Implementing Image-To-Text Engine With Usage Of Database And Google API

Rimma Niyazova

ABSTRACT

Every person has used some kind of medicine or will use some kind of medicine in the future and although pharma industry is generally well established and has been around for many decades, some technological improvements would make drug consumption more accurate and less dangerous. In this project we utilize a relatively new approach which can bring such improvement to the industry. By taking a photo of a single pill, our framework is able to identify the name of the pill, its ingredients and possibly more attributes. This paper shows that after using our framework for some time, the precision of the pill identification goes gradually up due to the number of photos being uploaded by the users.

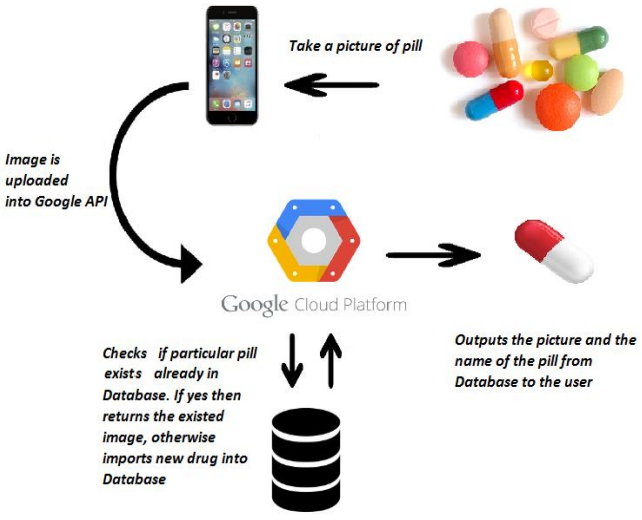
Index Terms — machine learning, multimedia, image retrieval, google API, drug, medicine, graph database, Neo4j, python.

1. INTRODUCTION

Before coming up with our idea, we invested a great thought in how it can be helpful to the society. We didn't want to make something redundant that already existed on the market. Given the recent growth in drug abuse in New York (and other states) we decided to address that issue in our project in a form of somehow controls drug consumption and prevents people from taking drugs they didn't need or were not prescribed with. We strongly believe that drug consumption can be crucial in everyone's day to day life and may affect one's life if not consumed with great care. We also believe that our idea can truly solve that issue or at least mitigate it to an extent. At first we thought that there may have already been something similar on the market and were very surprised to discover that such products didn't exist and that we would be the first ones to tackle that problem with technology and imagination. We searched on the internet for alternative solutions and were able to find several websites that provide great results but in our opinions they required much more steps to achieve the same results our app achieves.

2. PREVIEW THE SYSTEM

The diagram below shows the logic steps of our idea. It starts with the user uploading a photo of a single or multiple pills. In the second step the uploaded photo is passed to Google Vision API as a request in a form of image file. Third step, Google Vision API returns a response in a form of JSON file. That JSON file includes multiple fields with attributes that correspond to the pill image. At this step our Python code selects the first 2 results from the webEntity field and saves them as potential names for the pill. Next step we query our database by retrieved names. If our database already has a pill node with that name, we return this image with user's uploaded image for comparison. If user accepts result the program will return the name for this pill. Otherwise the program will return next associated image. If our database doesn't have any matchings it will display a sorry message. In any case we always store the uploaded image to expand our database.



3. REVIEW OF RELATED WORK

We have examined several websites such as Drugs.com and WebMD.com. In most cases these websites achieved some very satisfying results. Drugs.com website requires the user to input 3 fields in order to show results. Those fields are color, shape and the text on the pill (not always present).

WebMD.com operates in the same fashion.

For example we tested a pill called “Ritalin” on both website.

We inserted in the text field “CIBA 7”, in the color “Yellow” and in shape “Round”. The results were the following:

WebMD.com returned 1 images(figure 2).

Drugs.com returned 6 images but only 2 had the same name as WebMD.com(figure 3).

Different pictures showed different results on these sites. There were requests when one of the sites didn’t show any result.

Figure 2: Return result from WebMD.com

Figure 3: Return result from drugs.com

4. TECHNICAL DESCRIPTION

Python: We used Python version 2 for its simplicity and ease of use. Python requires very little code to integrate with various APIs, is light to run on almost every machine and relatively easy to debug. Our Python code has 4 functions.

Functions:

fix_url(URL) - a function that takes a URL as an argument, replaces some chars in it and returns a fixed URL.

annotate(path) - a function that takes a file path as an argument, makes a request to google vision API and returns web annotations that were returned by google as a response.

def show_result(url1, url2) - a function that takes 2 database image returns as argument, creates a UI pop up window and shows the 2 database image returns in that window.

report(annotations) - a function that takes a JSON object as an argument, parses it, makes calls to Neo4j database in order to store and/or retrieve images and data. Returns void.

Google Vision API: We tested several APIs for image recognition and came to a conclusion that Google provides the best accuracy and speed. Google makes its API very friendly and easy to use especially with Python. In order to be able to use Google’s service we had to create and specifically enable Google vision API. We then had to authenticate our client with the private key that we received from Google. We stored that key in a JSON file in the same location where we held our application. Once the Auth process was finalized we were able to make API calls with Google and receive Google responses accordingly. Below is an example of what Google response may look like.

Figure 4: example of return from Google vision API in a form of JSON file.

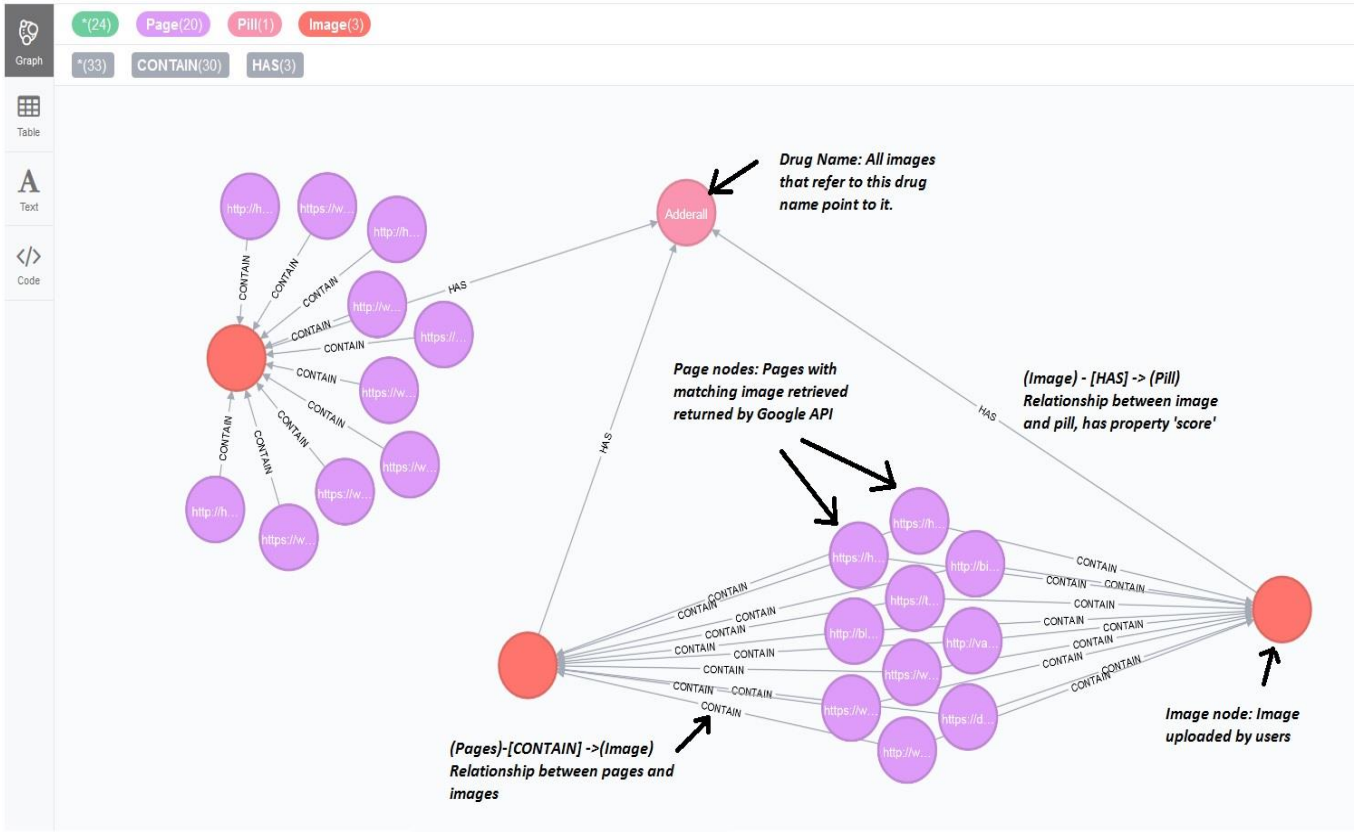


Figure 5: Return result from Neo4J database

```
$ MATCH (n:Pill { name: "Quinapril"})<-[:HAS]-(k:Image) RETURN k as Users_pictures_for_Quinapril
```

ou are currently not signed into Neo4j Browser Sync. Connect through a simple social sign-in to get started.

```
MATCH (n:Pill { name: "Quinapril"})<-[:HAS]-(k:Image) RETURN k as Users_pictures_for_Quinapril
```

"Users_pictures_for_Quinapril"
{"url":"C:/Users/Rimma/Desktop/pills/Quinapril.jpg"}
{"url":"C:/Users/Rimma/Desktop/pills/ACCUPRIL2.jpg"}
{"url":"C:/Users/Rimma/Desktop/pills/ACCUPRIL1.jpg"}

Figure 6: Query result from Neo4J database

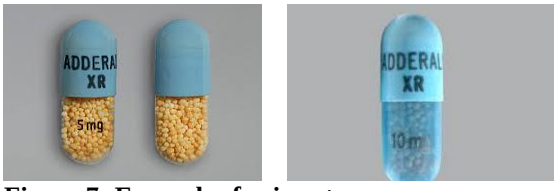


Figure 7: Examples for input

Neo4j Database: For this project we made a decision to store images and their attributes in Neo4j graph database. Neo4j is fairly flexible to use and can be manipulated later, should we add more functionality, something we could not do with relational database (i.e MySql, PostgreSql etc). Our database has 3 main node types: "Pill" nodes that stores the pill name, "Image" nodes that stores image (URL) that users have uploaded and "Page" nodes that stores URL of page which contain information about pill. The relationships are the following:

Image "contains" pages, meaning that a page has some information that is associated with the image.

(Image) - [Contains] -> (Page)

Image "has" pill, which means that the uploaded image has the name of the pill associated with it.

(Image) - [Has] -> (Pill)

5. QUERYING DATA

In figure 5 above there is the result of indexing of the three images that all contain the Adderall pill on them (as figure 7). The program created relationships and nodes which didn't exist prior.

There is the result of querying our populated database:

```
MATCH(n:Pill{name:"Quinapril"})<-[:HAS]-(k:Image)
RETURN k as Users_pictures_for_Quinapril
```

This query returns pages which contain information about Quinapril (printscreen of result in figure 6).

6. TESTING RESULTS

We have conducted several testing techniques but here we would like to focus of Precision and Recall tests.

Precision test results: We queried a drug called "Quinapril". We had 2/20 images that were relevant to this name in our database.
As a result 6 images were returned.

# of returns	1	2	3	4	5	6
R/N	R	N	N	R	N	N

Precision: P = tp/(tp + fp) = 2/(2+4) = 33%
Recall: R = tp/(tp + fn) = 2/(2+0) = 100%

We also queried a drug called "Adderall". We had 3/21 relevant images in our database. As a result 7 images were returned.

# of returns	1	2	3	4	5	6	7
R/N	N	N	N	N	R	R	R

Precision: P = tp/(tp + fp) = 3/(3+4) = 42.8%
Recall: R = tp/(tp + fn) = 3/(3+0) = 100%

7. ADAPTED CHANGES

After our presentation we improved our system in a way that the user can see his uploaded image and compare it with that image which our database has. User doesn't need to remember his image anymore.

8. FUTURE WORK

As we continued to develop our idea, we realized that more functionality can be added in the future. The main feature we would want to add soon is to give the user an option to order medicine from their nearest pharmacy based on their search results. This feature will require several additions to our database structure and strict compliance. Another feature is to allow users to automatically refill their prescription drugs via our app. In addition, we thought that being able to report drug abuse via our app would also contribute to better our society and perhaps prevent future tragedies. The last addition on our menu will be to offer our app in other languages which will expand our audience by a great margin.

9. REFERENCES

[1] "Google Vision API," Getting Started.[Online]. Available at: https://cloud.google.com/vision/. [Accessed: 10-November-2017].

[2] "Neo4j," [Online]. Available at: https://neo4j.com/. [Accessed: 3-October-2017].

[3] "WebMD," Pill Identifier. [Online]. Available at: https://www.webmd.com/. [Accessed: 6-November-2017].

[4] "Drugs.com," Prescription Drug Information, Interactions. [Online]. Available at: https://www.drugs.com/. [Accessed: 5-November-2017].