Poor Man's Rekognition

Implementing real-time facial analytics on video

Basic Information

Sarfaraz Mohammed Iraque (pronounced Iraqui)

Final year Computer Science student from India

GitHub/Slack/Gitter/SO: @sziraqui

Email: sarfarazghlm@gmail.com

Summary

Poor Man's Rekognition (PMR) is aimed to provide an Open Source alternative to the paid Amazon Rekognition API (Rekognition). This summer, I aim to initiate PMR project providing complete implementation of facial analytics and recognition as a Node.js library. A REST API will also be developed as demonstrated in my Proof of Concept. Node.js library will be a binding for C++ backed Machine Learning (ML) algorithms. OpenFace 2.0 C++ library will be used for face detection and analytics. Tensorflow will be used to implement ArcFace for face recognition. It will be converted to Tensorflowjs model and included in PMR Node.js library. Finally, support for videos will be added exploiting a Multi-Object Tracking algorithm for efficient face attribute and person annotation.

Scope and objectives

My focus is on building a library to provide a single API in Node.js for all face related tasks and a REST API for using the library in other languages. The REST API should be as close to Amazon Rekognition as possible for two reasons -

(1) Anyone familiar with Rekognition should be able to easily use PMR and

(2) Rekognition API is quite well developed and offers good documentation.

PMR must be scalable and low cost library with competitive performance on CPU to existing libraries that perform well only on GPU. That is the main reason I chose C++ and Node.js instead of Python (More on this later). High level API in Node.js will be provided

along with a REST API. For celebrity face recognition, publicly available datasets will be used to build a NoSQL database of tagged celebrities.

What this project will not include

I won't propose a new Neural Network architecture. Developing ML algorithms that beats state-of-the-art (SOTA) is not practically possible in 3 months. Instead, my focus is on high performance and scalability of PMR using current SOTA architectures. A GUI for end users is beyond the scope given the amount of work involved to achieve high CPU performance. Moreover, I am not a GUI expert. I can provide simple web pages for demonstrating REST API but a full blown web application is not something I am aiming at. Unit testing is also kept optional for now. I may do it post GSoC.

Video Annotation

Non-trivial things first

Let's assume face detection and recognition are already implemented in our library. Also assume that video annotation does not include facial attributes for the time being. They will be discussed in later sections in detail. The choice of programming tools (C++, Node.js) is primarily for providing real-time video annotation on both CPU and GPU hardware. This is essential in order to sustain PMR as a free service.

How Amazon Rekognition handles videos

Amazon Rekognition being an AWS API, all input data (image/video) have to be stored in S3 bucket. While images can alternatively be sent along with POST request body as a base64 string, videos have to be first stored in S3 bucket and parameters like bucket ID and file ID are sent in request body of REST call to the API. In response, a job ID is received which can be queried later to get final video annotation result. Naturally, video size is restricted to 60 MB. Refer

Can we avoid storing videos?

A better approach is to stream videos and provide frame-by-frame response. Rekognition supports this but the feature can be accessed only with AWS Java SDK. We can use Web Sockets to implement such feature in Node.js and eliminate the requirement of storing videos on our servers. However, Web Sockets implies real-time performance which is the real challenge here.

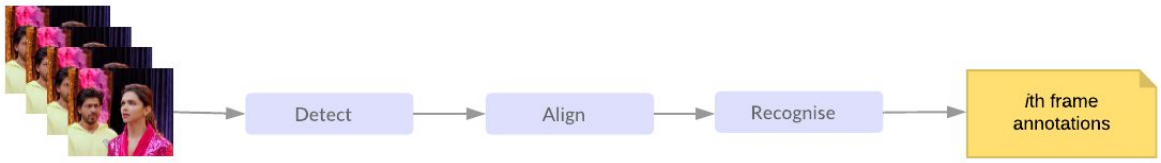
Video Annotation approaches

Steps involved in person and face attribute annotation

For each frame of video stream following steps are generally needed

1. **Detect:** Run a face detector to find bounding box coordinates of every face.
2. **Align:** Align every detected face such that a vertical line should easily fit centers of eyes, nose and mouth. Helps in recognition
3. **Recognise:** Classify each face into known label using a face representation.

The naive approach



But Align and Recognise steps are very expensive and hence require GPU for real-time performance. A brute-force solution like above will be very slow on a CPU.

Please Check my implementation of above method using face-api.js

Can we do better?

The Detect step has to be carried out for every frame. We can skip Recognise step by exploiting a method called object tracking. The basis of the idea is that *bounding boxes of same object in consecutive frames will have high overlap.* For example,

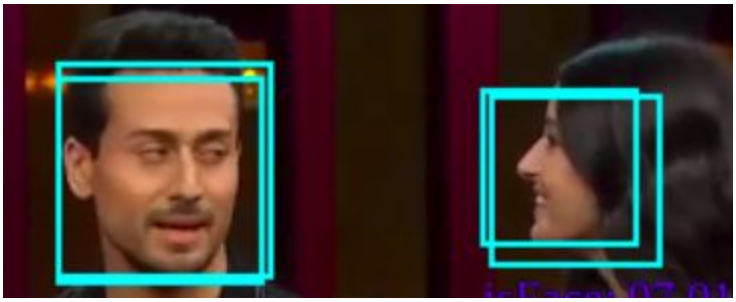
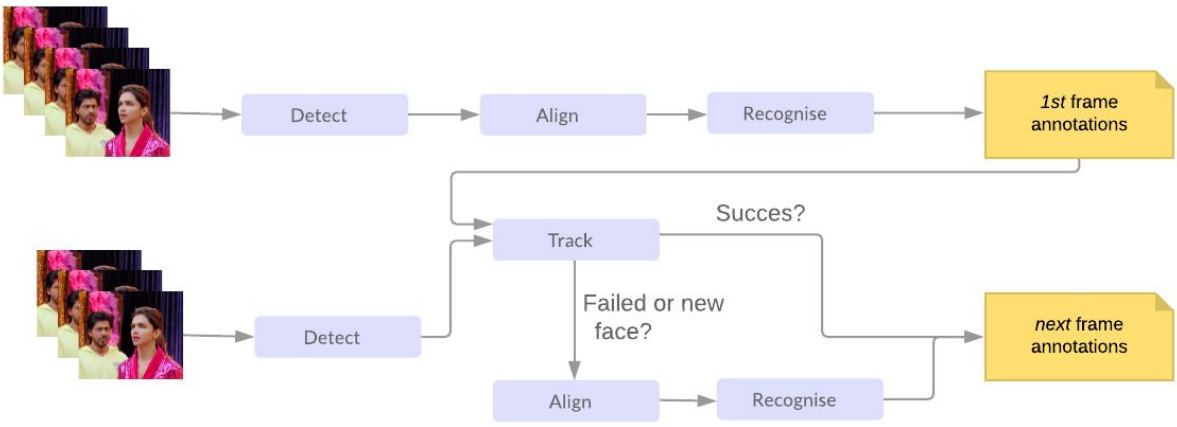


Image courtesy: 'Koffee with Karan' and Star World

When running my video annotation program implemented in POC, I kept last frame's bounding box. When current frame bounding box was drawn, it overlapped greatly with celebrities' past frame bounding box. Typically, overlap is measured using Jaccard's distance or Intersection Over Union (IOU). This idea is the basis for object tracking and person re-identification algorithms. Objects are assigned some IDs in the first frame. For next frames, IOU measurements help reassign the ID to new detections. There are many Multi-object tracking (MOT) methods but only few perform well in real-time viz. Simple Online Real-time Tracking (SORT) [8] and its extension DeepSORT [9].

Recog+MOT approach

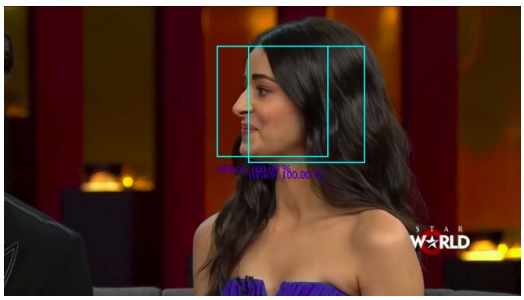


For the first frame, we will detect+align+recognise and assign labels to each detected face bounding box. For next frame, we will only perform detection and then measure overlap from previous frame boxes to see if we can track and label the new detections. If we fail to assign label to any box, we can simply run align+recognise for that box only. Face recognition involves very deep neural networks. (See section X). Being able to skip recognition step is a huge savings especially for CPU.

A corner case

MOT can result in false positives when there is no continuity in consecutive frames (eg. rapid scene transition). For example, consider below two consecutive frames from a Koffee with Karan promo.

Bounding boxes of both frames on current frame



MOT algorithm will assign label for person on left to the person on right for all the next frames! Even higher overlap can occur during scene transitions. This is a known issue which MOT algorithms cannot solve. To avoid long term incorrect annotations, we must run recognise step after some interval.

Recog+MOT+RecogCheck

Let N be the frame rate at which video stream will be processed. Let M (<=N) be the frame rate at which recognise step is executed. We will run Recog+MOT but after every Mth frame, we will verify the labels assigned for the current frame (RecogCheck). If false positive is detected, we can correct the annotation and prevent the error from propagating to next frames. With such a scheme, we will have M-1 incorrect annotations in worst case. We can choose M depending on the hardware. When M = N, the algorithm will be same as the naive brute-force approach.

Parallel processing with initial delay.

We can divide the video into P frames and annotate the first P frames while simultaneously fetching next P frames followed by their annotation. In such case, a response will be sent only after the first P frames are processed. Thus an initial delay of P frames/second will exist but the entire video will be processed much faster on a multi-core CPU. Thus we are not only saving CPU cycles with Recog+MOT+RecogCheck approach but also utilising idle CPU/GPU cores for extreme efficiency.

Building blocks of PMR

This section is a review of available technologies and techniques for face detection, attribute detection, alignment and recognition. All speed measurements are from my laptop running Intel dual core i5-5250U. I have used existing Python and Node.js implementations of these ML models.

Face Detection and Alignment

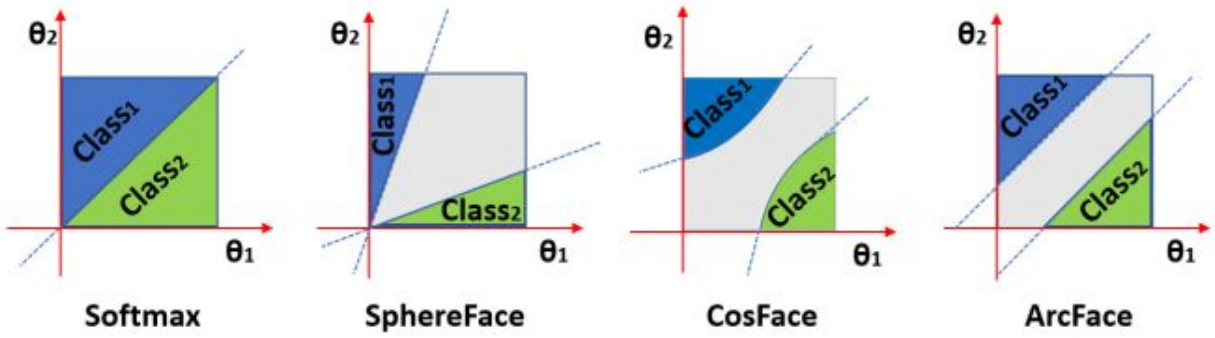
While there are many face detections ML models, I would focus on only those that allow face alignment with minimal overhead. Popular names like Haar Cascades and Histogram of Oriented Gradients (HOG) as simple low cost methods will come into mind. But these primitive methods do not generalise well for ‘in the wild’ faces [21]. Many Deep Learning solutions like SSD-MobileNet [12], YOLO [13] and a recent work MTCNN [5] have proved to be superior in terms of accuracy and to some extent speed too. SSD-MobileNet is very accurate face detector but has slow inference time (550-600 ms). YOLO is famous for real-time inference and it took roughly 160 ms for detections on same image. Both of these models are face detectors only. Additional step is needed for alignment. MTCNN is a joint model that performs both detection and alignment using a 5-point face landmark scheme.

The current SOTA for WIDER Face (hard) benchmark is held by DSFD [14] but it is not ideal for real-time inference. SRN [15] holds the 2nd spot but it's real-time implementation is not available. Many other face detectors were recently proposed but MTCNN is the only modern architecture whose Open Source implementations exist for real-time CPU inference by OpenFace 2.0 [7]

Face Recognition

It is impractical to make a classifier for every face present on this planet. Hence face recognition by verification is employed for most applications. Typically, a classifier is trained on 1000s of subjects and the classification layer is dropped after training. Output of feature extractor acts as a face embedding (face vector). Squared distance between two face embeddings is used to determine if two faces belong to the same person. Feature extractors developed in such manner have no guarantee for generalisation to unknown faces.

DeepFace [17] presented use of Siamese network to achieve ~97% accuracy on **LFW [23]** dataset of celebrity faces. Later on **FaceNet [4]** implemented a new method of training a feature extractor without a classification layer using Triplet loss and achieved a record 99.63% accuracy on LFW. But Facenet records only **86.47%** accuracy on **MegaFace [22]** and a very low **66.50%** on **IJB-C [27]**. This is because the proposed triplet loss does not help in separating two faces with a good margin. It helps in determining if a face matches another face but it fails to clearly determine if a face does not match another face. The latter is more important for face recognition. Hence the idea of *incorporating margins into a loss function to maximise class face separability* has been adopted by many researchers recently. **SphereFace [2]** introduced angular margin to softmax loss (A-Softmax) and the new loss function consistently influenced accuracy on different CNN architectures. **CosFace [3]** introduced cosine margin penalty to discriminate two classes more effectively. **ArcFace [1]** introduced additive angular margin penalty and became SOTA for many benchmarks including LFW (**99.83%**) and MegaFace (**98.48%**). The effect of margin penalty is visualised by ArcFace and attached below.



This also explains why previous approaches fail to determine if face **does not** match another face. There are other recent architectures like **AIM [19]**, **SeqFace [20]** and others but **ArcFace**'s novelty is it's **relatively easy implementation** without much loss in

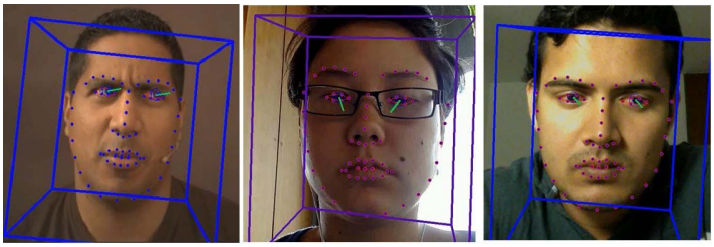
generalization to unknown data and faster convergence compared to CosFace and SphereFace.

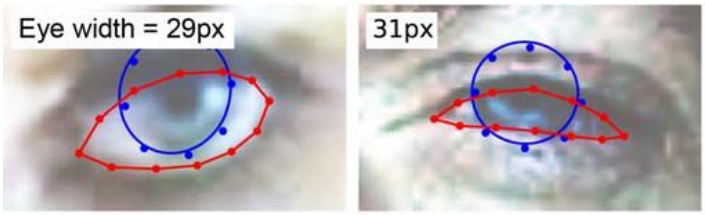
Facial attributes detection

Amazon Rekognition provides prediction of facial attributes viz -

Eye gaze, mouth open/close, expression (only smile) and head pose (pitch, yaw, roll). It also predicts age group, presence of eyeglasses, beard and moustache. Due to time constraints, it was not possible to do a survey on all of these but a general method is to perform object detection on each face. Fortunately, OpenFace 2.0 comes packaged with highly efficient implementation of predicting facial action units (AU). It currently provides head pose and eye gaze estimation and several important facial attributes listed below from original paper-

AU	Full name	Illustration
AU1	Inner brow raiser	
AU2	Outer brow raiser	
AU4	Brow lowerer	
AU5	Upper lid raiser	
AU6	Cheek raiser	
AU7	Lid tightener	
AU9	Nose wrinkler	
AU10	Upper lip raiser	
AU12	Lip corner puller	
AU14	Dimpler	
AU15	Lip corner depressor	
AU17	Chin raiser	
AU20	Lip stretched	
AU23	Lip tightener	
AU25	Lips part	
AU26	Jaw drop	
AU28	Lip suck	
AU45	Blink	





OpenFace 2.0 claims to have highly optimized implementations of all of the above for real-time performance on CPU.

Implementing PMR - The library

Why not Python?

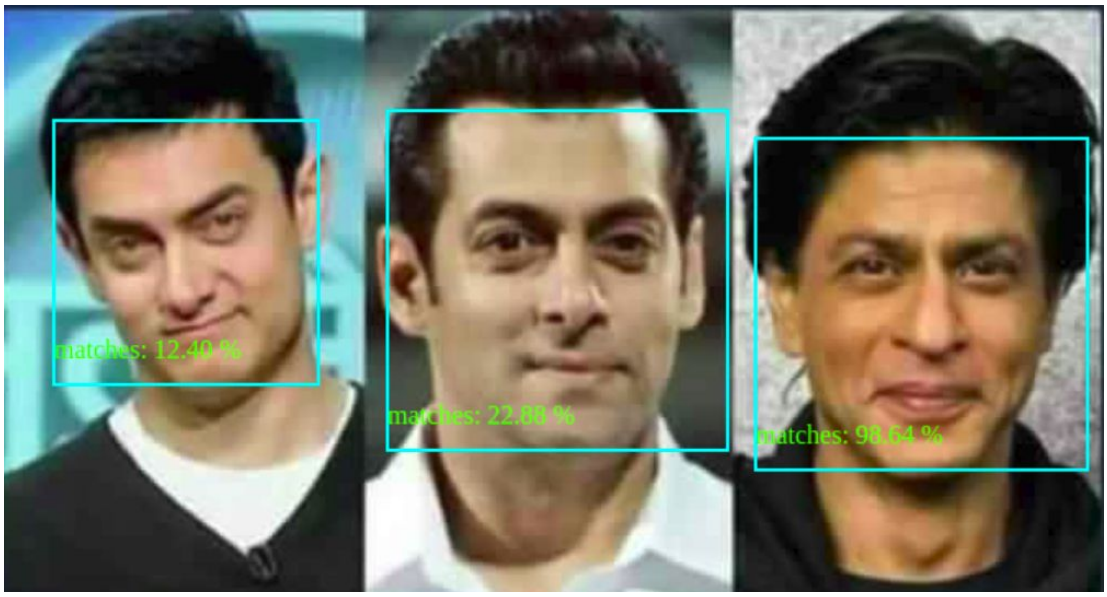
A lot of face recognition libraries are available for Python. Both Tensorflow and TensorflowJS run on the C/C++ engine under the hood. So why not use Python?

Even though Tensorflow and it's JS version have comparable performance on GPU, it is observed that Tensorflow's Node.js variant is 4x faster on CPU. Moreover, Node.js is excellent for IO operations because it performs all IO related tasks on a separate process contrasting to Python which uses threads to become asynchronous. The available Tensorflow models in Python can be easily converted to TensorflowJS models without rewriting code. Also the Node.js Express framework is the first choice of many web developers for REST API.

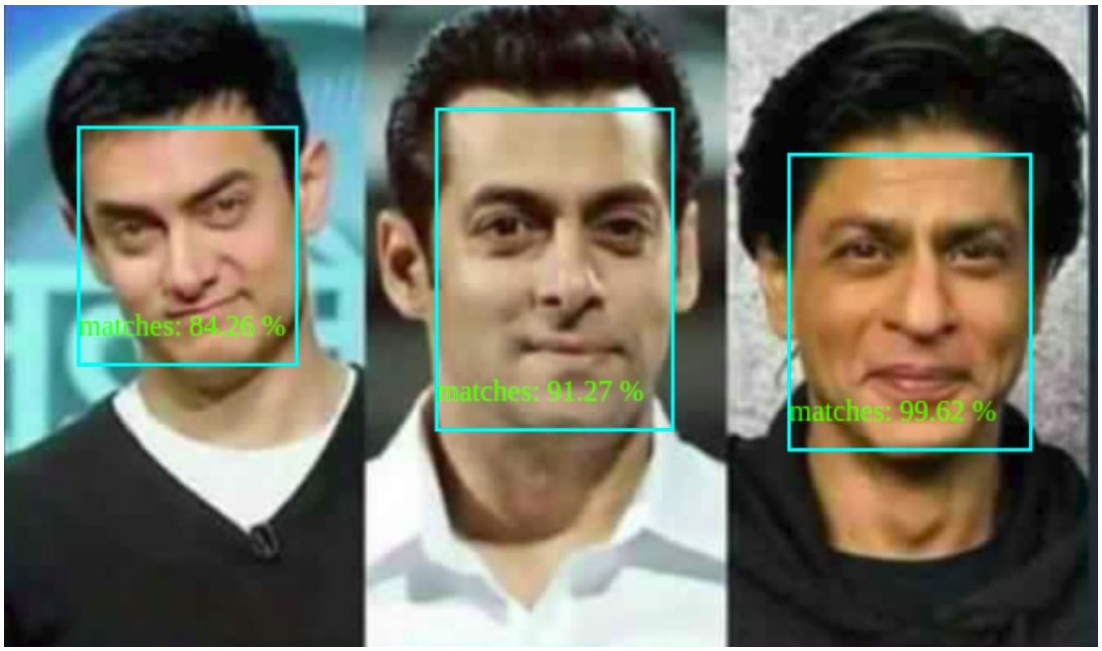
Existing libraries

I propose PMR as a Node.js library for high performance, scalability to add more features and sustainability of the free service. Almost all the features required in PMR are available in various programming languages and scattered all over GitHub. But most of them, in my opinion are not production ready especially for real-time CPU inference. In my POC, I used node-facenet (by Github user @huan) which is a Node.js wrapper to Tensorflow implementation of original Facenet paper (by GitHub user @davidsandberg). It provided excellent accuracy as mentioned in Facenet paper but the model is computation and memory expensive. I learnt TypeScript while reading node-facenet code which was a priceless experience. Face-api.js is a Node.js library having TensorflowJS implementations of SSD-MobileNet, YOLOv2, 68-landmarks face alignment model and a light version of Facenet. I switched to face-api.js in my POC to try the library. It was much better in terms of CPU performance. Here is the recognition result when using node-facenet

Source image:



Target image and recognition probabilities on Facenet original architecture



Target image and recognition probabilities on face-api.js's simplified Facenet model

Face-api.js is made for browser and later, support for Node.js was added by monkey-patching the Node.js environment. Moreover, image is read as HTMLImageElement object which is not present in Node.js hence dom and canvas packages have to be used to simulate a browser. A better and efficient way of reading images is by using OpenCV's Mat object. Face-api.js is excellent choice for edge devices but not for local computer and servers. Many libraries provide a some part of functionality required for PMR but they are mostly written in Python and are very slow on CPU.

OpenFace 2.0 [7] is a completely different version of the famous OpenFace [6] project. While the original OpenFace focused on face recognition, OpenFace 2.0 focuses on facial analytics and claim high performance on CPU due to the C++ API. Except face recognition, it has all other important functionality required by PMR. Moreover, Node.js supports extending the runtime with C++ Addon. In short, Node.js with C++ is a very efficient combination.

OpenFace 2.0 bindings for Node.js

Thus I will write Node.js Addons to bring all the functionality of OpenFace 2.0 without compromising performance. It might not be possible to write bindings for the entire OpenFace 2.0 library but the most important features like IO operation, face detection and face attribute prediction will be binded to Node. This will probably take a month.

ArcFace in Tensorflow.JS

ArcFace authors have implemented a Deep CNN architecture in MXNET and the model weights on different benchmarks are also available on GitHub. Since I have no experience with MXNET, I would like to implement the same DCNN architecture in Tensorflow. This is where I will learn a lot from the mentor Mr. Johannes Von Lochter. The pre-trained weights from MXNET model can be exported to Tensorflow format using MMdnn. I will still write scripts for retraining and fine tuning the model.

Implementing PMR - The REST API

Overview of Amazon Rekognition REST API

Amazon Rekognition defines several Data Types that are used in calling the API through POST requests. It also defined Actions such as DetectFaces and CompareFaces (Implemented in POC). We should follow similar Data types and expose similar Actions in our REST API because these are quite well designed. To understand this better, consider the DetectFaces Action's Request/Response syntax:

Request Syntax:

```
{
   "Attributes": [ "string" ],
   "Image": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   }
}
```

Image is sent as a **base64 encoded string** (blob) or as S3Object (when you have a stored image on AWS S3 storage). Incorporating image data in a JSON format makes it easy to call the API from several languages.

For calling the API from JavaScript client, a submitted image file can be converted to base64 using fileReader.readAsDataUrl() as illustrated in below snippet:

```
function encodeImageFileAsURL(inputElement, callback) {

    var filesSelected = inputElement.files;
    if (filesSelected.length > 0) {
        var fileToLoad = filesSelected[0];

        var fileReader = new FileReader();

        fileReader.onload = function(fileLoadedEvent) {
            var srcData = fileLoadedEvent.target.result; // <--- data: base64

            var newImage = document.createElement('img');
            newImage.src = srcData;

            document.getElementById("imgTest").innerHTML = newImage.outerHTML;
            callback(document.getElementById("imgTest").innerHTML);
        }
        fileReader.readAsDataURL(fileToLoad);
    }
}
```

Response Syntax:

```
{
   "FaceDetails": [
      {
         "AgeRange": {
            "High": number,
            "Low": number
         },
         "Beard": {
            "Confidence": number,
            "Value": boolean
         },
         "BoundingBox": {
            "Height": number,
            "Left": number,
            "Top": number,
            "Width": number
         },
         "Confidence": number,
         "Emotions": [
            {
               "Confidence": number,
               "Type": "string"
```

The BoundingBox data type store top left pixel coordinates and width and height of the bounding box as a normalised ratio.

From documentation:

> The top and left values returned are ratios of the overall image size. For example, if the input image is 700x200 pixels, and the top-left coordinate of the bounding box is 350x50 pixels, the API returns a left value of 0.5 (350/700) and a top value of 0.25 (50/200).
>
> The width and height values represent the dimensions of the bounding box as a ratio of the overall image dimension. For example, if the input image is 700x200 pixels, and the bounding box width is 70 pixels, the width returned is 0.1.

This design helps keeping the BoundingBox object independent of original image dimensions. Pixel coordinates of the box for any scaled version of original image can be easily calculated. Here's the code for conversion from my POC:

```
29      toRect(parentWidth, parentHeight): Rectangle {
30          const w = this.Width*parentWidth;
31          const h = this.Height*parentHeight;
32
33          return new Rectangle(
34              Math.round(this.Left*w),
35              Math.round(this.Top*h),
36              Math.round(w),
37              Math.round(h)
38          );
39      }
```

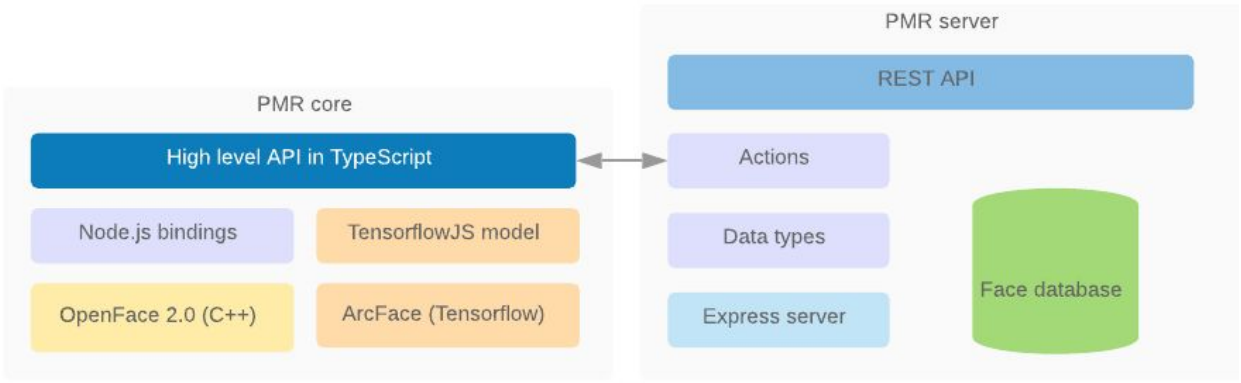
This is just one example of the smart design of their API. It is therefore best to follow their design of the REST API.

Server stack

The server will be developed using **Express** framework. Inspired by node-facenet, I will use **TypeScript** instead of vanilla JavaScript. We will also need a database for celebrity face recognition. An obvious choice is **MongoDB**. As mentioned in Scope, there won't be any GUI, so no frontend framework.

High level view of Server stack

The server implementation is well demonstrated in POC. A high level overview of the server stack is illustrated below.



High level view of PMR Server stack

Project Timeline

Till 5 May | Pre-GSoC period

I have explored CCExtractor projects. I have used CCExtractor and the explored the Realtime demo website codebase as I was initially interested in CCR project. I also went through the codebase of Sample Platform. I first thought of proposing my current research work "Automating Lip Reading" as my Summer project but it seemed impractical to finish in 3 months. I decided to work on PMR after evaluating its feasibility given my experience in OpenCV and Node.js and knowledge of Deep Learning.

In the remainder time from now, I will explore OpenFace 2.0 codebase in depth. I will also explore opencv4nodejs project which is a Node.js binding for OpenCV. This will help me learn industry practices in writing Node.js bindings to any C++ program. I have mid-term test on **April 18-20.** Except for these 3 days, I will actively learn and explore projects that involve writing Node.js Addons for C++ code. I will also spend some time understanding ArcFace's MXNET implementation.

May 6 - 27 | Community Bonding period

May 6 - 13 (1 Week)

I will discuss the implementation of ArcFace with Project mentor Johannes Lochter. I will spend time understanding CI tools used in CCExtractor and ways I can use it my PMR to increase productivity.

May 14 - 27 (2 weeks)

My Semester exam is from May 15 till June 3. I won't be available for this duration. I will still read all the discussions on Slack channel.
Refer Academic calendar

May 27 - June 28 | Coding Period 1

May 27 - June 3 (1 Week)

As stated earlier, I will be busy with Semester exam till June 3.

June 3 - June 10 (1 Week)

I will start writing Node.js bindings for OpenFace 2.0. I will start with bindings for IO operations of image and videos. I will spend 6 hrs/day coding. First 5 days will be spent writing the node-gyp code and last two days, I will expose the bindings. to high level TypeScript API

June 10 - June 17 (1 Week)

I will write bindings for face detection algorithms present in OpenFace 2.0. This will include the MTCNN model too. I will code for 8 hrs a day. First five days will be spent on writing the binding code and last two on high level TypeScript API.

June 17 - June 24 (1 Week)

I will code bindings for Eye gaze estimation and head pose estimation spending 8 hrs/day. I will attempt to complete bindings for age and gender prediction models. The week will be split in 5-2 scheme as stated for previous week. I will read about facial attribute detection in free time.

I have participated in a National level AI Hackathon (VesAIthon). If selected, I will be **unavailable on June 22**

June 24 - June 28 (4 days) | Evaluation

I will discuss the project status with mentor. I will ensure everything built so far is working by simple manual tests. I will also refactor TypeScript code to resolve any anomaly in naming conventions and ESlint. If time permits, I will write some unit tests especially for the Node.js code.

June 28 - July 22 | Coding Period 2

June 28 - July 12 (2 Weeks)

I will work on implementing ArcFace in Tensorflow. I will report to JV Lochter on every alternate day to ensure I am on right track. I will implement the 4 DCNN architectures mentioned by ArcFace authors and evaluate their performance on CPU and GPU.

July 12 - July 15 (3 days)

I will work on writing high level API for our ArcFace implementation. I will also expose APIs for retraining ArcFace models.

July 15 - July 22 (1 week)

I will implement video annotation. The said Recog+MOT+RecogCheck algorithm will be implemented for running locally.

July 22 - July 26 (4 days) | Evaluation

This time will be spent on evaluating the face detection with face recognition API developed so far. I will do major refactoring on the over all TypeScript API.

July 26 - August 19 | Coding Period 3

July 26 - August 2 (1 Week)

I will start working on REST API. Firstly, all the Data types of Rekognition and Request/Response interfaces will be coded. Then a database of face embeddings of celebrities in LFW dataset will be created by writing simple Node.js scripts.

August 2 - 16 (2 weeks)

I will expose REST APIs for face related Actions of Amazon Rekognition. Then I will work on Web Sockets for real-time video annotation REST API.

August 16 - 19 (3 days) | Evaluation

I will test the whole REST API manually. The performance test of the the whole project on CPU and GPU will be also be performed.

August 19 - August 26 | Final evaluation

I will prepare a docker image of whole project with sample inputs. The Docker image will be shared with mentors for evaluation. Final code will be submitted.

My Profile and experience

My last internship was at AitoeLabs, an AI Start-Up dealing with video analytics on low end hardware. I developed a Node.js application for image annotation in the first month of my internship. I spent next 3 ½ months to improve their multi-object tracking algorithms in C++ using Yolo face detector and SORT algorithm.
Please check my always up to date profile on Linkedin
And also my Github profile

References

[1] ArcFace: Deng, J., Guo, J., & Zafeiriou, S.P. (2018). ArcFace: Additive Angular Margin Loss for Deep Face Recognition. CoRR, abs/1801.07698.

[2] SphereFace: Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., & Song, L. (2017). SphereFace: Deep Hypersphere Embedding for Face Recognition. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6738-6746.

[3] CosFace: Wang, H., Wang, Y., Zhou, Z., Ji, X., Li, Z., Gong, D., Zhou, J., & Liu, W. (2018). CosFace: Large Margin Cosine Loss for Deep Face Recognition. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 5265-5274.

[4] Facenet: Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 815-823.

[5] MTCNN: Xiang, J.Q., & Zhu, G. (2017). Joint Face Detection and Facial Expression Recognition with MTCNN. 2017 4th International Conference on Information Science and Control Engineering (ICISCE), 424-427.

[6] OpenFace: Amos, B., Ludwiczuk, B., & Satyanarayanan, M. (2016). OpenFace: A general-purpose face recognition library with mobile applications.

[7] OpenFace 2.0: Baltrusaitis, T., Robinson, P., & Morency, L. (2016). OpenFace: An open source facial behavior analysis toolkit. 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), 1-10.

[8] SORT: Bewley, A., Ge, Z., Ott, L., Ramos, F.T., & Upcroft, B. (2016). Simple online and realtime tracking. 2016 IEEE International Conference on Image Processing (ICIP), 3464-3468.

[9] DeepSORT: Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. 2017 IEEE International Conference on Image Processing (ICIP), 3645-3649.

[10] LightCNN: Wu, X., He, R., Sun, Z., & Tan, T. (2018). A Light CNN for Deep Face Representation With Noisy Labels. IEEE Transactions on Information Forensics and Security, 13, 2884-2896.

[11] Deep face recognition survey: Wang, M., & Deng, W. (2018). Deep Face Recognition: A Survey. CoRR, abs/1804.06655.

[12] SSD-MobileNet: Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., & Berg, A.C. (2016). SSD: Single Shot MultiBox Detector. ECCV.

[13] YOLO: Redmon, J., Divvala, S.K., Girshick, R.B., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788.

[14] DSFD: Wang, Y., Wang, C., Tai, Y., Qian, J., Yang, J., Wang, C., Li, J., & Huang, F. (2018). DSFD: Dual Shot Face Detector. CoRR, abs/1810.10220.

[15] SRN: Chi, C., Zhang, S., Xing, J., Lei, Z., Li, S.Z., & Zou, X. (2019). Selective Refinement Network for High Performance Face Detection. CoRR, abs/1809.02693.

[16] FPN: Chang, F., Tran, A.T., Hassner, T., Masi, I., Nevatia, R., & Medioni, G.G. (2017). FacePoseNet: Making a Case for Landmark-Free Face Alignment. 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), 1599-1608.

[17] DeepFace: Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. 2014 IEEE Conference on Computer Vision and Pattern Recognition, 1701-1708.

[19] AIM: Zhao, J., Cheng, Y., Cheng, Y., Yang, Y., Lan, H., Zhao, F., Xiong, L., Xu, Y., Pranata, S., Shen, S., Xing, J., Liu, H., Yan, S., & Feng, J. (2019). Look Across Elapse: Disentangled Representation Learning and Photorealistic Cross-Age Face Synthesis for Age-Invariant Face Recognition. CoRR, abs/1809.00338.

[20] SeqFace: Hu, W., Huang, Y., Zhang, F., Li, R., Li, W., & Yuan, G. (2018). SeqFace: Make full use of sequence information for face recognition. CoRR, abs/1803.06524.

[21] Face detection survey: Zafeiriou, S.P., Zhang, C., & Zhang, Z. (2015). A survey on face detection in the wild: Past, present and future. Computer Vision and Image Understanding, 138, 1-24.

[22] MegaFace: Kemelmacher-Shlizerman, I., Seitz, S.M., Miller, D., & Brossard, E. (2016). The MegaFace Benchmark: 1 Million Faces for Recognition at Scale. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4873-4882.

[23] LFW: Huang, G.B., Mattar, M., Berg, T.L., & Learned-Miller, E.G. (2008). Labeled Faces in the Wild: A Database forStudying Face Recognition in Unconstrained Environments.

[24] WIDER Faces: Yang, S., Luo, P., Loy, C.C., & Tang, X. (2016). WIDER FACE: A Face Detection Benchmark. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 5525-5533.

[25] IJB-A: Klare, B., Klein, B.E., Taborsky, E., Blanton, A., Cheney, J., Allen, K., Grother, P., Mah, A., Burge, M., & Jain, A.K. (2015). Pushing the frontiers of unconstrained face detection and recognition: IARPA Janus Benchmark A. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1931-1939.

[26] IJB-B: Klare, B., Klein, B.E., Taborsky, E., Blanton, A., Cheney, J., Allen, K., Grother, P., Mah, A., Burge, M., & Jain, A.K. (2015). Pushing the frontiers of unconstrained face detection and recognition: IARPA Janus Benchmark A. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1931-1939.

[27] IJB-C: Maze, B., Adams, J.C., Duncan, J.A., Kalka, N.D., Miller, T., Otto, C., Jain, A.K., Niggel, W.T., Anderson, J., Cheney, J., & Grother, P. (2018). IARPA Janus Benchmark - C: Face Dataset and Protocol. 2018 International Conference on Biometrics (ICB), 158-165.