National College of Ireland

Reducing the Cloud Overhead and Latency for Artificial Intelligence Applications Using Hybrid Computing

MSc Research Project
Cloud Computing

Eromosele Idiagi Iriogbe
Student ID: 19242794

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Eromosele Idiagi Iriogbe
Student ID:	19242794
Programme:	Cloud Computing
Year:	2021
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	16/08/2021
Project Title:	Reducing the Cloud Overhead and Latency for Artificial Intelligence Applications Using Hybrid Computing
Word Count:	6138
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	20th August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Reducing the Cloud Overhead and Latency for Artificial Intelligence Applications Using Hybrid Computing

Eromosele Idiagi Iriogbe
19242794

Abstract

In developing artificial intelligence (AI) applications which require high compute resources for training deep learning and machine learning models, cloud is often adopted. Using the cloud in this way offers some benefits, however some limitations exist in terms of increased latency, privacy, reliability and computation overhead on cloud servers. An alternate approach is to run AI-based applications on local devices, but due to limited computing capacity of some local devices certain tasks like training some deep learning neural network models cannot be efficiently done on the everyday local devices, thus requiring the purchase of costly hardware which is not feasible for every user. Therefore, dependence on the cloud sometimes cannot be avoided. To address the limitations with each individual approach (local execution or cloud execution) for certain implementations, hybrid computing is proposed. To reduce the cloud computation cost and latency, a small portion of code is pushed to the client device where the less computationally intensive tasks like deep learning inference is carried out. This concept we refer to as hybrid computing. In this work, a human emotion classification application is developed in a cloud server based solution and a hybrid computing solution. The hybrid computing solution is shown to reduce the cloud overhead and provide better scalability of the application, enabling a larger number of users to utilise the service.

1 Introduction

Computation offloading has seen increased adoption in recent years. This can be attributed to the numerous benefits which can be gained by transferring compute tasks to more capable platforms. Some of these benefits include reduced operational costs, enhanced performance and greater efficiency. Computation offloading to the cloud via cloud computing received great adoption in this regard because of its inherent characteristics such as on-demand self service, rapid elasticity, and broad network access. These benefits led to quick adoption of the offloading of compute intensive tasks to the cloud. In more recent years however, some potential drawbacks with offloading to the cloud have been identified and more efficient computation approaches sought. These drawbacks affect implementations requiring lower latency as well as those with network bandwidth constrains.

To mitigate against these issues researchers have proposed frameworks which adopt unique methods to determine if at all to offload, what to offload, and how to carry out the

offloading. In some of these research efforts, local computation is adopted, whereas some others favour offloading the the cloud. Whilst these approaches have their individual benefits, they however posses individual limitations. For example, the cloud remains the more compute-resource rich option than local devices for most applications, but is affected by privacy issues for certain implementations. Offloading to the edge via edge computing has also been adopted by some researchers, but this is also met with peculiar challenges. As highlighted in Lin et al. (2019), issues with partitioning, task allocation, and resource management originate because of the distributed and heterogeneous edge computing environment. Therefore, identifying the ideal offloading approach for varied use cases and applications remains an open research area.

The resource-intensive nature of machine learning applications have made them a suitable candidate for computation offloading with offloading to the cloud being increasingly adopted. However, issues with this approach exist in terms of privacy, latency, reliability and even the accrued monetary costs. With local computation, there is no need to transfer input data across the network to the cloud hence ensuring reliability. Privacy is also ensured with local execution as data is managed locally. However, the amount of compute resources available locally is limited which can negatively impact accuracy and increase computation time.

Therefore, an approach which draws upon the strengths of both local and cloud execution while cutting out their individual limitations would be beneficial. This is what the hybrid computing approach proposed in this work aims to achieve. To address the identified issues, a hybrid computing approach which involves sharing the computation between the cloud and the local devices is adopted. This is possible because today, smart phones and personal computers (PCs) used on a day-to-day basis by individuals now ship with central processing units which are much more powerful than those available only a few years back. They also posses much greater memory capacity. Some of these devices even posses graphical processing units. Often time, a large portion of these compute resources are not fully utilised. Therefore, through the hybrid computing approach, both the cloud and the local devices would be used efficiently thus ensuring privacy and reliability is achieved without sacrificing on accuracy.

The hybrid computing approach proposed is tested over a machine learning application performing real time emotion classification from human faces. To demonstrate the effectiveness of this research, the emotion classification application is implemented with both a cloud server solution and a hybrid computing solution and both are critically compared. The deep learning model employed is trained over the facial emotion recognition (FER-2013) dataset. In the hybrid computing solution, the trained model is transferred to the local device and inference carried out locally, whereas in the cloud server solution inference is done on the cloud server. From experiments carried out, through the hybrid computing approach adopted, the emotion classification application is made available to more users with reduced overhead on the cloud resources.

1.1 Research Question

This project aims to answer the research question -

Can shared computation between the cloud and local devices through a novel hybrid computing approach be a suitable means of reducing cloud overhead for artificial intelligence applications?

The following objectives are derived to address the research question:

- Investigate the state of the art in computation offloading.
- Design and implement an artificial intelligence application which performs hybrid computing to reduce cloud overhead.
- Carry out suitable tests to evaluate the performance of the developed hybrid computing system.

The main contributions of this research are:

- A cloud server based emotion classification application solution.
- A hybrid computing emotion classification application solution.
- Utilising hybrid computing to improve efficiency and enable a larger audience to access and use an AI application.

This report is structured as follows. Section 2 presents related works, Section 3 describes the employed methodology, Section 4 discusses the design specification, Section 5 presents the implementation, Section 6 presents the evaluation, and in Section 7 conclusions and future work are discussed.

2 Related Work

In recent years, computation offloading has been well-researched owing to the benefits it offers. Different approaches have been adopted by researchers in determining what to offload, how the offloading would occur, and where to offload to. Recent efforts in this area are explored in detail in this section. Section 2.1 introducing computational offloading, and presents general surveys carried out on recent research efforts in the area. Section 2.2 discusses the various approaches adopted by researchers to identify the optimal offloading scheme. In Section 2.3, computation offloading of machine and deep learning to the edge via edge computing is discussed. In Section 2.4, client-side machine learning and deep learning is discussed alongside deep learning in browsers.

2.1 Computation Offloading

Computation offloading which is the transfer of part or all computation to another medium for execution has been adopted in recent times for several reasons. Some of these include empowering resource constraint devices, reducing execution costs (which could be in terms of time, financial, energy etc). This section introduces computation offloading and presents an overview of the recent efforts into computation offloading. Recent research efforts into computational offloading to the edge are surveyed in Lin et al. (2019). Edge computing is introduced by the authors as a possible solution to some issues that arise with offloading to the cloud. One of such issues is the latency. Increased latency is associated with computation offloading to the cloud because of the distance between the end devices which generate data and the cloud. Edge computing is further discussed, and its characteristics and challenges presented. The heterogeneous and distributed environment in which edge computing takes place results in unique challenges when offloading to the edge is adopted. Some of these challenges are in terms of partitioning, task allocation and resource management. To address these challenges, some technologies are proposed

which would be beneficial such as blockchain and severless computing. The importance of computation offloading to the edge is further buttressed using specific scenarios where it is particularly beneficial for improved performance, for example, in cloud gaming and real time streaming.

Similarly, Zheng et al. Zheng et al. (2020) discuss computation offloading to the edge highlighting offloading scenarios, factors which affect offload decision, strategies adopted in offloading and critical challenges with offloading to the edge. Program partitioning is elaborated on as well as the identification of components which can be offloaded. Varied offloading scenarios exist such as offload from IoT devices to the Cloud, offload from edge nodes to the cloud, and offload from the cloud to end devices such as IoT devices which is employed in applications which require extremely low response time, for example video analytics. This work supports the idea that offloading from cloud to end devices is beneficial for certain applications.

Computation offloading has experienced noticeable growth in significance in recent years. Akherfi et al. Akherfi et al. (2018) capture this graphically through a representation of published papers where the words "computation", "data offloading", and "offloading" were cited from 2004 to 2014. This can be seen in Figure 1. The author also discusses mobile cloud computing (MCC) which is an integration of mobile computing and cloud computing. The use of smart phones has grown exponentially in recent years and today smart phones are used in gaming, healthcare, and e-learning domains. However, despite the increased compute capacity of smart phones today, they still experience difficulty handling certain resource intensive tasks such as deep learning. Hence, the increased adoption of offloading to more capable platforms. A highlighted challenge faced by computation offloading frameworks developed for MCC is platform diversity which originates as a result of the variations in smartphones operating systems and architectures.

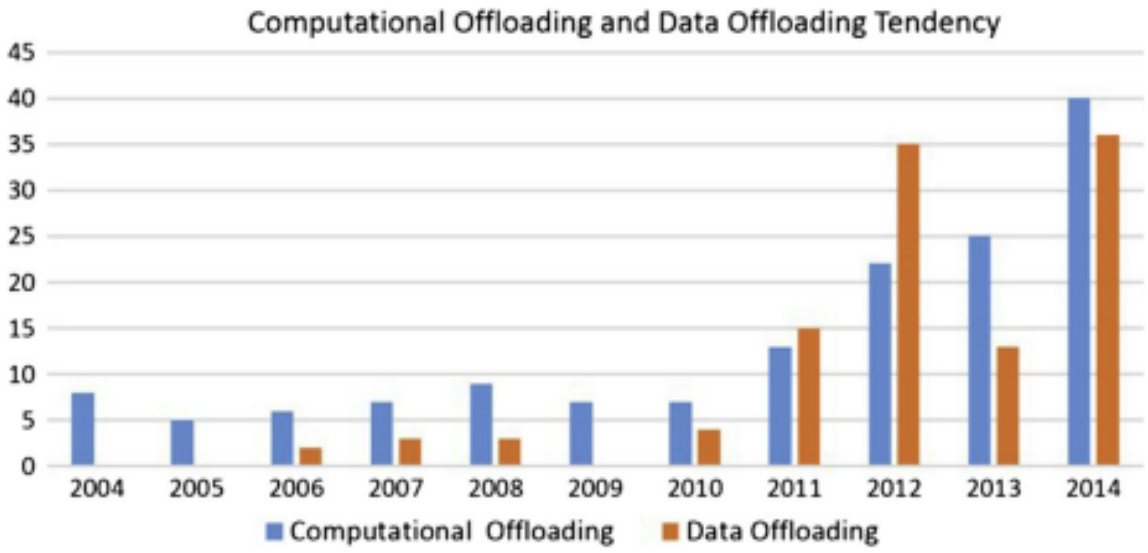

Figure 1: Number of computation and data offloading papers. Source Akherfi et al. (2018)

2.2 Approaches Adopted Towards Optimal Offloading

To arrive at an optimal offloading decision, different approaches have been adopted by researchers. In this section these approaches are reviewed. Research carried out by Yu et al. Yu et al. (2017) in the field of mobile edge computing (MEC) aimed to reduce the system cost in terms of network resource usage associated with computation offloading. Considering existing offloading schemes such as the random offloading scheme (what to offload is chosen randomly by the mobile device), the total offloading scheme (offloading all components), no offloading scheme (complete local execution) and the multi-label linear classier-based offloading scheme, they develop a deep supervised learning (DSL) algorithm to determine the ideal offload decision. The proposed system outperforms the schemes mentioned earlier over the experiments conducted by reducing the system cost considerably. Whereas their work aims to reduce system cost in terms of network resource usage, Chakrabarti et. al Chakrabarti et al. (2020) looked to mitigate the latency associated with real-time classification in an edge offloading environment. In their work, a Markov Decision Process(MDP)-based framework is proposed. To minimize network latency and maximize classification accuracy, they employ a token bucket to control the transmission to the edge device. To evaluate the framework, the ImageNet image classification benchmark Russakovsky et al. (2015) is utilised.

As often time offloading involves some form of performance trade-off between different metrics, Karim and Prevost (2017) proposes a framework which employs decision tree algorithm to decide if computation on a mobile device needs to be offloaded to the cloud. In the developed system the offload decision is taken as a binary classification problem, with user input, network conditions and the available device resources being considered. An image processing software is used to test out the developed system. A different approach is taken in Yao et al. (2020) where the large proportion of time which data transfer between local and edge devices consumes is addressed by the Deep Compressive Offloading framework proposed by the authors. Deep learning and compressive sensing theory are employed in the framework, with the process involving encoding the data to be offloaded into minute pieces, transmitting those pieces and then decoding them at the edge server. DeepCOD, an offloading system is developed by the author and through the system end-to-end latency is effectively decreased while accuracy is sustained with a loss of at most 1%. An observed drawback with the proposed system is that with decreased network transmission time, computation overhead increased on mobile devices.

To determine the ideal location to offload to between a near fog/edge node, a neighbour fog/edge node or the cloud server, a deep learning based framework is proposed in Alelaiwi (2019). The future response time is predicted by the framework and using the calculated response time, the offload location is decided by selecting the node which returns the least response time. MATLAB is used to simulate the proposed model.

2.3 Offloading Machine and Deep Learning to the Edge

Offloading machine learning tasks to the edge has been adopted by researchers to empower mobile devices as well as other resource constrained devices to carry out these tasks. This section explores these efforts. Li et al. Li et al. (2019) looked to exploit device-edge synergy to enhance deep neural network inference. They proposed the Edgent framework and tested it over an image recognition task. To reduce computing latency, adaptive DNN partitioning and DNN right-sizing are adopted in the framework. A drawback with the Edgent framework is that to achieve lower latency, accuracy is sacrificed. However,

in a more recent effort, the Cogent framework is proposed by Shan et al. Shan et al. (2020) and it successfully reduced the service latency without adversely impacting accuracy. Similar to the work carried out in Li et al. (2019), the Cogent framework looks to enhance deep neural network inference on devices with limited compute capacity. The frameworks consist of two distinct stages which are the pruning and partition stage, and the containerized deployment stage. In the developed architecture, inference is done on edge devices in close proximity to the local device.

Another similar research is carried out in Dey et al. (2019) where an optimal offloading system for the deep learning inference stage to edge servers is developed by the authors. However, in their work, the partitioning algorithm considers the available network bandwidth and the load on the local devices in determining the partition point. Offloading machine learning inference to edge servers is also proposed in Yang et al. (2019). The author proposes partial offloading of inference computation to edge servers in a bid to decrease the overall time consumed when carrying out convolutional neural network inference on mobile devices. Workload batch processing is done on graphical processing unit (GPU) on the edge server which from experiments carried out decreased the average inference time.

2.4 Machine Learning and Deep Learning on the Client-side

Computation on the client-side has been investigated by researchers based on their use cases and requirements. A review of these research efforts to carry out machine learning and deep learning on end devices (client-side) are discussed in this section.

Guo Guo (2018) carried out an empirical study of mobile deep inference. In the research work, the inference performance of the cloud is compared with that of a local device over three convolutional neural networks (CNNs). The performance of the cloud-based deep inference is compared with the on-device inference using a benchmark object recognition Android application. Deep learning driven mobile apps are categorized by the author into those employing cloud based inference and those employing on-device inference. The CPU or GPU of a mobile device is used to carry out inference tasks using CNN models that are stored on the device in on-device inference. In contrast however, models trained on cloud servers are used in cloud-based inference. Not surprisingly, from experiments conducted, the cloud-based approach out performs the on-device approach by at least 6 times. In addition, the battery consumed by the on-device approach accounted for twice that consumed by the cloud approach. The author therefore infers that the range of applications where on-device inference can be applied to are limited. A hybrid computation approach where the cloud and the mobile device both share the computation is however not considered in this work and this could be uniquely beneficial.

There exist some benefits to local inference, that is, carrying out inference on end devices such as mobiles and laptop computers and not solely on the cloud. A research carried out by Wu et al Wu et al. (2019) discusses this. In their work, how machine learning is applied at Facebook is discussed along side the approaches Facebook has adopted to enable machine learning inference on the end devices the Facebook application is run on. Some benefits looked to be gained by this include a reduction in inference time as well as a reduced dependence on fast and stable network connectivity. However, some technical challenges exist with carrying out inference on end devices so certain tools and optimizations are utilized to mitigate them. Microarchitecture specific performance tuning, weight compression, and quantization are some techniques employed to address

the aforementioned technical challenges. The study also reveals that on mobile, because of systems heterogeneity, it is difficult to port code to co-processors and as such inference is done mostly in CPUs.

In Ran et al. (2017), an Android application which carries out real-time object detection is developed by Ran et al. Trade-offs between battery usage, latency, and classification accuracy are presented when local object detection is done or when a remote server with more compute capacity is used. In their work, the offload decision is affected by factors such as a change in network bandwidth, change in the size of the neural network model, and a change in device battery level. As with some earlier reviewed work, a hybrid computation approach is not considered by the researchers, as computation is either done locally, or if a decision to offload is arrived at, completely on the remote server.

To address latency and privacy concerns that exist in the area of mobile health, Dai et al. Dai et al. (2019) develop an application which does inference on the mobile device using a pre-trained CNN model. They test the application over a dataset of skin cancer images using a CNN model earlier trained over 10,015 images of skin cancer. When new images are supplied to the application, computation is done locally without a need for the images to be transferred over a network. This approach, in addition to the benefits of reduced latency and improved privacy, enables the application to work even when a network connection is not present.

A different approach is adopted by Teerapittayanon et. al Teerapittayanon et al. (2017) to reduce communication cost. In their work, they propose a distributed deep neural network (DDNN) involving the end device, edge devices, and cloud servers with increasing neural network layers in the listed order. They train a single DNN model and map sections of it across these levels and enable early exit points.

Today, deep learning can be carried out in browsers. Ma et al. Ma et al. (2019) explore in detail, deep learning is browsers. In their work, they survey what currently can and what currently cannot be achieved when carrying out deep learning in browsers, and compare the performance of different deep learning tasks when run in browsers and when run on native deep learning platforms. They postulate that deep learning in browsers offers an advantage over native platforms when devices with integrated graphic cards are used. This owes to the fact that native deep learning frameworks require standalone graphic cards for acceleration whereas both integrated and standalone graphic cards can be accessed when browsers are used. Another advantage which carrying out deep learning in browsers offers is the mitigation of cross-platform portability challenges. By developing web applications, artificial intelligence (AI) driven applications would not need to be maintained across different platforms.

2.5 Research Niche

Through the review of literature, it is evident that computation offloading can be employed in diverse scenarios to meet diverse requirements. The viability of computation offloading has made it a well researched area in recent years with authors proposing various approaches to it in addressing a range of applications and scenarios. Table 1 provides a summary of some of the offloading frameworks proposed in the reviewed literature, comparing and contracting them.

Through computation offloading, resource-constrained devices are empowered to more efficiently carry out compute intensive tasks. Machine learning is a classic example of a compute intensive task which has enjoyed the benefit of computation offloading. Ap-

Reference	Paper Title	Method	Merit	Limitation
Yu et al. (2017)	Computation offloading for mobile edge computing: A deep learning approach	DSL algorithm to determine optimal offload decision in a MEC environment in order to reduce network resource utilization	Reduces network resource usage and out-performs random, total, and no offloading schemes	System cost is considered only in terms of network resource usage
Karim and Prevost (2017)	A machine learning based approach to mobile cloud offloading	Framework using decision tree algorithm considering network condition, device resources and user input. Offload is between mobile device and a cloud server	Considers a variety of factors in making offload decision	Does not consider the potential benefits of sharing computation between the mobile device and cloud server
Yao et al. (2020)	Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency	Deep Compressive Offloading framework to address large proportion of time consumed during data transfer to edge device. Developed DeepCOD offloading system	Decreases end-to-end transmission latency with accuracy loss of at most 1%	Approach resulted in increased computation overhead on the mobile devices
Li et al. (2019)	Edge AI: On-demand accelerating deep neural network inference via edge computing	Edgent framework exploiting device-edge synergy to enhance deep neural network inference by reducing computing latency. Adopts adaptive DNN partitioning and DNN right-sizing	Framework successfully reduced computing latency	Sacrifices accuracy to achieve lower compute latency

Table 1: Comparison of Offloading Frameworks

proaches adopted in the reviewed literature include offloading to edge servers, offloading to cloud servers, and on-device computation. These individual approaches however have advantages and disadvantages which pertain to each of them. Also reviewed is deep learning in browsers which is proposed as a solution to cross platform portability challenges. Therefore, an approach which can pull together the advantages of the different approaches adopted towards carrying out machine learning tasks today such as total offloading to the cloud or edge, and on-device computation, whilst effectively cutting out their individual limitations would be advantageous. This is what the hybrid computing solution developed in this work achieves. In this work, the cloud server and the end device share the computation and to mitigate against cross platform portability issues, the web browser is used.

3 Methodology

Artificial intelligence (AI) based applications which requires high computation resources for training deep learning and machine learning models often time utilize cloud services. However, along with advantages there exist some limitations associated with cloud computing services such as increased latency, accrued financial costs, and privacy concerns for certain implementations. An alternate solution for AI-based applications is to run on local devices, but due to limited computing capacity of local hardware devices, high computation tasks such as training certain deep learning and machine learning models over large datasets is often time not feasible, making reliance on the cloud sometimes necessary. Therefore, to reduce cloud computation overhead a portion of computation is pushed to the client device where the less computational intensive tasks such as Inference is then done. The above concept, is referring to as hybrid computing. In this work, we are using two different approaches for performing the computation. They are:

- Cloud server based solution
- Hybrid computing solution (Offloading computation to client-side)

For real-time applications, the cloud server based solution will induce high network usage and huge server costs for computing. To demonstrate this, in this work we have developed a real-time emotion classification application. The developed application utilizes a pre-trained deep neural network model. In this section, we discuss the design methodology.

3.1 Cloud Server Based Solution

In the cloud server based solution, computation and data transfer dependency of the application is placed on the cloud server. On the client end, the user will grant access to their device's camera or the web-cam associated with local device and the application will identify the emotion in real-time. For the server based system, WebRTC [1] is used to send and receive video stream between the Web browser and the cloud server. Inference is done using the deep learning model which is saved on the cloud server.

As the video stream is transmitted to and from the cloud server, there is much network utilisation and good internet connection is required. As the application is completely

[1] https://webrtc.org/

dependent on the cloud for all computation, the application can run on devices with low compute capacity, but where there is low network bandwidth, much latency is observed. The architecture of the cloud server based solution is shown in Figure 2.

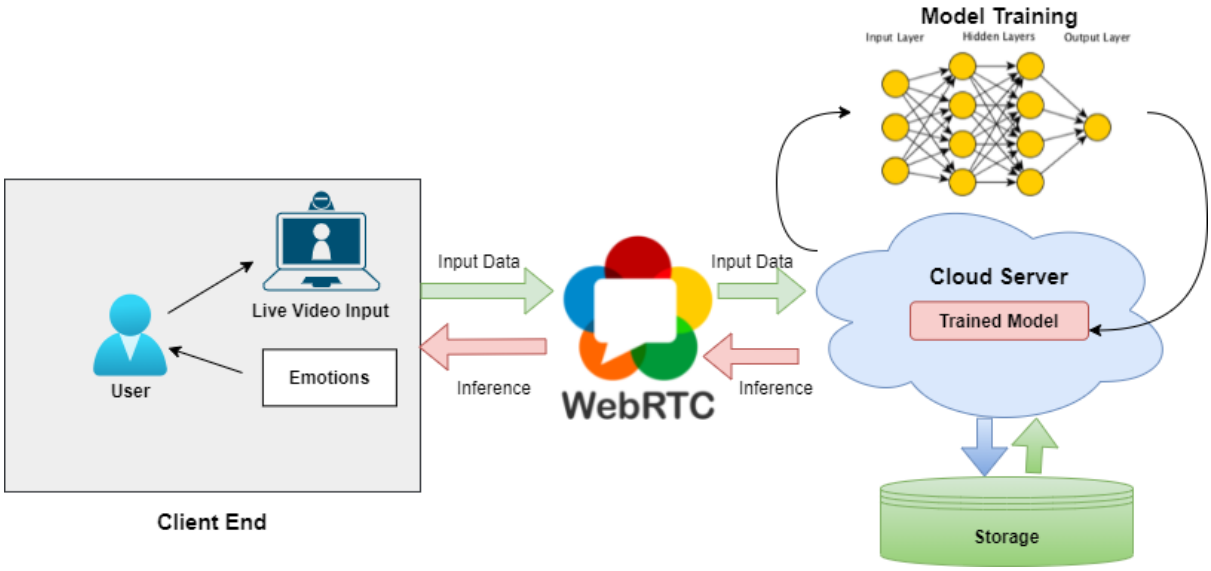

Figure 2: Server Based Solution

3.2 Hybrid Computing Solution

In the hybrid computing solution, rather than complete dependency on the cloud server, the trained deep learning model from the cloud server is transmitted to client end in JSON format. Model transfer from the cloud server to client end is one-time process. Once the model is transferred to client end, the inference tasks can be performed at client end itself. In the hybrid computing approach, the high computational task such as training the deep learning model on large volume of image data is performed at server end and the inference task which is less compute intensive is performed at client end.

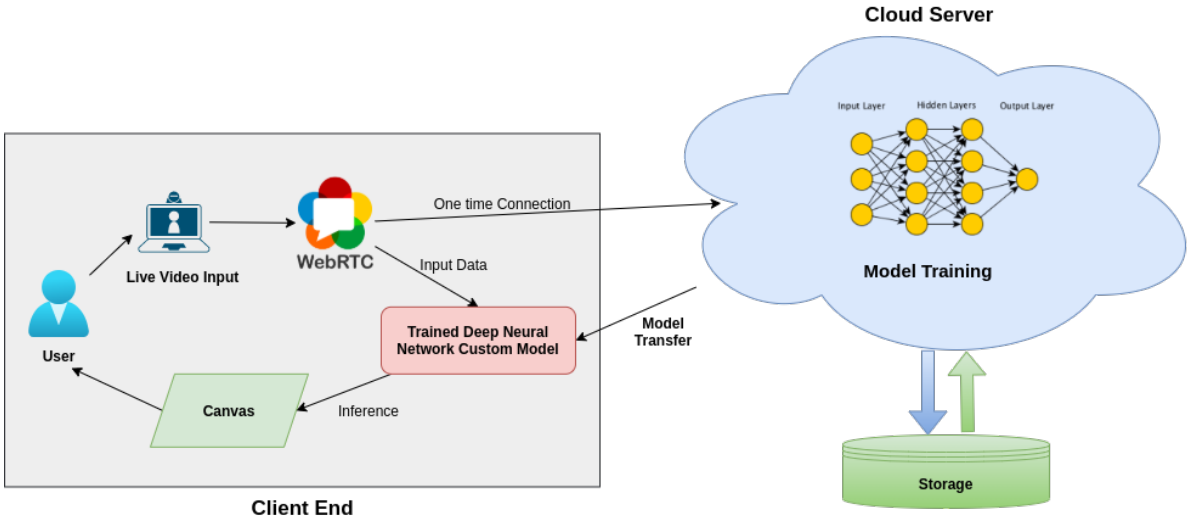

Figure 3: Proposed Hybrid Approach

Overall, using hybrid computing approach, when a user accesses the application, a one-time connection from server end to client is established and after successful establishment of the connection, the trained deep learning model is transmitted to the respective client. Once the model is transferred to the client end, it is used to carry out the inference. The emotions of the user over the video is classified. This approach greatly reduces the computational overhead on the cloud server, reduces amount of network bandwidth consumed, also avoids the latency which results from transmitting large data over the network. The approach is simple yet effective, as the developed application works effectively even when there is low network bandwidth.

3.3 System Components

In this section, the system components are discussed.

- **Application Users :** The application users access the application using their computing devices such as smartphones, laptop or tablet. As it is a Web application, it is run in a Web browser and no additional installation is required on the users device. In order to utilise the application, the user's device has to have a camera as well as a stable internet connection.

- **WebRTC :** In order to enable real-time communication capabilities for the application, WebRTC is used. WebRTC stands for Web Real-time Communication, an open-source project and it is a collection of several APIs and protocols. WebRTC is available on major browsers and it enables the application to capture and stream video media.

- **Cloud Server :** Cloud server refers to a cloud instance. This could be provided by a public cloud provider like Amazon Web Services, Google Cloud, or Microsoft Azure. The cloud server is connected with cloud storage, in a two-way communication. The large volume image dataset used for training deep learning models is stored on the cloud storage. In case of server-based system cloud server will also perform the inference task.

4 Design Specification

In this section the design specification of the emotion classification application is discussed. The overall architecture for training the deep learning model is shown in Figure 4.

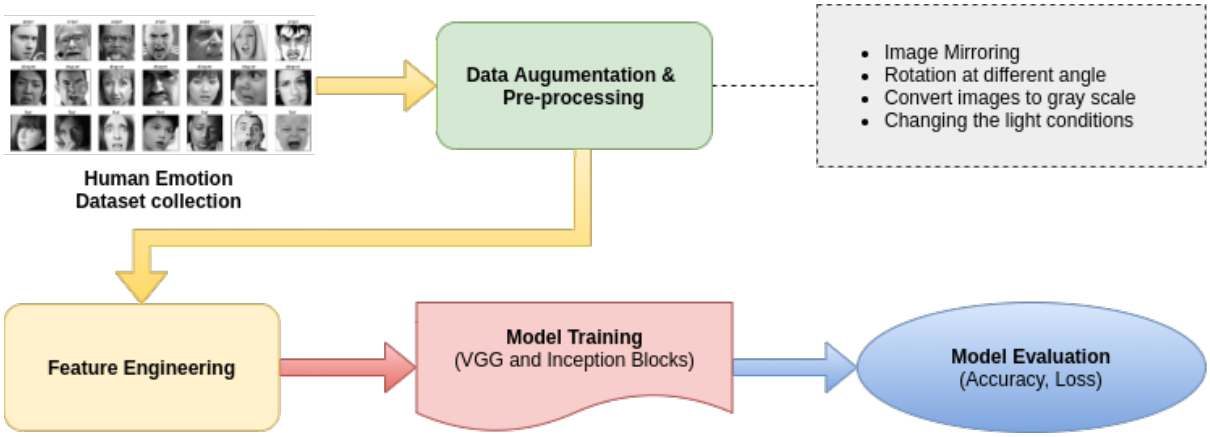

Figure 4: Training the Deep learning model for Emotion Classification

The pre-trained deep learning model is then used in the application to classify emotions. In the first instance, the application identifies the human face from the video data and then the emotions are classified by analysing the human face.

4.1 Dataset Collection

In this step, the FER-2013 (Facial Emotion Recognition) [2] dataset for performing emotion classification is collected. The main labels for emotions are 'Fear', 'Angry', 'Happy', 'Disgust', 'Sad', 'Surprise', 'Neutral'. The dataset contains grayscale images of faces with size of 48 X 48 pixel. The training set contains 28709 examples and testing set contains 3589 examples.

4.2 Data Augmentation & Pre-processing

In order to improve the classification accuracy, data augmentation and pre-processing plays an important role. In this process, we are applying certain techniques such as image mirroring, rotation of images from different angles, changing the images to different light conditions. Data pre-processing step also includes the mapping of the images with the correct labels.

4.3 Model Training & Feature Engineering

We train a CNN model created by combining VGG blocks and Inception blocks. The model has approximately 3 million parameters and is able to classify 7 emotions with an accuracy of 63%. The model utilizes 9 convolutional layers, 5 max pooling layers and 3 dense layers. The model has been trained over 25 epochs. Rectified linear activation function (ReLu) is used as the activation function. The implementation is carried out using TensorFlow functional API

[2]https://www.kaggle.com/msambare/fer2013

4.4 Model Evaluation

The model has been trained over 25 epochs. Accuracy and loss score has been calculated for analysing the performance of the model. The fine tuning of the hyper-parameters also has been performed. The result obtained for training and validation set before hyper parameter tuning and after hyper-parameter tuning are shown in Figure 5, Figure 6, Figure 7 and Figure 8.

The highest accuracy obtained for FER-2013 dataset before hyper parameter tuning is 61.24% over the validation set. On the other accuracy obtained over the training set is 63.27%. On calculating the training and validation loss, the values obtained after training the models are 0.9786 and 1.052 before performing hyper-parameter tuning. It has been found that on increasing the number of epoch the accuracy was increasing and in an inversely proportional manner the loss was found to be decreasing.

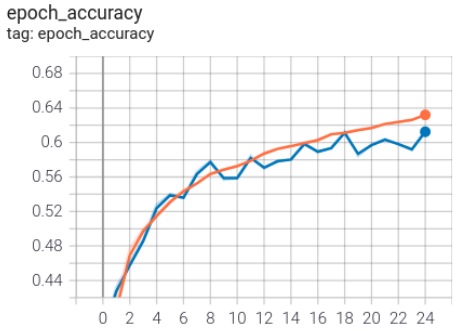

Figure 5: Accuracy Score (Before Parameter Tuning)

Figure 6: Loss Score (Before Parameter Tuning)

After performing the hyper-parameter tuning improvements are found in the performance score. The highest accuracy obtained after hyper-parameter tuning for training and validation set is found to be 68.36% and 63.39% respectively. Over every epoch the accuracy was found to be increasing. On the other hand, on comparing the loss score after hyper-parameter tuning it is found that, over every epoch the loss score decreased. The loss score obtained after 25 epoch for training and validation set is 0.83 and 1.07 respectively.

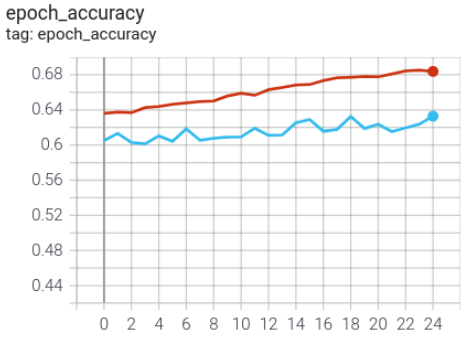

Figure 7: Accuracy Score (After Parameter Tuning)

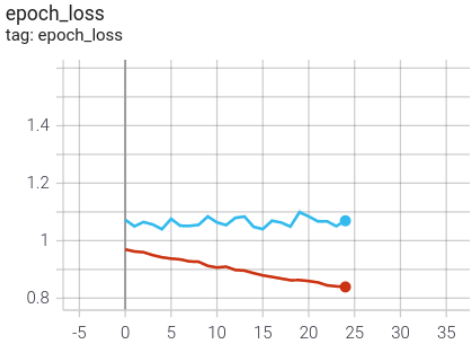

Figure 8: Loss Score (After Parameter Tuning)

It is observed that that hyper-parameter tuning helps to improve the performance of the model. After successfully training the model, the model is saved in the cloud server storage. In this way, we have successfully trained the model for development of the application which classifies 7 different emotions of human faces in the real-time.

5 Implementation

5.1 Cloud Server Based Solution

In order to acquire the media streams from the camera the WebRTC MediaStream API is utilized. For security reasons, the browser only provide the data through encrypted https connections. We are utilizing the ngrok [3] for creating a secure https tunnel to access our application. In transferring data from the local device to the server, the aiortc library [4] is used. In order to perform the image processing, OpenCV- Python library [5] has been used. All the operations such as extracting the face, resizing, rotation, mirror operation can be performed with OpenCV-python library. For face detection Haar-cascade algorithm [6] is utilized. For inference, we are using TensorFlow 2.5. Rendering is done using OpenCV drawing functions [7]

[3]https://ngrok.com/
[4]https://aiortc.readthedocs.io/en/latest/
[5]https://docs.opencv.org/4.5.2/
[6]https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
[7]https://docs.opencv.org/3.4/d6/d6e/group__imgproc__draw.html

5.2 Hybrid Computing Solution

In the hybrid computing based solution some functionalities and use of libraries are same as the cloud server solution. However, as the architecture is different the method of implementation also differs. In the hybrid computing architecture, we convert the saved model into JSON using TensorFlow.js converter. Inference is run in the client's browser using the model assets which are fetched using Tensorflow.js model loading API. Same as with the server based solution, WebRTC MediaStream API is for media streaming and ngrok has been used in order to establish a secure https tunnel. In the hybrid computing architecture transfer of video stream is not required, as we are locally processing the input. In order to process the image, ml5.js library [8] has been used which is used to recognise the face. In the inference process, the input is taken from the pixeled of canvas, the conversion to gray-scale images and resizing is managed by TensorFlow.js library. The input is fed to the model and the inference run. For rendering Canvas API is used.

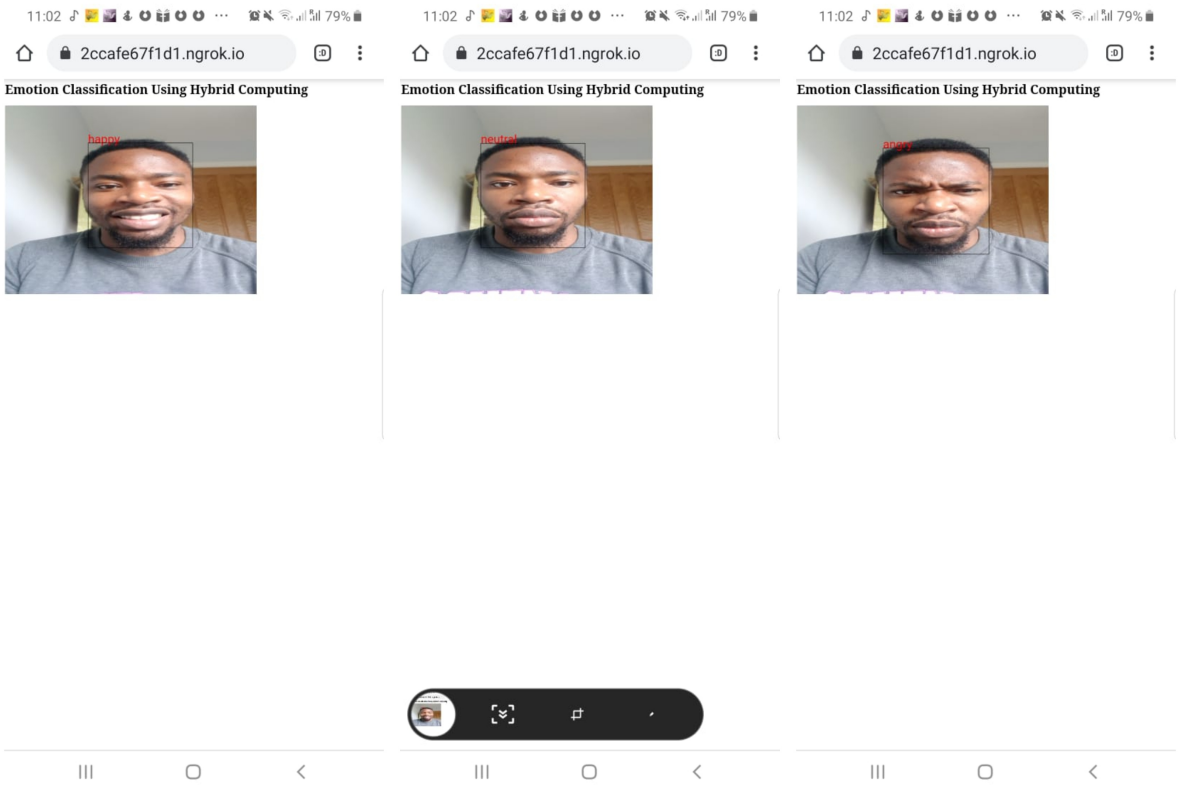

Figure 9: Classification of Different Emotions (Mobile Device)

[8] https://github.com/ml5js

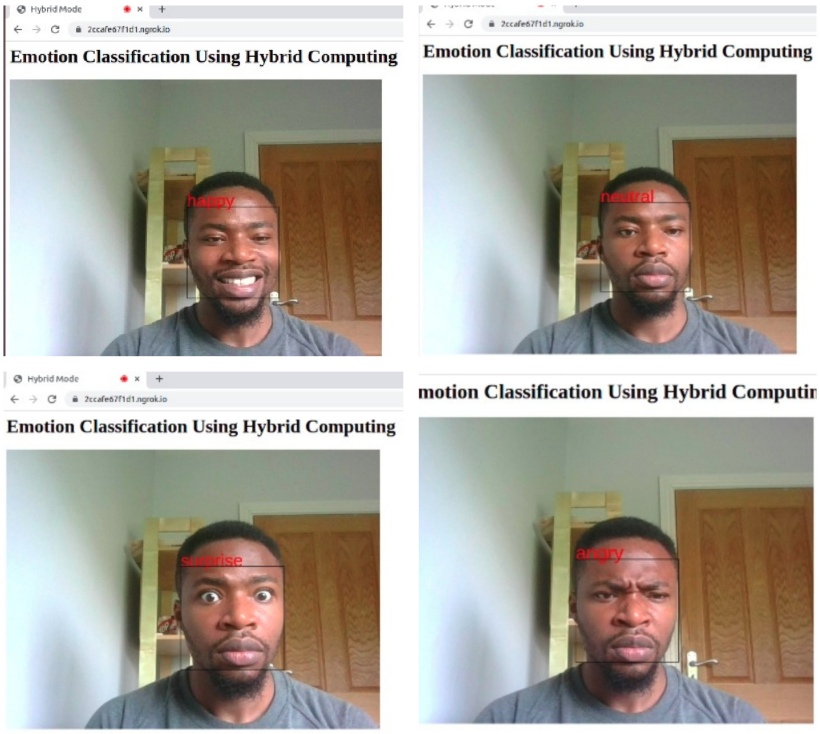

Figure 10: Classification of Different Emotions (Laptop Computer)

The application can run on any modern browser whether on a mobile phone, laptop, or tablet. The application identifies the emotion from the human face in the real-time. The core difference between the server and hybrid based approach lies in where inference is run. In cloud server based solution, the inference takes place at the cloud server. Whereas, in the hybrid computing approach the inference takes place at client machine. Emotion classification on the application is shown in Figure 10 and Figure 9.

6 Evaluation

As the application is cloud-based it can be served to as many users depending on the resource availability of the cloud server. In this section, the CPU utilisation and network bandwidth utilisation on the cloud server for the complete cloud server solution and the hybrid computing solution is evaluated, and results from the experiments carried out are presented. In the experimental setup, the cloud server has 16GB of RAM, 4 cores (8 logical processors), and a base speed of 2.4GHz.

6.1 Experiment 1/ Cloud Server-based Solution

From the cloud server based solution's architecture discussed in earlier sections, it is clear that the video data is transferred to the cloud server and inference is performed at the server end. After inference is performed, the video data is transmitted back to the client end. In this manner, two-way communication between the server and the client is taking place where the client is transmitting data to the server and the server again is transmitting data back to the client. Network utilisation is occurring at both client

end and server end. As inference is carried out on the server, the CPU utilisation of the server increases as the number of users of the application increases. The graph shown in Figure 11 represents the relationship between the CPU utilization and the number of users.

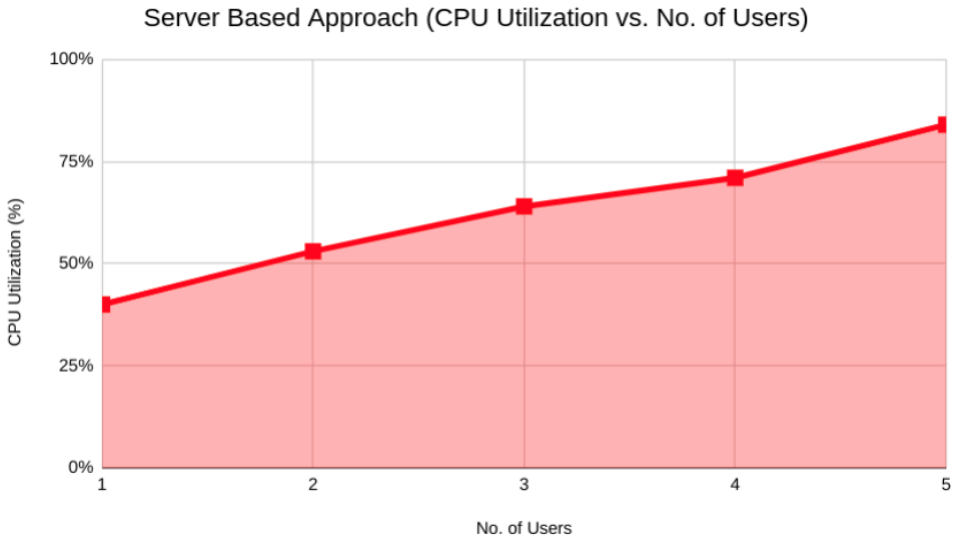

Figure 11: CPU Utilization in Server Based System

From the graph it can be observed that the CPU utilisation increases as the number of users increases. When the emotion recognition application is serving a single user, the CPU utilization of the server is found to be 40%, and as more users are connected, this increases, reaching up to 82% when 5 users are connected. Upon adding more number of users, the application crashed as the cloud server stopped responding to the requests. Therefore, with the cloud server-based approach, the emotion classification application served at the maximum 5 users.

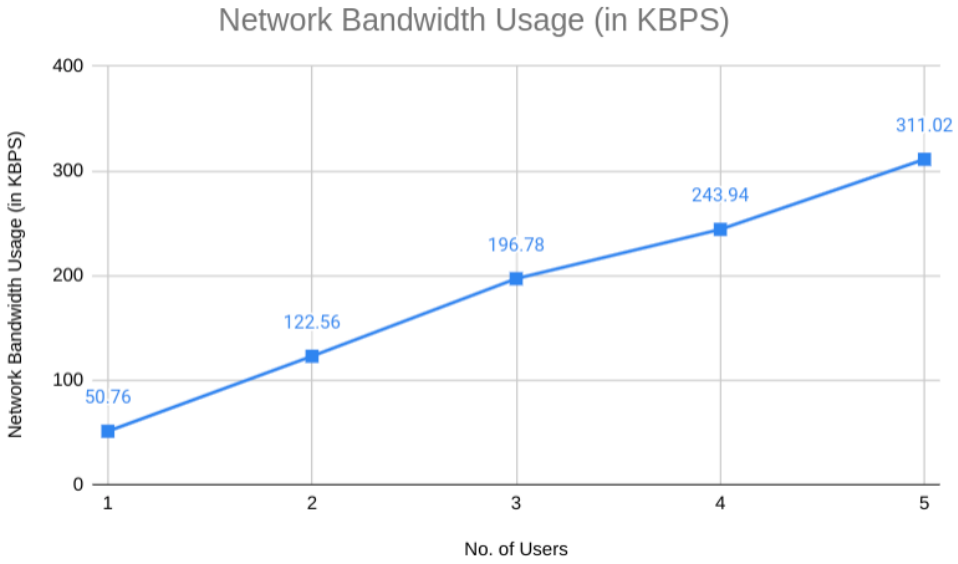

Figure 12: Network Bandwidth Usage in Server Based System

On observing the network bandwidth utilisation with respect to the number of users, it has been observed that as the application works in the real-time environment, the network bandwidth is continuously utilized as continuous data transmission is taking place between the client and server. The line graph shown in Figure 12 represents a relationship between the number of users and network bandwidth usage. On analysing the graph it is to be observed that as the number of users are increasing the network bandwidth utilization is also increasing in a linear manner. Additionally, when the application is executed using the server based approach, there is observable latency in the emotion classification because of the round trip time from the client device to the server and back again.

6.2 Experiment 2/ Hybrid Computing Solution

The architecture of the hybrid computing solution is discussed in earlier sections. In the hybrid approach, the client establish a one-time connection with the cloud server and in response the cloud server offloads a small portion of code to the client end. When the cloud server offloads the model to client end, small amount of network utilization takes place. Once the model is transmitted to client end, inference can be performed at client end, without dependency on the server. Therefore, using the hybrid computing approach, it is observed that the CPU utilisation remains relatively constant even as the number of users increased. There is no notable increase in the CPU utilisation as the number of users connected increased up to 10 users.

As for the network utilisation, this also remains relatively constant and low on the server as only when application users initially access the application is a small portion of code offloaded to their device. There is also little or no observed latency in emotion classification on the hybrid computing solution.

6.3 Discussion

From the experiments carried out on the cloud server and hybrid computing solutions developed, it is evident that the hybrid computing solution produced better results. Through the hybrid solution, emotion classification can be performed in the real-time with much lower latency. The cloud server network bandwidth utilization and CPU utilization is also found to remain relatively low and constant. Using the Hybrid computing approach, the application can be served to a greater number of users unlike with the cloud server solution where the number of users who can effectively use the application at a particular time is small. These results confirm that the hybrid computing solution greatly reduces the overhead on the cloud server and makes the emotion classification application available to a larger number of users. Using the hybrid computing approach, the overhead on cloud servers can be reduced without compromising on the services provided by the application.

7 Conclusion and Future Work

In this work we investigate the viability of a hybrid computing approach which involves offloading some computation to the client side to reduce cloud overhead in artificial intelligence applications. To achieve this we have investigated recent development in computation offloading, designed and implemented a real time emotion classification application, and carried out suitable tests of the hybrid computing approach. Classification of human

emotions using deep learning is computationally intensive. Usage of the application by multiple users at the same time, generates huge workload on the cloud server. In order to reduce the cloud overhead, hybrid computing is used. Using the hybrid computing method, the application can be served to more number of users without an overhead on the cloud resources. This way, the cloud is used for the compute intensive model training, however, deep learning inference is done on local devices. As demonstrated, for real time applications the hybrid computing approach proves to be the optimal solution. In future work, more real time applications can be developed and the performance using the hybrid computing approach compared against the cloud server based approach.

References

Akherfi, K., Gerndt, M. and Harroud, H. (2018). Mobile cloud computing for computation offloading: Issues and challenges, *Applied computing and informatics* **14**(1): 1–16. JCR Impact Factor: 4.44.

Alelaiwi, A. (2019). An efficient method of computation offloading in an edge cloud platform, *Journal of Parallel and Distributed Computing* **127**: 58–64. JCR Impact Factor: 2.296.

Chakrabarti, A., Guérin, R., Lu, C. and Liu, J. (2020). Real-time edge classification: Optimal offloading under token bucket constraints, *arXiv preprint arXiv:2010.13737* .

Dai, X., Spasić, I., Meyer, B., Chapman, S. and Andres, F. (2019). Machine learning on mobile: An on-device inference app for skin cancer detection, *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, IEEE, Rome, Italy, pp. 301–305.

Dey, S., Mondal, J. and Mukherjee, A. (2019). Offloaded execution of deep learning inference at edge: Challenges and insights, *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, IEEE, pp. 855–861. CORE2020 Ranking: A*.

Guo, T. (2018). Cloud-based or on-device: An empirical study of mobile deep inference, *2018 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, Orlando, FL, USA, pp. 184–190. CORE2020 Ranking: B.

Karim, S. A. and Prevost, J. J. (2017). A machine learning based approach to mobile cloud offloading, *2017 Computing Conference*, IEEE, London, UK, pp. 675–680.

Li, E., Zeng, L., Zhou, Z. and Chen, X. (2019). Edge ai: On-demand accelerating deep neural network inference via edge computing, *IEEE Transactions on Wireless Communications* **19**(1): 447–457. JCR Impact Factor: 6.779.

Lin, L., Liao, X., Jin, H. and Li, P. (2019). Computation offloading toward edge computing, *Proceedings of the IEEE* **107**(8): 1584–1607. JCR Impact Factor: 10.252.

Ma, Y., Xiang, D., Zheng, S., Tian, D. and Liu, X. (2019). Moving deep learning into web browser: How far can we go?, *The World Wide Web Conference*, San Francisco CA USA, pp. 1234–1244. CORE2020 Ranking: A.

Ran, X., Chen, H., Liu, Z. and Chen, J. (2017). Delivering deep learning to mobile devices via offloading, *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, Los Angeles CA USA, pp. 42–47. CORE2020 Ranking: A.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. et al. (2015). Imagenet large scale visual recognition challenge, *International journal of computer vision* **115**(3): 211–252. JCR Impact Factor: 5.698.

Shan, N., Ye, Z. and Cui, X. (2020). Collaborative intelligence: Accelerating deep neural network inference via device-edge synergy, *Security and Communication Networks* **2020**. JCR Impact Factor: 1.288.

Teerapittayanon, S., McDanel, B. and Kung, H.-T. (2017). Distributed deep neural networks over the cloud, the edge and end devices, *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, Atlanta, GA, USA, pp. 328–339. CORE2020 Ranking: A.

Wu, C.-J., Brooks, D., Chen, K., Chen, D., Choudhury, S., Dukhan, M., Hazelwood, K., Isaac, E., Jia, Y., Jia, B. et al. (2019). Machine learning at facebook: Understanding inference at the edge, *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, pp. 331–344. CORE2020 Ranking: A*.

Yang, Q., Luo, X., Li, P., Miyazaki, T. and Wang, X. (2019). Computation offloading for fast cnn inference in edge computing, *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, Chongqing China, pp. 101–106.

Yao, S., Li, J., Liu, D., Wang, T., Liu, S., Shao, H. and Abdelzaher, T. (2020). Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency, *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, Virtual Event Japan, pp. 476–488. CORE2020 Ranking: A.

Yu, S., Wang, X. and Langar, R. (2017). Computation offloading for mobile edge computing: A deep learning approach, *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, IEEE, Montreal, QC, Canada, pp. 1–6. CORE2020 Ranking: B.

Zheng, T., Wan, J., Zhang, J., Jiang, C. and Jia, G. (2020). A survey of computation offloading in edge computing, *2020 International Conference on Computer, Information and Telecommunication Systems (CITS)*, IEEE, Los Angeles CA USA, pp. 1–6.