Fachhochschule
Dortmund
University of Applied Sciences and Arts

Research Project (Thesis) MOD3-03

PROJECT AVATAR

Facial Landmarks Detection and Lip Reading on a Distributed System Based on an Android Device and a Remote Server Based on Tensor Flow

Submitted by:
Ahmad Hassan Mirza [7104716]
Embedded Systems for Mechatronics, FH Dortmund
ahmad.mirza001@stud.fh-dortmund.de

Supervisor:
Prof. Dr.-Ing. Jörg Thiem

Abstract

According to National Institute on Deafness and Other communication Disorders(NIDCD) 7.7 percent of children between the ages of 3 to 17 in the US alone have had a disorder related to voice, speech, language or swallowing with in the duration of 12 months of the report being published[1]. The idea behind project AVATAR is to develop a standalone logopaedics system which can be utilized for speech therapy for children suffering from these speech impediments. For the first part of this project the objective is to extract visual information about the shape and movement of the lips and the jaw that is associated with the pronunciation of the word and attempt to infer the word that has been said regardless of any accent or mispronunciation present. This could then be used as a base and stepping stone for further development towards a complete system for speech therapy.

With the advancements in mobile devices and their wide availability and popularity, android platform has been chosen for the development of the prototype application. The application developed employs several image processing techniques to extract the relevant information which is then passed to a remote computer for further processing that is required to infer the words or sentence said using only visual data (lip reading).

The scope of the project involves research and evaluation of the useable image processing methods, development of the application using the results from phase-1, performance optimizations to achieve a near real-time processing and evaluation and analysis of the results.

Keywords: *Image Processing, face detection, facial landmarks detection, speech therapy, logopaedics, OpenCV, dLib.*

Table of Contents

Table of Figures

Abbreviations

API	– Application Programming Interface
JNI	– Java Native Interface
NDK	– Native Development Kit
IDE	– Integrated Development Environment
UI	– User Interface
ROI	– Region of Interest
XML	– eXtensible Markup Language
ROS	– Robot Operating System
APK	– Android Package Kit
CNN	– Convolutional Neural Networks
RNN	– Recurrent Neural Networks
LSTM	– Long Short-Term Memory
vLSTM	– Vanilla LSTM
sLSTM	– Stacked LSTM

1. Introduction

According to the findings of National Health Interview Survey [2] nearly 8 percent of the children aged 3-17 in the US had a communication disorder with-in the past 12 months of the study, and among these children only 55% received any intervention service during the time period of 12 months. These statistics show that the problem of speech impediment is very prevalent in children and young people, while almost half of the people suffering from this problem receive any help.

The motivation behind this project is to develop a standalone system which can be used by laymen without any extra help to address certain speech impediments, and the most important part is that the system should be easily and widely available for everyone.

In order to achieve this objective, a smartphone application on the android platform has been chosen as the medium for the first phase of the project. The reason behind this is that in recent years smart phone industry has seen a large boom and by 2020 there are expected to be 3.5 billion smartphone users worldwide [3], apart from this the advancement in technology has made the smart phones very capable and tasks that required specialized equipment before can be performed by a simple application running on the smart phone. The introduction of wearable tech like smart watches and other health monitors that can be connected to a smartphone via Bluetooth and the data analysed by a paired app have also made diagnostics and health monitoring very easy and does not require highly skilled medical personnel to perform these tasks as in the past.

The first part of the project deals with the development of the android application. This part has further been divided into four phases for better management and implementation. *Phase-1* was research and performance evaluation of the methods which could be used to develop the application. *Phase-2* was the development of the application to the point that it could use the on-board camera to capture live images and perform image processing tasks to extract useful information. *Phase-3* was to establish real-time communication between the mobile application and a remote PC to transfer the extracted data to a PC to perform the resource extensive processing that is required to infer the words from lip reading. *Phase-4* was implementation of the Matlab scripts running on the remote PC using Hidden Markov Model and its training using data set developed using the android

application. In the future the tasks done in phase-3 and 4 would be moved to be performed on the device itself or on the cloud to enable portability.

For image processing OpenCV's android SDK is used in conjunction with dlib c++ library to extract facial landmarks. The details of the implementation are provided in the following sections.

2. Theory

2.1 Literature Review

Digital Image Processing is the collection of techniques used to manipulate images to extract some results that might be of interest out of the image for example object detection, classification etc. Image processing itself can be further sub-divided into sub.-classes lie image analysis, image restoration, image enhancement etc. In the past decades the usage of these digital image processing techniques in production applications has increased tenfold. The applications of image processing range from medical technology, automated driving to space technology.
The following sections provide a brief overview of the state of art of the techniques that have been used or could be used in this project.

2.2 Image Processing Techniques – Theory

In the scope of this project object detection and classification techniques have been used to extract facial features from the images captured from the device camera which are then further used for image analysis using machine learning techniques:

2.2.1 Face Detection

Face detection is the first step towards all facial analysis algorithms which in turn play a huge role in today's technologies that interact with human's; these facial analysis algorithms can range from facial recognition, authentication/security applications, lip reading and expression classification. Face detection is an easy task for human beings but for machines considering all the variable could prove to be a difficult task. The scale, orientation, pose and the lighting conditions of the image are among the many variables that need to be accounted for in order to perform a successful facial detection. There are multiple ways that have been a topic of research for performing face detection. They can be divided into four main methods: (i) Feature-based (ii) Appearance based (iii) Knowledge based (iv) Template matching. Out of these the more popular ones are feature based and knowledge based. For the first technique face detection can be done by using self-defined features in conjunction with HAAR cascade machine learning algorithm which is the techniques used in OpenCV's face detection algorithm. The other technique is to use neural networks and train them on a huge dataset of faces, which

are readily available now, one of the most influential face databases is the FERET or COLOR FERET database [5]. Tensor Flow is an open source platform for machine learning which is commonly used for research and applications involving deep neural networks. Tensor flow allows users to create dataflow graphs and the training API extracts features automatically from the training dataset which allows for detection and classification of objects of interest.

In 2001 P.Viola and M.Jones published a paper describing what is now known as Viola-Jones (V-J) Object detection framework. V-J algorithm is one of the first real time object detection algorithms and provides very high detection rate and very less percentage of false positives.

The algorithm has some limitations related to the pose of the face but the classifiers can be trained to recognize slight tilts in the face. V-J detector utilizes Haar-like features that are a scalar product between the image and some Haar-like templates. Fig. 1 shows the Haar-like templates used in the V-J algorithm and also employed by OpenCV.

The Haar features signify that almost all the human faces share similar visual

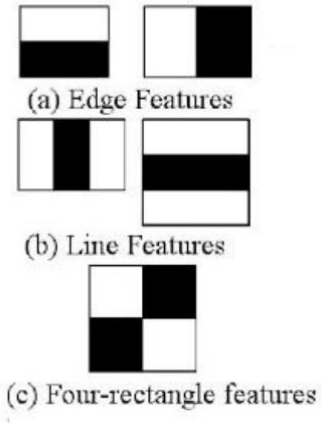

Figure 1 Haar-Features used in VJ-Algorithm implementation in OpenCV

outlooks for example the bridge of the nose is lighter than the eyes which can be defined by the Haar template figure 1(b-1) and the eyes region is darker than the upper cheeks and which is defined by inversion of Haar template of fig. 1(a-1). This way the regions of interest i.e. different features on a face can be detected using the gradients in pixel intensities. The V-J algorithm uses a 24x24 window as the base window size to start the evaluation of these features which gives rise to up to

160,000+ features to be calculated. The steps put forward by the algorithm brings this immense calculation down to a smaller size. The steps involved after selection of appropriate Haar-Feature templates in V-J algorithm are [7]:

- Image Integral and feature extraction
- AdaBoost Learning
- Cascading Classifiers

The first step of the V-J algorithm is to obtain an integral image from the original image. The integral image provides a quick and efficient way of calculating the sum of pixel values in a subset of a pixel grid which contain the pixels located above and to the left of x, y inclusive (with x, y being the coordinates of the pixel of interest) [8]. This can be represented mathematically by the following equation:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

Where ii(x, y) is the integral image and i(x, y) the original image. The computation of integral image can be done in one iteration over the entire image. The advantage of computing an integral image is that the sum of any rectangle with in the image can be calculated very quickly and efficiently by the following equation.

$$\sum_{(x,y) \in ABCD} i(x, y) = ii(D) + ii(A) - ii(B) - ii(C)$$

The second part of the algorithm is application of AdaBoost to remove unnecessary features. For example the same Haar-feature template that applies to the eyes and the bridge of the nose can also apply to other parts of the face and using AdaBoost algorithm a best threshold can redefined which classifies the faces while selecting the features with minimum error rate so that the features that most accurately classify the faces and non-face parts of the image are selected. V-J algorithm combines a small number of these features to form an effective classifier, by employing a weak learning algorithm which determines the optimal threshold classification function so that a minimum number of misclassifications occur. All the images used in the training the classifiers are given an equal weight and every feature is applied on all the training images. After each iteration of classification weights assigned to misclassified images is increased and the iteration of classification is repeated. After each iteration new error rates and weights are calculated and assigned until the required accuracy is achieved. After the

application of AdaBoost algorithm is successfully completed one feature is selected out of the 160,000+ features for each iteration.

The classifiers obtained from the AdaBoost stage are not strong enough to be used individually for object detection but the V-J algorithm suggests the work around to this by using the classifiers obtained in a cascade. Each stage of classifiers checks the given stage as to whether it definitely does not contain a face or maybe it contains a face. As soon as a sub window is classified as not containing a face it is discarded. The sub- window classified as maybe is then passed to the next stage in the cascade. It can be deduced that the more stages a sub-window passes the higher is the probability of it _containing a face. All the stages in this cascade are obtained by training classifiers by using AdaBoost algorithm and then adjusting the threshold to minimize false negatives. OpenCV already provides a cascade classifier for face, eyes and other features and also provides a way to train custom classifiers for other objects that a user might want to detect [9].

The other method to perform face or object detection is to use neural networks. A lot of research is available on this technique and some trade-offs have been listed between using template based detection and neural networks. HA.Rowley et.al show in their paper that although neural network based object detection can enhance the results of the face detection system but it causes an increase in the execution time [10]. Although this paper was published in 1998 and with the evolution of neural networks e.g. tensor flow, this trade-off might not exist at this time and a new evaluation need to be performed. The system described in this paper is a two stage detection system. A filter is applied on a 20x20 pixel region of the image which generates an output from 1 to -1 based on the presence of a face in the image, in case the faces or the image is larger than the filter size the image is reduced in each repetition and the filter is applied on every point of the image on each iteration. In the second stage overlapping detections are merged, resulting in removal of false detections in neighbouring areas of the true faces, to further reduce the number of false positives, multiple networks can be used and their outputs are arbitrated to produce a final output.

M.Wu et. al. [11] developed a system for face detection and feature localisation based on case structured classifier, template matching, feature space analysis method and AdaBoost. They also incorporated human face super resolution algorithm for helping in detection of faces in low-resolution images. The system

performs image pre-processing with histogram equalization, the pre-processed image is fed to the face detector, which is based on the V-J's cascade framework for robust and rapid object detection. The detector returns the number of faces and the locations of all the faces in the image. A resolution threshold of 64*48 has been defined, if the faces are larger than this threshold value then they are passed on to the organ detection system otherwise they employ the face super-resolution algorithm to get a higher resolution image. Before this algorithm is applied a rough location of eyes needs to be detected, if a non-human face is detected in this step then that result could be discarded making the algorithm more robust. For organ location (eye location), this part is also interesting for our application for landmarks detection, the team used 8-eye-pair template as per the minimum relevant match theory. The team report an accuracy of 95.8% for eye detection, 89.8% for nose, 87.3% for mouth and 91.7% for chin detection.

Anamika Singh et.al. [12] Proposed a system for face and eyes detection using Sobel edge detection and morphological post processing steps for extracting eyes, by analysing the eye regions and the shape of the region i.e. the distance from the left and right of the face and given the fact that the eyes are generally in a straight line on the face. They report a 92.5% success rate for eyes extraction from the IMM frontal face database.

Another algorithm for face detection and landmark localization is presented by Prathap Nair et.al [13]. The proposed algorithm detects and segments 3-d face meshes and localizes landmarks based on Point Distribution Model which removes any dependency on prior knowledge of texture, pose or orientation. They perform face detection by analysing the transformation between model points and points of interest in the sample image based on the deviation of the parameters from the mean model, afterwards landmark localization is performed on the extracted face based on the transformation that gives the minimum deviation of the model from the mean shape. They reported a 99.6% accuracy for face detection.

The facial landmarks detection scheme implemented in Dlib is based on the method described by Vahid Kazemi et.al [20] where they use a cascade of regression trees to refress the location of facial landmarks from pixel intensity values extracted from the input image, with a handling for partial and missing labels as well. The topic of facial landmarks detection has been under research and a lot of literature is available on this as well. More recently Yue Wu et. al [21] published a study where they

digress the working of several CNN based neural network configurations trained to extract Facial landmarks, they evaluated both 5-points landmarks as implemented in Google's machine learning for mobile developer's toolkit as well as 68-point landmarks. Amir Zadeh et.al [22] reported a system for landmarks detection using Convolutional Experts-Constrained Local Model, which is a CLM model which at its base uses Convolutional Experts Network (CEN) as a local detector. such CLM based models extract the location of each landmark using local detectors and utilize a shape model to do constrained optimization.

2.3 Machine Learning/ Artificial Intelligence

Artificial Intelligence is a very broad concept and revolves around the idea that machines or computers should be able to perform tasks in a way that could be classified as smart without human intervention. Conventionally this smart behaviour of the machines was programmed covering all the possible scenarios possible, the idea to enable machines to learn for themselves given they have access to large amount of training data gave rise to Machine Learning. Machine Learning is the application of AI based on this idea.

Machine learning problems can be classified into three categories: supervised learning, unsupervised learning and reinforcement learning.

In supervised learning, the system is expected to accurately predict targets from unseen features, and to achieve the machine must learn the mapping from feature vectors to outcome classes. Object classification problem comes under this category.

Unsupervised learning only a set of features are available without any mapping to output classes, and the system is expected to learn the data structure. An example problem for this case could be classification of a dataset of records into their specific categories.

Reinforcement learning is done in a dynamic environment and based on the actions taken by a system the system is rewarded or punished.

Neural networks are multi-layered network of neurons that are used to enable machine learning. Fig. 3 shows a depiction of a neural network. The circles in the image depict the neurons and how they are interconnected. A neural network at its base tries to make the best prediction about the outcome depending on the input with a level of certainty.

Section below provide a brief insight into some Machine Learning algorithms.

2.3.1 **Hidden Markov Model**

The Hidden Markov Model (HMM) is a statistical model for a system with unobservable or hidden states. In simpler versions of Markov models for example the Markov Chain, the state is visible, and the only parameters to be considered are state transition probabilities. In the HMM since the states are not visible, instead the output dependent on the state is accessible. So each state has a probability distribution for possible outputs.

- *Markov Chains*

  Markov Chains are used to describe types of random processes, giving information about the probabilities of sequences of random variables called states, which can take up values from a set (chain) for example the weather forecast for the day. Different states are then defined for the chain for example in case of weather the states could be {Sunny, Overcast, Rain, and Snow}. For a random process to be a Markov Chain it has to fulfil the property called the **Markov Assumption** which is defined as: the probability of being in a state j depends only on the previous state, and not on what happened before, or simply put only the present weighs in on predicting the future, while the past does not matter [4]. Mathematically this can be stated as:

$$P(q_i = a | q_1 \dots q_{i-1}) = P(q_i = a \vee q_{i-1})$$

  Table 1 below shows the essential components for a Markov chain:

Table 1 Components of a Markov Chain

Component	**Explanation**
$S=\{q_1,q_2 \dots q_n\}$	A set of 'n' states
$P=\begin{bmatrix} a_{11} & \cdots & a_{1j} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ij} \end{bmatrix}$	A transition probability matrix, containing probabilities of moving from state i to state j.
$v=(v_1,v_2 \dots v_n)$	Initial probability distribution vector.

Fig. 2 shows a state diagram with the transition probabilities, the states are shown as the nodes in the graph. The transition probabilities shown in the

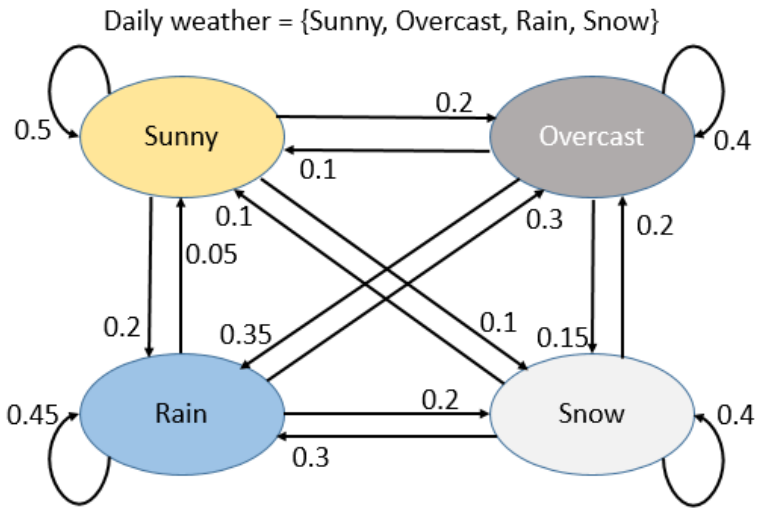

Figure 2 Transition probabilities' state diagram for a weather model

figure can also be shown in a matrix as:

$$P = \begin{bmatrix} 0.5 & 0.2 & 0.2 & 0.1 \\ 0.1 & 0.4 & 0.35 & 0.15 \\ 0.05 & 0.3 & 0.45 & 0.2 \\ 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix}$$

In the stochastic probability matrix P each element (i,j) represents the transition probability. The deductions that can be made from this matrix are:

- P^k: each entry will represent the probability to arrive from i to j in k steps.
- If probability of being at a state at time t=0 are given by a vector 'v'. The probability of being at state x at time t can be calculated as $P^k v$, the required result will be denoted by the x^{th} entry in the matrix.

- *Hidden Markov Model*

As opposed to the simple Markov Model, in an HMM, Markov chains cannot be observed directly. In contrast the outputs generated from each stage are visible to the observer. Taking the example of a speech recognition system, the parts of the speech are non-observable events the words and their positions in a sentence are characterized as observable events and also can be the inputs of such a system. Table below describes the components required to specify an HMM.

Component	**Explanation**
$S=\{q_1,q_2\ldots q_n\}$	A set of 'n' states
$P=\begin{bmatrix} a_{11} & \cdots & a_{1j} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ij} \end{bmatrix}$	A transition probability matrix, containing probabilities of moving from state i to state j.
$V=(v_1,v_2\ldots v_n)$	Initial probability distribution vector.
$O=(o_1,o_2 \ldots o_T)$	A sequence of T observations
$B=b_i(o_t)$	A sequence of observation likelihoods, they represent the probability of an observation o_t being generated from a state i

The Markov assumption as stated in the previous section also applies to HMM, but a 2nd assumption also comes into play which is termed as Output Independence and states that the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other state or observation [5,6]. This assumption is mathematically described as:

$$P(o_i|q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i \vee q_i)$$

2.3.2 **Neural Networks**

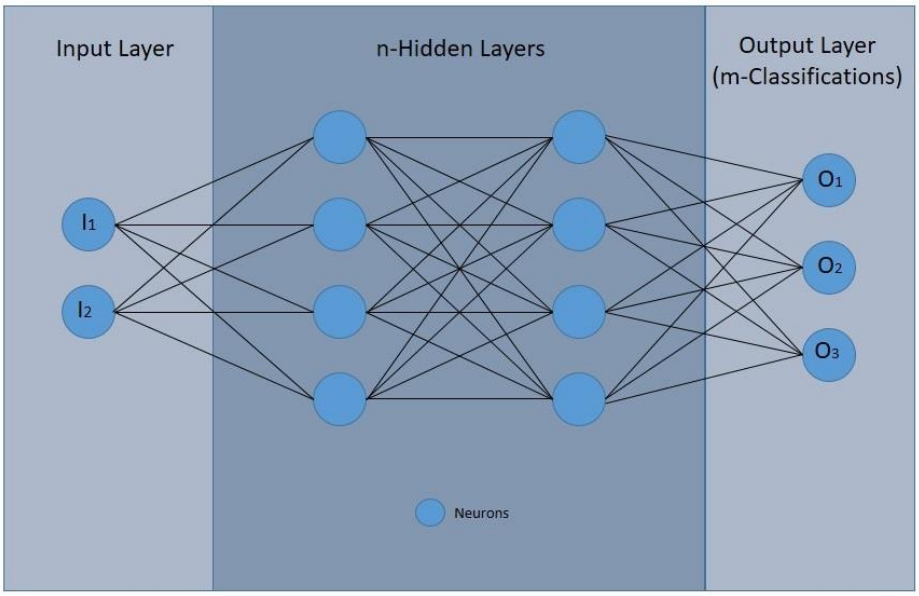

Figure 3 Pictorial depiction of a neural network

Fig above shows a neural network that can be used for a classification problem which can take some inputs e.g. an image and classify it into different categories e.g. classification of cats, dogs or none of the above. Neurons holds a weight corresponding to the body of the input or in case of the image the pixels and it is called an activation. The number of neurons in the input image will correspond to the size of the input data and in the output layers correspond to the number of classes the data could be classified into. Each connection between a neuron of one layer to the next has an assigned weight. The activations from each neuron from the first layer is then multiplied with the weight to get the weighted sum

$$WeightedSum = w_1a_1 + w_2a_2 + w_na_n$$

To compute the weighted sum between the range of 0-1 associated with the activation, a sigmoid function is usually applied, which is mathematically represented as

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Another alternative to using sigmoid function which is being used more and more in the modern neural network is ReLU (Rectified Linear Unit), this function is represented as:

$$ReLU(a) = \max(0, a)$$

Which is relatively simple as in any number below the set bias (0 in the equation above will be treated as inactive. Sigmoid activation leads to slow learning in deep neural networks while ReLu activation results in much faster models.

A bias is also usually applied to adjust the activation. The learning part of this process is finding the right weights and biases by looking over huge dataset automatically.

So in total the activation for a neuron in the next layer comes out to be:

Activation = sigmoid (Weighted Sum ± bias)

Mathematically this can be written as:

$$a^1 = \sigma\left(\begin{bmatrix} w_{00} & w_{01} & w_{0n} \\ w_{10} & w_{11} & w_{1n} \\ w_{k0} & w_{k1} & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_n \end{bmatrix}\right)$$

$$a^1 = \sigma(Wa^0 + b)$$

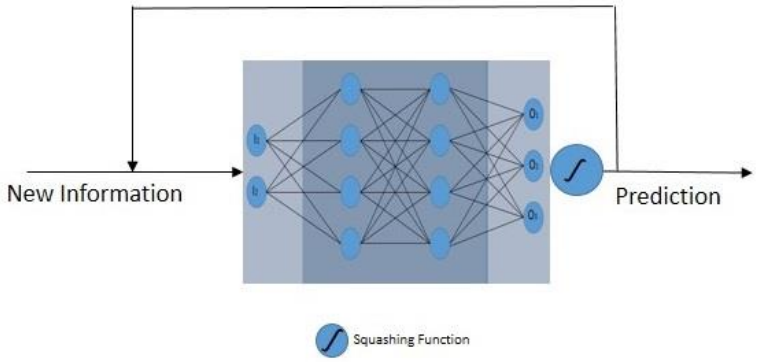

Figure 4 Simple representation of an RNN Model

There are several types of hidden layers that are employed in a neural networks. Convolution Layers are most commonly used for pattern recognition problems and Convolutional Neural Networks CNNs are able to recognise patterns in images using convolutional layers. Each conv. layer has a specific number of filters defined by the designer which are used to detect patterns e.g. edges, circles etc. The filters for identifying basic shapes are usually in the start of the networks and the deeper we go into the network the more complicated the filters become so much so that they are able to distinguish between actual features like eyes, mouth, face etc. Another important kind of neural networks is RNN (Recurrent neural network) and

LSTM (Long Short-Term Memory) which can be used for sequence identification and categorization.

2.4 Robot Operating System

ROS short for Robot Operating System is a collection of tools and libraries which is mainly designed to allow for writing robot software [30]. It provides the developers with low-level device control, drivers, package management and an infrastructure for message passing between processes. ROS systems are based on Publisher-Subscriber model. Active ROS processes are represented in a graph architecture and divided into different nodes. Each node can do its independent processing and can further communicate with other nodes. This communication is achieved by passing ROS messages. A node can be configured to be a publisher or/and a subscriber. In this scenario a publishing node does some processing and publishes some relevant information on a specific topic. This information could range from sensor data to images (as in our case). Any other node which subscribes to this same topic name will have access to this data as soon as it is published by the first node.

2.4.1 ROS-Master:

A ROS-Master node must always run for the ROS based systems to work. This master node provides naming and registration services to all the other nodes in the system and manages publishers and subscribers to different topics.

2.4.2 ROS Nodes:

A node in a ROS system is any process that performs some computations.

A ROS eco system is made of multiple nodes. The advantage of using nodes is break down of code into simpler components. Added isolation so that if a fault occurs in one node, other nodes usually stay unaffected by it etc. Node are combined together into a graph and communicate with each other by publishing and subscribing to topics or services.

2.4.3 ROS Topics:

Topics are named buses which are utilized by nodes to exchange information via ROS messages. Node in themselves are not aware of which other nodes they are communicating with instead they subscribe to the Topic containing the data that is relevant to them and publish the generated data to another relevant Topic if needed. Topics are generally used for unidirectional communication. ROS supports both TCP/IP based and UDP based message transport TCPROS is the default protocol used for communication.

For this project, the idea was to distribute the system between image processing tasks and machine learning (ML) tasks. The image capture and processing is done on the android device. Once the ROI has been extracted it should be transferred to a remote server running the ML module of the project which can then infer from the captured image-sequence the word or phrase that has been said.

For implementing this architecture several alternatives are available. For example the ML component (MLC) could be run on a cloud server (for example on Amazon Web Services) exposing the interface of the MLC via APIs. The android device can then transfer the images to the MLC via API calls and once the processing has been done on the cloud server the resulting string can be passed back to the android device via another API in JSON format.

Another option for implementation which does not require AWS, and in addition API development, was to use ROS. ROS provides the same architecture but on a simpler and lower scale. The application will again be divided into two parts Image-Processing Component which is to be run on the android device, and MLC to be run on a remote PC which must be on the same network as the android device running ROS-master node. For this architecture the Android application must also act as a ROS-node and once the ROI has been extracted it publishes the images to a ROS-Topic. The MLC running on the remote PC subscribes to this same topic. As soon as the images are available on the topic they are picked up by the MLC and the processing is performed. The results can either be shown on the PC or again published to another Topic to which the android device is subscribed to, so that the results can be shown on the device.

3 Application Requirements

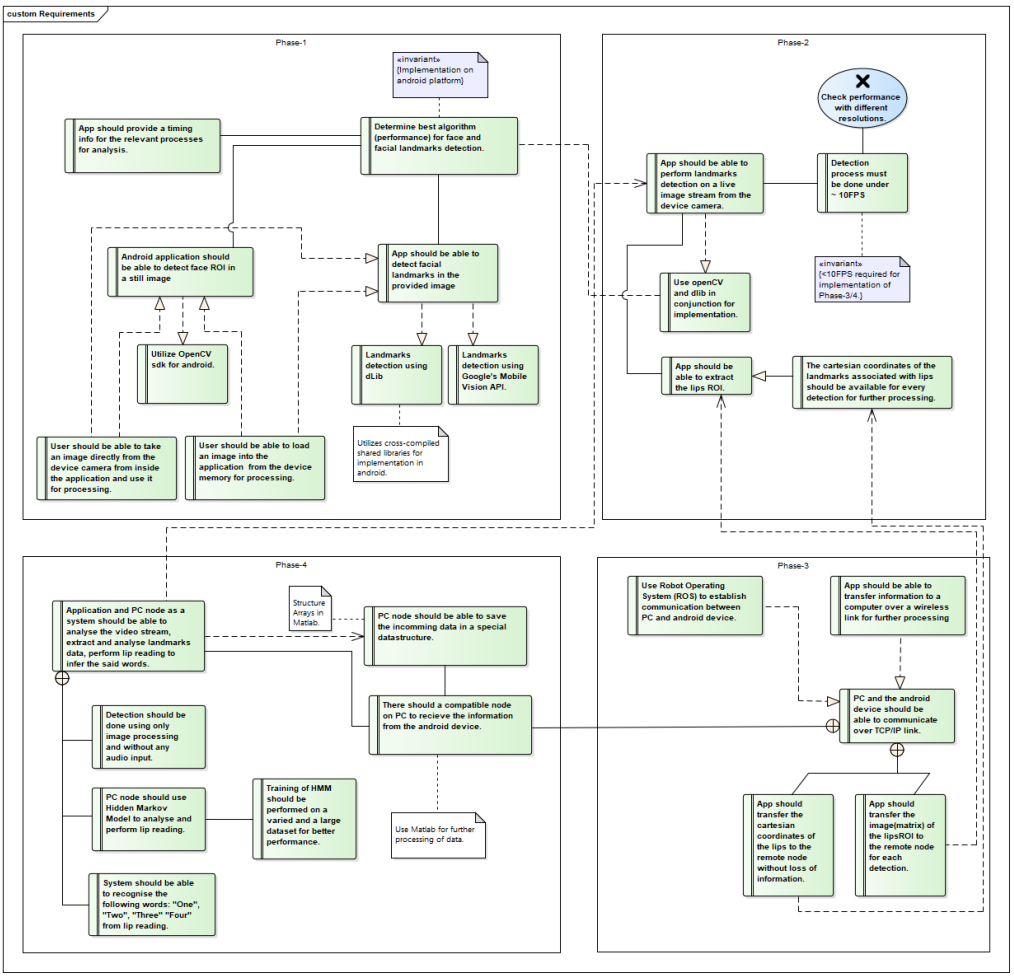

Figure 5 Requirements Model

The requirement analysis for the application was done to come up with what needed to be done and to manage the development lifecycle of the application. The figure above shows a UML representation of the developed requirements. As can be seen from the requirements model, the development has been divided into four phases to allow for easier development and clear goals. Each phase and the requirements contained in it are described in the following sections.

3.1 Phase -1:

Analysis and comparison of facial and landmarks detection techniques:

Phase-1 encompasses study of relevant image processing libraries and techniques which could be used for the application. A look into the timing and performance parameters of these techniques is to be performed so that the best approach could

be decided for the development of the application in the later stages. The requirements corresponding to this stage are listed below:

Req-1.1: Determine best algorithms for face and facial landmarks detection, identifying parameter is timing performance.

Req-1.1(a): Application should run on android devices.

Req-1.2: Application should provide execution times for the relevant processes for analysis.

Req-1.3: Application should be able to detect faces in still images.

Req-1.3(a): User should be able to take an image using the device camera and use it for processing with in the application.

Req-1.3(b): User should be able to load an existing image from the gallery and use it in the application.

Req-1.3(c): OpenCV sdk for android should be used for face detection.

Req-1.4: Application should be able to detect facial landmarks in the given image.

Req-1.4(a): Utilize dLib for facial landmarks detection.

Req-1.4(b): Utilize Google's Mobile Vision API for facial landmarks detection.

3.2 Phase-2:

Real-Time Facial Landmarks Detection on Live Camera Stream:

In phase-1 once the best approach has been identified, phase-2 is to extend the existing implementation from its application of still images to live feed from the on-board device camera. And to make the application as close to real time as possible given the hardware constraints of the target device.

Req-2.1: Android application should be able to perform facial landmarks detection on live image stream from the device camera.

Req-2.1(a): App should utilize OpenCV and dLib in conjunction for facial landmarks detection

Req-2.2: Application should run close to or greater than 10FPS.

Req-2.3: App should be able to extract Lips ROI.

Req-2.3(a): Cartesian coordinates of the lips detection should be available for further processing in later stages.

3.3 Phase-3:

Implementation of wireless data transfer to a remote computer:

Phase-3 was to incorporate distributed processing capability in the application. Such that the image processing is to be performed on the android device but for further processing, the extracted images and information will be transferred to a remote computer running the Machine Learning Component of the application. This component will be responsible for processing the transferred images and deducing the said word or phrase from the given information.

Req-3.1: App should be able to transfer information over a wireless connection to a remote computer for further processing.

Req-3.1(a): Remote PC and the android device should be able to communicate over a TCP/IP link

Req-3.1(b): Use Robot Operating System for establishing data transfer capabilities.

Req-3.2: App should transfer the Cartesian coordinates of the lips to the remote node.

Req-3.3: App should transfer the images of the Face/Lips ROI to the remote computer.

3.4 Phase-4:

Implementation of Machine Learning Component:

Phase-4 is the final phase which brings together the work done in the previous parts and ties them together to produce the whole application. In this phase the Image processing results done on the device should be processed further on the computer. The chosen ML algorithm should then infer from the data the said word or phrase.

Req-4.1: App and PC should work as a system to analyse the video stream, extract facial landmarks and lips ROI and perform lip reading to infer the said word/phrase.

Req-4.2: A ROS node on PC should be able to fetch and save the incoming data from the mobile device.

Req-4.3: PC Node shall use Tensor Flow to analyse the incoming images and perform lip reading.

Req-4.3(a): Training of Tensor Flow model should be performed on a varied and a large dataset for better results.

Req-4.4: System shall be trained on a set of pre decided words and phrases.

4 Architecture Design

As defined by the requirements the system is designed as a distributed system with two main parts.

- Android Application
- Machine Learning Module

The android application is to act as the interface to the user. It has been designed to utilize the on-board device camera of the android device to capture images/videos. No recording or processing is performed unless the user initiates the process from the Start/Stop button.

Once the process is started by the user indicated by the press of the Start button the algorithm starts to process the video feed from the camera frame by frame. The application is developed using OpenCV framework. *CameraBridgeViewBase* class of the OpenCV is used to control the camera operation. This class implements the behaviour for interactions between android camera and the OpenCV library, including but not limited to the processing of the frames, enabling and disabling of the camera sensor and drawing of the image on the screen. Apart from this *CascadeClassifier* class from the OpenCV *objdetect* package is used for detecting faces in the frames. Although all the faces present in the frame are detected the further implementation on MLC is limited to processing a single face in the frame for now.

Dlib is used for detection facial landmarks. The dLib shared libraries created by Ttzutalin are imported in to the android application and the interface defined by jni functions are used to create the structure of our application.

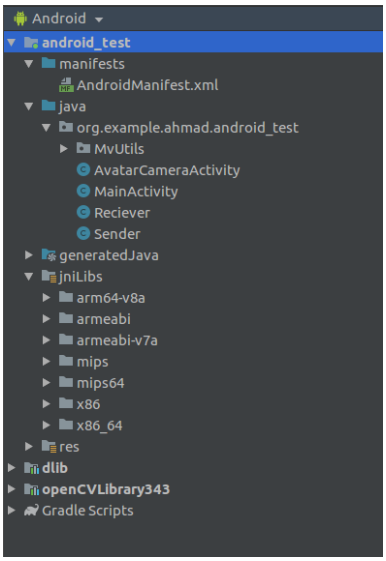

Figure 6 Modules of the Android application.

Fig.6 shows the modules of the Android Application. Following sections give some details into the modules and their contents:

4.0.1 AndroidManifest.xml

AndroidManifest.xml file is a must have file for each android project and it contains information about the application's package name and

components contained therein among other things. This information is shared with other tools and applications in the android eco system like the operating system, build-tools and Google play which is used to publish apps to end users. Some of the important information that is contained within this file is the application's package name as mentioned earlier which is used to infer a unique app identifier for the application, The names of all the components in the application for example activities, services etc., device configurations and permissions that the app can have by default are also contained in this file like for our App, permission is granted to access the camera to capture video feed and to read/write to the storage by the following code snippet in Fig.7

```
package="org.example.ahmad.android_test">

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
```

Figure 7 Preset permissions for the application in Android Manifest

4.0.2 Java Component

The folder named java contains all the source code files associated with the project. The project in its current state has only two activities, and is divided further into four classes containing the code used for performing various functions, which are described in a latter section.

4.0.3 JniLibs

Android provides the capability to build or use pre-built native libraries in the android application. In order to allow Gradle to package prebuilt native libraries along with the APK the .so files are placed in the JniLibs folder corresponding to their respective hardware architecture.

4.0.4 Dlib

This folder contains the prebuilt shared libraries and the Jni-Interface for the dlib library imported as a sub module in the project. The API of this library is used with the source code of our application to build the required logic.

4.0.5 OpenCVLibrary343

OpenCV SDK version 3.4.3 is imported into the project as a sub module as well and is later used for performing face detection and handling camera related tasks

4.1 Application Flow

Fig.8 shows the flow of the application. Once the application starts *MainActivity* is started by the Android system which in turn starts the *AvatarCameraActivity* via Android Intent. The *onCreate()* method of this class initiates the required libraries as well as the device camera. In case the *processImage* flag is set to one (user input), the current frame is converted in to a bitmap and sent to the OpenCV face detector to extract the face ROI, if a face is not detected in the current frame the system moves on to the next frame and does the process again if a face is found in the frame the input frame is cropped to the faceROI and sent to the dlib module for detecting landmarks the results are returned and shown on the device screen in a picture-in-picture mode. The results are also packaged into a ROS message (details in a later section) and published to the defined ROS topic.

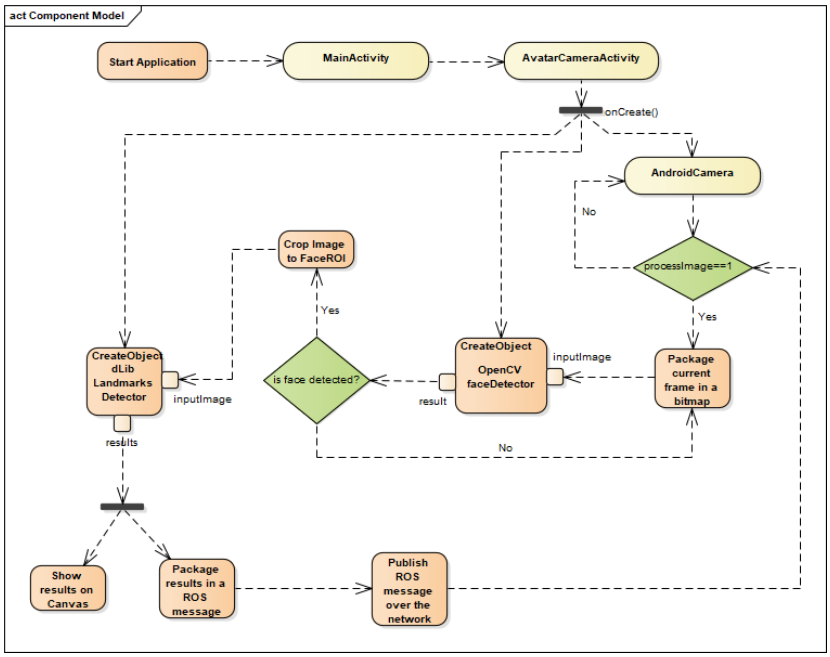

Figure 8 UML diagram – State Flow Diagram

4.2 Component Diagram

The application is divided into twelve components communicating with each other via different interfaces ranging from functions calls to ROS topics. The combination of all these components makes up the application work as a unit. The fig.9 below shows the information flow between the components as well as their associated interfaces.

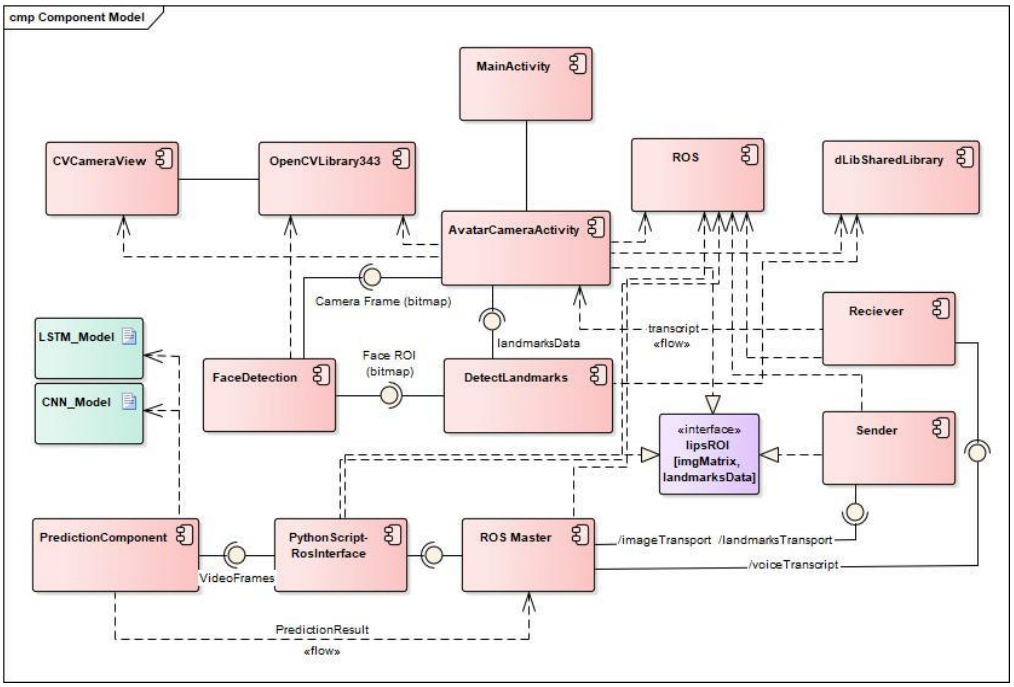

Figure 9 UML diagram - Component Model

4.3 Class Diagram

The developed system mainly consists of five classes, at the beginning an extra *FaceDetection* class was also implemented but in a later revision the code to extract face was moved from this class to *AvatarCameraActivity* class. As described earlier the application contains OpenCV android sdk version 3.4.3 and dlib prebuild shared libraries and the ROS package. The logic implemented by the code uses the API of these packages to perform its functions. There are further two dependencies on pre trained models for both OpenCV's face detection algorithm and dlib's landmark detection. In Fig.10 they are shown as artefacts:

- lbpcascade_frontalface.xml
- shape_predictor_68_face_landmarks.dat

The classes and their functions are explained in the sections below:

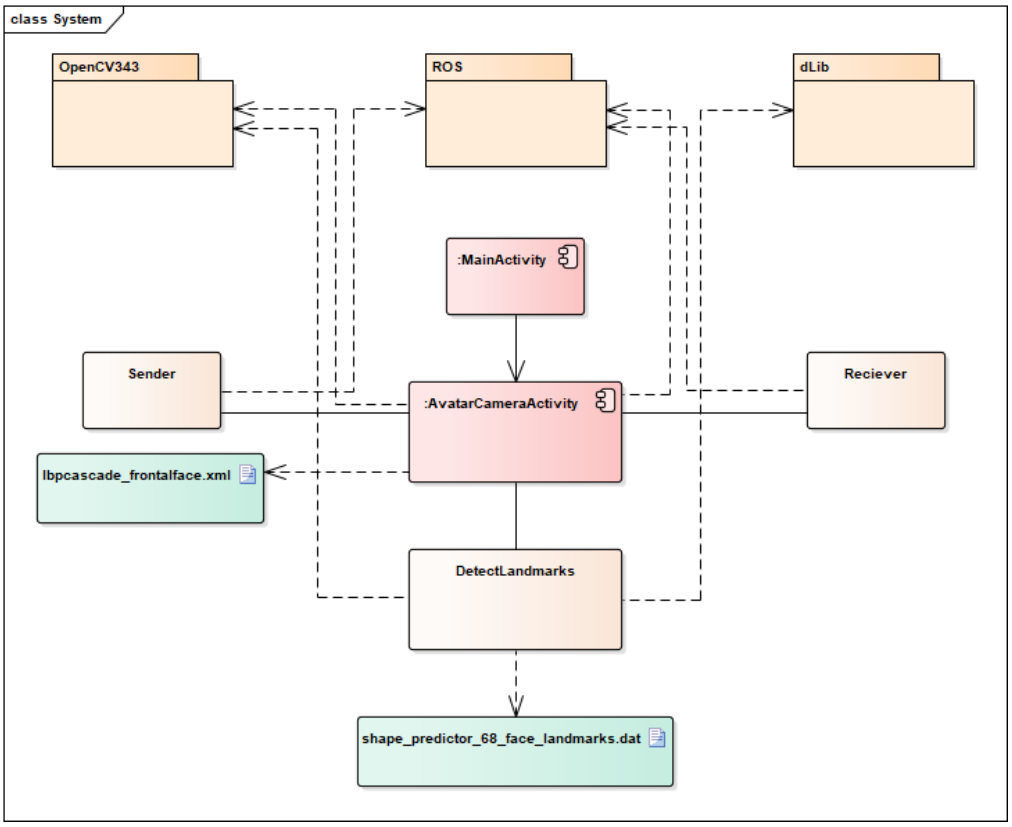

Figure 10 UML diagram - Class Diagram

4.3.1 MainActivity

The functionality of the *MainActivity* in this instance is not much except for calling the *AvatarCameraActivity* via Android intent. *MainActivity* is also needed in the android architecture as this is the first activity that is started by the operating system when the application is started.

4.3.2 AvatarCamerActivity

This Activity is the backbone of the whole application. It handles the starting up and stopping of the camera, detection of faces and landmarks and also publishing of the landmarks data over the ROS network.

For handling the frames captured from the on-board camera the following OpenCV function is used:

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame)
```

This is a pre-built OpenCV method and is invoked the frames need to be processed and delivered on screen. The function returns a modified frame in matrix form ready to be displayed on the screen. This function periodically calls the *faceDetection* method and then the *landmarksDetection* methods to extract the required information.

The landmarks data is then sent to the Sender Class to be published over the ROS network to the set Topic.

4.3.3 DetectLandmarks

In the context of application's architecture this class has been placed in the *MvUtils* package as this class is a helping class and provides and interface to the underlying dlib package and its java native interface. The class has two constructor methods, one provides the option to initialize the object with the application's context object and a bitmap image. This constructor is provided if the implementation for landmarks detection on a single image is needed, and the constructor was developed in the previous version of the app. The second constructor only takes the context object to initialize the class's object.

Context [15] in Android system provides an interface to global information related to the application's environment e.g. app's resources and classes and other operations like passing and receiving intents for launching activities.. The implementation of this class is provided by the Android system. The constructor checks if the shape model file for dlib is already present on the device storage or not, in case it is not present the model is downloaded from an external link and saved. For this operation permission to access the internet and read/write permission to the storage is needed. It has already been set in the *AndroidManifest* file but might have to be set in the device settings.

There are several different methods implemented in the class for detecting faces using OpenCV as well as dlib separately, the method being actively used in the current implementation is

```
public Bitmap detFacesFromBitmap(Bitmap b)
```

The method takes in a bitmap image and identifies the face in the image and then detects landmarks on the detected face. As per the dlib documentation landmark points labelled between 49 and 68 correspond to lips on the face. For our application these points are the ones in focus so the method only extracts these points and adds them to an array-list named *landmarkPoints*. This function only return the bitmap

with an overlay of the detected landmark points. The interface to get *landmarkPoints* is provided by the method:

```
public ArrayList getLandmarkPoints()
```

An important part defined by this class is the location where the dLib shape model should be in the device and is defined in the code by the following line:

```
NewFile(Environment.getExternalStoragePublicDirectory
(Environment.DIRECTORY_DOWNLOADS).getPath()+File.separator+modelFileName);
```

As can be seen in the definition the shape model is checked in the External storage in the directory defined by the *DIRECTORY_DOWNLOADS* constant in the Environment class of android system and is defined as “Download”

4.3.4 Sender

The sender class is a ROS class for publishing data to topics on ROS network. The class has a simple constructor that does not take any external parameters and initializes the names of the topics on which the data is to be published. The topic is set to *“image_transport”*. The publisher object for compressed images is defined as:

```
Publisher<sensor_msgs.CompressedImage> imagePublisher;
```

Sensor_msgs/CompressedImage.msg [16] is a message type defined by the ROS system that deals with transferring different formats of compressed images, in a typical ROS system this is used to handle data from on-board cameras on a robot. The message body is defined to contain a Header, format of the image in string format, currently supported formats are jpeg and png, a data buffer of type uint8[] to contain the image data.

4.3.5 Receiver

The receiver class is implemented to receive a string message of ROS message type *std_msgs.String*. As per the current implementation the remote module does not publish any Strings back but as per the architecture the result inferred by the ML component running on the remote computer will publish the results on a ROS topic and the Receiver module shall pick it up to be displayed on the device to the user.

4.4 Sequence Diagram

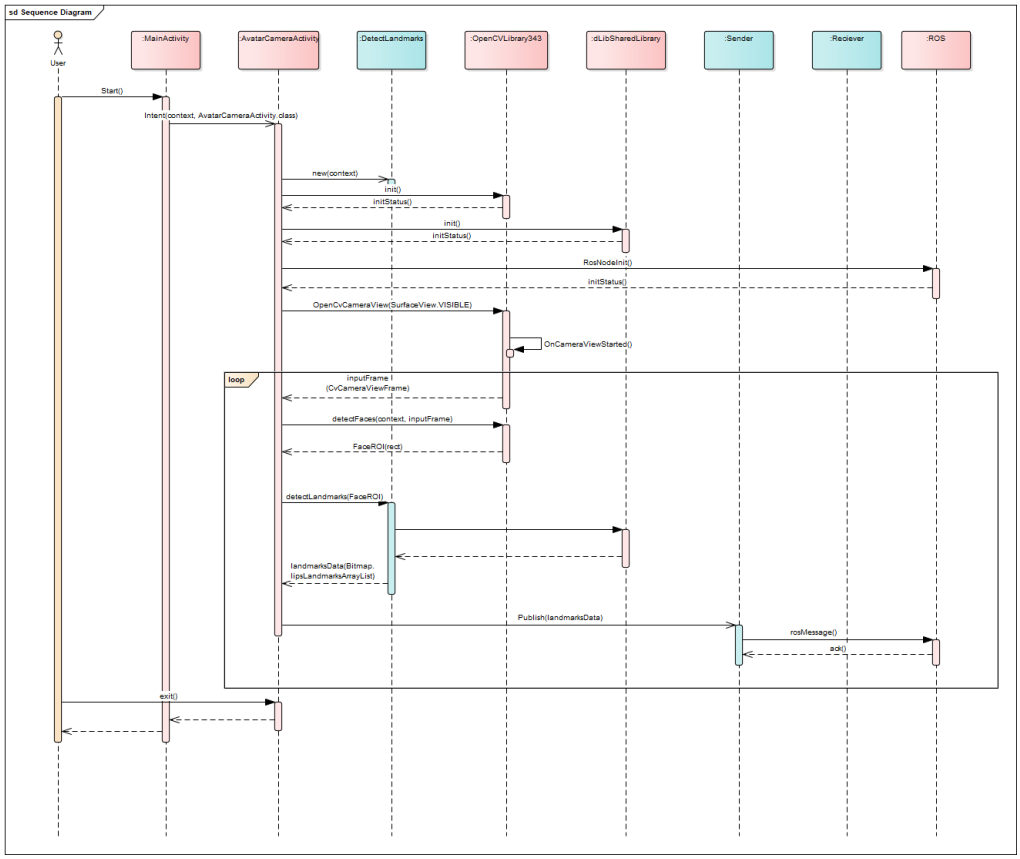

Figure 11UML Sequence Diagram

Fig.11 above shows the UML sequence diagram. This sequence diagram illustrates the flow of the process on the device side only. The application starts in the *MainActivity* from which the *AvatarCameraActivity* is initialized. The initialization process includes *init* of both the OpenCV and dLib libraries as well as the defined ROS nodes. During this initialization process, user is prompted to enter the ip address of the ROS master node as well.

Once all the initiations have been run properly the on device camera is started by the *OpenCVCameraView* class and the frames are drawn on the screen. Further process from this point onwards only takes place upon the request of the user which is indicated by a button press on the screen. Once this processing starts it continues to run in a loop which can only be stopped by a user interrupt also indicated by the press of the “Stop” button.

Inside the processing loop, the frames are captured by the *OpenCVCameraViewFrame* and transferred to the *AvatarCameraActivity*, which is the main activity class for handling all the image processing tasks and routing the

outputs. *AvatarCameraActivity* calls the *faceDetection()* function passing the *inputFrame* image matrix as the parameter, under the hood of this function is the OpenCV face detection algorithm. This method detects faces in the input frame and returns a cropped image of the face back to the *AvatarCameraActivity*.

The image containing the FaceROI is then passed to *detectLandmarks* function which passes the image to an object of the *DetectLandmarks* class. An image with an overlay of 64 landmark points, their coordinates and the coordinates of the landmarks associated with the lips is returned from this method call:

```
landmarkDetector.detFacesFromBitmap(croppedFaceImage);
```

Once the landmark points have been received the next step in the process is to perform the lip reading part using the ML component. To accomplish this the data is transferred over the ROS network to the topic named "image_transport"

The ML component running on the remote PC takes over from there, these sequence

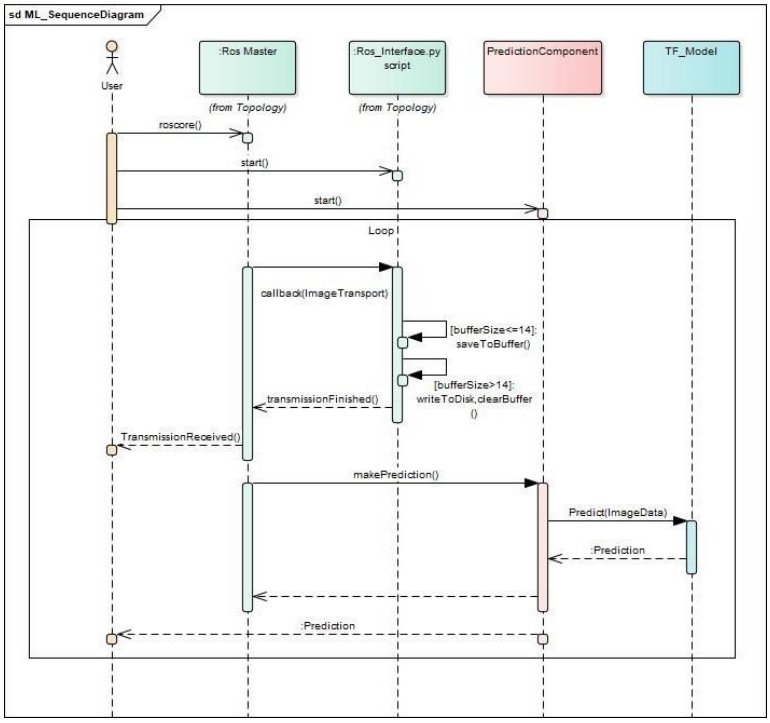

Figure 12 Sequence Diagram - Remote ML Component

diagram for the ML component is shown in Fig.12.

5 Application Development and Challenges

5.1 UI development

The android system draws the user interface from xml files placed in the resources/layout directory. XML is the abbreviation for eXtensible Markup Language and is defined by www consortium's XML specs and several others, this markup language is meant to be both human and machine readable.

For version1 of the project the goal was to compare several different image processing techniques and document their performance parameters to determine the best approach for implementing the application described in this document. Fig.12 shows the first design of the application UI

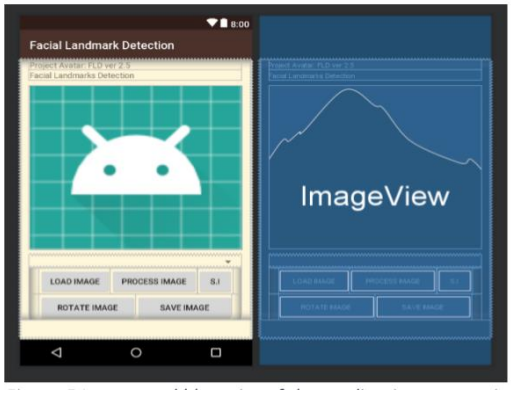

Figure 13 UI of Avatar app-ver1

In the current version of the application there was no need for all of the UI elements so the UI was revised with the *MainActivity* only including a button to start the main process while the *AvatarCamera* activity now has the Layout as seen in fig.13. The UI layout is quite simple with The Parent block being a *FrameLayout* containing three further child elements.

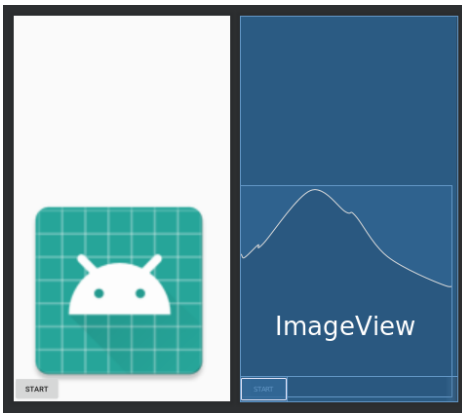

Figure 14 Current UI of the application

JavaCameraView fills the entire screen and the frames from the camera are drawn on this view by the OpenCV module.

An *ImageView* is embedded as a child element of this. By default this view is hidden. When the image processing starts the frames with the processed image with the landmarks drawn over the original face is displayed on this *ImageView* and it acts as the Picture-in-Picture.

A button is provided inside a *LinearLayout* and gives the user control over starting and stopping the detection process.

5.2 Performance improvements

With the normal implementation at the time of application's development the average frame rate without any image processing was 23-25 FPS, and with the undelaying image processing algorithms running the frame rate dropped to 3-4 FPS. This frame rate is not enough for performing lip reading or speech processing. Improving the frame rate was the major challenge in the development process of the application. Following steps give a brief overview of the steps taken to improve the frame rate.

- Convert the Input frame from RGB to Gray-scale.
- Face-detection skips some frames instead of running on every frame.
- Lower frame resolution (optimized for good results and performance).

Face detection is performed using OpenCV in order to crop the input frame to the FaceROI, removing the unnecessary portions of the image hence making the part of landmark detection faster. Assuming the usage of the application it is highly unlikely that the position of the face will displace too much during the usage. Keeping this assumption in mind the face detection process does not need to be executed on every frame. To achieve best results some setting were tried out to achieve a balance between performance and accuracy. Setting the rate of face detection to once per 10 frames brought the FPS while detection process up to 7, decreasing it to once per 20 frames brought it up to 9-11 FPS and the accuracy remained intact. Decreasing it further beyond this point had a negative effect on the accuracy as the position of face was too easily lost during the process. The setting to set the frame rate is done by the following line of code in *AvatarCameraActivity*:

```
private int frameSkipCount=20;
```

The associated public get and set functions for the private variable *frameSkipCount* are also provided within the scope of the class.

The resolution by default for OpenCV's camera handle is set to 960x720, and the resolution for the testing device the Huawei Media Pad M5 Pro is 2k 2560x1600. These resolutions are very large and images of this size take a huge amount of

processing power to go thorough image processing algorithms. To make the performance of the application better different resolutions were tried and 800x800 resolution produced the best result. Following code snippets set the frame resolution in the AvatarCameraActivity class:

```
private int MAX_WIDTH = 800;
private int MAX_HEIGHT = 800;
mOpenCvCameraView.setMaxFrameSize(MAX_WIDTH,MAX_HEIGHT);
```

The private constants define the resolution to be used by the OpenCV's *CameraBridgeViewBase* and the method call to *setMaxFrameSize* sets the parameters.

5.3 Machine Learning Module

For implementing the AI module to infer which phrase has been said from the input video frames from the device Tensor Flow was chosen as the ML. Tensor Flow is an open source library that can be used to develop and train ML models. Tensor Flow is an open source software library for numerical computation using data flow graphs. The architecture of Tensor flow allows to deploy the ML model onto multiple CPUs/GPUs, personal computers as well as mobile devices using Tensor Flow Lite. Tensor Flow Lite converts a model into a compressed flat buffer which can then be deployed on a mobile device e.g. and android application or an IOT device like a raspberry Pi. This advantage is one of the main reasons for choosing TF as further down the line the application can be made completely standalone without any dependency on remote PCs by moving the ML module from the PC to the device.

Two model architectures are implemented to do a comparison as to which approach produces better results, following section describes the implementations for both architectures:

5.3.1 Data Preparation:

Data Preparation for training the ML model is the most important and tedious step in machine learning development. For our problem of lip reading some data sets like VGG Lip-Reading dataset and MIRACL-VC1 data sets are available. For this problem I have used dataset obtained by myself from friends and colleagues, so the data set was not adequately big but for demonstrating the software and the process it should be enough. Three phrases as follows were recorded from 6 people and each phrase was repeated 8-10 times. The table below shows the data classification:

Table 2 Data to class mapping

Phrase	Key	Class
How are you?	P01	1
Nice to meet you!	P02	2
Thank you.	P03	3

For pre-processing of the arranged data python scripts were written. The scripts are available on the github repo of the project. The details are given below:

- *ExtractFaces.py*

  The first script to be run in the data preprocessing step. This script utilizes OpenCV and dlib libraries for python to go through all the directories containing the data set images (Frames extracted from the videos) extracts the faces and the lips ROI and crops the input image to the lips ROI and saves the image in a specified directory.

- PrepTrainingData.py

  The second script takes all the new saved images from the first step and arranges them into directories corresponding to each class so the data is arranged for generating a csv file containing data to class mapping.

- *CreateMappingfile.py*

  This script crawls through all the arranged training dataset and generates a .csv file that contains the mapping of each image to its respective class specified in an input configuration file.

  A file named "ImageToClassMappings.csv" is generated from this script which contains the names of each image paired with the class they belong to as shown in the table above. This file is used by the training script to generate a one-hot encoding sequence.

- *TrainAndTest.py*

  The script takes in the data pre-processed by the previous scripts and the .csv mapping file and creates a TF model, trains, evaluates and saves the model to disk.

Data preprocessing and training the model for LSTM models was handled differently and presented its own set of challenges. The scripts for data preparation and training the model are available on this link.

The training data used for training the LSTM model consists of 130 data sequences in total for the three classes, each containing a varying number of frames. The important constant that have to be defined before the data pre-processing step were. SAMPLES, TIME_STEP and FEATURES. In the prepared dataset, as stated earlier, there are 130 samples. The time steps were set to 42 defining that each sequence is expected to have 42 frames. As the TF model expects the sequences to be of the same size. In case a video containing more of less frames than 42 is received it is padded with a zero matrix on both sides, in case of more than 42 frames extra frames are dropped from both ends. FEATURES define the number of pixels in the input image i.e. 244*244.

5.3.2 Convolutional Neural Network:

For implementing a CNN model, VGG16 model for Keras was used employing the concept of transfer learning. This is a 16-layer network used by the VGG team. The details about the model can be found in the cited paper [17]. As it is a pre-trained model, extra layers can be added in front of it including the output layer and the model can be retrained to extract the features given in our custom data set. The VGG16 model works on images of size 224*224 so a resizing operation is performed on all the incoming images. Once the training process has been completed the model is saved to disk, so that it can be used by the prediction module to process the incoming images from the device.

5.3.3 Long Short-Term Memory Recurrent Neural Network:

As out problem is more of a sequence classification problem, using an LSTM model for solving this makes more sense. For LSTM three configurations were originally chosen to be implemented as below:

- Single Layer LSTM Model (Vanilla LSTM)
- Stacked LSTM
- CNN-LSTM Hybrid

Due to inadequate processing power of the PC available the third model could not be tried out as it always ran out of memory during the training process.

The Single Layer LSTM model as apparent from its name contains just one LSTM layer which is trained on the input sequential data. A variable number of parameters

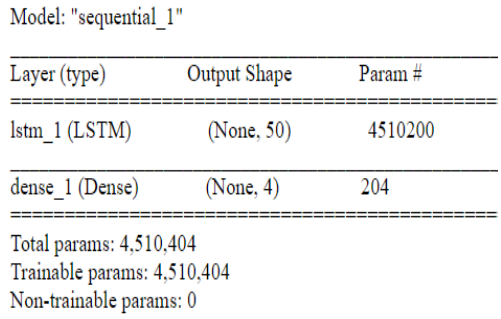

Figure 16 Model summary - vLSTM

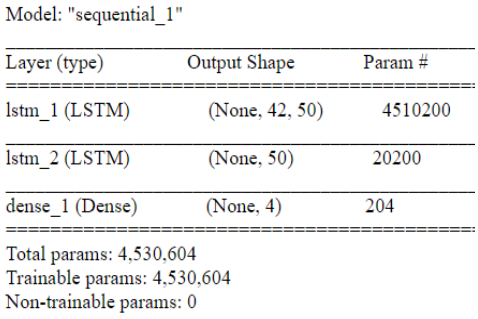

Figure 15Model Summary - sLSTM

(neurons, epochs, batch size) were tried out to obtain the best possible configuration. The stacked LSTM model contains multiple LSTM layers, the model used in this case contained two LSTM layers. The same process was used on this model as well. A comparison is provide in the results section. Fig.15 and Fig.16 show the model summary for both implementations.

6 Results

6.1 Android Application Performance Statistics

Getting optimum performance from the application so that a decent frame rate for the video during the image processing tasks were underway was one of the most important and challenging tasks. Implementing the steps described in one of the previous section a frame rate of 9-11FPS was achieved which is good enough for the application at the moment although a more optimum FPS for speech processing application is considered between 20-24FPS.

During Testing the application executes perfectly without any memory leakages and the memory consumption of the application is also normal as can be seen from the Android Device Analysis plot in fig.16

Figure 17 Device's Memory Usage Analysis

As can be seen from the plot, the memory consumption of the Camera Activity while the image processing tasks were running is less than 128MB including other device tasks like native code and graphics engine.

Some problem with the energy consumption of the Application were observed as during running and also while on stand-by warning messages from the Android system are shown that the application is draining the device's battery. A solution for this has yet not been implemented. The plot of energy consumption also shows that the application pull in considerable amount of energy while running and still consumes energy while on stand-by.

The CPU usage of the application is also quite optimum, only using 10.1% of the

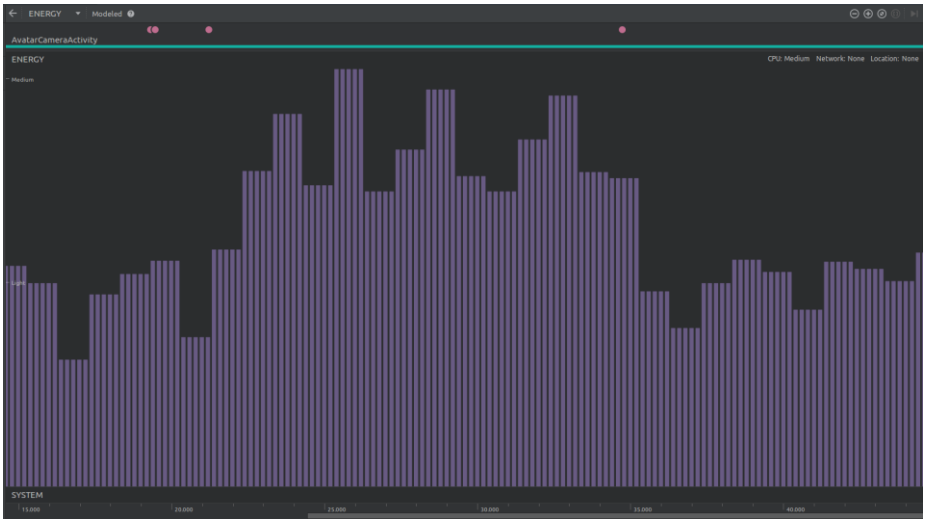

Figure 18 Energy consumption of the application

available processing power in total and only some small peaks can be seen in the plot when the image processing tasks were running.

Figure 19 Application's CPU usage

6.2 Machine Learning Module

As mentioned in an earlier section the data set was small as compared to what is normally used for training ML models for such sequence classification problems with about 150 samples covering 3 phrases. This same data set was divided into training and test sets for training and validation. The results of training the models described in Section-5.3 are described in this section.

The CNN network produced very abnormal results for both Loss and Accuracy as it shows total convergence and an Accuracy of 100% although the results obtained showed quite a different story. The plot for loss and accuracy for this model is shown in fig.19. The actual results obtained were quite inaccurate on real data from the device, amounting to about a correct prediction in 1 in 20 samples.

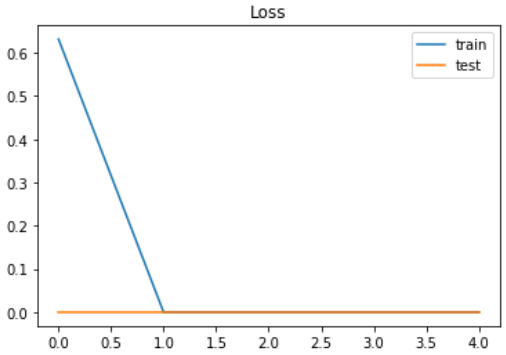

Figure 20 CNN model's loss graph for train and test batches.

The LSTM models showed promising behavior with accuracy ranging from 55-73% approximately.

Several configuration were tried of for LSTM models by changing the number of epochs and neurons a summary of the results is shown in the table below:

Table 3 Stats for different LSTM model configurations

Model	Neurons	Epochs	Batch Size	Loss (%)	Accuracy (%)
vLSTM	30	150	10	14.7	50
	50	100	10	13.2	53.8
	50	500	10	12.7	65.4
	200	500	10	13.7	65.4
sLSTM	50	500	10	13.5	53.8
	200	500	10	12	65.4

The configurations with the best results for both models are highlighted in green. The training time for the models ranged between 2-4 hours depending on the epochs and number of neurons with the size of the data set being constant.

Fig.21 and Fig.22 show the plots of Loss and Accuracy for both configurations highlighted in Table-3

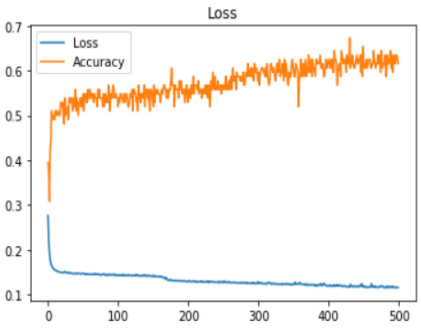

Figure 22 Loss vs. Accuracy vLSTM Model

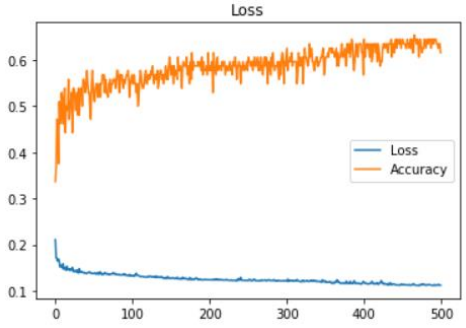

Figure 21 Loss vs. Accuracy sLSTM Model

The result of the prediction are shown in a terminal window at the moment. The process to setup the application on the PC and the device is given in the Documentation Section-8. Fig.22 shows the output terminal windows showing the results and the other running processes for the ML-Module.

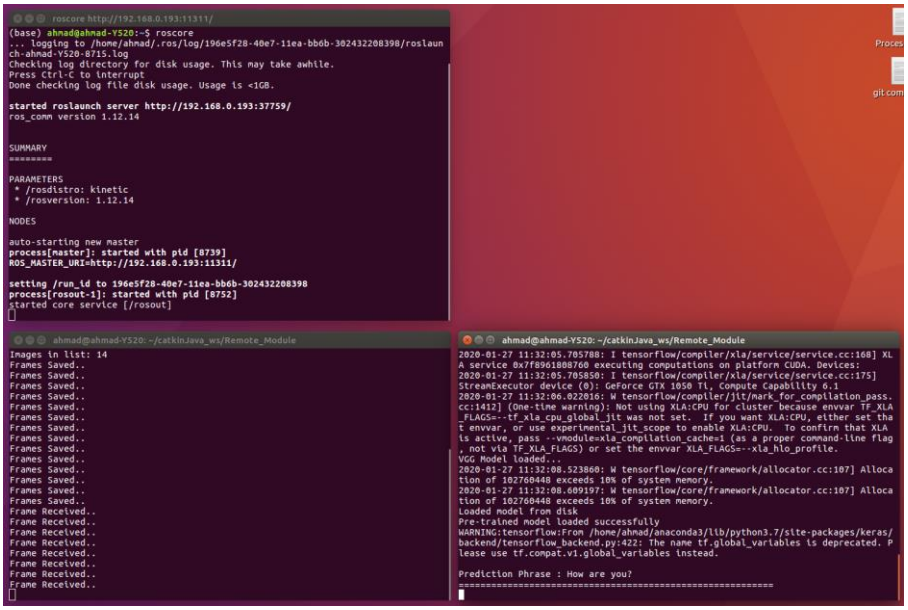

Figure 23 UIs showing Prediction, RosCore and ROS-APP Interface

7 Conclusion

The use case that we have tried to tackle in this project was to develop a tool for logopaedics and speech therapy. The challenge behind this idea was to make this tool a cheap solution that is easily accessible and readily available to the masses. The solution for this – A smart phone application that employed the latest in image processing technologies and Artificial Intelligence tool set, gave rise to Project Avatar.

Optimizing the performance metrics of the application to produce a somewhat close to real time application was one of the major challenges in the development cycle, distributing the system between two nodes with the remote PC running the AI module and the mobile device running the image capturing, processing tasks as well as acting as a UI with the end users, gave very good results.

In the current application OpenCV and Dlib libraries have been used for performing feature extraction, one alternative for this could be to develop and train a single Tensor flow based model to extract lips from the images directly. This approach would remove the dependencies on these two libraries and having to maintain extra code in the future.

Due to the limited data set used to train the results obtained for ML model training were not optimal, A larger data set for further training the model will improve the results by a huge factor. Although amassing a large enough data set is a challenge in itself, unless a dataset that is already available from 3rd parties is used which required which in case of BBC's lip reading data set required some legal proceedings.

In conclusion the distributed architecture proved to have a good effect on the application's performance. Using TF based machine learning models was also a good choice instead of using Matlab toolboxes for Machine Learning which makes the application more independent (removes Matlab dependency), and LSTM based models should be preferred for the speech analysis problem.

8 Future Work

The application in its current state only aims to infer phrases said from performing image analysis. In the future the application can be extended to also take in to account speech input to improve the results. The ultimate goal of the application is to be a tool for speech therapy which is also further down the pipe-line once the speech analysis part has been perfected.

The current solution in place for the remote PC is based on ROS which imposes the limit that the ip address of the remote PC running the ROS core must be known and the two devices should be on the same network. To remove this dependency the ML component can be moved to a remote server e.g. Amazon web services and access to the Model and the Prediction component and the results produced by these components can be provided via RestAPIs (Also the topic of my thesis). With this approach ROS will not be needed anymore.

Another approach that can be explored in the future is to convert the model to TensorFlowLite version using the APIs provided by TensorFlow and deploy it directly on the device. Although this approach can have an adverse effect on the application's performance as well as remove the distributed architecture of the application which will in turn make it impossible to update the model without effecting the application's code base.

9 Documentation

9.1 End-User Documentation

The application can be installed by placing the provided apk file on the android device storage. An Android Package Kit (APK for short) is the package file format used by the Android operating system for distribution and installation of mobile apps. In some cases installing from apk files not from the google play store in not allowed by the Android system to bypass this the following steps can be followed:

Go to Menu > Settings > Security > and check Unknown Sources to allow your phone to install apps from sources other than the Google Play Store.

Once the application has been installed successfully, it can be started from the icon as shown in the screenshot in fig.23.

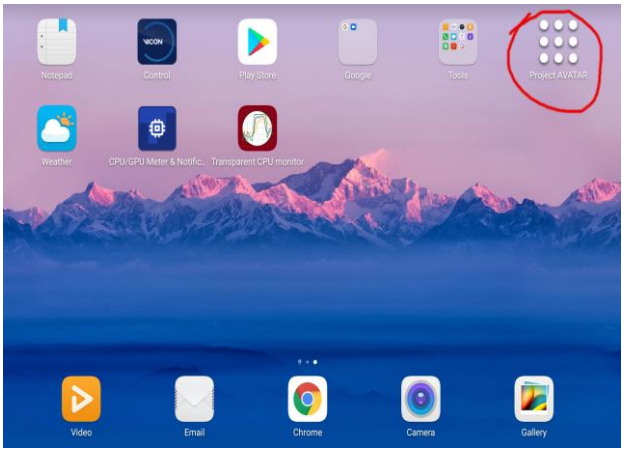

Figure 24 Application Icon

The start screen only gives a button to start the process, clicking on the button named “Project AVATAR” will take you to the next process shown in Fig.24. This Page allows you to connect the application to the remote host running the machine learning algorithm to perform the phrase detection part.

At this point the ROS-master node should be running on the remote PC (instructions for installing the required ROS package on your system are given in the Appendix).

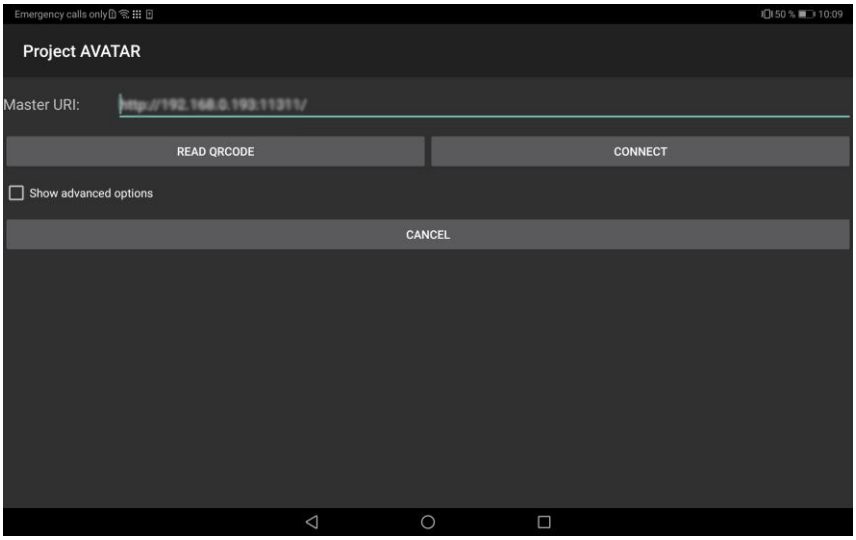

Figure 25 Interface for establishing connection to ROS-master

Open a terminal window and type the following command first:

~$ hostname -I

This will print the ip-address of the computer. Then enter the following command to set the ip address of the ROS-master node to correspond to the remote PC's ip address:

~$ export ROS_IP=<*ip-returned-from command-1*>

The next command starts up the ROS system:

~$ roscore

Fig.25 shows how the running ROS core should look like. The set IP address can be verified from the label *ROS_MASTER_URI*.

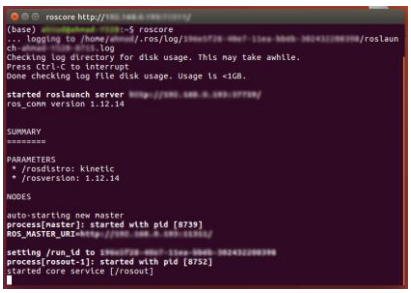

Figure 26 Terminal window running ROS-core

The URI from the above step should be entered into the ROS connection activity on the app. If the connection is successful you will see a toast appear telling that the connection has been successful, and the Camera activity will start.

At this point in order to make deductions regarding lip reading, the remote components should also be running on the PC. The Path on the remote PC for these components is */home/<User>/catkinJava_ws/Remote_Module*.

- Open this directory in a new terminal window.
- Type the following command to start the ROS interface to the application:
  - ~/catkinJava_ws/Remote_Module$ python Ros_Interface.py
  - If the script is started up successfully you should see Ok printed on the terminal window.
- Open a new terminal window in the same directory
- Type the following command to start up the ML component.
  - ~/catkinJava_ws/Remote_Module$ python Ros_Prediction_Component.py
  - Running this command will start up the tensor flow library. Once the program has been initialized completely “Ok” will be printed on the terminal window. The three terminal windows can be seen in fig.26 below.

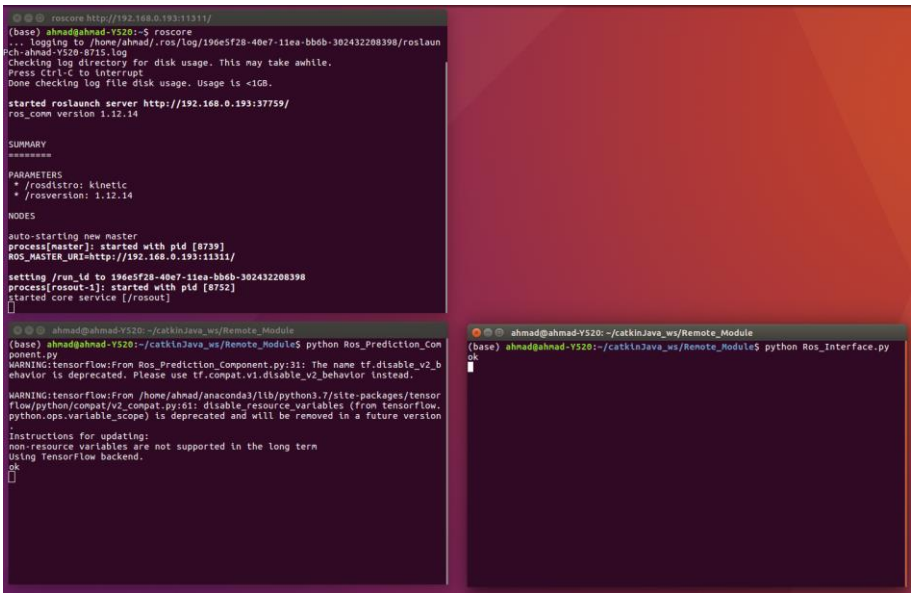

Figure 27 Terminal Windows showing the status of Remote module as Ready

  - A dependency is the directory */home/<User>/Avatar/Prediction_Path* on the remote PC. The frames sent from the Android device are saved in this path and then after the prediction has been performed the images are deleted.

At this point all the required modules are up and running. The start button can be pressed and the phrase can be said, once the required phrase has been said press the same button again to indicate the end of sentence. Now the detection process

will begin on the transferred images. And the result will be shown on the terminal window as shown in fig.27

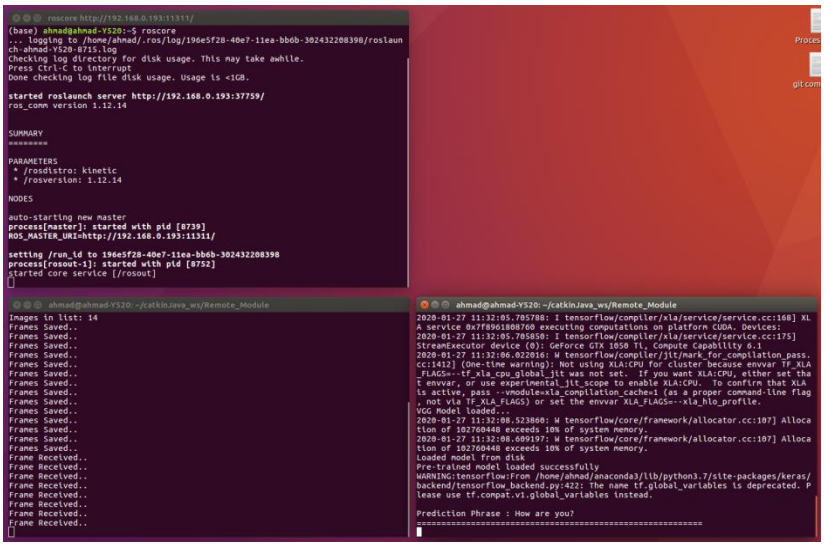

Figure 28 Remote Module showing Prediction Results

10 Java Doc

API documentation generated from java source code. HTML version is available on github.

10.1 MainActivity

- *Constructor Detail*

MainActivity

public MainActivity()

- *Method Detail*

avatarStart

public void avatarStart(android.view.View view)

This function is Triggered on the button press from the user. Creates an Intent object to start AvatarCameraActivity

Parameters:

view - object - basic building blocks of User Interface(UI) elements in Android. They are simple rectangle box which responds to the user's actions, e.g. EditText, Button, CheckBox etc. View refers to the android.view.View class, which is the base class of all UI classes.

onCreate

public void onCreate(android.os.Bundle savedInstanceState)

Overrides:

onCreate in class android.support.v7.app.AppCompatActivity

10.2 AvatarCameraActivity

- *Constructor Detail*

AvatarCameraActivity

public AvatarCameraActivity()

- *Method Detail*

detectLandmarks2

public void detectLandmarks2(org.opencv.core.Mat inputFrame)

This function takes in a Mat object and performs landmarks detection on the input image matrix once the detection has been performed it also publishes the data over the ROS network, and draws the resulting image on the canvas

Parameters:

inputFrame - -Mat - CvCameraViewFrame matrix

faceDetection

public org.opencv.core.Mat faceDetection(org.opencv.core.Mat inputFrame)

Uses the FaceDetection Class to perform face detection This method also sets the global variable FaceRoi after performing the face detection steps for further processing e.g. cropping the full image to the face ROI region

Parameters:

inputFrame - input frame: CvCameraViewFrame matrix

Returns:

inputFrame :MAT - Processed inputFrame, cropped to extracted faceROI

getFaceRoi

public org.opencv.core.Rect getFaceRoi()

Returns:

faceROI rect object

init

protected void init(org.ros.node.NodeMainExecutor nodeMainExecutor)

Function to initialize Ros Nodes and corresponding classes

Specified by:

init in class org.ros.android.RosActivity

Parameters:

nodeMainExecutor -

load_cascade

public void load_cascade()

This function loads the cascade classifier for face detection to be used by OpenCV facedetection module. the cascade classifier is placed in /res/raw/lbpcascade_frontalface_improved.xml The classifier is loaded in the global variable of type CascadeClassifier classifier

onCameraFrame

public org.opencv.core.Mat onCameraFrame(org.opencv.android.CameraBridgeViewBase.CvCameraViewFrame inputFrame)

Description copied from interface: org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2

This method is invoked when delivery of the frame needs to be done. The returned values - is a modified frame which needs to be displayed on the screen. TODO: pass the parameters specifying the format of the frame (BPP, YUV or RGB and etc)

Specified by:

onCameraFrame in

interface org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2

onCameraViewStarted

public void onCameraViewStarted(int width,
int height)

Description copied from interface: org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2

This method is invoked when camera preview has started. After this method is invoked the frames will start to be delivered to client via the onCameraFrame() callback.

Specified by:

onCameraViewStarted in

interface org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2

Parameters:

width - - the width of the frames that will be delivered

height - - the height of the frames that will be delivered

onCameraViewStopped

public void onCameraViewStopped()

Description copied from interface: org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2

This method is invoked when camera preview has been stopped for some reason. No frames will be delivered via onCameraFrame() callback after this method is called.

Specified by:

onCameraViewStopped in interface org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2

onCreate

public void onCreate(android.os.Bundle savedInstanceState)

Called when the activity is first created. Several imporant variables are initialized here

Overrides:

onCreate in class android.app.Activity

onDestroy

public void onDestroy()

Overrides:

onDestroy in class org.ros.android.RosActivity

onPause

public void onPause()

Overrides:

onPause in class android.app.Activity

onResume

public void onResume()

Overrides:

onResume in class android.app.Activity

publishToROS

public void publishToROS(android.view.View v)

Placeholder function for a button functionality for publishing data

setFaceRoi

public void setFaceRoi(org.opencv.core.Rect faceRoi)

Sets global variable faceROI

Parameters:

faceRoi - - Rect object

startProcess

public void startProcess(android.view.View v)

This function sets the global variable process and is called from button press action from the user, This variable is used to device whether to perform image processing tasks on the inputFrame.

10.3 Sender

- *Constructor Detail*

Sender

public Sender()

Constructor for the class, initializes the topic names on which data will be published

Sender

public Sender(java.lang.String topic)

Function to set a custom topic name for the publishers

Parameters:

topic - - String type

- *Method Detail*

getDefaultNodeName

public org.ros.namespace.GraphName getDefaultNodeName()

onStart

public void onStart(org.ros.node.ConnectedNode connectedNode)

Initializes the publisher objects with Topic names, and message types

Specified by:

onStart in interface org.ros.node.NodeListener

Overrides:

onStart in class org.ros.node.AbstractNodeMain

Parameters:

connectedNode - Name of the connected node, passed internally

publishFlag

public void publishFlag(int flag)

publishImage

public void publishImage(android.graphics.Bitmap b)

Converts the incomming bitmap image to a compressed image format - jpeg populates the image CompressedImage object with the received data publishes image on the specified topic

Parameters:

b - Bitmap image to be published

publishImageAndLandmarks

public void publishImageAndLandmarks(android.graphics.Bitmap b)

Publishes the landmarks data on the defined ROS topic

Parameters:

b - object of type Bitmap, to be published

10.4 Receiver

- *Constructor Detail*

Reciever

public Reciever()

- *Method Detail*

getDataReceived

public java.lang.String getDataReceived()

getDefaultNodeName

public org.ros.namespace.GraphName getDefaultNodeName()

onStart

public void onStart(org.ros.node.ConnectedNode connectedNode)

Specified by:

onStart in interface org.ros.node.NodeListener

Overrides:

onStart in class org.ros.node.AbstractNodeMain

10.5 DetectLandmarks

- *Constructor Detail*

DetectLandmarks

public DetectLandmarks(android.content.Context context)

DetectLandmarks

public DetectLandmarks(android.content.Context context,
android.graphics.Bitmap b)

- *Method Detail*

convertPixelsToDp

public static float convertPixelsToDp(float px,
android.content.Context context)

converts a float type pixel value to the corresponding value in dp also float type

Parameters:

px - Pixels value to be converted

context -

Returns:

corresponding dp value float data type

detFaces

public android.graphics.Bitmap detFaces()

Face Detection and Facial Landmarks detection, along with performance measurements are done in this method

Returns:

bm - Bitmap image - Processed results

detFacesFromBitmap

public android.graphics.Bitmap detFacesFromBitmap(android.graphics.Bitmap b)

Face Detection and Facial Landmarks detection. This method extracts the landmark points related to the lips ROI. Provision to draw the detected landmark points is available but disabled for performance improvement. If this feature is needed the line

needs to be uncommented. Populates landmarkPoints Arraylist with points for LipsROI

Returns:

bm - Bitmap image - Processed results

getFaceDetections

public void getFaceDetections()

Function to get bounds of ROI containing the face using OpenCV Not being used, only written for checking performance using this path of implementation

getLandmarkPoints

public java.util.ArrayList getLandmarkPoints()

Returns the arraylist containing the point objects for Landmarks

Returns:

ArrayList landmarkPoints

10.6 FaceDetection

- *Constructor Detail*

FaceDetection

public FaceDetection(android.content.Context current)

FaceDetection

public FaceDetection(android.content.Context current,
org.opencv.core.Mat img)

- *Method Detail*

detectFacefromMatrix

public org.opencv.core.Mat detectFacefromMatrix()

detectFaces

public android.graphics.Bitmap detectFaces(android.graphics.Bitmap b)

Parameters:

b - of type Bitmap, Image to be processed

Returns:

Bitmap image with overlay of a bounding rectangle on the detected faces.

getFaceDetections

public org.opencv.core.MatOfRect getFaceDetections()

getFaceRoi

public org.opencv.core.Rect getFaceRoi()

load_cascade

public void load_cascade()

Function to load the cascade classifier with the xml file containing the data from training the algorithm to detect faces. The XML is named lbpcascade_frontalface.xml and is included in the OpenCV sdk, for the application it is placed in res/raw directory. This function also has a handle to check if cascade classifier is empty making the one in the detectFaces() redundant.

setFaceRoi

public void setFaceRoi(org.opencv.core.Rect faceRoi)

11 Appendix

Source code of the project is available on the following git hub repo:

https://github.com/ahmadhmirza/Avatar-ROS

11.1 Android Studio:

For installation on 64 bit Ubuntu some external libraries are required to be installed first:

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386
```

The latest version of Android Studio can be downloaded via this link.

The download archive of android studio can be installed using the following command to a suitable location:

```
sudo unzip android-studio-ide-141.2178183-linux.zip -d /opt
```

To run Android studio navigate to this */opt/android-studio/bin* directory in a terminal and execute *./studio.sh* to start Android Studio.

11.2 ROS Kinetic:

ROS kinetic is the 10th ROS distribution release and targets Ubuntu 16.0.4(Xenial). Information about ROS packages related to this release can be found in this link:

For installation of the required ROS version following steps need to be executed in a Linux terminal.

- Set computer to accept packages from ROS:

  ```
  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
  ```

- Set up the api keys:

  ```
  sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
  ```

- Update the debian package index for the computer

  ```
  sudo apt-get update
  ```

- Install the full version of ROS-Kinetic

  ```
  sudo apt-get install ros-kinetic-desktop-full
  ```

- Initialize rosdep package which is necessary for executing roscore

  ```
  sudo rosdep init
  rosdep update
  ```

- edit .bashrc so that ros environment variables are automatically added to a bash session:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

11.3 Python packages and Tensorflow:

Python and Tensor flow are needed for running the machine learning component, following steps should be followed to install all the required packages.

```
sudo apt update
sudo apt install python3-dev python3-pip
sudo pip3 install -U virtualenv
pip install --upgrade pip
pip install --upgrade tensorflow
```

The scripts also depend on the following python packages that will need to be installed via pip:

- numpy
- OpenCV – cv2
- roslib, rospy
- scipy
- matplotlib
- pandas
- keras
- skimage
- dlib
- pypac

12 References

[1] https://www.nidcd.nih.gov/health/statistics/quick-statistics-voice-speech-language

[2] https://www.cdc.gov/nchs/products/databriefs/db205.htm

[3] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[4] https://towardsdatascience.com/introduction-to-hidden-markov-models-cd2c93e6b781

[5] https://www.nist.gov/itl/products-and-services/color-feret-database

[6] https://github.com/tensorflow/tensorflow

[7] Kaiqi Cen “Study of Viola-Jones Real Time Face Detector”; https://web.stanford.edu/class/cs231a/prev_projects_2016/cs231a_final_report.pdf

[8] P.Viola, M.Jones; "Rapid Object Detection using a Boosted Cascade of Simple Features" ; Conference on Computer Vision and Pattern Recognition, 2001

[9] AH.Mirza; "AVATAR: Implementation of facial landmark detection on an android device"

[10] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade; "Neural Network-Based Face Detection"

[11] Moshu Wu, Guangda Su, Jun Zhou ; "A robust system of face detection and precise face organ location"

[12] Anamika Singh, Manmindar Singh, Birmohan Singh; "Face detection and Eyes extraction using Sobel Edge Detection and Morphological Operations"

[13] Prathap Nair, Andrea Cavallaro; "3-D Face Detection, Landmark Localization, and Registration Using a Point Distribution Model."

[14] ROS Documentation; http://wiki.ros.org/ROS/Concepts

[15] Android Documentation; https://developer.android.com/reference/android

[16] ROS message documentation; http://docs.ros.org/melodic/api/sensor_msgs/html/msg/CompressedImage.html

[17] Very Deep Convolutional Networks for Large-Scale Image Recognition K. Simonyan, A. Zisserman arXiv:1409.1556

[18] dLib shared Libraries and JNI @ https://github.com/tzutalin/dlib-android

[19] Tutorials and Documentation for ROS-JAVA @ http://wiki.ros.org/rosjava

[20] One Millisecond Face Alignment with an Ensemble of Regression Trees by Vahid Kazemi and Josephine Sullivan, CVPR 2014

[21] Yue Wu, Tal Hassner et.al; "Facial Landmark Detection with Tweaked Convolutional Neural Networks, IEEE Transactions on pattern analysis and machine intelligence vol. 40 No. 12,Dec 2018

[22] Amir Zadeh et.al. "Convolutional Experts Constrained Local Model for 3D Facial Landmark Detection", IEEE International Conference on Computer Vision Workshops, 2017