Multimedia Tools and Applications (2023) 82:8447–8466
https://doi.org/10.1007/s11042-022-13555-y

1207: INNOVATIONS IN MULTIMEDIA INFORMATION PROCESSING & RETRIEVAL

Deep learning model based multimedia retrieval and its optimization in augmented reality applications

Yash Prakash Gupta[1] · **Mukul**[1] · **Nitin Gupta**[2]

Received: 31 January 2021 / Revised: 11 March 2022 / Accepted: 14 July 2022 /
Published online: 5 August 2022

Abstract

With the uproar of touchless technology, the Virtual Continuum has seen some spark in the upcoming products. Today numerous gadgets support the use of Mixed Reality / Augmented Reality (AR)/ Virtual Reality. The Head Mounted Displays (HMDs) like that of Hololens, Google Lens, Jio Glass manifested reality into virtuality. Other than the HMDs many organizations tend to develop mobile AR applications to support umpteen number of industries like medicine, education, construction. Currently, the major issue lies in the performance parameters of these applications, while deploying for mobile application's graphics performance, latency, and CPU functioning. Many industries pose real-time computation requirements in AR but do not implement an efficient algorithm in their frameworks. Offloading the computation of deep learning models involved in the application to the cloud servers will highly affect the processing parameters. For our use case, we will be using Multi-Task Cascaded Convolutional Neural Network (MTCNN) which is a modern tool for face detection, using a 3-stage neural network detector. Therefore, the optimization of communication between local application and cloud computing frameworks needs to be optimized. The proposed framework defines how the parameters involving the complete deployment of a mobile AR application can be optimized in terms of retrieval of multimedia, its processing, and augmentation of graphics, eventually enhancing the performance. To implement the proposed algorithm a mobile application is created in Unity3D. The mobile application virtually augments a 3D model of a skeleton on a target face. After the mentioned experimentation, it is found that average Media Retrieval Time (1.1471 μs) and Client Time (1.1207 μs) in the local application are extremely low than the average API process time (288.934ms). The highest time latency is achieved at the frame rate higher than 80fps.

Keywords Media retrieval · Augmented reality · Deep learning · Latency · Medical augmented reality · Cloud computation · MTCNN · OffLoading

✉ Nitin Gupta
nitin@nith.ac.in

Extended author information available on the last page of the article.

1 Introduction

Evident from the notion of Reality-Virtuality Continuum, the AR technology spawns the actual world by rendering simulated annotation material in the camera view of the device and aligning the virtual object pose, with the camera pose [34]. As a result, it overlays the rendered visual objects to the top of the real-world camera view and provides an intuitive user interface. The entire collection of the virtual continuum can be defined using Milgram and Kishino's reality–virtuality continuum [20], where Mixed Reality (MR) is the parent of all the sub-realities as shown in Fig. 1. When a blend of real and virtual worlds occurs, a new reality emerges, known as mixed reality which provides vast opportunities with regard to accessibility and time.

The framework of MR and AR systems has opened a universe to the development of substantially intuitive mobile applications. The recently released AR development kits worldwide ARCore (Google) and ARKit (Apple) [21], SparkAR (Facebook) [27], Vuforia [23], CatchAR SDK [29] have compelled developers to work and build completely intuitive virtual interaction. In the currently prevailing situation of COVID-19, it has become essential to reiterate physical services.

While most of the companies are resorting to VR solutions, the generic users are not equipped with the costly VR headsets. Currently, Jio Telecommunication in India has introduced its AR/VR headset as JioGlass, which is 75 grams in weight and has exceedingly low cognitive strain [11]. Therefore, the upliftment in the AR technology will manifest the physical world into a virtual and extremely interactive world, where some essential services like online study, entertainment, office meetings, sport studies, marine and aviation monitoring and training, all can happen by augmenting virtual assets to the physical world.

Since AR is the interaction with the physical world, it becomes essential to understand the happenings of the physicality around the operating mobile device. This will not only enable intuitive graphical interaction, but also enhance the user experience. Therefore, in order to achieve this and infer about the physical assets near the device, the AR application needs to capture the surroundings in the form of media [26]. Owing to the vast amount of data being captured in the AR applications, the best choice to study the same is the deep learning concept. Therefore, the proposed work elucidates how to achieve that with the help of deep learning. Due to the flexibility of deep learning methods in the approach of inferring the results, they can substantially affect about how the AR applications can be optimized and work seamlessly.

In the proposed work it is intended to use an augmentation mechanism to ease our understanding of the skeleton of the head. The graphical mode of education enhances grasping and retention capacities in young learners. The problem lies in limited resources available to study the system in 3D space, the solution lying in manifesting the physical system to digital reality being the solution. Skull possesses peculiar bony prominences, mastoid process, occipital protuberance, foramens, sutures, which can only be studied on a 3D object [22]. The imagination of these structures solely by studying the text and diagrams could

Fig. 1 Milgram and Kishino's Reality–Virtuality Continuum

pose difficulties in learning for most of the students. Similar is the case with specimens of the human body, knowledge of anatomical position and spatial placement of the heart lungs, spleen liver, and other internal organs within the body are essential. 3D models of anatomical specimens produced by augmented reality will aid the anatomical study. Easily accessible 3D models in every handset will eliminate the requirement of labs when labs are not accessible, thus easing out the midnight study.

Many industries pose real-time computation requirements in AR, as in the case of medical applications, but do not implement an efficient algorithm in their frameworks. One such startup AiBorne Tech [2] is working on an AI-AR vehicle assessment application. This will involve communication with a wide variety of automobiles and sensing the issues in them. Definitely, AI models will come into play here for the anomaly detection. Therefore, it becomes important to enable smooth communication in the mobile app for the best output. On a second note, Abhiwan Technologies [1] is another startup working on VR Rooms for virtual networking. However, the networking feature will be implemented best when their communication with multiplayer networking APIs will not be a hindrance. Hence, usage of optimized and fast APIs will be a bottleneck for many industries resorting to AR/VR mobile apps. These issues can be solved in future work for such kinds of applications.

Offloading the computation of deep learning models involved in the application to the cloud servers will highly affect the processing parameters. The optimization of communication between local application and cloud computing frameworks needs to be optimized. The proposed work presents a proof of concept where the user scans the head of a human, and the feed of the image is continuously sent to OpenCV API. The API analyzes the image and locates the shape of the user's head, returns the essential coordinates to the local mobile application, and finally, the skull will be deployed in Augmented Reality.

This paper defines how the parameters involving the complete deployment of a mobile AR application can be optimized in terms of retrieval of multimedia, its processing, and augmentation of graphics, eventually enhancing the performance. The primary goal of the proposed work is to study how the ARCore SDK can be equipped with a deep learning model to implement face tracking feature, which otherwise is not present as a plugins in Unity, unlike ARKit Face Tracking plugin.

Furthermore, this paper elucidates the paradigm wherein the variation in the latency of the application is studied with the change in the frame rate of the device. The latency parameters are divided into three categories; **Media Retrieval Time**, which is the time taken to retrieve the media from the surroundings of the device, second **API Process Time** which is latency variations in the API to process the image, and **Client Time** which is the time taken to change the position of the 3D model (skull) in mobile device.

The remaining paper is organized as follows. Section 3 gives an overview of the related work in the field of AR, deep learning, medical applications. Then, Section 4 emphasizes on the tech stack involved in the development of the application, where the pipeline for the implementation of the work and the methodologies to execute the optimization in the framework. Section 7 evaluated the results followed by the conclusions.

2 Motivation

The upsurge in the AR/VR market has led to the development of numerous game engines like Unity3D [12], Unreal Engine [13], Amazon Lumberyard [3], and CryEngine [6]. They have provided SDKs to assist mixed reality developers. Unity3D is the most widely used platform and has plugins of ARFoundation, ARCore, ARKit, and Vuforia. Many mobile AR

applications need high computing resources for heavy graphical computations like lighting, mesh rendering, material shading, and mesh filtering. It is evident that software grows manifolds with respect to the hardware. It, therefore, becomes essential to exploit the capabilities of the software to access the multi-processing power of the hardware. For instance, Unity3D uses single thread operation by default and that poses a bottleneck for heavy processing applications like that of AR. This also limits the capability to use the hardware of the device. To solve this issue, Unity provides a provision of the job system to invoke multithreading using the Entity-Component System [7]. Hence it becomes essential to resort to software prowess to smoothen the applications. In a similar way, a system backed by a cloud-based deep learning model can be helpful for face tracking applications.

Since the primary purpose of such graphical applications is to provide the best user experience, it is important to enhance their functioning. A combination of both, local mobile app and processing at cloud or edge can result in a hassle-free experience. Therefore the need to strengthen the intuitiveness led us to research this prevailing topic and provide a noble framework to achieve the results.

3 Related work

Over the past decade, the uproar in the MARS applications [31] has transformed the course of many industries. The cognition of young students is mostly exploited in many applications of AR. Saidin et al. [28] defined the scope of AR in education. The crux of the paper elucidates the influence of interaction and intuition among students. This innovative technology allows users to interact with virtual and real-time applications and brings a natural experience to the user. In addition, the fusion of AR and education has recently attracted research attention because of its potential to encourage students to immerse themselves in practical experiences. However, the access to such applications is limited by the hardware required for rendering heavy graphics. Therefore, a trade-off between graphics and smoothness is a prime factor for programming AR applications.

In addition to this, the scope of AR/VR has influenced the study of biological specimens and anatomies. The potential of AR to demonstrate everything in 3D and real-time, enables budding doctors to study and engage with the concepts simultaneously. It is evident that current trends for animal and human rights have limited the use of actual specimens for analysis, course study, and scenario writing process for professional studies. Arslan et al. [14] defined that although the AR industry had made its mark in the field of physics, chemistry, engineering etc., yet the biological field of study remains unexplored with respect to MARS applications. It has proved that the pipeline of developing 3D models for complex laboratory biological experiments and then integrating them into AR technology for university students is plausible. On an industrial level, the accuracy of the AR applications is vital. As in case of biology, a precise and accurate application will enhance the user experience and an inaccurate and imprecise application would do otherwise.

Now that it is evident that AR can significantly boost the anatomical and skeletal study in the field of biology, the necessary thing is to make applications compatible for all users. Wang et al. [32] has covered the major aspect of compatibility issues in MARS frameworks. Perception, interpretation, and cognition being the major factors affecting the working of the applications, were studied in their work. The device on which the application is running, like a mobile device in our proposed work, must complement the perceptual environment involved in the manual task. The choice of spawned models affects cognitive compatibility and how the user perceives the physical aspects of the same. Varied cognitive capabilities of

users produce different perceptions of the same application and cognitive model. Finally, the type of task determines the display device being used for the application on the basis of the egocentric and exocentric task continuum. Although the device is expected to complement the perceptual environment, it is important to program a responsive application that can quickly personalize the experience for the user.

Further, a vital obstacle lies in the fact that backend computations involve substantial time consumption. The quality of object tracking is a crucial factor in providing a seamless AR experience, as defined by Zhang et al. [34]. More degrees of freedom will provide a better personalized user experience. An edge cloud server is used to compute the large amount of data in the process. Lampropoulos et al. found that the efficiency of the deep learning models increase only when a large amount of data is trained and processed [19]. Specifically for a personalized AR application, that too for anatomical and skeletal study, acquisition of robust and appropriate data is essential [25]. This can not be covered by using publicly available datasets on the internet [15]. Further, to tackle the dynamic changes in appearances of neighbourhood, Datta et al. in [16], proposed to use Content-based Image retrieval technique. This will enhance the image detection in multiple outdoor and indoor conditions, making the application flexible to changes.

The proposed work primarily focuses on proposing a paradigm to equip the medical industry with a dynamic AR face tracking application to study a personalized skeletal system. The proposed application can completely be deployed on a mobile device and works similar to real-time while retaining excellent recognition efficiency. Further, to maintain a flexible recognition system, the Multi-Task Cascaded Convolutional Neural Networks (MTCNN) algorithm [33] has been used to run on a mobile device. However, the computations are offloaded to the cloud GPU, deployed on an AWS (EC2) instance. As of now, most of the deep learning computations are held locally within the device. This adversely makes the application slow. Therefore, by using a cloud network for the computations, the latency effect can be resolved. Moreover, the essential factor studied in this work is related to the repercussions of the frame rate of the device. In order to deploy a robust and versatile AR application, it is necessary to consider the varying device parameters. The application should run in a similar fashion in all the target devices. Therefore collectively, this work deals with challenging the frame rate and deduce the prime consequences of the frame rate on the application latency.

4 Technologies used

The most popular platform to develop AR applications is Unity3D. It is equipped with all sorts of plugins and assets to create MARS applications. Some of them require third-party APIs and SDKs to develop. The application being described in this paper makes use of ARCore for extraction of multimedia. An OpenCV API, as a cloud computation framework processes this multimedia. Finally, ARCore again spawns graphics based on the processed multimedia. These concepts are described below in detail.

4.1 ARCore framework

Unity provides numerous XR frameworks to deploy MARS applications. For the face tracking applications, ARKit provides ARKit Face Tracking plugin. However, instead of a separate plugin, a feature is embedded in ARCore itself to use face tracking in AR applications [4]. The features that ARCore XR Plugin (4.0.4) supports are efficient background

rendering, horizontal planes, depth data, anchors, and hit testing. It uses a face subsystem for face tracking. Face subsystems work on the principle of tracking subsystems [8]. However, the compatible Unity versions are limited - 2019.4, 2020.3, 2021.1, 2021.2. Due to this limitation, this paper intends to create a universal algorithm for face tracking independent of unity versions.

Moreover, when a plugin is imported into a unity project, many unrequired files and scripts are added to the project. This increases the size of the application and hence poses a hindrance in the app at run time. ARCore face tracking system will eventually trade-off features with performance. The provided solution will cater to this anomaly too. In this use case of AR, we developed an OpenCV API to interact with the AR application for face tracking feature. All the computations of the API are implemented on the AWS server.

4.2 ARCore vs ARKit

Google launched its platform for AR developers to build augmented reality experiences - ARCore. Its key capabilities are motion tracking, environmental understanding, and light estimation, which enable a vast pool of use cases like depth sensing, face tracking, image tracking, occlusion mapping, floor-wall mapping, and many more. It supports both, android and iOS devices.

However, Apple's ARKit enables the integration of AR applications and motion features in iOS devices only. A complete pooling of features like camera scene capture, device motion tracking, display conveniences, and advanced scene processing smoothens the task of building AR experiences.

4.3 Computer vision and deep learning model

The unity application captures the image and the binary data retrieved from it is sent to the API. The next step in the proposed work is to detect the human face coordinates on the received image. In computer vision, face detection involves locating and localizing one or more faces in a photograph. A number of deep learning methodologies have been developed and demonstrated for face detection over the past decade but what makes MTCNN stand out is its high accuracy and low latency. MTCNN uses a cascade structure with three multi-tasks networks. Figure 2 presents the architecture of the layers used in each of the stages in the cascaded MTCNN model.

As shown in (3), firstly the image is re-scaled to a range of different sizes (also referred to as an image pyramid), then in the MTCNN, the three cascaded stages follow ascending steps:

1. **P-Net:** The first network (Proposal Network or P-Net) proposes candidates for facial regions. The feature map obtained by the forward propagation is a 32-dimensional feature vector at each position. It is used to ascertain whether or not grid cells of 12×12 contain a human face. If a grid cell contains a face, the Bounding Box of the face is regressed, and the Bounding Box corresponding to the area in the original image is also obtained. The Bounding Box with the highest score is retained by a Non-maximum suppression (NMS) step and the rest of the Bounding Boxes with an excessively large overlapping area are excluded.
2. **R-Net:** The second network (Refine Network or R-Net) filters out the bounding boxes. Similar to the previous stage (O-Net), the 24×24 and the resulting Bounding Box area are up-scaled to 48×48. It is then used as input to the R-Net stage to get the highest detection confidence of Bounding Box detection and facial landmark extraction.

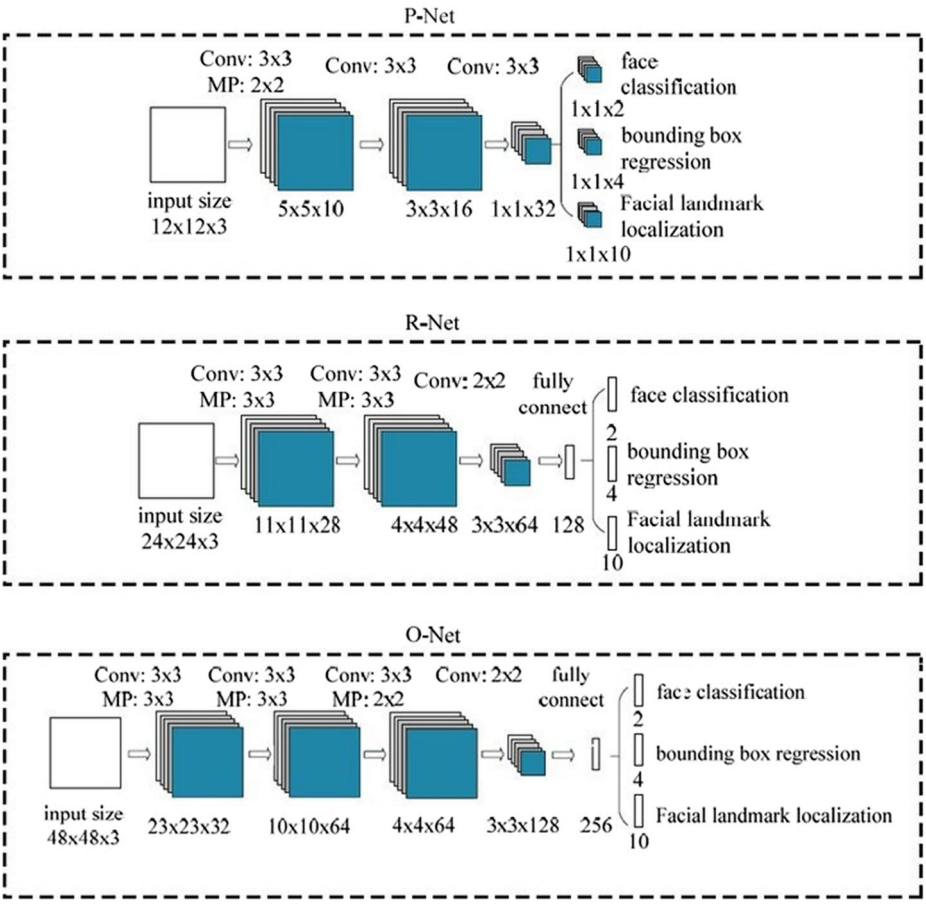


Fig. 2 Stage architecture of the MTCNN model used for face detection

3. **O-Net:** The third network (Output Network or O-Net) proposes facial landmarks in the image. The Bounding Box that the P-Net step gives may or may not contain a human face. This box as well as the 12×12 input are then up-scaled using a bi-linear interpolation method to 24×24, which is then used as the input to the O-Net to find the face coordinates. If a human face is present, the Bounding Box is regressed, which is then followed by the NMS step.

An optional feature of MTCNN is detecting facial landmarks, i.e. nose, eyes, and corners of the mouth. It comes at almost no computational expense since they are used anyway for face detection in the process (Fig. 3).

4.3.1 Training methodology

MTCNN uses a cascade structure with three multi-tasks networks. The three main tasks performed by each network are as follows:

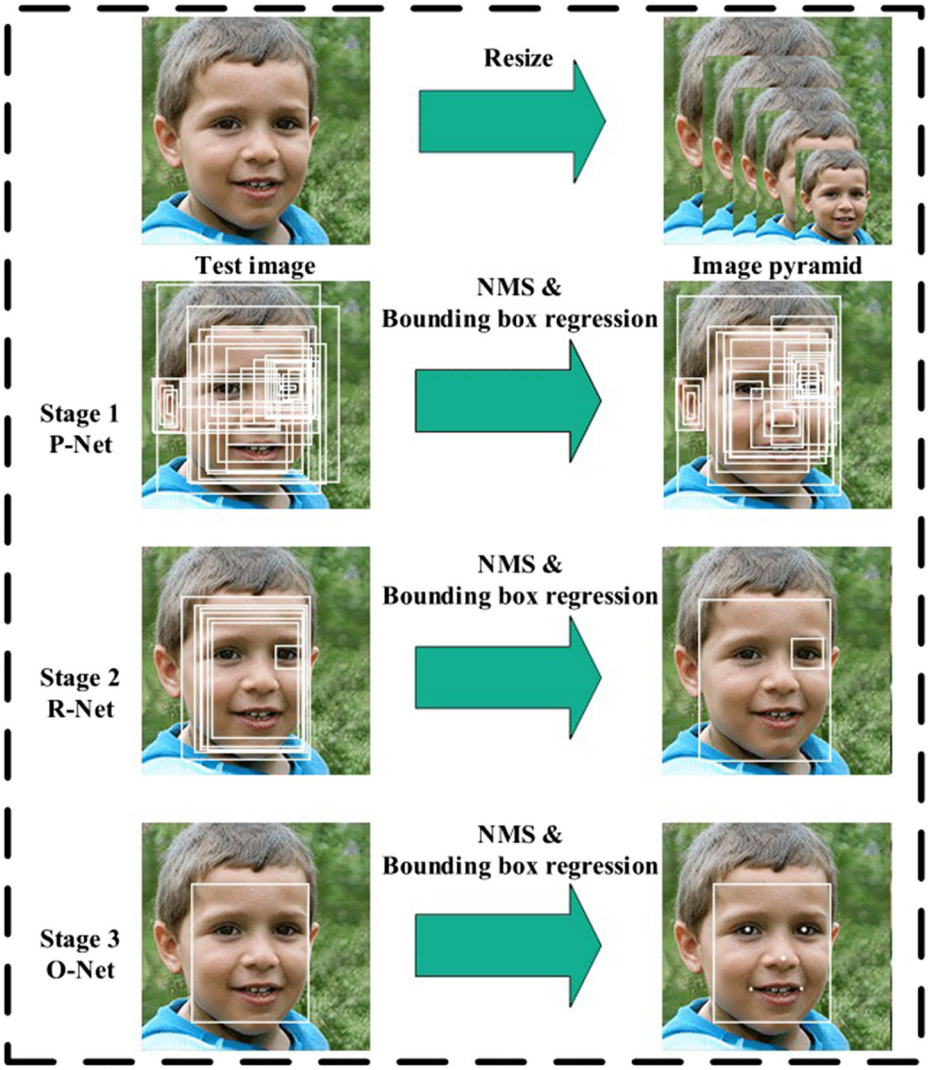


Fig. 3 Stage architecture of the MTCNN model used for face detection

a) **Face Judgement:** This is a binary classification problem. For each sample it uses cross-entropy loss function given as [17]:

$$L_i^{det} = -\left(y_i^{det} log\,(p_i) + \left(1 - y_i^{det}\right)(1 - log\,(p_i))\right) \tag{1}$$

p_i is the probability produced by the network that the i^th face sample predicted by the MTCNN model is a real face, y_i^{det} presents the ground-truth for the i^{th} sample, and $y_i^{det} \in 0, 1$.

b) **Bounding-Box Regression:** For each candidate window, the offset (such as the bottom right and top-left coordinates, the height, and the width) between it and the nearest ground-truth is predicted. The loss function is the square loss function [17]:

$$L_i^{bb} = ||y_i^{\wedge bb} - y_i^{bb}||_2^2 \tag{2}$$

$y_i^{\wedge bb}$ is the regressed target from the network. y_i^{bb} is the ground-truth four-dimensional coordinate, including the top-left coordinate, the height, and the width.

c) **Feature Location:** It is similar to Bounding-Box Regression. The loss function is as follows:

$$L_i^{landmark} = ||y_i^{\wedge \text{landmark}} - y_i^{\text{landmark}}||_2^2 \tag{3}$$

Likewise, $y_i^{\wedge \text{landmark}}$ is the regressed feature coordinate from the network. $y_i^{landmark}$ is the ground-truth containing five coordinates: two eyes, two corners of the mouth, and the nose.

Caffe framework is used to implement the original MTCNN face detection model and it uses 'PReLU' layers within its architecture. TensorRT library does not support parsing 'PReLU' layers directly from the prototxt/caffemodel files, which poses a big problem since TensorRT is likely the fastest way to run a model at present time. To solve this problem we converted all PReLU layers into 2 branches of ReLU layers as shown in Fig. 4 [24]. As a result, the TensorRT API can directly parse and optimize the converted Caffe models.

4.4 Cloud computation

In the proposed work, the main computation involves the use of a deep learning model. The data of the image is sent to the API, which in turn returns the coordinates of the face. The primary quality of APIs is based on, on-demand working. The APIs work only when the mobile application tends to interact with it. It enables design of expandable network design for a simple mobile system.

The proposed work makes use of Amazon Web Service Elastic Compute Cloud (EC2) [18] for the deployment of API. Using this service, a remote server is enabled for the OpenCV model. Based on seamless network security protocols, AWS cloud computing platform not only eases out the deployment but also assists in debugging the interaction between main application and the deep learning model.

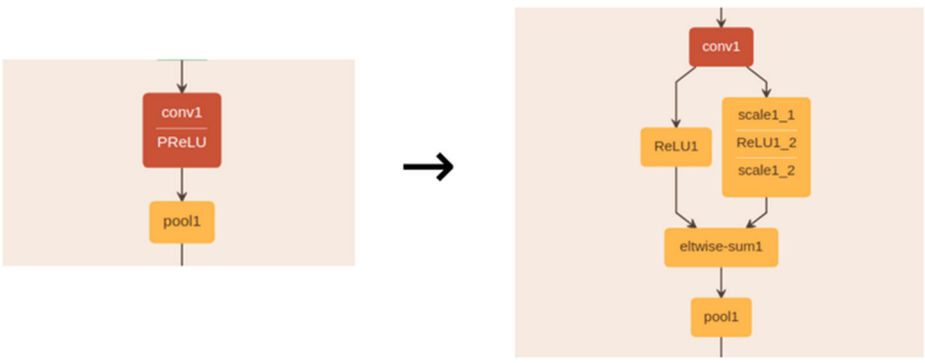


Fig. 4 Conversion of all PReLU layers into 2 branches of ReLU layers

4.5 Edge computation

The new age notion of edge computing has applications which can substantially affect the development of computational frameworks [30]. It finds applications in fleet management, autonomous vehicles, voice assistant devices such as Alexa and Google Home, and predictive maintenance. Instead of sending the data to cloud data centers, edge computing decentralizes processing power to ensure real-time computation. This drastically reduces the latency by affecting the bandwidth and storage consumption on the network.

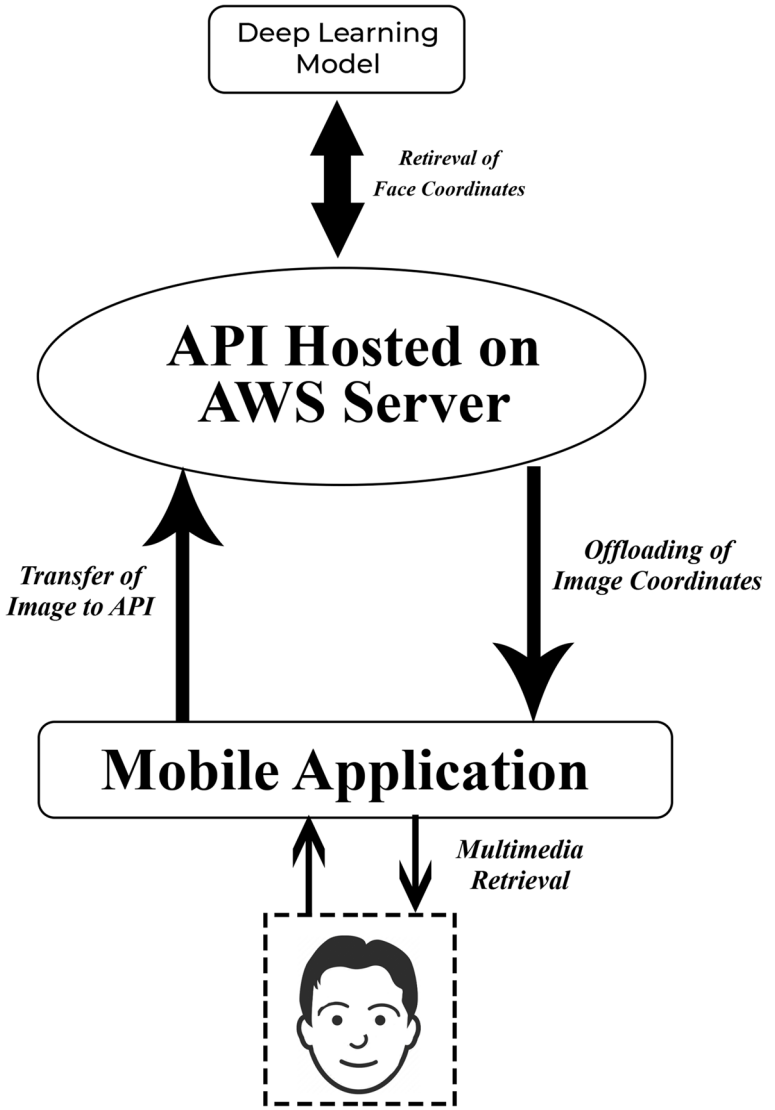


Fig. 5 Block Diagram of the Application Pipeline

Therefore it is evident that edge computing finds use cases in the scenarios where fast processing of data is required for better communication among the devices. In the use case of this paper, a mobile device running this application is a standalone node. However, if the application is deployed in a multiplayer learning environment, then an edge computational model can be deployed for communication between all the devices. Additionally, parameters that are sent to a cloud server and an edge server are the same in the case of face tracking API.

5 Working pipeline

After the tools and technologies being worked upon are understood, their actual implementation to realise the concept is described in this section. The Fig. 5 depicts the flow of the processes in the application.

As it can be inferred from the Fig. 5, the retrieval of multimedia data starts from the subject present in front of the mobile application. The Unity mobile application captures the position and orientation of the person present in front of the camera. The data of the image is converted into binary form and then sent to the API. Henceforth, the API interacts with the deep learning neural network to fetch the coordinates of the face. The processing of the data results in generation of two sets of coordinates *top_left_corner* and *bottom_right_corner*. These are sent to the application, where the center of the image is located by the general midpoint formula of cartesian geometry. Finally the 3D model of the skull is spawned on the midpoint coordinates perceived. Primarily two blocks processes needed to be taken care of - the code running in the mobile device and the algorithm of API on the server. The end nodes for this system are human face target on the end of the mobile device and a deep learning face tracking model on the end on the API.

The x and y coordinates of the top-left corner (X_t, Y_t) and bottom right corner (X_b, Y_b) are extracted. Then the midpoint formula is applied to get the center of the face (X_c, Y_c),

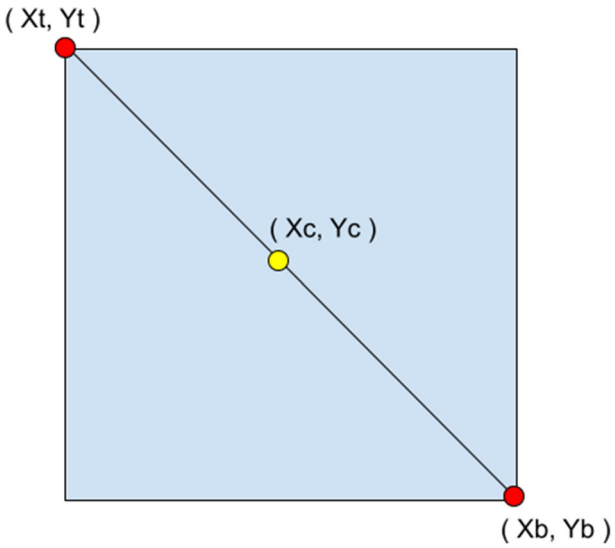


Fig. 6 Points on the Cartesian Plane

as shown in Fig. 6. Since the points are with respect to 2D mode, they have to be converted into 3D space using $ScreenToWorldPoint()$ function.

Cartesian mid-point formula:

$$X_c = \left(\frac{X_t + X_b}{2}\right) \tag{4}$$

$$Y_c = \left(\frac{Y_t + Y_b}{2}\right) \tag{5}$$

6 Optimization parameters

The prime target of this work is to show the effect of external API on the latency in the AR mobile system. The two major functions during the complete process are as follows:

a) Retrieval of Image from the Unity application and then calling of the cloud API to detect the face.
b) Processing of the media feed in the deep learning model.

Therefore, the factors on which latency tends to depend on, is the number of times the media is sent to the API (Frame Rate) and how much time it takes to process the same (API Processing Power)

6.1 Frame rate

Unity applications tend to work faster on 30fps rather than 60fps mobile applications [12]. This is true primely for AR applications, and as in our proposed work, this factor affects the latency when the API is hit. At the instant when the media is transferred from the application towards the cloud, a screen-shot is taken to determine the position of the face. This is taken in a Unity Coroutine. In the coroutine, a statement *yield return WaitForEndOfFrame()* is used to capture the screen. This function is dependent on the frame rate of the mobile system. Hence this affects the timing of the initial image retrieval.

Unity provides a function *void Update()*, that runs on every frame. It seems that the proposed API can be hit using this function at every frame to change the position of the 3D model of the skull with respect to the face. However, the results of this practice are opposite of that expected. This will increase the RAM usage, hence increasing the latency further. To tackle this problem, a flexible timer is created, upon completion of which the retrieved media is sent to the cloud for processing. Moreover, it gets reloaded automatically every time.

6.2 API processing power

Next, that the binary coded image data is handed over to the API, the whole application waits for the deep learning model to process the image. The whole process stops until the image gets processed and the coordinates are returned. Eventually the API efficiency becomes an essential part of the process.

Flask [10] is currently the prime choice for writing these APIs but instead FastAPI [9] is considered in this work. FastAPI is a high-performance, web framework for building APIs with Python [9]. It increases the speed to develop features by about 200% to 300% and also

reduces about 40% of human (developer) induced errors. It significantly minimizes code duplication and bugs. FastAPI is built on top of Uvicorn, an Asynchronous Server Gateway Interface (ASGI) server, whereas Flask is designed for Web Server Gateway Interface (WSGI) servers like Gunicorn. Under the hood, FastAPI uses Pydantic for data validation and Starlette for tooling, making it much faster than the Flask. Its performance is comparable to high-speed web APIs in Node or Go. The whole work is summarized in Algorithms 1 and 2

As it can be inferred from the algorithm 1 that a coroutine in unity is used. Coroutines in unity are invoked when there is a process whose completion time is not specific or it requires multiple frames to process [5]. Since it is unknown that when the API will send the processed data, hence the code requires a coroutine. Moreover, the statement - yield return w, pauses the application to receive the HTTP web request before resuming the application.

1: Wait for end of frame.
2: Capture the screen-shot as a Texture 2D type object.
3: *EncodeTOJPG()*.
4: Store JPG as a byte array
5: Create an instance of *WWWForm form*
6: Add the encoded binary data of byte array to the form using *form.AddBinaryData()*
7: Generate a *WWW* request *w*
8: Initiate the web request *w* by - yield return *w*
9: Extract the *top_left_corner* and *bottom_right_corner* coordinates from the string response.
10: Calculate the centre of the image.
11: Convert screen coordinates to world point form using *ScreenToWorldPoint()* function
12: Update the local position of skull with regard to the world point coordinates.
13: **OUTPUT:Position of skull is changed according to the coordinates obtained from the API**

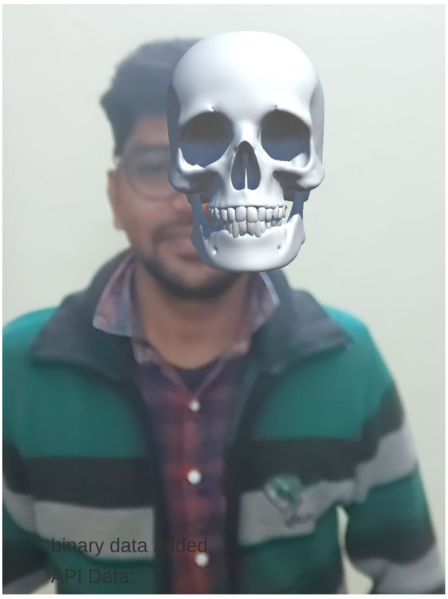
Algorithm 1 Coroutine to capture image send it to API.

Algorithm 2 MTCNN algorithm.

1: Binary image data is received for the API post request.
2: The Image is passed onto the MTCNN model.
3: Resizing the image with different scales and constructing image pyramid.
4: Proposal Network or P-Net proposes candidates for facial regions.
5: Refine Network or R-Net filters out the bounding boxes.
6: Output Network or O-Net proposes facial landmarks in the image.
7: Face coordinates received from this MTCNN model are sent as response for further processing.
8: **OUTPUT:Image coordinates are extracted and sent to C# application**.

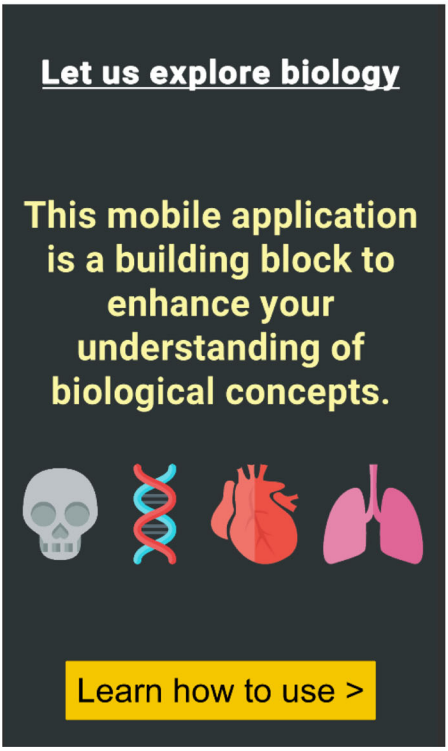
7 Performance evaluation

In this section, the performance of proposed algorithm is evaluated against various parameters. An android application was first created and then tested on SM-A307FN (Android API Level 10). The results are interpreted based on what the user sees in the application

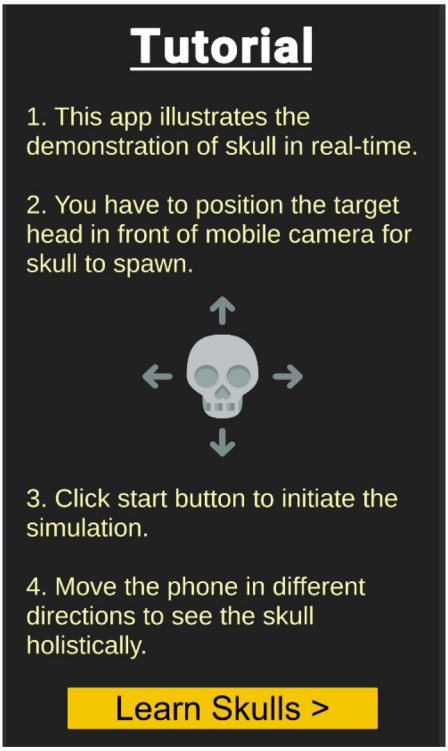


(a) Changing Position of Face

(b) Skull follows the Face



(c) Home Screen of Mobile Application



(d) Tutorial Screen of Mobile Application

Fig. 7 Screenshots of the Mobile Application

Table 1 Key system parameters

Parameter	Quantitative Value
API hitting frequency	Once per second
Mobile Device	SM-A307FN
Image Resolution	720 x 1560 pixels, 19.5:9 ratio (268 ppi density)
Device RAM	16 GB
Device CPU	CPU Octa-core (2x1.8 GHz Cortex-A73)
Device Camera (Rear)	25 MP, f/1.7, 27mm (wide), PDAF

while using it and graphical representation of latency parameters. Some of the parameters are listed in Table 1.

Other than the parameters mentioned above, the testing of the mobile application required some specific conditions to be fulfilled. Target was asked to sit on a chair and the mobile phone was mounted to ensure a fixed position.

In the proposed experiment, the generated dataset consists of the following parameters and the data is extracted as follows:

a) **Time taken to capture the image (Media Retrieval Time)** - In Algorithm 1, a time stamp was included, enclosing the step numbers 2 and 3. This provided the time difference for the capturing of image from the application.
b) **Time taken to process the image and obtain the coordinates (API Process time)-** In Algorithm 1, a time stamp was included, enclosing the step numbers 7 and 8. This provided the time difference for obtaining the end points of the image.
c) **Time taken to calculate the exact location and spawn the 3D model (Client Time)-** In Algorithm 1, a time stamp was included, enclosing the step numbers 9 to 12. This provided the time difference for augmenting of 3D model.

The Unity AR android application spawns a 3D model of the skull after the complete backend processing. The user can see these images in the sequence, like Fig. 7a and then b.

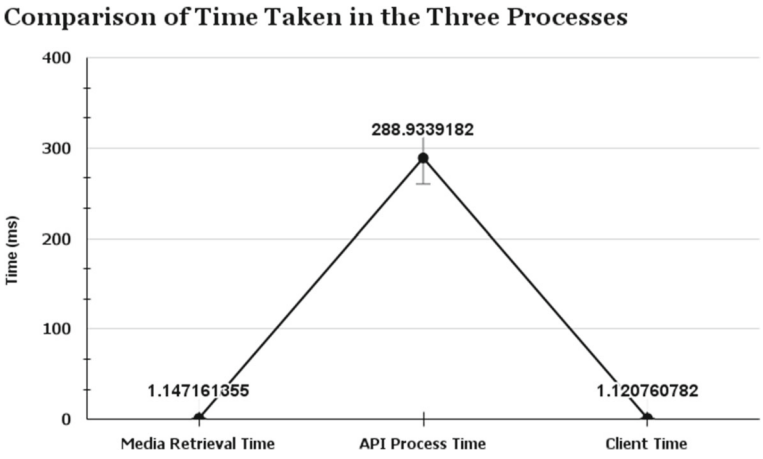


Fig. 8 Comparison of the Average Time Taken during the Three Processes: Media Retrieval, API Process, Client Process

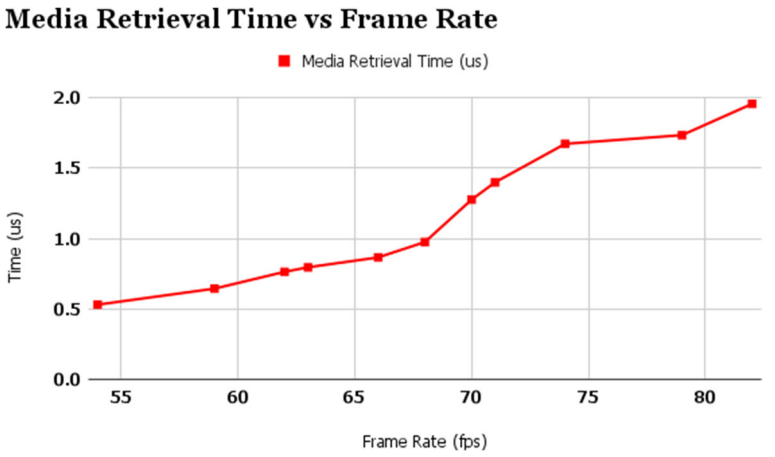


Fig. 9 Media Retrieval Time vs Frame Rate

Deducing form the same, as the target face changes its position, the skull reorients itself in the 3D space. Moreover, since we have the extreme coordinates of the face, the size of the skull can be modified with respect to the human body acting as the target. This has made the application more reactive.

Next, comparison of average time taken during the three processes Media Retrieval, API Process, and Client Process is done. It can be observed collectively about the three sub-processes in the Fig. 8 where it clarifies the time difference between the processes. Clearly, the average Media Retrieval Time (1.1471ms) and Client Time (1.1207ms) in the local application are extremely low than the average API process time (288.934ms). The average time of latency is based on the changes incurred due to the change in the frame rate. The highest time latency is achieved at the frame rate higher than 80fps.

Next for more clarity, these three processes are compared against different frame rates. As evident from Figs. 9, 10, and 11, the time taken by the proposed algorithm in the different

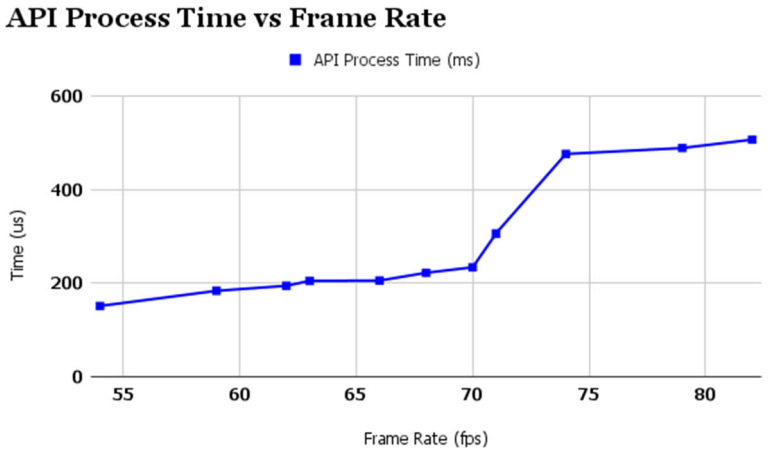


Fig. 10 API Process Time vs Frame Rate

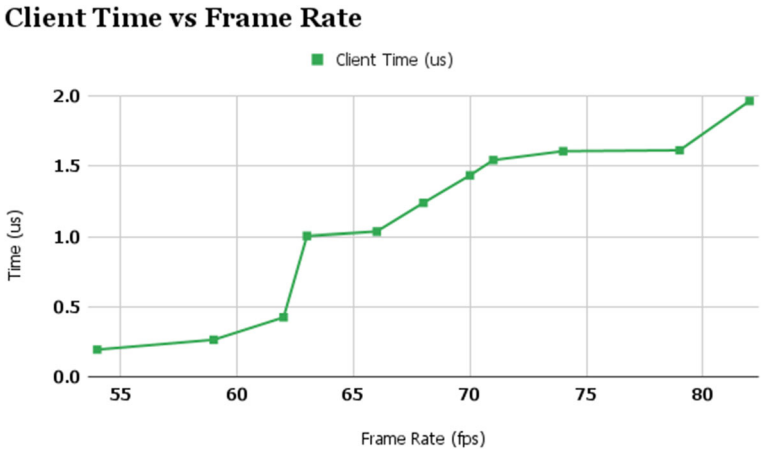


Fig. 11 Client Time vs Frame Rate

segments keeps on increasing with the frame rate of the device. The sample size of frame rate considered here ranges from 54fps to 82fps. Although the device is deployed on a cloud GPU, yet the latency is affected by the device's frame rate. The effective influence of frame rate on the three sub-processes is evaluated.

It is evident from Fig. 9 that Media Retrieval time solely depends on the mobile application. The algorithm mentioned in Algorithm 1 is responsible for the part of capturing the data from the surroundings. The screenshot is captured only after a frame ends. $WaitForEndOfFrame()$ function takes care of this. As evident from the function name, the frame rate directly affects the rate of capturing the screen. In this process, the minimum latency exists for 54fps - 0.5305176 μs and extends to a maximum at 82fps - 1.95752 μs. These results are recorded in Table 2.

Next, API Process Time is evaluated against frame rate in Fig. 10. API process time is the main part of the complete application. As evident from the data presented in the Fig. 8, it takes up most of the time in the application. The effect of frame rate can be seen from the graph as, the minimum time is around 151.5467 ms at 54 fps and maximum is around 507.3544 ms at 82fps. This is the prime sub-process to be optimized. This is done by offloading the computation to the cloud GPU server. The face detection algorithm used here has the least latency for a constant frame rate as it is deployed using FastAPI. As in the case of frame rate, the API is hit only after the frame ends, similar to the case of media retrieval. Therefore subsequently, as the frame rate increases, the time taken by the application automatically increases. Next, Client Process Time is evaluated against frame rate if Fig. 11. After getting the coordinates from the API locally, the process of changing the position of the skull is dependent on Algorithm 1. C# code handles the processing of the

Table 2 Time Taken for Different Parameters

Parameter	Time
Media Retrieval Time (μs)	1.147161355
API Process Time (ms)	288.9339182
Client Time (μs)	1.120760782

Table 3 Time Taken for Different Parameters

Frame Rate (fps)	Media Retrieval Time (μs)	API Process Time (ms)	Client Time (μs)
54	0.5305176	151.5467	0.1936035
59	0.6444092	183.9989	0.263916
62	0.763916	194.7776	0.4238281
63	0.7960205	205.3095	1.003906
66	0.8662109	205.899	1.035156
68	0.9752197	222.6494	1.239014
70	1.2771	234.3984	1.435791
71	1.399414	306.3757	1.614502
74	1.67334	476.5838	1.608154
79	1.735107	489.3797	1.544434
82	1.95752	507.3544	1.966064

coordinates, determining the centre of the face and subsequently altering the position of the skull. The function $ScreenToWorldPoint()$ converts the coordinates of the pixel to a 3D coordinate system. Since, the alteration in the position of the skull happens at every frame, frame rate largely affects this process. Hence, the latency value ranges from $0.1936035\mu s$ to 1.966064 μs for the frame rate range 54fps to 82fps.

With the increase in frame rate, the smoothness increases, but simultaneously this increases the computational cost required by MTCNN to process every frame, thus boosting the latency of our application. As a prior analysis, the algorithm was formulated to use $WaitForEndOfFrame()$ to hit the API. The API gets called after the end of frame. So, if the frame rate increases, then the API processing will increase the API Process time. Therefore, the results show the expected output. This is also true for media retrieval time as well, as the media (image) is captured after the end of frame. Lastly, the client time also depend on frame rate as the position of skull is updated after every frame. Hence, the results of client time are also as expected. Although it is required to have a higher framerate for a smooth execution, the computational cost will be an added disadvantage. Consequently, there has to be an optimum framerate to decide the tradeoff between latency and smoothness. For instance, a high graphic mobile game will prefer quality over the delay, and an AR medical application will work on precision. Result of achieved Media Retreival Time, API Process Time and Client Time against different frame rate is recorded in Table 3.

8 Conclusions

The retrieval of multimedia in AR plays a significant role in deploying real-time intuitive systems. AR combines the essence of digital reality with the physical world. Inferring the physical world where the application is being used has become crucial. Due to this, the application interacts with a substantial amount of data. To cater this, proposed work uses deep learning technique. The objective of the proposed work is to develop a framework to study the effects of varying frame rate on the mobile AR application. The results conclusively proved that the latency in the application gradually increases with increase in the frame rate of the application. The API takes up the maximum time in the application, compared to Media Retrieval Time, and Client Time. The highest time latency is also achieved at

the frame rate higher than 80fps. It was seen that performance was improved by off-loading the deep learning computation to a cloud GPU server. In future, the AR model can also be employed for the study of physiological processes (cardiac cycle, counter current mechanism, blood clotting, neuromuscular transmission, muscular transmission); pathology of diseases, effects of drugs on a body at the molecular level, enzymatic actions, the study of fractures and other aspects of medicine.

Declarations

Conflict of Interests The authors declare that there is no conflict of interest with this work.

References

1. Abhiwan https://www.abhiwan.com/portfolio, Accessed 10 June 2021
2. Aiborne tech https://aiborne.tech/, Accessed 10 June 2021
3. Amazon lumberyard https://aws.amazon.com/lumberyard/, Accessed 10 June 2021
4. Arcore face tracking https://docs.unity3d.com/Packages/com.unity.xr.arcore@4.1/manual/index.html, Accessed 10 June 2021
5. Coroutines https://docs.unity3d.com/Manual/Coroutines.html, Accessed 10 June 2021
6. Cryengine https://www.cryengine.com/, Accessed 10 June 2021
7. Entity component system https://docs.unity3d.com/Manual/JobSystem.html, Accessed 10 June 2021
8. Face subsystems https://docs.unity3d.com/Packages/com.unity.xr.arsubsystems@4.1/manual/index.htmltracking-subsystems, Accessed 10 June 2021
9. Fastapi https://fastapi.tiangolo.com/, Accessed 30 Nov 2020
10. Flask https://flask.palletsprojects.com/en/2.0.x/api/, Accessed 10 June 2021
11. The future of reality is here: Jio glass https://tesseract.in/, Accessed 30 Jan 2021
12. Keep it cool: three fast optimization tips for ar https://unity3d.com/how-to/fast-optimizations-for-AR, Accessed 30 Nov 2020
13. Unreal engine https://www.unrealengine.com/en-US/, Accessed 10 June 2021
14. Arslan R, Kofoğlu M, Dargut C (2020) Development of augmented reality application for biology education. J Turk Sci Educ 17(1):62–72
15. Barbosa IB, Cristani M, Caputo B, Rognhaugen A, Theoharis T (2018) Looking beyond appearances: synthetic training data for deep cnns in re-identification. Comput Vis Image Understand 167:50–62
16. Datta R, Li J, Wang JZ (2005) Content-based image retrieval: approaches and trends of the new age. In: Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval, pp 253–262
17. Du J (2020) High-precision portrait classification based on mtcnn and its application on similarity judgement. In: Journal of Physics: Conference Series, vol 1518. IOP Publishing, p 012066
18. Emeras J, Varrette S, Plugaru V, Bouvry P (2016) Amazon elastic compute cloud (ec2) versus in-house hpc platform: A cost analysis. IEEE Trans Cloud Comput 7(2):456–468
19. Lampropoulos G, Keramopoulos E, Diamantaras K (2020) Enhancing the functionality of augmented reality using deep learning, semantic web and knowledge graphs: a review. Vis Inform 4(1):32–42
20. Milgram P, Takemura H, Utsumi A, Kishino F (1995) Augmented reality: a class of displays on the reality-virtuality continuum. In: Telemanipulator and telepresence technologies, vol 2351. International Society for Optics and Photonics, pp 282–292
21. Nowacki P, Woda M (2019) Capabilities of arcore and arkit platforms for ar/vr applications. In: International conference on dependability and complex systems. Springer, pp 358–370
22. Oishi M, Fukuda M, Yajima N, Yoshida K, Takahashi M, Hiraishi T, Takao T, Saito A, Fujii Y (2013) Interactive presurgical simulation applying advanced 3d imaging and modeling techniques for skull base and deep tumors. J Neurosurgery 119(1):94–105
23. Peng F, Zhai J (2017) A mobile augmented reality system for exhibition hall based on vuforia. In: 2017 2nd International conference on image, vision and computing (ICIVC). IEEE, pp 1049–1052
24. PKUZHOU Pkuzhou, https://github.com/PKUZHOU/MTCNN_FaceDetection_TensorRT. Accessed 26 Feb 2022

25. Planche B, Wu Z, Ma K, Sun S, Kluckner S, Lehmann O, Chen T, Hutter A, Zakharov S, Kosch H et al (2017) Depthsynth: real-time realistic synthetic data generation from cad models for 2.5 d recognition. In: 2017 International conference on 3D vision (3DV). IEEE, pp 1–10
26. Radu I (2014) Augmented reality in education: a meta-review and cross-media analysis. Personal Ubiq Comput 18(6):1533–1543
27. Rice M, Wright V, McMath J, McNeill L, Wilson M (2019) Virtual reality: offering virtually unlimited application. In: E-Learn: world conference on e-learning in corporate, government, healthcare, and higher education. Association for the Advancement of Computing in Education (AACE), pp 567–571
28. Saidin NF, Halim NDA, Yahaya N (2015) A review of research on augmented reality in education: advantages and applications. Int Educ Stud 8(13):1–8
29. Schuir J, Vogel J, Teuteberg F, Thomas O (2020) Understanding the augmented and virtual reality business ecosystem: an e 3-value approach. In: International symposium on business modeling and software design. Springer, pp 240–256
30. Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge computing: vision and challenges. IEEE Int Things J 3(5):637–646
31. Verhey JT, Haglin JM, Verhey EM, Hartigan DE (2020) Virtual, augmented, and mixed reality applications in orthopedic surgery. Int J Med Robot Comput Assisted Surg 16(2):e2067. https://onlinelibrary.wiley.com/doi/abs/10.1002/rcs.2067
32. Wang X, Dunston PS (2006) Compatibility issues in augmented reality systems for aec: an experimental prototype study. Autom Construct 15(3):314–326
33. Zhang K, Zhang Z, Li Z, Qiao Y (2016) Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Process Lett 23(10):1499–1503
34. Zhang W, Han B, Hui P (2018) Jaguar: low latency mobile augmented reality with flexible tracking. In: Proceedings of the 26th ACM international conference on multimedia, pp 355–363

Affiliations

Yash Prakash Gupta[1] · Mukul[1] · Nitin Gupta[2]

Yash Prakash Gupta
yasharsh.28@gmail.com

Mukul
mukulthakur17.mt@gmail.com

[1] Department of Electronics Communication and Engineering, National Institute of Technology, Hamirpur, Himachal Pradesh, India

[2] Department of Computer Science and Engineering, National Institute of Technology, Hamirpur, Himachal Pradesh, India