Problem Statement

27,356 children in Delhi went missing during 2015-2018. And 63,407 children went missing in 2016, according to data from the National Crime Records Bureau, which translates to an average of 174 every day. More disquietingly, 50% of the children who have gone missing until 2016 have stayed untraced. This is a big issue and most of this childern get caught up in trafficing or child labour.

Solution

For solving this problem to some extent we have developed ReLice. A software solution for the problem of missing children.

The major problem is that there is a lot of workload on the authorities. They can't be at every place. But the people are present almost everywhere. So the main aim of our project is to transfer this load on the authorities to the masses.

We are using machine learning algorithms along with a lot of Azure services. The users across the country can send images of suspected missing children which we compare to the database of reported missing children. If a match is found authorities are immediately informed.

Azure Services used

1. Azure Cognition Services - Face API
    a. Used for extracting features from the image of the child
    b. Used for comparing 2 images to detect a missing child
    c. Extract FaceId from an image to ease comparisons and reduce image uploads
2. Azure Cosmos DB
    a. Used to store details of a child reported(eg. Skin color, hair color etc.)
    b. Used to store user details
3. Azure App Services
    a. Used to Deploy the web app for reporting a missing child by the authorities
    b. Used to Deploy the web app for serving the machine learning model
    c. Used to Deploy the web app for the serving api calls the database.

Steps:

Register a missing child -

1. Open link http://relice-test.azurewebsites.net/admin/dashboard
2. Enter user-id as 'admit@gmail.com' and password as 'admin'
3. Click on Register Complaint button on the left pane
4. Click on 'Pick an image' to open a file picker to select missing child's image
5. Fill all the details of the missing
6. Click Submit

Check if a child is missing -

1. Download ReLice.apk file to an android device
2. Install the above downloaded file
3. Open the installed app ReLice
4. Login with userId 'test.email@gmail.com' and password 'abcd1234'
5. A camera interface opens
6. Click the photo of a child that you suspect of being lost
7. Then press on Submit
8. The app extracts essential information from the image using Azure Congnition Services Face API
9. The app then uploads the image and initiates the check to detect if the child has been reported.
10. If found reported, the concerned authorities are notified immediately.

When a reported child is found the authorities are notified. It can be viewed in the Notifications sections on the website http://relice-test.azurewebsites.net/admin/dashboard

The demo videos are present in the "Video" folder in the "Waitlisted" folder of the main repository.

Edge Cases Covered

1. The project maintains a temporary database where queries are stored for 72 hours. It deals with the case in which a child's photo is submitted for check before the child has been reported missing by the parents. When a child is reported missing his images are compared with the temporary database.
2. We have secured the data of the children by not storing the images of the child at all. We submit the image to the Face API of Azure and it generates a face id. Now the image can be accessed by our APIs only and not by anyone else. Even our API accesses the images for comparison via the face ids and not the actual images. This ensures total security of everyone's data.

Future Work

We can extend this to pets. We can easily use this software solution to find the missing pets. Then app would not require the police intervention and can easily works between user and consumers.

ReliceAI!

ReliceAi is a python flask app **deployed on azure using app services** which serves the deep learning model as well as process other **Azure Cosmos DB** request.

What it does

It provides routes to both **Azure Face API** and **FaceNet-MTCNN-blend**. We have used **Azure Face API** for face detection as well as face matching and we have used FaceNet-MTCNN-blend network for image conversion to embeddings and image similarity calculation. It also provieds routes for the file extraction from **Azure Cosmos DB**.

Azure Face API

Azure face api is a part of **microsoft azure cognitive services** it provides face detection and face matching and many other featuers. We have used the face detection and face matching api from the cognitive services for our application.

How the image searching works

Our model does not directly extract all the images but it first search of sutiable images based on the meta data from our user the meta data includes the age, gender, hair color, skin color, and other features. This decreases the search tree size exponentially and we get the response in very short time. Then we just match the user given image with extracted images from the database(**Azure Cosmos DB**).

Access Model Files and API website

Model files are prototyped in jupyter notebook with different kids of deep learning frameworks like PyTorch, Keras and we have also used other types of scientific python packages.

[Relice API Link](https://reliceai.azurewebsites.net)
[Relice Notebook Link](https://github.com/ANUBHAVNATANI/LNMHacks4-Submissions/blob/master/Waitlisted/DeeplearningAndAzure/ML_Models/SiameseNetwork.ipynb)

RELIC API DOCUMENTATION

Base url : https://reliceapi.azurewebsites.net/

Report rotutes

1. [POST] /api/post/report
    a. Input :

```
{
  face_id : String,
  image : String (Base64 Image),
  age: Number,
  location: String,
  gender: String,
  hair_color: String,
  user_id: String
}
```

    b. Output.
        i. Error : `{ error: err }`
        ii. Otherwise : created report is returned

Processing Routes

1. [GET] /api/get/reports (for temp data access)
    a. Input : empty
    b. Output :
        i. Error : `[{error : err}]`
        ii. Otherwise : Array of reports

```
[
{
  face_id : String,
  image : String,
  age: Number,
  location: String,
  mgender: String,
hair_color: String,
user_id: String
}
]
```

2. [GET] /api/get/complaints (to be used with permanent database)
    a. Input : empty

b. Output :
    i. Error : `[{error : err}]`
    ii. Otherwise : Array of complaints

[

```
{
    face_id : String,
    victim_name : String,
    age: Number,
    location: String,
    gender: String,
    height: String,
    complainant_name: String,
    contact: String,
    skin_color: String,
    hair_color: String,
}
```

]

3. [POST] /api/post/matched
    a. Input : {

```
    complaint_id : String,
    report_id : String,
    victim_name : String
}
```

    b. Output :
        i. Error : `[{error : err}]`
        ii. Otherwise : registered match

```
{
    complaint_id : String,
    report_id : String,
    victim_name : String
}
```

Complaint Routes

1. [GET] /get/complaints
    a. Input : {}
    b. Output :
        i. Error : {error : <error mesage>}

ii. Otherwise : Array of complaints

[

```
{
    face_id : String,
      image : String,
    victim_name : String,
    age: Number,
    location: String,
    gender: String,
    height: String,
    complainant_name: String,
    contact: String,
    skin_color: String,
    hair_color: String,
```

]

2. [POST] /post/complaints
    a. Input : {

```
    image : String,
    victim_name : String,
    age: Number,
    location: String,
    gender: String,
    height: String,
    complainant_name: String,
    contact: String,
    skin_color: String,
    hair_color: String,
}
```

    b. Output :
        i. Error : {error : err }
        ii. Otherwise : {
            Complaint_id : String,
            Matched_id : String
            }

3. [GET] /get/matched
    a. Input : empty
    b. Output : Array of all the matched outcomes

[{

```
    complaint_id : String,
    report_id : String,
    victim_name : String
}
```

]
