Contact Tracing using Neo4j and Face Recognition

Sathwik Kondapuram
University of Georgia
Athens, Georgia
Sk67015@uga.edu

Nikitha Jambula
University of Georgia
Athens, Georgia
sj04187@uga.edu

ABSTRACT

Contact tracing is very important in many use cases. For example, potential victims for Covid-19 will be those who came in close contact with a person who already has covid-19. In other case a suspect detail can be fetched by contacting their close contacts. In both the cases contact tracing plays crucial role. In today's world there is an abundance of picture data with government as well as individuals. We suggest using image data to do contact tracing. Our proposed solution is to maintain a graph database with each node representing an individual with two properties, Name and URL (Containing a Face shot of individual). When we get to know that an individual is tested positive to COVID, we can trace all the close contacts by just analyzing the images containing multiple faces. Each image will be fed to a face recognition model and based on detected labels a relation will be created among nodes in neo4j database. Now we will have a graph data with nodes indicating people and relation indicating that they were in a close contact. By developing cypher queries we can quickly check for direct, second degree, third degree so on until nth degree relations based on the requirements and seriousness of the case.

KEYWORDS

Nodes, Relations, Face Recognition Model, Connecting Neo4j and python.

INTRODUCTION

A detailed Step by step process for implementing our proposed Contact tracing technique are as follows:

1. Create a Neo4j Node for each person
2. Add Name and Face image URL properties to node
3. Create a connection between Neo4j and Python

4. Fetch All Node details from Neo4j Database and store them into a data frame.
5. Train a Face recognition model using known Face-Name Data tuples stored in data frame.
6. Upon giving an image containing multiple people Face recognition model will detect all the faces and outputs a list of detected faces
7. Using this list, a relation will be generated among the nodes in neo4j database.
8. Generate a cypher query to fetch all Nth degree relations. The results will be fetched quickly compared to traditional relational database queries.

RELATED WORK

Creating a Noe4j Node:

A node is a data/record in a graph database.

We can create a node in neo4j using the following cypher query:

CREATE (node)

As we know that each node represents a person it will be handy to give each node a label “Person”

CREATE (node:Person)

Adding propertied to Node:

Properties are name-value pairs that are used to add qualities to nodes and relationships.

For our use case we need our person node to have two properties

1. Name
2. URL

Query Example:

```
CREATE (node:Person {
name: ' Cristiano
Ronaldo', URL:
"https://upload.wikimedia.org/w
ikipedia/commons/8/8c/Cristiano
_Ronaldo_2018.jpg"})
```

Our Graph database will now contain several nodes each indicating an individual along with name and URL properties.

Accessing Neo4j from python:

Neo4j can be accessed from python using neo4j.GraphDatabase class present in neo4j library. Once we install neo4j library we can import GraphDatabase class.

By calling the driver method we can establish the connection.

Syntax:

driver = GraphDatabase.driver("bolt://localhost", auth=("neo4j", "password"))

Now with driver.session method we can start our session. We can pass the cypher query as an argument to session.run method.

Fetch all node details from Neo4j Database and store them into a data frame:

This can be done using the following syntax:

```
with driver.session(database="neo4j") as session:
  result = session.run("""
  MATCH (p:Person)
  RETURN p.name AS name, p.URL as URL
  """)
  X = pd.DataFrame([list(record) for record in result])
```

Face Recognition:

Face recognition is a visual pattern recognition problem which means it is the problem of recognizing and confirming individuals by their face in a photograph. A face recognition system with the input of an arbitrary image can search in depth in the database to generate the identification of people in the input image. Basically, it is a technology capable of matching a human face from a digital image or a video frame against a facial database, usually used to authenticate users, by recognizing and measuring facial features from a given image through ID verification services. It is a task that humans, even under varying light and when faces are altered by age or obstructed by accessories and facial hair, perform trivially.

Although people can recognize faces without much effort, facial recognition in computing is a difficult pattern recognition problem. Based on its two-dimensional image, facial recognition systems attempt to recognize a human face, which is three-dimensional and varies in appearance with lighting and facial expression. As shown in Figure 1, a face recognition system typically consists of four modules: Face Detection, Face Alignment, Feature Extraction and Face Recognition

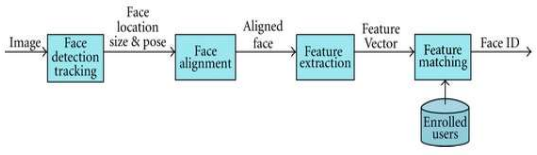


Fig-1 Structure of Face Recognition System

Face Detection: The non-trivial first step in face recognition is face detection. It is an object recognition issue that involves the identification of both the position of each face in a photograph (e.g. the location) and the location of the face (e.g. with a bounding box). Object recognition itself is a difficult issue, but it is similar in this case as there is only one form of object to be localized, e.g. faces, although faces can differ wildly.

In addition, since it is the first step in a wider scheme for face recognition, face detection must be robust. For example, it is not possible to identify a face if it cannot be identified first. This means that faces need to be marked with all sorts of orientations, angles, light ratios, hairstyles, caps, glasses, facial hair, makeup, ages, etc.

Face alignment attempts to achieve more precise localization and thus to normalize faces, while face detection provides coarse estimates of each detected face's position and size. The facial elements, such as the eyes, nose and mouth and facial outline, are located; the input face image is normalized according to the position points with regard to geometric properties such as size and posture, using geometric transformations or morphing.

With regard to photometric properties, such as illumination and gray scale, the face is typically further normalized. After a face is geometrically and photometrically normalized, **feature extraction** like eyes, nose and mouth are pinpointed and measured in the picture and then it is carried out to provide efficient knowledge that is useful in distinguishing between faces of different individuals and stable in terms of geometric and photometric variations.
The extracted feature vector of the input face is matched against those of enrolled faces in the database for **face matching**; it outputs the identity of the face when a match is identified with enough confidence or otherwise indicates an unknown face.

Train a Face recognition model using known Face-Name Data tuples stored in data frame:

When we pass the image to a face_locations method present in face_recognition library all the face locations(top left, top right, bottom left and bottom right) pixel values list will be returned.

Now these all face locations list is passed to a face_encodings function which will generate the face encodings of each face.

Detecting All the Faces present in a image:

When images containing multiple people is given, by using the following compare_faces method we can detect all the faces present in the image and then this list of detected faces(names) is stored in a persons_detected list.

Generating Relations among detected Faces:

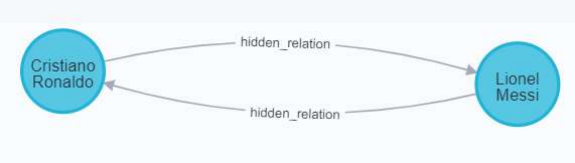
Again, using the session.run method a cypher query is passed as an argument which will generate the relations among all the person nodes with a name in persons_detected list.

Syntax:

```
with driver.session(database="neo4j") as session:

        result = session.run("""

        match(p1:Person),(p2:Person)

        where p1.name in $persons_recognized and p2.name in $persons_recognized and p1.name <>

        p2.name

        Merge (p1)-[:hidden_relation{name: $rel_name, url:$url_value}]->(p2)

        Merge (p1)<-[:hidden_relation{name: $rel_name, url:$url_value}]-(p2)

        return p1,p2

        """,{"persons_recognized":persons_recognized, "rel_name":rel_name, "url_value":pic_url})

        print(result)
```

Now, a relation will be generated in graph database. When we give a cypher query asking for a direct contact i.e., victim and people we are looking for are in a same image a graph will be returned.

Example: When we give an image containing Messi and Ronaldo, Given Ronaldo tested positive, Our answer set should return two nodes(messi and ronaldo) with a relation.



Images used are:

Ronaldo:

Messi:

Image with multiple people:

RESULTS:

We have used a random data set which we have generated randomly. Our data set includes 150 individuals. We have fed 100 images each containing 2-7 people. Based on our face recognition model output we have generated 480 relations.

CONCLUSION AND FUTURE WORK:

In this paper, we are assuming that an image containing multiple people as a sole reason for generating a relation among respective person nodes. However, we can improvise this by adding a Date measure indicating the Date and time at which picture was taken. Given a person tested positive today, we can feed only images taken 3-5 days prior to today. We can further add additional properties to person nodes and relations based on requirement. This model can be provided as a stand-alone application to users so that the victim can give images from gallery as an input and application will return the names of all the persons, he was close with, using this list victim can alert all the close contacts instead of updating everyone and creating a panic.

REFERENCES:

->https://neo4j.com/docs/getting-started/current/graphdb-concepts/

->
https://neo4j.com/docs/api/python-driver/current/api.html#api-documentation

->https://pypi.org/project/face-recognition/