "Thousands of people are dying although we have technology today to prevent these errors. Can this technology help us to reduce such medical errors?". As a step to mitigate the errors, we propose an approach called DigiCon as an initiative of digital convergence and we call it as "Digitization of Doctors' Handwritten Prescriptions", which we have detailed in the solution overview section above.

Pipeline



Pre - processing

OCR

Google Cloud Vision API

Azure Vision API

Entity Extraction

Crude NER

Draw boundary boxes

Fix Orientation

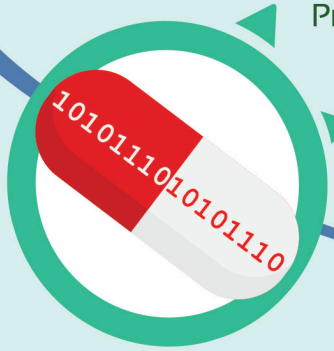
Fix Spelling

- Character level
- Sentence level

Extract Medical Data

Generate Digital Prescription

- Corrected
- Fresh



Get dosage data

Paracetamol	500g	10
Albendazole	10g	14
Metacin	70g	15
Acetaminophen	150g	5
Hydrocodone	100g	7
Azithromycin	50g	1

Create PDFs

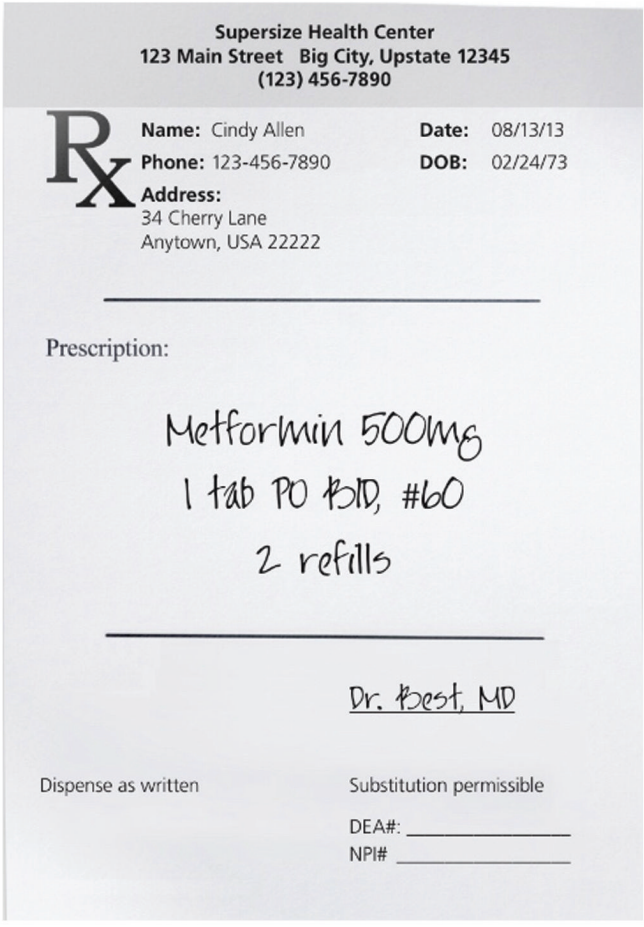
Pre-processing

Motivation

A prescription provided as an image may have a lot of noise and contrast issues due to which the results could get affected. This calls for the image to be processed and cleaned before it is passed for character recognition.

Overview

Below is a an example prescription passed through the processes. Any bleed through the paper or paper defects is also compensated for in the processes.



Supersize Health Center
123 Main Street Big City, Upstate 12345
(123) 456-7890

Rx

Name: Cindy Allen **Date:** 08/13/13
Phone: 123-456-7890 **DOB:** 02/24/73
Address:
34 Cherry Lane
Anytown, USA 22222

Prescription:

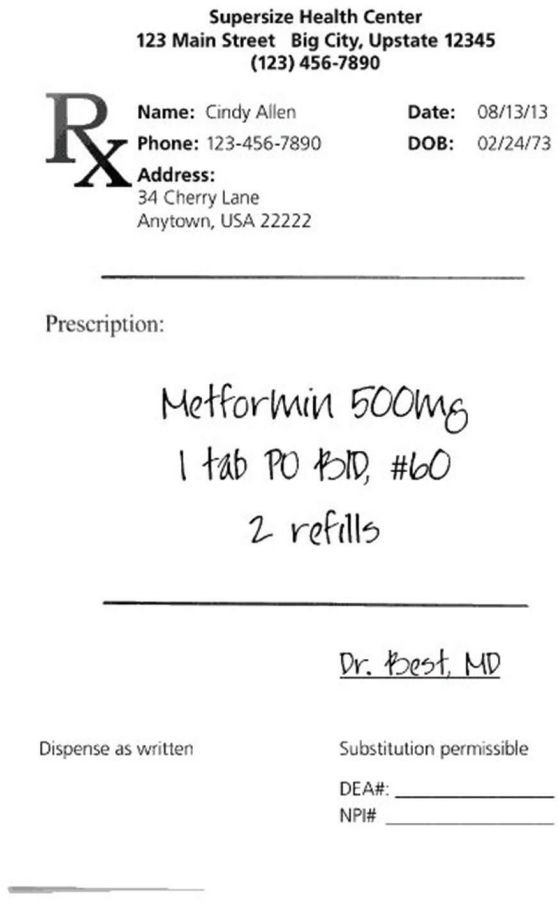
Metformin 500mg
1 tab PO BID, #60
2 refills

Dr. Best, MD

Dispense as written Substitution permissible

DEA#: _____
NPI# _____

Input



Supersize Health Center
123 Main Street Big City, Upstate 12345
(123) 456-7890

Rx

Name: Cindy Allen **Date:** 08/13/13
Phone: 123-456-7890 **DOB:** 02/24/73
Address:
34 Cherry Lane
Anytown, USA 22222

Prescription:

Metformin 500mg
1 tab PO BID, #60
2 refills

Dr. Best, MD

Dispense as written Substitution permissible

DEA#: _____
NPI# _____

Output

Processes

Below is the pathway that the algorithm follows:

1. Identity the background color of the original scanned image.
2. Isolate the foreground by thresholding on difference from background color.
3. Convert to an indexed color PNG by choosing a small number of “representative colors” from the foreground.

Parsing Background Color

Since the majority of the page is free from text, we might expect the paper color to be the one that appears most frequently in the scanned image but random variations in color can appear due to dust specks and smudges on the table, color variations of the page itself, sensor noise, etc. So in reality, the “page color” can spread across thousands of distinct RGB values.

Essentially, we solve this by reducing the bit depth, by grouping similar pixels into larger “bins”, which makes it easier to find a strong peak in the data. There’s a tradeoff here between reliability and precision: small bins enable finer distinctions of color, but bigger bins are much more robust.

Isolating the foreground

Once we have identified the background color, we can threshold the image according to how similar each pixel in the image is to it. For this purpose we use HSV space, which deforms the RGB cube into the cylindrical shape:

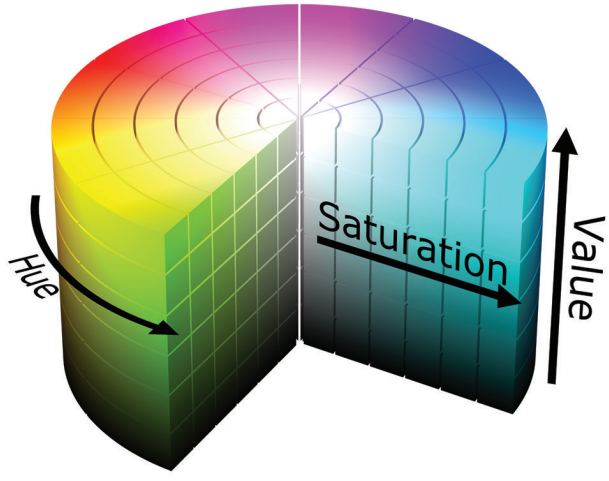
Color Palette Quantisation

In this part, we’ll be solving a color quantization problem (which is itself just a special case of vector quantization), through the use of cluster analysis. The particular methodological tool for the job that we picked is k-means clustering. Its overall goal is to find a set of means or centers which minimizes the average distance from each point to the nearest center.

Contrast Enhancement

The preprocessing also increases the vividness and contrast of the final palette by rescaling the minimum and maximum intensity values to 0 and 255, respectively. Without this adjustment, the 8-color palette for the scan above would look like this:

The adjusted palette is more vibrant:



Intelligent character recognition

We call this phase of the pipeline “Intelligent Character Recognition (ICR)” as this is an advanced Optical Character Recognition (OCR), or – rather more specific – handwriting recognition system that allows for context specific mistakes and intelligently corrects them.

Remove Text from bounding box

This operation is essential before the text inside the bounding boxes are replaced with the text detected by the OCR API and corrected using spellcheck and lexigram.

This is achieved by cropping out that part of the image containing the relevant bounding box using 4 sets of coordinates and using cv2.dilate function on the resulting image for a suitable number of iterations. This operation also helps in maintaining the background colour of the bounding box.

State of the art OCR

Generic ‘OCR for handwritings’ is not a task we could take up and complete in a couple of weeks independently and beat existing work, so we started by going out and looking at the current state-of-the-art models for OCR. Specifically, OCR that has been trained on hand written images. We found a lot of great work like RNNLIB and recent student efforts like ml_HR. But from our experiments, the industry giants Google and Microsoft had leading work in the field with state of the art results and provided access to their models through their APIs.

Google Cloud Vision API

Google Cloud Vision API is a great OCR tool, which is expectedly running on Google’s powerful datacenters and is thus pretty fast and accurate. Their ‘Documentation text detection’ endpoint is specifically meant for documents with text and returns a formatted protobuf result that returns the parsed text in several layers of hierarchy (page, block, paragraph, word and character), along with the bounding box where it found them. The text is contained only at the character level.

We created our own classes to store all the data:

```
class coordinate:
   x = 0
   y = 0

class boundingBox:
   bound_text = ''
   box_type = ''
   tl = coordinate(0, 0)
   tr = coordinate(0, 0)
   br = coordinate(0, 0)
   bl = coordinate(0, 0)
   bb_children = []
   lexi_type = ''
   lexi_label = ''
   dosage = {}
   language = 'en'
```

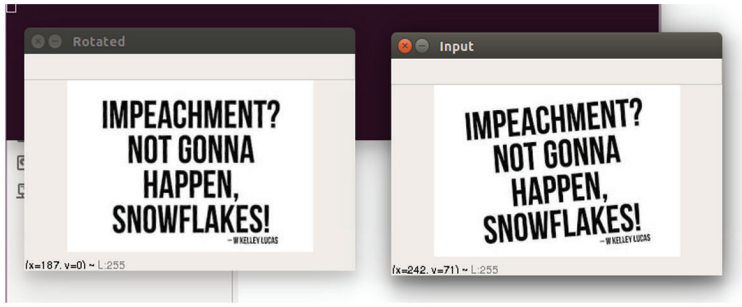
Microsoft Azure Vision API

Microsoft’s API responds similar to Google’s, but is less feature rich and has 2 limitations: It doesn’t detect cases with multiple languages in the same image and also doesn’t work with rotated images. But we still have it in our pipeline as a backup, in case Google’s models ever have an issue.

Orientation detection

We define 2 levels of orientation detection in our pipeline:

1. Images which are minorly titled (< 30 degrees)
2. Images which are completely rotated in a different direction (i.e. multiples of 90 degrees)



Images with minor tilt

This class of images are further dealt in 2 categories:

1. Images inclined on the right by an acute angle
2. Images inclined on the left by an acute angle

For determining the skew angle of the text in the image, cv2.minAreaRect function is used which for (a) returns angle with the vertical but the negative value and for (b) returns angle with the horizontal and this value is negative as well. Appropriate changes are made accordingly.

However, before applying this function the image is converted to grayscale and the foreground and background are flipped i.e. foreground is made white and background black to compute the coordinates of the rotated bounding box containing the text.

To rotate the skewed images we use the cv2.getRotationMatrix2D function which given an image and angle of rotation rotates it in the anti-clockwise direction.

Images rotated by right angles

Again, for this case, we have 2 different heuristics, but use only one of them in the interest of time.

If our OCR was done with Google Cloud Vision API, we then get the bounding boxes with their corners oriented in the same direction as the detected text. So, we average out the angle at which the bounding boxes are directed for 10 randomly selected bounding boxes and then use the result to detect which direction the main text is all pointing towards and consider that to be the top side.

When using a different model, we use Hough transform to detect the general direction of the near straight lines in the image. This can help us tell between images rotated by 90 or 270 degrees and images rotated by 0 or 180 degrees. If the former, then we rotate the image once in both directions, check with Azure's API and pass the results to get a quick edit distance run for the words in the text and use the one with the lesser edit distance.

Spelling correction

Yet again, we have 2 levels in this part of the pipeline: Sentence level spelling correction and word level. For word level, we implemented our much improved version of Russel Norwig's brilliant blog post on 'How to write a Spelling Corrector'.

Custom word level spell check (Hard enforcing)

This is a custom spell check, which tries to correct the word not based on the context of the sentence, but on the frequency of the occurrence of the word in the source dictionary. We have manually created 2 dictionaries: 1 based on the English dictionary and 1 on a publicly available dictionary of common medical terminology.

Since this will always try to fit a given word within the words in the dictionary, we'd like to call this a "Hard enforcing" spellcheck. Initially, the given word undergoes some preprocessing. It converts the entire word to lower, If there are more than 2 numbers in the whole word, it returns the number as it is. This is so as to prevent the spell check from messing up the dosages and other numbers such as mobile numbers, pin codes, etc.
If the word has less than 2 numbers, it will convert the numbers to their visually similar counterpart alphabets.

After the processing, if the given word is already present in the dictionary (medical + english), the function returns the same word. If not, the spell check tries to match the incorrect word to its most suitable correction in a series of steps.

Here comes the idea of **edit distance**. Let's see what edit distance 1 consists of.

Deletes: All words after removing 1 letter. For example, if the given word is "tsry", it generates a list consisting of "sry", "try", "tsy", "tsr".

Transposes: All words with their adjacent characters exchanged. If the given word is "bor", it generates a list of words containing "obr", "bro" as it's elements.

Insertions: All combinations with a letter inserted at each possible position. For example, if the given word is "ht", the program generates 78 elements in the list "'a-z'ht"(26), "h'a-z't"(26), "ht'a-z'"(26)

Replacements: Here, we consider all the words that are generated by replacing one letter with another possible character. In this context, we give way more importance to visually similar characters, since an OCR is more likely to confuse between two visually similar characters than between ones that are not. For example, "a" to "o" is more likely than "a" to "t" or "x". Thus, this assumption is reasonable. So, we generate a list of words, with each letter of the original word replaced by a visually similar character. To achieve this, we use a dictionary in python. In the dictionary, each letter in the dictionary, maps to a list of characters it is visually similar to.

```
dicts['a'] = list('bdou')
dicts['b'] = list('hlt')
dicts['c'] = list('eo')
dicts['d'] = list('ao')
dicts['e'] = list('c')
dicts['f'] = list('l')
dicts['g'] = list('pqy')
dicts['h'] = list('tbn')
...
dicts['9'] = 'g'
dicts['('] = 'c'
dicts['5'] = 's'
dicts['1'] = 'li'
dicts['0'] = '0'
```

Now we merge all the generated words to a single list. This is edit distance 1. By this logic, edit distance 2 is applying the steps of edit distance 1 to every word generated in the edit distance 1 algorithm. Similarly, edit distance 3 applies edit distance 1 to all the words generated by the edit distance 2 algorithm. As we can see, with each edit distance, the words generated increases exponentially (and thus the processing time), we tried to cap the edit distance to 3 (an optimal midway mark).

For words upto word length 3, 1 edit distance is considered. For words upto word length 6, edit distance of 2 is considered and for the rest, edit distance 3 is considered.

Then among the generated lists, the words which are present in the source dictionary (english + med), are kept and the rest are filtered. Then among all these possible words, the word with the maximum occurrence in the dictionary is considered as the most suitable match for the given word.

Sentence level spell check with Azure (Soft enforcing)

The Azure SpellCheck API has the ability to read a given sentence and analyze the required changes in words, based on possible spelling / grammatical checks. As the Azure SpellCheck API doesn't necessarily try to fit the given words into the vocabulary and gives suggestions with sentence-based context, we'd like to call this a "Soft enforcing" spellcheck.

We use Azure's API at a sentence level since it can correct sentences with more contextual information and can also detect examples where a single word is broken into multiple parts.
For example, "Hw are yu? Isn't thi s cool!" gets converted to "How are you? Isn't this cool!"

Name

We use NLTK for the purpose of recognising names present in the prescription. NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

The process of extracting names uses contextual data along with dictionary to figure out if a word is a name. Suppose, in "Metha wants to give Lata her keys", "Metha" and "Lata" are keys. The module identifies the context using Natural Language Processes and hence segregates the names and provides them as a list.

Metadata Recognition

Name

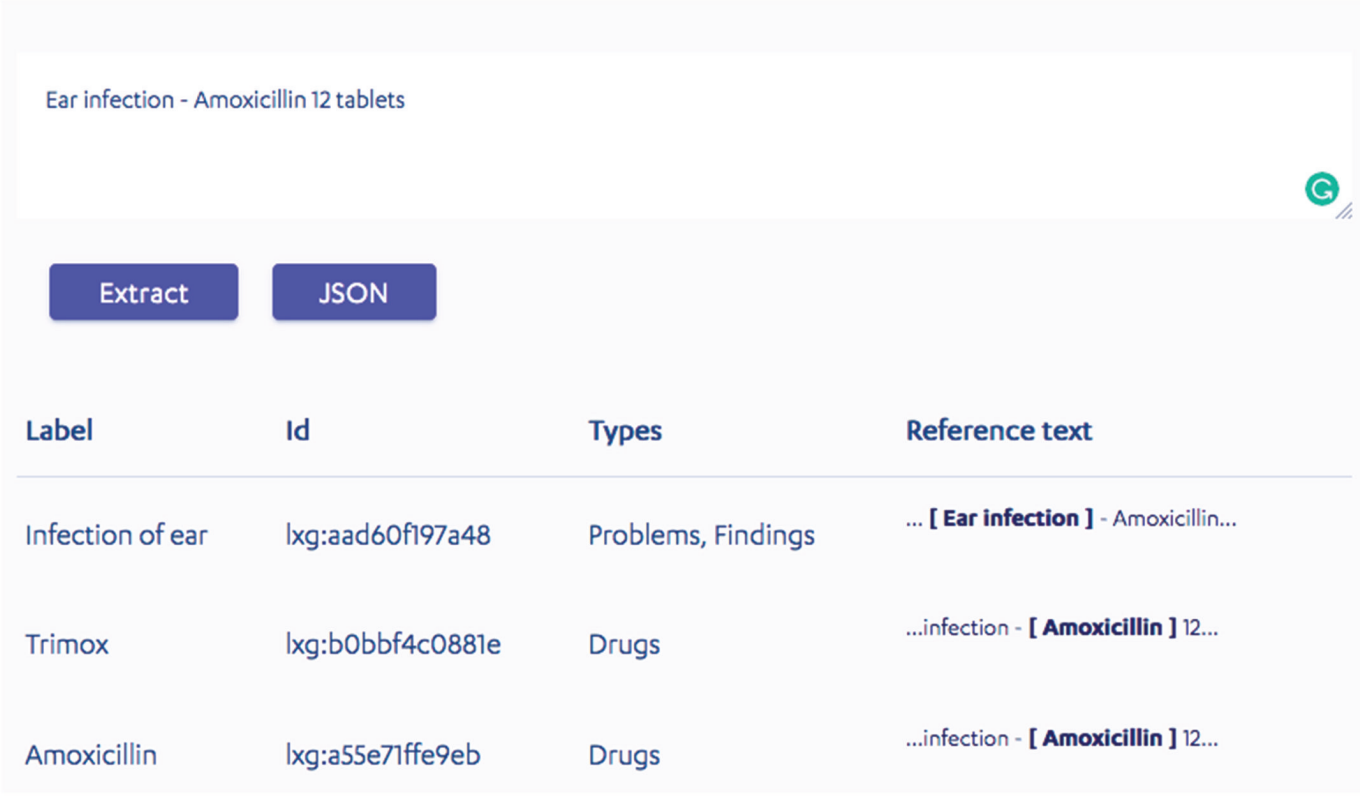
We use NLTK for the purpose of recognising names present in the prescription. NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

The process of extracting names uses contextual data along with dictionary to figure out if a word is a name.

Suppose, in “Metha wants to give Lata her keys”, “Metha” and “Lata” are keys. The module identifies the context using Natural Language Processes and hence segregates the names and provides them as a list.

Dosage and Medical Condition

For this task, we use the pre-existing Lexigram API to get medical-related insights. The extraction API endpoint provides data about the dosage name, and metadata about the medical condition (illness, if any).



Detecting useful metadata

Our task here was very specific and focused:

- Identifying Hospital/Clinic and address, if present
- Contact details mentioned on prescription, if any
- Identifying Doctor's name and specialisation, if any

The major challenge here was that a prescription sheet may or may not contain one or all of following :

- Name of the doctor
- Clinic/Hospital where the medical prescription was issued
- Location/address of the hospital/clinic visited
- Specialization of the doctor
- Phone number and email address of the medical institution visited

A prescription has no fixed format to be followed, it can't be simply extracted by using heuristics considering name of hospital/clinic on top, then address, doctor's name and specialization which although is common but not necessary always the case. The text can't be just distinguished using Natural Language Processing as these are not in complete sentences, but as phrases and stand alone words. **So the task wasn't achievable by merely applying geometric and linguistic heuristics.** The name of hospital could be any combination of Proper and common nouns or even adjectives (Eg: Gentle Care Clinic), so training the machine by feeding data was also ruled out.

After this analysis it was obvious that the task completion would be achieved by a combination of Natural language Processing and Image processing concepts. The NLP Tool used is the Stanford CoreNLP.

Stanford CoreNLP provides a set of human language technology tools. It can **tokenize, identify Part of Sentence(POS), Named Entity Recognition(NER)**, Parsing, Lemma etc., mark up the structure of sentences in terms of phrases and syntactic dependencies, indicate which noun phrases refer to the same entities, indicate sentiment and so on. However only the first three were needed for the task.

Next challenge was the **time consumption of Named Entity Recognition**. This function has a peculiar nature as its processing time didn't depend on the size of the input. Owing to this ,its use had to be minimised .It identifies objects and put them under tags like 'PERSON', 'OBJECT'(o),'NUMBER' and so on. Thus its used had to be minimised. Another problem faced by using it was that NER functions are written to take sentences as input and not phrases or blocks of text identified. As in a medical prescription, **the words detected are never in a (proper) sentence**, so identifying the tags and working of NER was important to completing this task. The attempt was made to reduce the use of this and increase that of the less time consuming Parts_of_Sentence(POS) tag. The POS tag isn't as advanced as NER but sometimes it is prefered as it gets the job done at cost of less time consumption.

Another factor to take into consideration was that this step is executed at the last, so the faults and imperfections of detecting regions of interest, inclined bounding boxes, fault in Optical Character Reader (OCR) and limits of rectifying words by spellcheck and Medic_Vocab had to be all kept in mind. Even though this step does not rectify the misidentifications, but it had to make sure that it does not compounds the mistakes.

Detecting contact details:-

This was simplest of the three subtasks, but some considerations had to be made. Although the phone_no could be detected as set of numbers and email of format xyz@abc.com, but here it was preferred to use NER as it tags a token as 'NUMBER' and 'EMAIL' as the NER alloted tags were used later.The entire text was run for NER as it saved time rather than running it for individuals. Also it is more flexible than the former approach mentioned. Here, some considerations had to be kept in mind if not taken care of could produce wrong outputs, for example length of number string could be used to differentiate the phone numbers from other places where numbers were used example (Dosage, patients age, etc.).Then the NER list was searched for tags 'NUMBER' and 'EMAIL' and were assigned to their corresponding variables.

Detecting hospital details:

This was restricted to top 30% of the page input, setting it as threshold as it is common in most of prescriptions. Detecting hospital/clinic name was approached by the fact that if it is present, and accompanied by the set of words... hospital, clinic, center, etc. and location is nearby it. It is not always on the top as there may be prescription number, stray mark or not even present, hence getting it by heuristically extracting the text in topmost box. on the medical prescription. Once we identify whether the box has the name of hospital, an issue raised up was that what part of a bounding_box was name of hospital. NER wasn't the best solution for this as it may be any combination of nouns, adjectives... However if comma or fullstop or dashes appear then it may signify end of the name and beginning of address or contact details or another box.

The address was again a very big challenge as there was no exact tag for address, it could be either under tag of 'CITY', 'STATE' or 'LOCATION' tag under NER. The address details were limited to bounding boxes immediately after the hospital's name. Next step was to identify what part of box is address and which one is the miscellaneous details following like telephone number, email-id etc. The best possible solution seemed to be to consider the POS identified as 'NNP' (Noun Phrase) or 'NN' tags, and the corresponding text is taken as address.

Detecting name of doctor and specialization:

The name of the doctor is accompanied by a prefix of Dr. and the bounding text detected with the prefix contains the name of doctor. This box may already contain specialization, qualification and other details. So only the part of the text of the bounding box identified as a 'PERSON' tag by NER as well as 'proper noun phrase' by part_of_sentence tells us the name of the doctor. The latter was necessary because sometimes OCR detects small letter instead of block and NER wont classify it as 'PERSON' tag. NER tags the specialization as 'TITLE' but it is not exhaustive as it classified for example gynaecologist as 'OBJECT' (O). So an exhaustive list of specialization was used to identify specialization of doctor.

Generating Output

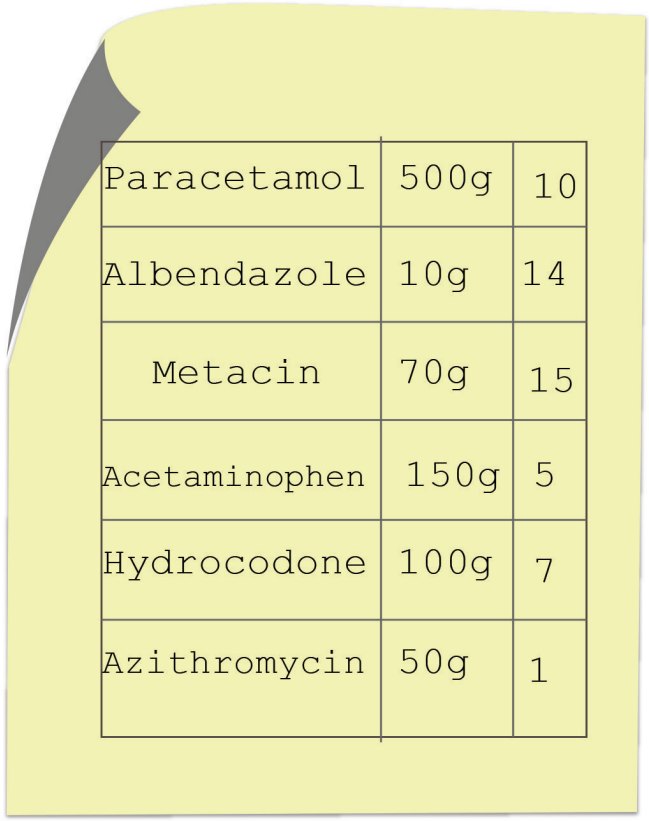
Replacing Doctor's handwriting with text

As we now have the coordinates of the bounding boxes of text and the spell-checked text present in the bounding boxes, it becomes possible to remove the boxes with doctor's handwriting and replace it with typed text for more readability.

To do this, we apply erosion on the area inside the bounding boxes to remove only the text from the image without disturbing the color of the background.

Then, to write back to the image, we first detect the orientation in which the bounding box is. This is because most often, the bounding boxes aren't parallel to the image dimensions.But the challenge here was that it is not straightforward to write back text to the original image in a titled direction using either OpenCV or PIL. So instead, we first create a transparent image of the same dimension as the bounding boxes and write text to it, which is then finally 'pasted' back over the original image.

Another challenge here was that the font size varies all over the image, and so do the size of the bounding boxes.



Paracetamol	500g	10
Albendazole	10g	14
Metacin	70g	15
Acetaminophen	150g	5
Hydrocodone	100g	7
Azithromycin	50g	1

So keeping the font size constant while writing back was no option. So we dynamically calculate the font size for all the bounding boxes in the image by smartly calculating the length of the text to be written within the bounding box and adjusting the font size to fit inside the box.

Installation

Docker

Docker is a computer program that performs operating-system-level virtualization also known as containerization. We use Docker to build and run DigiCon. Docker can be easily installed by executing the following script.

```
curl -fsSL https://get.docker.com | sh\
```

Docker should now be installed. If working behind a proxy, follow the steps to setup your proxy environment for Docker.

- Create a systemd drop-in directory for the docker service:

```
sudo mkdir -p /etc/systemd/system/
docker.service.d
```

- Create a file called /etc/systemd/system/docker.service.d/http-proxy.conf that adds the HTTP_PROXY environment variable

```
[Service]
Environment=
"HTTP_PROXY=http://proxy.com:port/"
```

- Flush changes

```
sudo systemctl daemon-reload
```

- Restart Docker

```
sudo systemctl restart docker
```

Proxy should be now set for Docker. For further details visit
https://docs.docker.com/config/daemon/systemd/#httphttps-proxy

ENV files

Some components of the pipeline use external services, such as the Vision & Text API. In such cases, a secret key needs to be obtained from the service, and stored in an environment file / .env file. These secret keys cannot be shared on the repository in public, as they can be misused by anyone.

Running Everything

- Install Docker-compose

```
sudo apt-get install -y docker-compose
```

In case of error,

```
Cannot connect to the Docker daemon at
unix:///var/run/docker.sock. Is the docker
daemon running?
```

Run

```
sudo dockerd
```

- Run the docker-compose from the main directory

```
sudo docker-compose up
```

Interface

Website

The website is built on top of React, and uses Yarn for dependency management. The web interface allows the user to upload a scanned prescription of a doctor, processes the image in the back-end and shows the processed image again to the user.

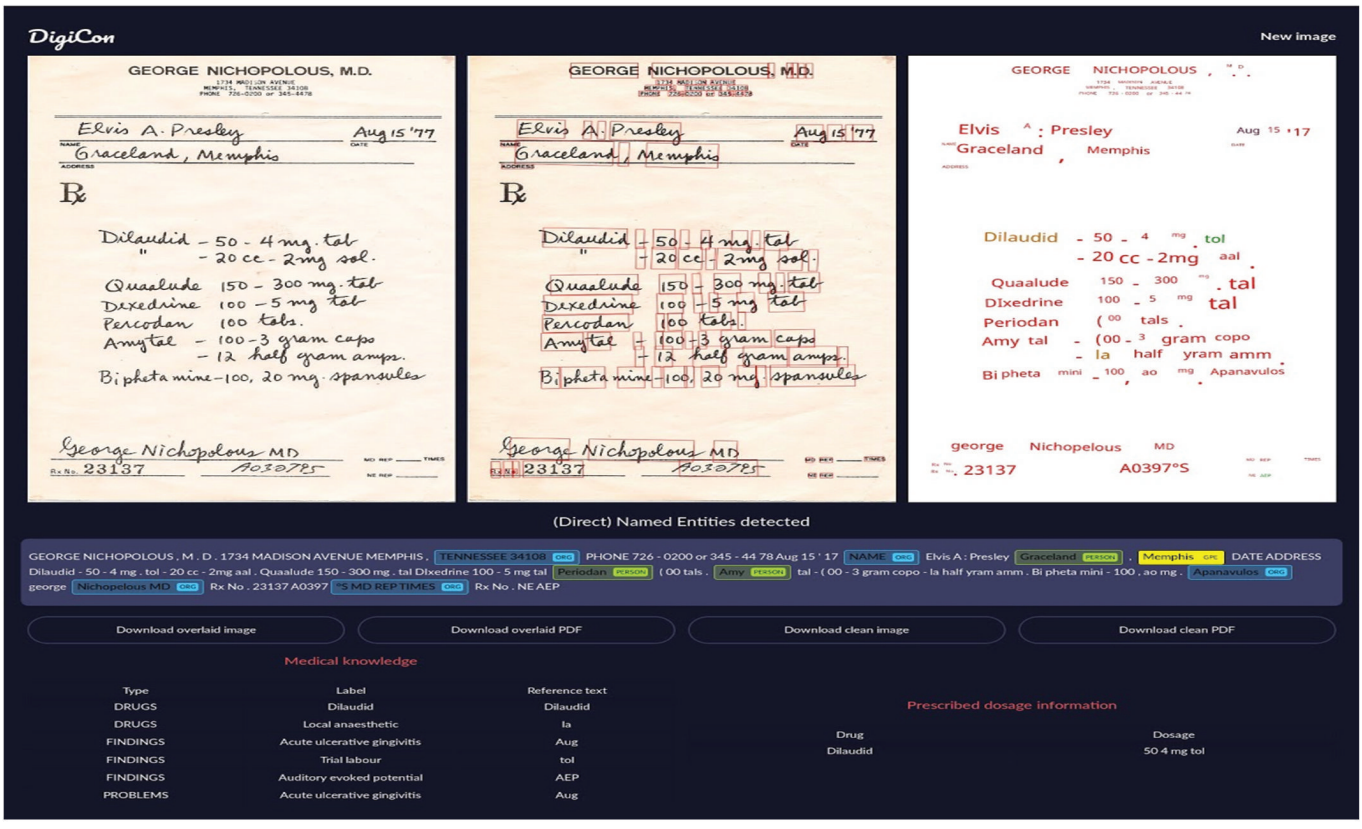
Dynamic updates

The app uses sockets to establish a bi-directional communication channel with the server. By this, the server emits status of the recognition process at every checkpoints. The app catches the messages emitted by the server and displays the updates respectively. With this, we can know at what stage the process is.

Team 3

digicon

It is hard to change a doctor's handwriting. But it shouldn't be hard to read them.



References

- Custom - Spellcheck : https://norvig.com/spell-correct.html
- Preprocessing : https://github.com/mzucker/noteshrink/
- Azure-vision-API : https://azure.microsoft.com/en-in/services/cognitive-services/computer-vision/
- Google-vision-API : https://cloud.google.com/vision/
- Azure - Spellcheck : https://azure.microsoft.com/en-in/services/cognitive-services/spell-check/
- Lexigram : https://www.lexigram.io/
- NLTK : https://www.nltk.org/
- Stanford - CoreNLP : https://stanfordnlp.github.io/CoreNLP/