



Birla Institute of Technology
Department of Computer Science and Engineering

Project Write-up – Intensive Summer Research Experience (ISRE)
(Summer Research Internship)

Advisor: Dr. Abhinav Bhushan

Submitted By:

Name: Saad Aziz Zaidi	
Roll No: BTECH/10012/18	Semester: NA
Branch: CSE	Section: B
Submission Date: 30 th Oct 2021	Session: Summer of 2021

Project Title

Title: Point of Care Tests for diagnosing periodontitis

Program Theme: Medical Devices and Sensors

Professor name: Dr. Abhinav Bhushan – [Department of Biomedical engineering, Armour college of Engineering, Illinois Institute of Technology, Chicago]

Overview

I, under my advisor from Illinois Institute of Technology, came up with concepts and possible solutions to the problem of point of care testing for diagnosing periodontitis, I then presented the proof of concepts every week in the sessions with Dr. Abhinav Bhushan. We then proceeded to draft an application for our method to get approved by the FDA.

My Advisor

Dr. Abhinav Bhushan was my advisor during my time at Illinois Institute of Technology. To say he was supportive and encouraging of my efforts would be an understatement, the kind of environment he created during his sessions was fabulous and I shall remember it all my life. His constant encouragement and positive attitude were extremely helpful in pushing me past my limits and go beyond.

I was heartbroken, when it dawned on me that the Intensive Summer Research Experience this year wouldn't be offline, but the fact that I do not feel I could have extracted more from the Intensive Summer Research Experience tells you how successful it was, despite being online.

Truly and sincerely, ISRE has changed me for the better and I feel selfish knowing many of my peers did not get to experience this.

Goals of the Summer Research

- Research about periodontitis and current testing for diagnosing it
- Suggest point of care tests for periodontitis
- Show proof of concepts for it
- Draft FDA applications to get them FDA approved
- Learn about BIOMEDICAL MICROELECTROMECHANICAL SYSTEMS
- Learn about the different manufacturing and working of medical devices and sensors

ENGR 498-07 Research in Artificial Intelligence and Deep Learning

FINAL PROJECT REPORT

Machine Vision Covid-19 Norms Surveillance System based on
Artificial Intelligence
(**MaViSS AI**)

By: Ritika Nigam (A20498268)

Advisor: Dr. Jafar Saniie

Summer 2021

Abstract

In March 2020, **World Health Organization (WHO)** had declared pandemic due to **COVID-19** since then the coronavirus outbreak has caused a global disaster with its deadly spreading. The economic and social disruption caused by the pandemic is devastating. Though vaccines have been developed by various nations, but as stated by the World Health Organization (WHO), vaccines rarely protect 100% of the recipients and vaccinated individuals still run the risk of contracting the disease. And also with the increase in mutation of the virus, the new variants of coronavirus are being emerged which is eventually decreasing the effectiveness of vaccines against the coronavirus. In order to curb this pandemic it is important to monitor whether people are abiding by all the necessary precautions i.e. maintaining **social distancing norms, wearing face masks and crowd management**.

Manual monitoring of these norms is difficult and tends to be quit inefficient and inaccurate. This necessitates the urge of an automated machine vision system for monitoring the covid norms in real time. This encouraged us to to design an Artificial Intelligence based machine vision surveillance system (MaViSS AI) for **real-time monitoring of COVID-19 norms** which would be **cost effective, accurate, feasible and secure** and would overcome the real time challenges faced during manual monitoring of norms.

Introduction

MaViSS AI is a Machine Vision Surveillance System based on Artificial Intelligence which would be used for real-time monitoring of COVID-19 norms and thus would help in alleviating the COVID-19 surge. This system would replace many physical eyes with computer visions and thereby providing an accurate and efficient monitoring system. The system will be used for monitoring three different tasks.

- **Detecting and tracking** humans for monitoring **social distancing** norms and **counting** the total **humans for crowd management**.
- **Detecting face mask** and keeping the track of face mask usage by the detected people.
- **Raising real-time alerts** using a **telegram bot** whenever any of the following norms are breached.

Thus, the aim of our project is to develop a framework that tracks and counts humans for monitoring social distancing and detects face mask. To accomplish this objective, we developed an algorithm using object detection method. For object detection method, we used **YOLO(You Only Look Once)** neural network to detect person and count them. And for social distancing monitoring we used the concept of centroid i.e. calculating the distance between pairs of centroids, and thus checking whether there is any violations of threshold or not. This approach of social distancing algorithm will red mark the persons who are getting closer than a permissible limit. In order to detect the face mask, a **YOLO V4** deep learning used as the mask detection algorithm. The system also raises alerts when any suspicious event occurs. In view of this alert, security personnel can take relevant actions. Therefore, the automated surveillance system will surpass several limitations of the manual monitoring systems.

This research aims to limit the impact of the coronavirus epidemic with minimal harm to economical artifacts. Monitoring social distancing in real-time scenarios is a challenging task. It can be possible in two ways: manually and automatically. The manual method requires many physical eyes to watch whether every individual is following social distancing norms strictly. This is an arduous process as one can't keep their eyes for monitoring continuously. However, automated machine vision surveillance system replaces many physical eyes with computer vision.

The primary **application** of our system MaViSS AI is that it could be used as **covid-19 norms machine surveillance system** for monitoring both indoor and outdoor surveillance scenarios. It can be used significantly in various busy places like railway stations, airports, megastores, malls, streets, etc. where manual monitoring is very difficult. Apart from COVID-19 norms monitoring, MaViSS AI can be used for **broader applications** as **generic human detection and tracking system** in various real-world applications. It can be significantly used as human action and anomaly detection in security systems such as in banks, ATM and also in residential areas, pedestrians detection and tracking in autonomous vehicles, crowd management in shops, lifts, public transports, etc.

Description

System Modules

The system MaViSS AI is the integration of three different modules. Each module perform a different monitoring task with the help of real-time object detection method (YOLO) and OpenCV library of python.

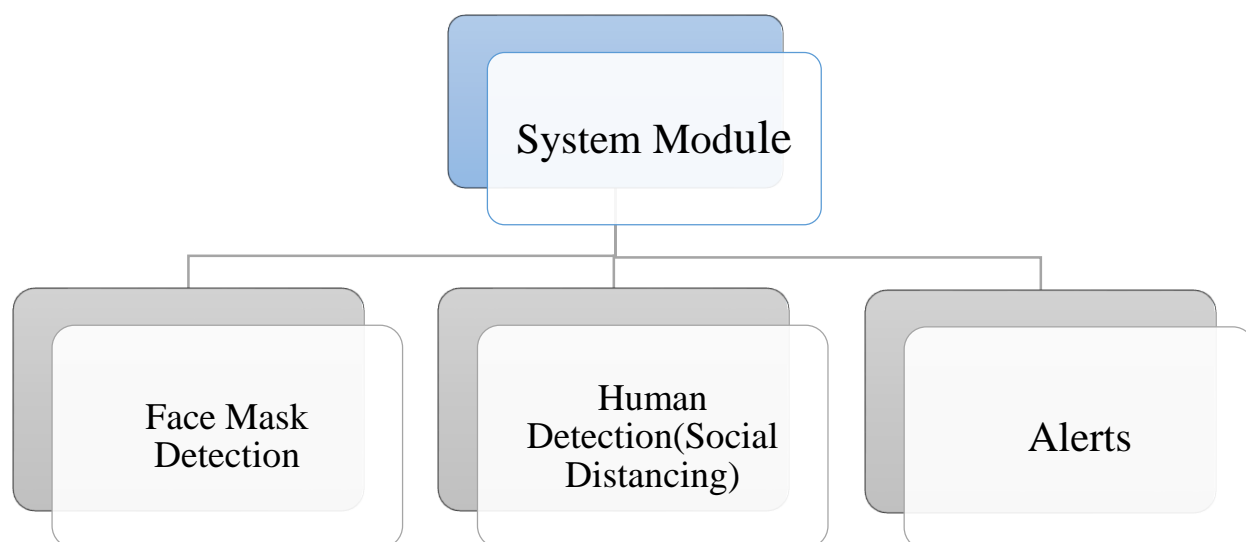


Fig 1: Flowchart representing different system modules

The three different modules are used for monitoring three different norms:

- I. **Face Mask Detection** - Face mask detection module uses **mask-YOLOv4-tiny** model to detect the face mask usage and classifies it into three classes using different shades of bounding box:
 - **Good** – The green bounding box is annotated with good remark which represents that the person is well masked with nose and mouth fully covered.
 - **Bad** – The orange bounding box is annotated with bad remark which represents that the person is not well masked i.e. his/her nose or mouth is not fully covered.
 - **None** – The red bounding box is annotated with none remark which represents that the person is not wearing a mask and is violating the norms.

II. **Human Detection** (For Social Distancing) - Human detection uses **YOLOv3-608** model to detect and track humans in the scene and calculate the distance between each pair of humans. This information is then calibrated with the safe distance set by the user and each person is classified into three classes using different colors of bounding box:

- **Green**- The person enclosed within green bounding box denotes that person is at safe distance (i.e. 2m) from others (**No Violation**).
- **Yellow**- The person enclosed within yellow bounding box denotes that person is at minimum safe distance (i.e. 1m) but not at safe distance (i.e. 2m) from others (**Abnormal Violation**).
- **Red** - The person enclosed within red bounding box denotes that person is not at minimum safe distance (i.e. 1m) from others (**Serious Violation**).

III. **Alerts** - The **alerts** module of our system uses urllib and requests packages and is connected to a **Telegram bot** using its chat id and authenticated with the token id. Alert messages can be delivered through this bot both to individual users and groups. Whenever there are any serious violations in COVID19 norms, either social distancing or face mask usage, the same is communicated to the user's smartphone in realtime through this bot.

Workflow

The basic workflow of our system comprises of six different phases:

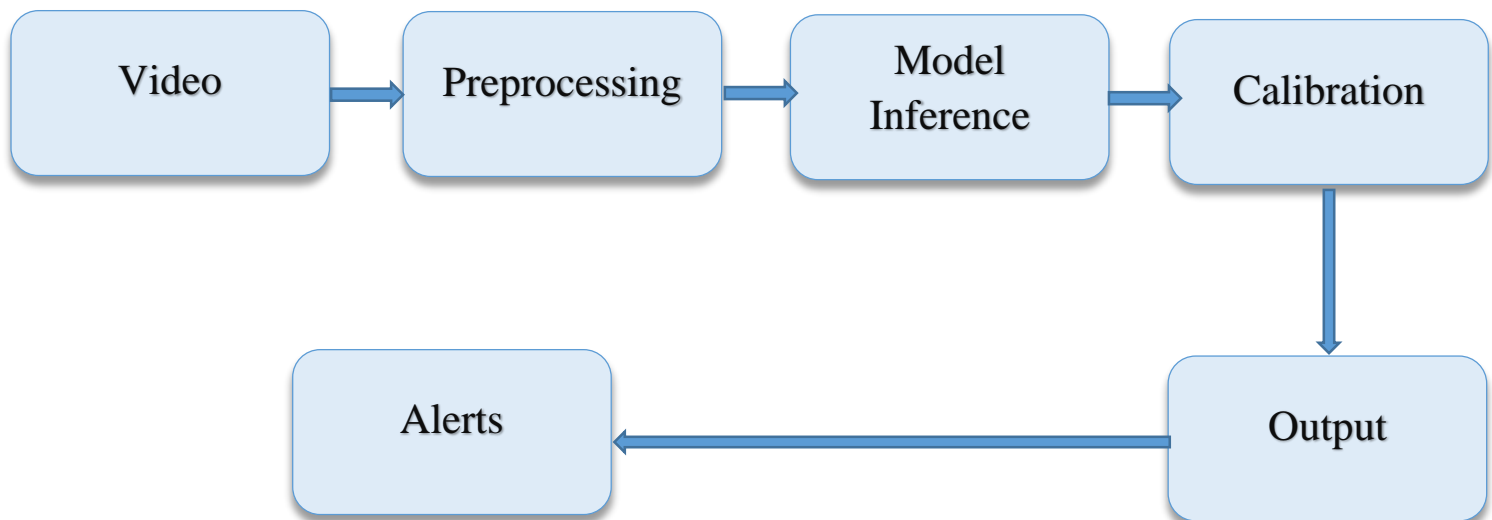


Fig 2: Flowchart representing the workflow of the system

In the diagram above the basic overflow of the system is shown. According to the flowchart six phases are involved in the process:

- **Video** – The first phase is the video phase in which the frames are extracted from the video sources obtained from CCTV or IP cameras.
- **Preprocessing**- After extracting the frames, these frames are sent to the second phase which is the preprocessing phase. In this phase resizing of the frames are done for the model inference.

- **Model Inference-** Resized frames are then sent to the third phase which is the model inference phase. The model inference is done by using the YOLO architecture (trained on COCO dataset) for state-of-the-art, real-time humans and face mask detection.
- **Calibration-** This is the fourth phase which involves computing parameters like social distancing & face mask metrics, validating it with the norms and identifying violations.
- **Output-** After calibrating the frames, these frames are sent to the fifth phase which is the output phase. In the output phase, output is generated in real-time to the monitoring user, displaying the social distancing metrics, color coded bounding boxes for persons detection & tracking, and information regarding any violations.
- **Alerts-** This the last phase of the workflow. This phase generates real-time alerts messages using a telegram bot which is directly sent to the user's smartphone whenever any of the norms are breached.

Hardware and Software Tools

Hardware Components

- I) **NVIDIA Jetson Nano** – The major hardware component utilized in our project is NVIDIA Jetson Nano for computation of our system. NVIDIA Jetson Nano Developer Kit is a small, powerful computer that runs multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing. It is an easy-to-use platform that runs in as little as 5 watts. It is booted by inserting a microSD card with system image. It is used for building practical AI applications.

The Jetson Nano is specifically powered by a powerful **NVIDIA Maxwell GPU** comprised of **128 CUDA cores** along with **quad-core Arm Cortex-A57 CPU** MPCore processor. It has lots of IO options including one USB 3.0 Type-A, 2 USB 2.0 Type-A, 1 USB 2.0 Micro-B, a 40 GPIO header, 12-pin power/UART header, 4-pin fan header, a Gigabit Ethernet RJ45 jack, full-size HDMI port, and an included 802.11ac wireless USB dongle. Also on board is a microSD card slot for storage (card not included) and a MIPI CSI-2 connector to attach a camera, to give the Nano a set of eyes. The kit is setup in a mezzanine-style IO board and SODIMM slot configuration, where the processing engine clips into the IO board with an integrated heat sink. The operating system used in Jetson Nano is Linux Ubuntu 18.04.



Fig 3: Image of Nvidia Jetson Nano

Technical Specification of Jetson Nano:

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 4x 1080p @ 30 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 2x 4K @ 30 8x 1080p @ 30 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I ² C, I ² S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

- II) Camera-** The second hardware component used in our project is an IMX 219-77 camera or a webcam for capturing videos. An IMX 219-77 camera is a high-quality camera with an 8 megapixel Sony IMX219 image sensor. It is capable of viewing images at a high resolution of 3280x2464. It has a high FOV (field of view) to capture more area. It is suitable to use with the NVIDIA Jetson Nano and NVIDIA Jetson Xavier NX Development Kits.

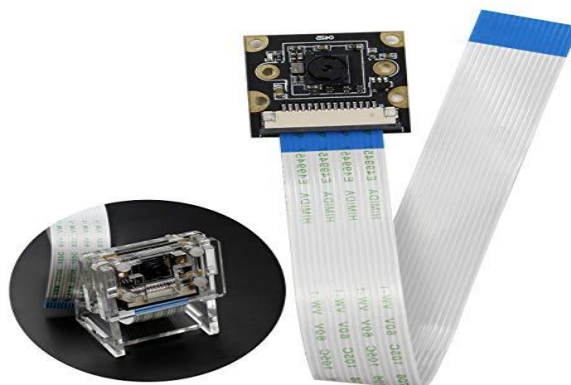


Fig 4: IMX 219-77 Camera

Technical Specification of IMX 219-77 camera

Specification	Description
Megapixels	8 Megapixels
Photosensitive chip	Sony IMX219
Assembly Technique	SMT (ROSH)
Resolution	3280 × 2464
Pixel Size	1.12µm x1.12µm
CMOS size	1/4 inch
Aperture (F)	2.0
Focus	fixed
Focal Length	2.96mm
Lens Construction	4P
Diagonal field of view (FOV)	77 degrees

III) External monitor- The third hardware component used in our project is an external monitor used for visualizing the output of our system and also for monitoring the norms. This external monitor is connected to Jetson Nano using the HDMI cable. Along with the external monitor, a USB mouse and a keyboard is also connected to the Jetson nano through USB cables.



Fig 5: An External Monitor

Thus all of the above components integrate together to form the hardware of our system MaViSS AI which provides a **computational unit**, **monitoring unit** and the **visualization unit**. The figure below represents all the different components of hardware connected to the Jetson Nano.



Fig 6: Hardware Components of the system MaViSS AI

Circuit Design

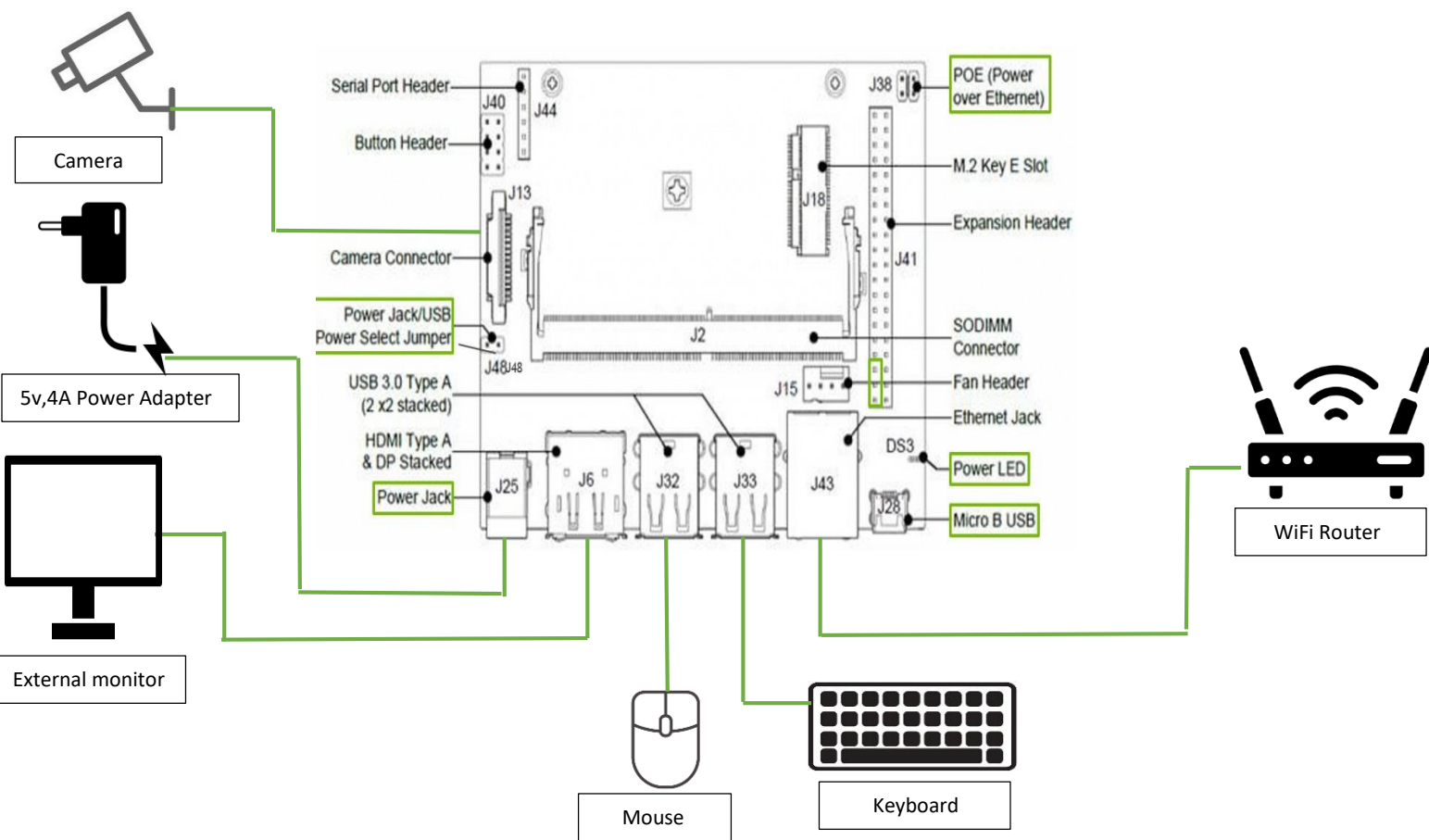


Fig 7. Schematic of Circuit design

Software Components

- I.) Python-** The programming language used in our project is Python. **Python** is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.^[31]

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

In our project we have utilised various python libraries available for building different parts of the system. For video and frames capturing and processing, we have used OpenCV and imutils packages. Various calculations and calibrations are facilitated by packages like scipy and numpy.

- II.) OpenCV-** The major python library used in our project is OpenCV for image processing. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

- III.) YOLO-** The main object detection algorithm used in our project is YOLO (You Only Look Once). YOLO is an abbreviation for the term 'You Only Look Once'. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images. YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects. This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

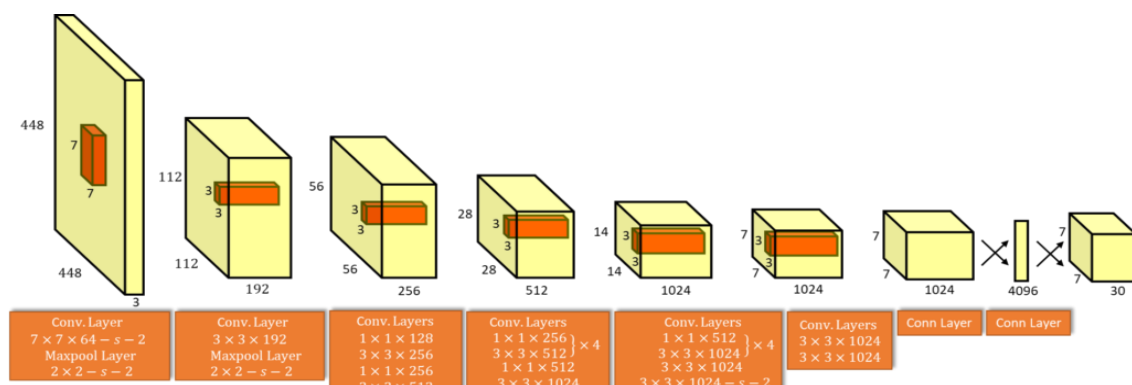




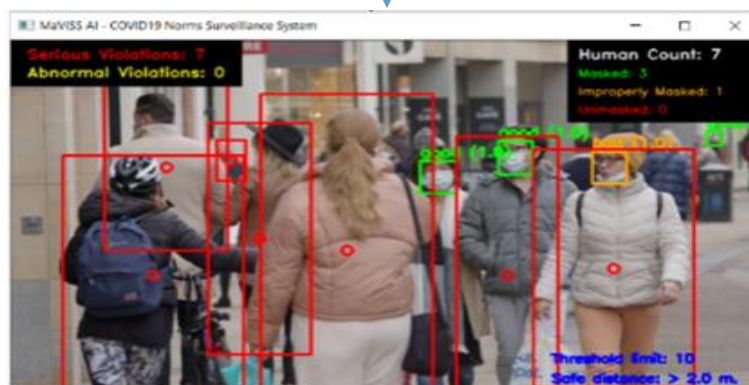
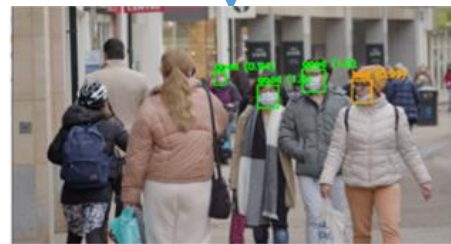
Fig 8. YOLO Architecture

Software Implementation


Coded the system module using python programming language


OpenCV
Processed the video using the OpenCV library


YOLO- Real-time object detection method used for model inference



Calibrating and Integrating together the two module into the final output

Challenges Overcome

Real-Time Constraints

While building up the project we faced real-time constraints based on the performance of our system. When we tested our system utilizing the CPU of Jetson Nano, we got a very low rate of frames per second because of which the performance of our system was tremendously dropped.

However this constraint was overcome by installing OpenCV with CUDA support as this enabled us to use the GPU of Jetson Nano which increased our performance nearly 4 times. Modern GPU accelerators has become powerful and featured enough to be capable to perform general purpose computations (GPGPU). OpenCV includes GPU module that contains all GPU accelerated stuff. Supported by NVIDIA the work on the module, started in 2010 prior to the first release in Spring of 2011. It includes accelerated code for significant part of the library, still keeps growing and is being adapted for the new computing technologies and GPU architectures.



Fig 9: Jtop indicating OpenCV compiled with CUDA on Jetson Nano

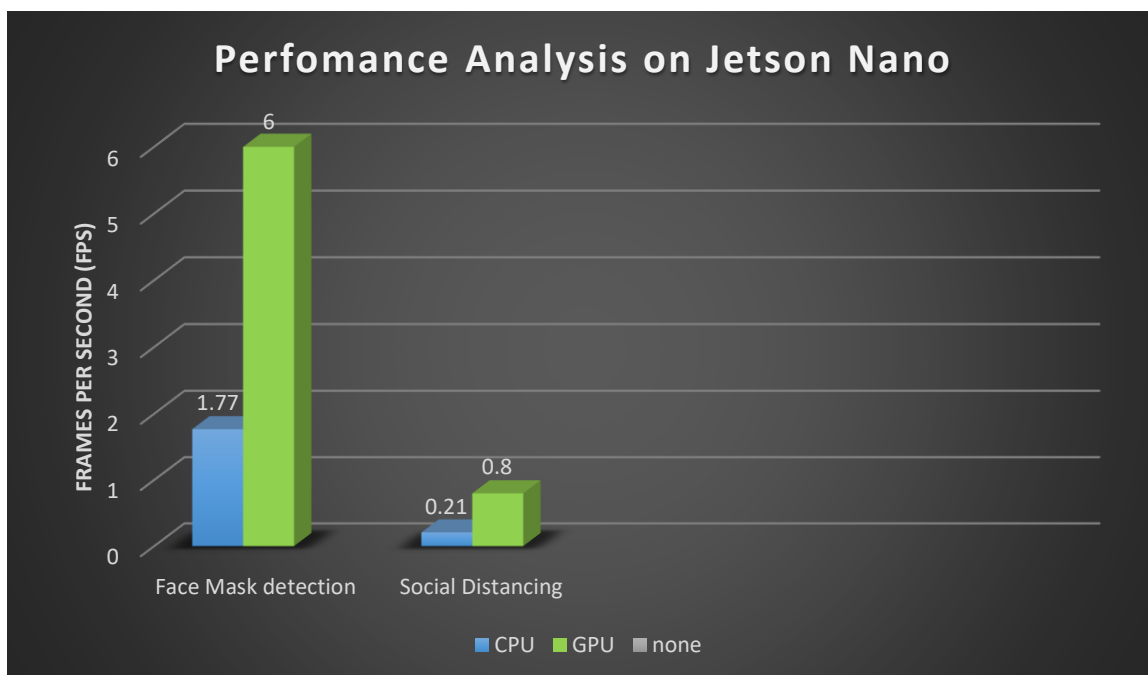
After enabling CUDA for OpenCV we did **performance and accuracy analysis** of our system for the two given modules:

Module 1: Face Mask Detection Module:

Model	Frames per second (FPS)	
	CPU	GPU
mask-YOLOv4-tiny	1.37-1.77	3-6

Module 2: Human Detection and Tracking (Social Distancing)

Model	Frames per second (FPS)	
	CPU	GPU
YOLOv3-608	0.21	0.71-0.79



Thus the chart above shows the analysis of system modules on CPU and GPU respectively. These models of YOLO are selected on the parameter of performance and accuracy.

Security

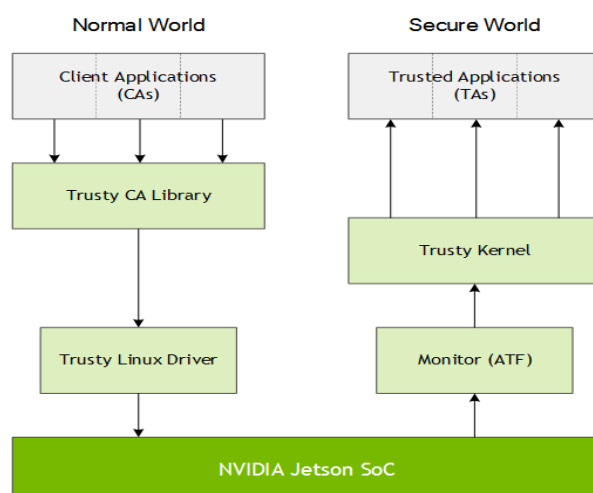
Security and privacy issue is one of the major issues faced but with the help of NVIDIA's Trusty Execution Environment(TEE) available for Jetson products we can overcome this challenge.

Trusty resides in a separate storage partition and boots as part of a chain of trust or a secure boot sequence. It creates two environments in a device with different security modes:

- Non-Secure Environment (NSE):** An environment for running software components in non-secure mode. This mode is known as the "normal world." A rich OS, such as Linux, typically runs in this environment.
- Trusted Execution Environment (TEE):** A separate environment, that provides trusted operations and runs in secure mode enforced by hardware. This mode is known as the "secure world." Trusty runs in this environment.

The normal world OS and Trusty software operate in a client-server relationship, with Trusty as the server.

The bootloader allocates a dedicated carveout, TZ-DRAM, to run a secure OS. All secure operations are initiated by a client application running in the non-secure environment. A trusted application, in the secure world, never initiates contact with the non-secure environment.

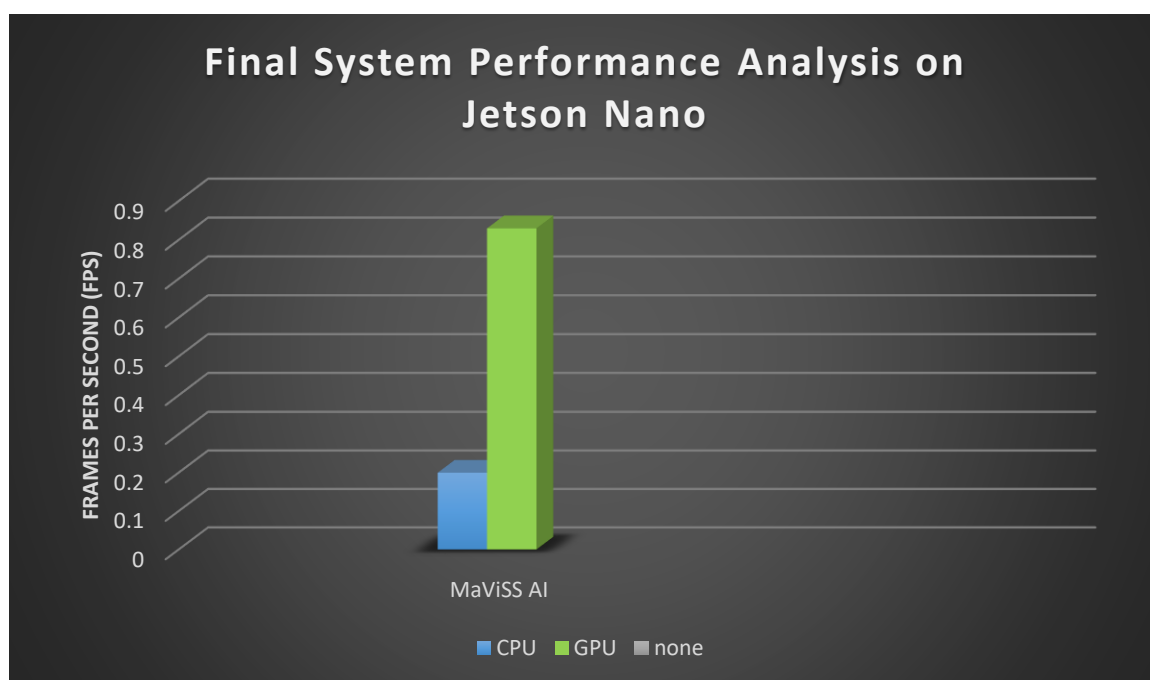


Results and Discussions

On combining all the modules together the final performance of the system on **Jetson Nano** are as follows

System	Frames per second (FPS)	
	CPU	GPU
MaViSS AI (YOLOv3-608 + mask-YOLOv4-tiny)	0.15-0.2	0.65-0.83

Thus the final inference is that **MaViSS AI** (mask-YOLOv4-tiny + YOLOv3-608) utilizes the **powerful GPU of Jetson Nano** with **CUDA** backend to improve its performance by approximately **4 times better** than that achieved on CPU and runs at frame rate of **0.65-0.83 FPS**.



The above chart shows the comparative analysis of our system MaViSS AI vs its performance in CPU and GPU of the Jetson Nano. The bar graph clearly indicates that the the fps obtained on GPU is much higher than that achieved on the CPU. Thus, with the help of the NVIDIA Maxwell 128 cores GPU of Jetson Nano we improved the overall performance of our system.

Therefore, with the help of MaViSS AI the user is able to monitor social distancing norms and face mask usage in the scene captured by the surveillance camera and any norm breach is reported directly to the user as an alert message.

The results obtained can be found in the demonstration video,the link for which has been attached below

[Demonstration Video](#)

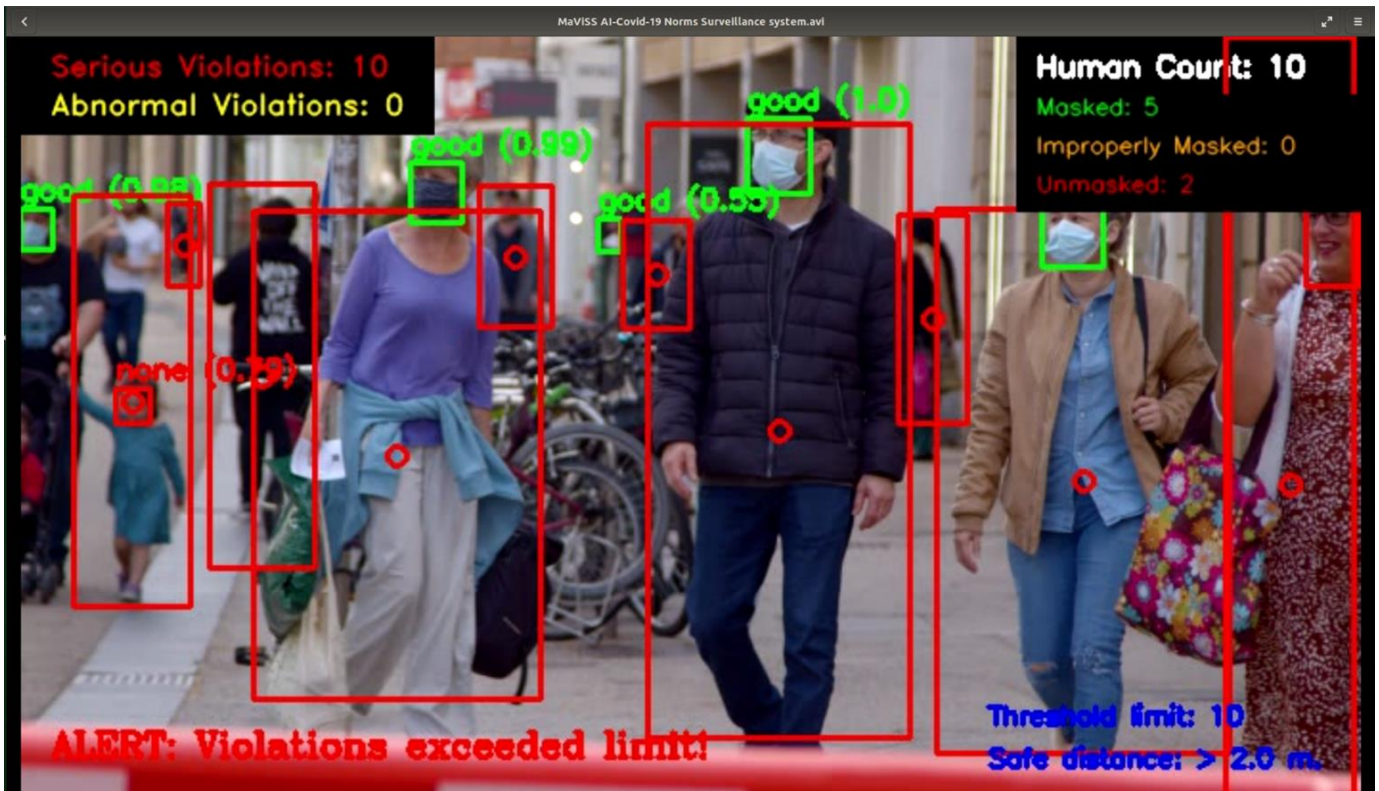


Fig 10: Monitoring window of our system MaViSS AI

The figure shown above is the monitoring window of our system MaViSS AI with all the metrics displayed on the screen. The metric at the right hand corner represents the human counter which counts the total number of humans present in the scene at any particular moment. It also represents the counter for masked, improperly masked and unmasked parameter which shows number of humans wearing mask, not wearing a mask or is improperly masked. On the other hand, at the left corner there is a metric for social distancing parameter. It represents the counter for serious and abnormal violations present in the scene. Whenever these counters exceed the threshold value an Alert message is displayed on the monitoring screen and as well as an alert message through the telegram bot is directly sent to the user on their smartphones.



Fig 11: Telegram bot sending alerts to the smartphone of the user

Conclusion

Taking into account the importance of social distancing and face mask usage in managing and reducing the probability of COVID-19 disease from continuously spreading which can cause the healthcare system to collapse due to high number of patient, **MaViSS AI** provides an integrated system for monitoring all the necessary norms that are needed to be followed. It monitors the face mask usage, social distancing parameters and also counts the total number of humans present in the scene. The system raises real-time alerts through telegram bot whenever any of the following norms are breached. Thus MaViSS AI surpasses several limitations of the manual monitoring systems and provides an **efficient** and **accurate** way of monitoring and reporting breaches in COVID19 norms.

Future Work

In future, additional backend process will be included that allow advanced statistical analysis to be done which can be used by the authority, facilities or building owner to monitor the level of compliance among the people or visitors. The data received from the monitoring system can be collected and based on this data a live dashboard can be built which provides visualizations of the norms that would be dynamic in nature and keeps on updating based on the data received. Moreover, system performance can be improved by using higher end hardware and more optimized detection algorithms. Also, distance calculation can be made more accurate by using depth and aspect information. In addition to this a more advanced camera with zooming and adjusting capabilities can be used to detect the distant faces for face mask usage.

References

- [Neuralet's smart social distancing](#)
- YOLO v3 (for human detection) - <https://pjreddie.com/darknet/yolo/>
- mask YOLO v4 tiny (for mask detection) - <https://github.com/AlexeyAB/darknet>
- Jetson Nano - <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit>
- M. M. Rahman, M. M. H. Manik, M. M. Islam, S. Mahmud and J. -H. Kim, "An Automated System to Limit COVID-19 Using Facial Mask Detection in Smart City Network," 2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), 2020, pp. 1-5, doi: 10.1109/IEMTRONICS51293.2020.9216386. (<https://ieeexplore.ieee.org/document/9216386>)
- [J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition \(CVPR\), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.](#)
- <https://abhatikar.medium.com/make-your-nvidia-jetson-nano-deep-learning-ready-a8f5fcd7b25c>
- Opencv with CUDA and dnn modules- <https://www.pyimagesearch.com/2020/02/03/how-to-use-opencvs-dnn-module-with-nvidia-gpus-cuda-and-cudnn/>
- Configuring Jetson Nano- <https://www.pyimagesearch.com/2020/03/25/how-to-configure-your-nvidia-jetson-nano-for-computer-vision-and-deep-learning/>

Appendix

Alerts Script:

```
#=====AlertsModule\=====
=====
# importing required libraries
import urllib, requests
from lib.config import chat_id, token

"""
This script initiates the telegram alert function.
"""
def trigger(arr, typ):
    message1 = 'Social distancing violations exceeded!\n\nSerious
Violations : {}'.format(arr[0])
    message2 = 'Face Mask violations exceeded!\n\nMasked : {} \nImproperly
Masked : {} \nUnmasked " {}'.format(arr[1], arr[2], arr[3])

    if typ == 1:
        url=
'https://api.telegram.org/bot%s/sendMessage?chat_id=%s&text=%s' % (token,
chat_id, urllib.parse.quote_plus(message1))
        _ = requests.get(url, timeout=10)
    if typ == 2:
        url=
'https://api.telegram.org/bot%s/sendMessage?chat_id=%s&text=%s' % (token,
chat_id, urllib.parse.quote_plus(message2))
        _ = requests.get(url, timeout=10)
#=====
=====
```

Configuration Script:

```
#=====ConfigurationScript/=====
===#
# base path to YOLO directory
YOLO_PATH = "yolo"

# minimum object detection probability
Min_Prob = 0.3

# minimum threshold for non-maxima suppression
NMS_Threshold = 0.3
```

```

# to count number of people in frame (True/False)
Human_Counter = True

# set the threshold value for violations
Violations_Threshold = 10

# set the ip camera url (e.g. url = 'http://192.168.43.39:4747/video')
# set url = 0 for webcam
url = 0

#-----|TELEGRAM ALERTS|-----
-----
# toggle telegram alert feature (True/False)
Alert = False

# telegram bot's chat ID and token
chat_id = ''
token = ''
#-----
-----

# toggle GPU usage for computations (True/False)
# CPU used by default
Use_GPU = False

# set minimum safe distance between 2 people (in cm.)
MAX_DISTANCE = 200 # (i.e. safe distance)
MIN_DISTANCE = 100 # (1.e. minimum safe distance)

# set average height of a person (in cm.)
avg_height = 170
#=====
=====

```

Detect Function Script:

```

#=====
#=====

# importing necessary libraries
from lib.config import NMS_Threshold, Min_Prob, Human_Counter
import numpy as np
import cv2

# defining the detect_humans function
def detect_humans(frame, net, layer_names, human_idx = 0):
    # extracting the dimensions of the frame and
    # initializing the results list
    (H, W) = frame.shape[:2]

```

```

results = []

# constructing a blob from the input frame and performing a forward
# pass of the YOLO object detector
# gives us the bounding boxes and associated probabilities
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
                             swapRB = True, crop = False)
net.setInput(blob)
layerOutputs = net.forward(layer_names)

# initializing the lists of detected bounding boxes,
# centroids and confidences
boxes = []
centroids = []
probabilities = []

# iterating through the layer outputs
for output in layerOutputs:
    # iterating through each of the detections
    for detection in output:
        # extracting the class ID and object detection probability
        scores = detection[5:]
        classID = np.argmax(scores)
        probability = scores[classID]

        # filtering detections by:-
        # (1) ensuring that a human was detected and
        # (2) that the minimum probability criteria was satisfied
        if classID == human_idx and probability > Min_Prob:
            # scaling the bounding box coordinates back relative to
            # the size of the image, as YOLO returns
            # the center (x, y) coordinates of the bounding box
            # followed by the width and height
            box = detection[0:4] * np.array([W, H, W, H])
            (centerX, centerY, width, height) = box.astype("int")

            # using the center (x, y) coordinates to find the
            # top-left corner coordinates
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))

            # updating the list of bounding box coordinates,
            # centroids and confidences
            boxes.append([x, y, int(width), int(height)])
            centroids.append((centerX, centerY))
            probabilities.append(float(probability))

# applying non-maxima suppression (NMS) to suppress weaker,
# overlapping bounding boxes

```

```

idxs = cv2.dnn.NMSBoxes(bboxes, probabilities, Min_Prob,
NMS_Threshold)

# calculating the total humans in frame
if Human_Counter:
    human_count = "Human Count: {}".format(len(idx))
    cv2.rectangle(frame, (520, 0), (700, 30), (0, 0, 0), -1)
    cv2.putText(frame, human_count, (530, 20),
cv2.FONT_HERSHEY_DUPLEX, 0.5, (255, 255, 255), 1, cv2.LINE_AA)

# ensuring at least one detection exists
if len(idx) > 0:
    # iterating through the indexes
    for i in idx.flatten():
        # extracting the bounding box coordinates
        (x, y) = (bboxes[i][0], bboxes[i][1])
        (w, h) = (bboxes[i][2], bboxes[i][3])

        # updating the results list to contain
        # detection probability, bounding box coordinates and
centroid
        res = (probabilities[i], (x, y, x + w, y + h), centroids[i])
        results.append(res)

# returning the list of results
return results

```

```

=====
=====

```

Detect Facemark Script:

```

===== /FaceMaskDetectionmodule \=====
=====

```

```

# importing necessary libraries
import time
import cv2
import numpy as np
from lib.config import Use_GPU

class DETECT_FACEMASK:

    def __init__(self, config, model, labels, size=416, confidence=0.5,
threshold=0.3):
        self.confidence = confidence
        self.threshold = threshold
        self.size = size

        self.labels = labels
        self.net = cv2.dnn.readNetFromDarknet(config, model)

```

```

# checking if there's GPU usage
if Use_GPU:
    # set CUDA as the preferable backend and target
    print("")
    print("[INFO] Looking for GPU")
    self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

def inference_from_file(self, file):
    mat = cv2.imread(file)
    return self.inference(mat)

def inference(self, image):
    ih, iw = image.shape[:2]

    ln = self.net.getLayerNames()
    ln = [ln[i[0] - 1] for i in self.net.getUnconnectedOutLayers()]

    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (self.size,
self.size), swapRB=True, crop=False)
    self.net.setInput(blob)
    start = time.time()
    layerOutputs = self.net.forward(ln)
    end = time.time()
    inference_time = end - start

    boxes = []
    confidences = []
    classIDs = []

    for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
            # extract the class ID and confidence (i.e., probability)
of
            # the current object detection
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]
            # filter out weak predictions by ensuring the detected
            # probability is greater than the minimum probability
            if confidence > self.confidence:
                # scale the bounding box coordinates back relative to
the
                # size of the image, keeping in mind that YOLO actually
                # returns the center (x, y)-coordinates of the
bounding
                # box followed by the boxes' width and height
                box = detection[0:4] * np.array([iw, ih, iw, ih])
                (centerX, centerY, width, height) = box.astype("int")

```

```

        # use the center (x, y)-coordinates to derive the top
and
        # and left corner of the bounding box
        x = int(centerX - (width / 2))
        y = int(centerY - (height / 2))
        # update our list of bounding box coordinates,
confidences,
        # and class IDs
        boxes.append([x, y, int(width), int(height)])
        confidences.append(float(confidence))
        classIDs.append(classID)

        idxs = cv2.dnn.NMSBoxes(boxes, confidences, self.confidence,
self.threshold)

        results = []
        if len(idxs) > 0:
            for i in idxs.flatten():
                # extract the bounding box coordinates
                x, y = (boxes[i][0], boxes[i][1])
                w, h = (boxes[i][2], boxes[i][3])
                id = classIDs[i]
                confidence = confidences[i]

                results.append((id, self.labels[id], confidence, x, y, w,
h))

        return iw, ih, inference_time, results

```

Main Sript:

```

#=====/MAIN EXECUTION
FILE\=====
# importing necessary libraries
from lib import config
from lib.detect_facemask import DETECT_FACEMASK
from lib.alerts import Alerts
from lib.detect import detect_humans
from imutils.video import FPS
from scipy.spatial import distance as dist
import numpy as np
import argparse, imutils, cv2, os, time

#----- |ARGUMENTS PARSING|-----
-----
# argument parser to parse command line arguments
ap = argparse.ArgumentParser()

ap.add_argument("-i", "--input", type=str, default="",

```

```

    help="path to (optional) input video file")

ap.add_argument("-o", "--output", type=str, default="",
    help="path to (optional) output video file")

ap.add_argument("-d", "--display", type=int, default=1,
    help="whether or not output frame should be displayed")

args = vars(ap.parse_args())
#-----
-----

# loading YOLO facemask detector classes & object
classes = ["good", "bad", "none"]
detect_facemask = DETECT_FACEMASK("yolo/mask-yolov4-tiny.cfg",
    "yolo/mask-yolov4-tiny.weights", classes)

# initializing facemask detector size & confidence
detect_facemask.size = 416
detect_facemask.confidence = 0.5

# facemask detector component colors
colors = [(0, 255, 0), (0, 165, 255), (0, 0, 255)]

# loading the COCO class labels
labelsPath = os.path.sep.join([config.YOLO_PATH, "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# deriving the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([config.YOLO_PATH, "yolov3.weights"])
configPath = os.path.sep.join([config.YOLO_PATH, "yolov3.cfg"])

#weightsPath = os.path.sep.join([config.YOLO_PATH, "yolov3-
tiny.weights"])
#configPath = os.path.sep.join([config.YOLO_PATH, "yolov3-tiny.cfg"])

# loading the YOLO object detector trained on COCO dataset (80 classes)
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# checking if there's GPU usage
if config.Use_GPU:
    # set CUDA as the preferable backend and target
    print("")
    print("[INFO] Looking for GPU")
    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

# determining only the *output* layer names that we need from YOLO
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

```



```

# if a video path was not supplied
# creating a reference with source as the camera
if not args.get("input", False):
    print("[INFO] Starting the live stream..")
    vs = cv2.VideoCapture(config.url)
    #vs = cv2.VideoCapture()
    #vs.open('https://r6---sn-ci5gup-
25us.googlevideo.com/videoplayback?expire=1624996766&ei=PifbYKTWiu6HjuMP
1qi98AQ&ip=2401%3A4900%3A3b36%3Afb6a%3A38fa%3Ac081%3A771b%3Aff51&id=o-
AK-h5uev53-
NYfF8tnV3J6y5fqUroHPKowvjMbQ4pZh6&itag=22&source=youtube&requiresl=yes&
mh=82&mm=31%2C29&mn=sn-ci5gup-25us%2Csn-ci5gup-
h55e&ms=au%2Crdu&mv=m&mvi=6&pcm2cms=yes&pl=48&initcwndbps=191250&vprv=1&
mime=video%2Fmp4&ns=qsoYgVHBUDFsTRZPfjEhcCwG&cnr=14&ratebypass=yes&dur=1
76.262&lmt=1580187947492520&mt=1624974769&fvip=9&fexp=24001373%2C2400724
6&beids=9466588&c=WEB&txp=5535432&n=2y79okbryw8kCkR7XE5JS&sparams=expire
%2Cei%2Cip%2Cid%2Citag%2Csource%2Crequiresl%2Cvprv%2Cmime%2Cns%2Ccnr%2C
ratebypass%2Cdur%2Clmt&sig=AOq0QJ8wRQIgzSKYkk1__Yyh1iCaLwHA4LHJLLicJgh1a
ITU11Er488CIQCUaHppAydb-EpZtEOB7kGGITVuFCOa-
qcUeRGJx5eTfw%3D%3D&lsparams=mh%2Cmm%2Cmn%2Cms%2Cmv%2Cmvi%2Cpcm2cms%2Cpl
%2Cinitcwndbps&lsig=AG3C_xAwRQIgy0GLCvdD5Cq-
mFrX7dNYHHOaAtou5MtyC7Rz2m_m01ECIQDMo87oMGA-5sqkeFozpq-OV-
kY_QUrg7K6EZzxC547g%3D%3D')
    time.sleep(1.0)

# else, creating a reference with source as the video file
else:
    print("[INFO] Starting the video..")
    vs = cv2.VideoCapture(args["input"])

writer = None

# starting the FPS counter
fps = FPS().start()

# iterating through the frames from the video stream
while True:
    # reading the next frame from the file
    (grabbed, frame) = vs.read()
    # if the frame was not grabbed, then we have reached the end of the
    stream
    if not grabbed:
        break

    # resizing the frame
    frame = imutils.resize(frame, width=700)

    # calling detect_facemask function to detect face & masks usage in
    frames

```

```

width, height, inference_time, fm_results =
detect_facemask.inference(frame)

# counter for mask usage
masked = 0
improper_masked = 0
unmasked = 0

# looping through facemask detector results
for detection in fm_results:
    id, name, confidence, x, y, w, h = detection
    cx = x + (w / 2)
    cy = y + (h / 2)

    # updating counters
    if id == 0:
        masked = masked + 1
    if id == 1:
        improper_masked = improper_masked + 1
    if id == 2:
        unmasked = unmasked + 1

    # drawing a bounding box rectangle and label on the image
    color_fm = colors[id]
    cv2.rectangle(frame, (x, y), (x + w, y + h), color_fm, 2)
    text_fm = "%s (%s)" % (name, round(confidence, 2))
    cv2.putText(frame, text_fm, (x, y - 5),
cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, color_fm, 2)

# formatting counters text
masked_text = "Masked: {}".format(masked)
improper_masked_text = "Improperly Masked:
{}".format(improper_masked)
unmasked_text = "Unmasked: {}".format(unmasked)

# displaying counters on screen
cv2.rectangle(frame, (520, 30), (700, 90), (0, 0, 0), -1)
cv2.putText(frame, masked_text, (530, 40), cv2.FONT_HERSHEY_SIMPLEX,
0.40, (0, 255, 0), 1)
cv2.putText(frame, improper_masked_text, (530, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.40, (0, 165, 255), 1)
cv2.putText(frame, unmasked_text, (530, 80),
cv2.FONT_HERSHEY_SIMPLEX, 0.40, (0, 0, 255), 1)

# calling detect_humans function to detect only humans in the frames
results = detect_humans(frame, net, ln,
human_idx=LABELS.index("person"))

```

```

# initializing the set of indexes that violate the max/min social
distance limits
serious = set()
abnormal = set()

# ensuring there are *at least* two people detections (required in
# order to compute our pairwise distance maps)
if len(results) >= 2:
    # extracting all centroids from the results and computing the
    # Euclidean distances between all pairs of centroids
    centroids = np.array([r[2] for r in results])
    # extracting heights of all detected bounding boxes
    pixel_heights = np.array([r[1][3]-r[1][1] for r in results])
    D = dist.cdist(centroids, centroids, metric="euclidean")

    # loop over the upper triangular of the distance matrix
    for i in range(0, D.shape[0]):
        for j in range(i + 1, D.shape[1]):
            # calibrating the pixel distance to centimeters
            calib_factor = (1/pixel_heights[i] + 1/pixel_heights[j])
/ 2 * config.avg_height
            D[i, j] = D[i, j] * calib_factor
            # check to see if the distance between any two
            # centroid pairs is less than the configured number of
pixels
            if D[i, j] < config.MIN_DISTANCE:
                # update our violation set with the indexes of the
centroid pairs
                    serious.add(i)
                    serious.add(j)
                # update our abnormal set if the centroid distance is
below max distance limit
                    if (D[i, j] < config.MAX_DISTANCE) and not serious:
                        abnormal.add(i)
                        abnormal.add(j)

# iterating through the results
for (i, (prob, bbox, centroid)) in enumerate(results):
    # extracting the bounding box and centroid coordinates, and
    # initializing the color of the annotation
    (startX, startY, endX, endY) = bbox
    (cX, cY) = centroid
    color = (0, 255, 0)

    # if the index pair exists within the violation/abnormal sets,
then update the color
    if i in serious:
        color = (0, 0, 255)
    elif i in abnormal:
        color = (0, 255, 255) #orange = (0, 165, 255)

```

```

# drawing:-
# (1) a bounding box around the person and
# (2) the centroid coordinates of the person
cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
cv2.circle(frame, (cX, cY), 5, color, 2)

# drawing some of the parameters
Safe_Distance = "Safe distance: > {}".format(config.MIN_DISTANCE/100)
cv2.putText(frame, Safe_Distance, (505, frame.shape[0] - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 0, 0), 2)
Violations_Threshold = "Threshold limit:
{}".format(config.Violations_Threshold)
cv2.putText(frame, Violations_Threshold, (505, frame.shape[0] - 37),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 0, 0), 2)

# drawing the total number of social distancing violations on the
output frame
cv2.rectangle(frame, (0, 0), (215, 50), (0, 0, 0), -1)

text = "Serious Violations: {}".format(len(serious))
cv2.putText(frame, text, (15, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.50,
(0, 0, 255), 2)

text1 = "Abnormal Violations: {}".format(len(abnormal))
cv2.putText(frame, text1, (15, 40), cv2.FONT_HERSHEY_DUPLEX, 0.50,
(0, 255, 255), 2)

#----- |Alert function|-----
-----
if len(serious) >= config.Violations_Threshold:
    cv2.putText(frame, "ALERT: Violations exceeded limit!", (15,
frame.shape[0] - 20),
        cv2.FONT_HERSHEY_COMPLEX, 0.60, (0, 0, 255), 2)
    if config.Alert:
        print("")
        print('[ALERT] Sending alert...')
        Alerts().trigger()
        print('[ALERT] Alert sent')

#-----#
# checking to see if the output frame should be displayed
if args["display"] > 0:
    # displaying the output frame
    cv2.imshow("MaViSS AI - COVID19 Norms Surveillance System",
frame)

key = cv2.waitKey(1) & 0xFF

# breaking loop if 'ESC' key is pressed

```

```

        if key == 27:
            break
# updating the FPS counter
fps.update()

# if an output video file path has been supplied and the video
# writer has not been initialized, doing so now
if args["output"] != "" and writer is None:
    # initializing the video writer
    fourcc = cv2.VideoWriter_fourcc(*"MJPG")
    writer = cv2.VideoWriter(args["output"], fourcc, 25,
        (frame.shape[1], frame.shape[0]), True)

# if the video writer is not None, writing the frame to the output
video file
    if writer is not None:
        writer.write(frame)

# stopping the timer and displaying FPS information
fps.stop()
print("=====")
print("[INFO] Elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] Approx. FPS: {:.2f}".format(fps.fps()))

# closing any open windows
cv2.destroyAllWindows()
#=====
=====

```

Artificial Intelligence System to Analyse Human Facial Emotions and Text Sentiments

Final Report

Meghna Narwade

ENGR-498-07

ABSTRACT

The aim of this project is to build a Response Sentiment Analyser using **Artificial Intelligence** and **Deep learning**. Its main functions are detecting facial expression in real time video and using the audio to measure the polarity of their response. The proposed standalone system detects the facial expressions in real time video with an accuracy of about **86.75%**. This system can be used to help users analyze the facial expressions and the content of their responses within the text using text sentiment analysis during debate competition, interview, meeting or conversation. This report presents the common techniques of analyzing sentiment from machine learning and deep learning perspective.

Keywords: Artificial Intelligence, Deep Learning, Facial expression, Text sentiment analysis.

INTRODUCTION

Humans share a universal set of fundamental emotions. These emotions are significantly expressed through facial expressions. Facial emotion recognition is a task that can also be accomplished by computers. For a detection approach, it is important to have a taxonomic reference for classifying the eight basic emotions which consist of anger, contempt, disgust, fear, happiness, sadness, surprise as well as neutral. The proposed system uses python libraries to detect the face in real time video and extracts 68 facial landmarks and classifies the facial expression using deep neural network(DNN). In addition, the audio is extracted by the system and converted to text which uses Textblob sentiment analyzer library to estimate the polarity of their answer/response as positive, negative or neutral. In light of this, the literature review explores and discusses the concept of facial emotion recognition and text sentiment analysis by undertaking a systematic review of scientific research papers, journals, and articles.

OVERVIEW

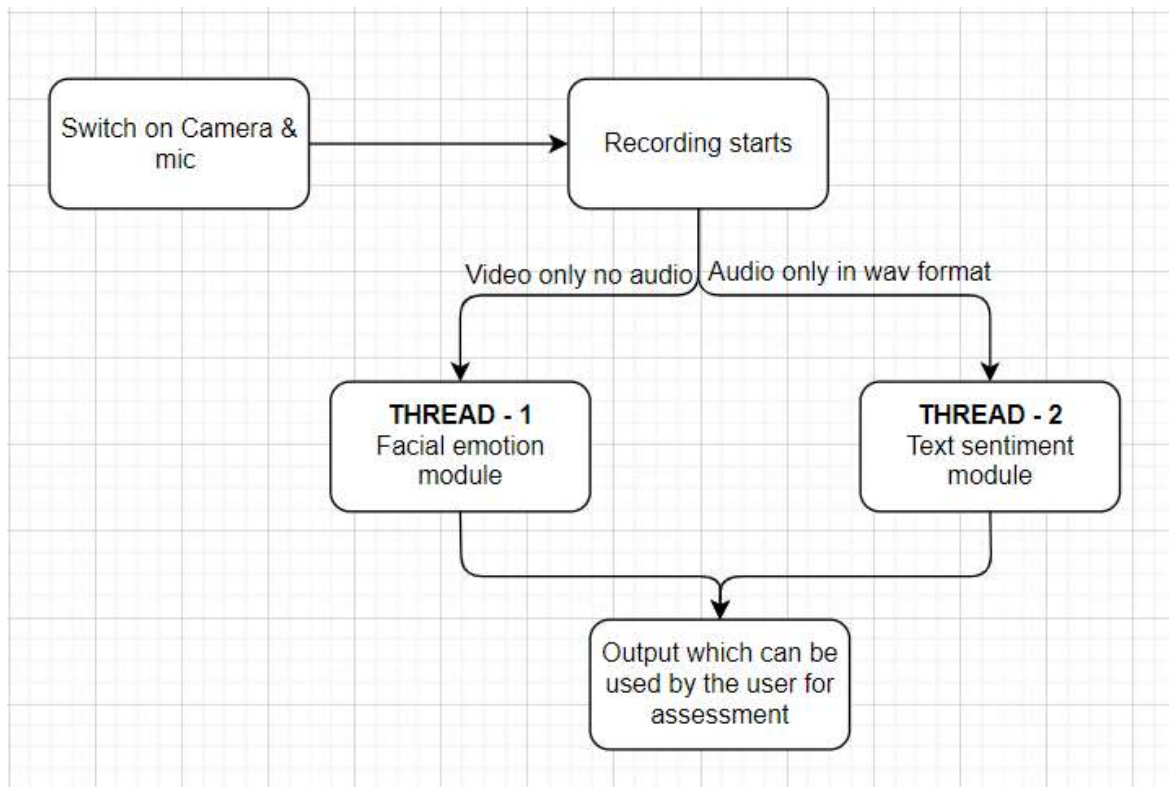


Figure: Flowchart showing working of the project

LITERATURE REVIEW

Techniques for facial emotion detection using landmark extraction.

Research Paper	Number of landmarks	Method of landmark detection	Dataset used	Classifier used	Accuracy
<u>Real time emotion recognition system using facial expression and EEG</u>	10	Manually placed through optical flow algorithm	Own database	CNN	93.02% (for facial emotion detection)
<u>Real time facial expression recognition in Video</u>	22	Manually placed using feature displacement approach	CK+ database	SVM	86.0%
<u>Real-time Mobile Facial Expression Recognition</u>	77	Extracted using STASM library	CK+ database	SVM	85.8%

System					
<u>A fuzzy logic approach for real time facial recognition of facial emotions</u>	68	Extracted using DLIB library	CK+ database	FURIA	83.2%
Our approach: Response sentiment analyser	68	Extracted using DLIB library	CK+ database JAFEE database TFEID database Additional images via manual web scraping	DNN	86.75%

TECHNICAL DESCRIPTION WITH FLOWCHARTS

Facial emotion detection

The facial emotion recognition consists of two parts (i) Image processing that extract facial landmarks. (ii) Neural network for emotion recognition.

The proposed system uses Dlib library which is one of the most utilized packages for face recognition. The Dlib python library is used to detect faces from images and extract 68 facial landmarks from the detected face. The detected facial landmarks is an array of 68 points. The order of these points is consistent, point 1 is the right chin, 34 is the tip of the nose, etc.

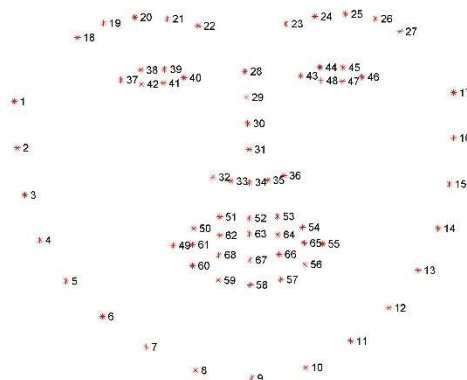
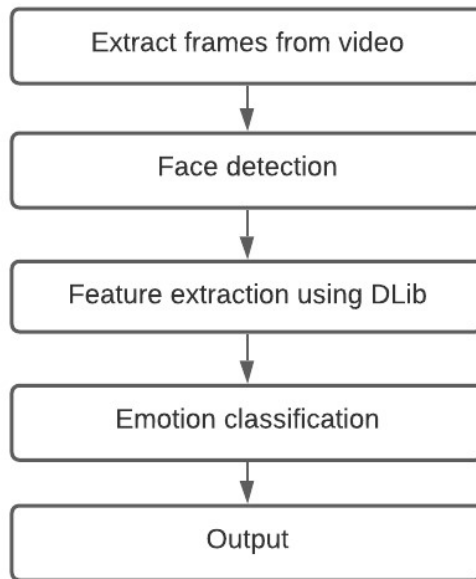


Figure: Facial landmarks

The Dlib library uses the Histogram of Oriented gradients (HOG) function to detect the face. The predictor function in Dlib then places 68 landmarks on the detected face. The Dlib library accurately detects the facial landmarks at the angle of -30 to +30 degrees in any direction. These landmarks are normalized and saved in a .csv file. This file is then used to train and test the deep neural network. The normalized coordinates of the facial landmarks are then passed to the deep neural network which classifies the emotion from the image.



Flowchart showing workflow of real time facial emotion detection

Output stages:



Training the Neural network

Vectorized facial landmarks are used to train a DNN. The vectorization of facial landmarks is achieved by putting tensors of 2-dimensional coordinates into a vector. Since these coordinates are normalized, when the vectorized facial landmarks being fed into the DNN, the network can be trained properly.

$[(x_1, y_1), (x_2, y_2), \dots, (x_{68}, y_{68})] \rightarrow [x_1, y_1, x_2, y_2, \dots, x_{68}, y_{68}]$

Neural network perform the best when the data is concentrated within small radius like $[0,1]$. The detected landmarks are then scaled to $[-1,1]$ and aligned using the tip of nose.

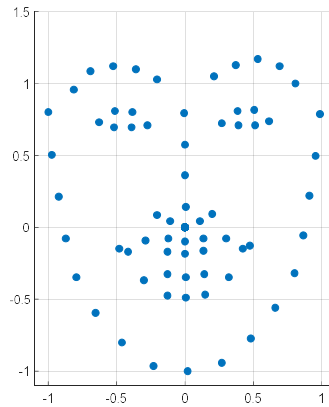
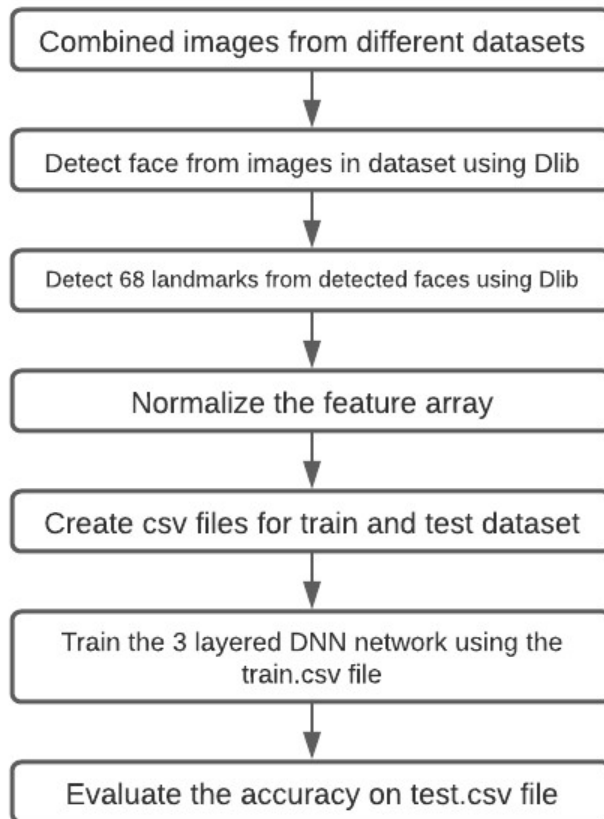


Figure: Normalize

The result data is stored in a CSV file with an integer indicating the emotion.



Flowchart of training the neural network

The dataset used is a combination of CK+, JAFFE, TFEID and RaFD (3000 images, eight classes). The model used in building the deep neural network is a sequential model with three hidden layers. The type of layers used is dense which implies that every neuron in the dense

layer receives input from all neurons of the previous layer. The activation function used is sigmoid. Adam optimizer allows the framework to adjust the step size depending on the loss. The accuracy attained after testing the neural network was about 86.75%.

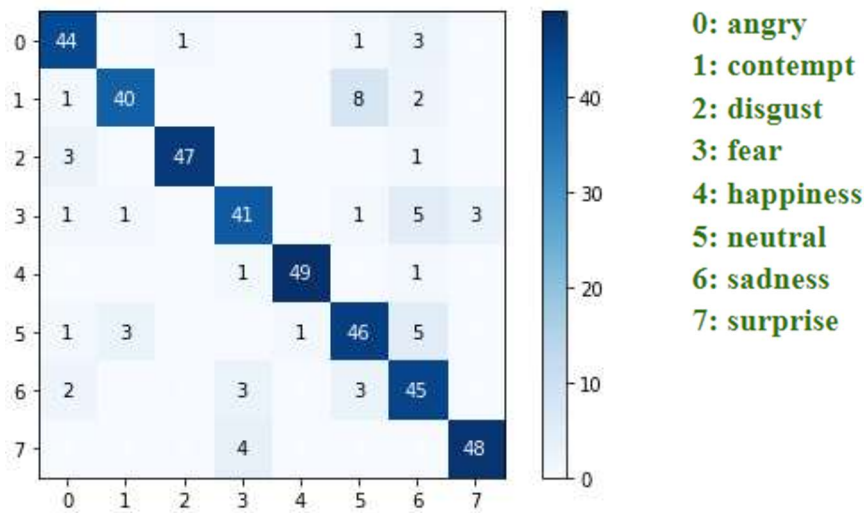


Figure: Confusion matrix

Neural network Summary

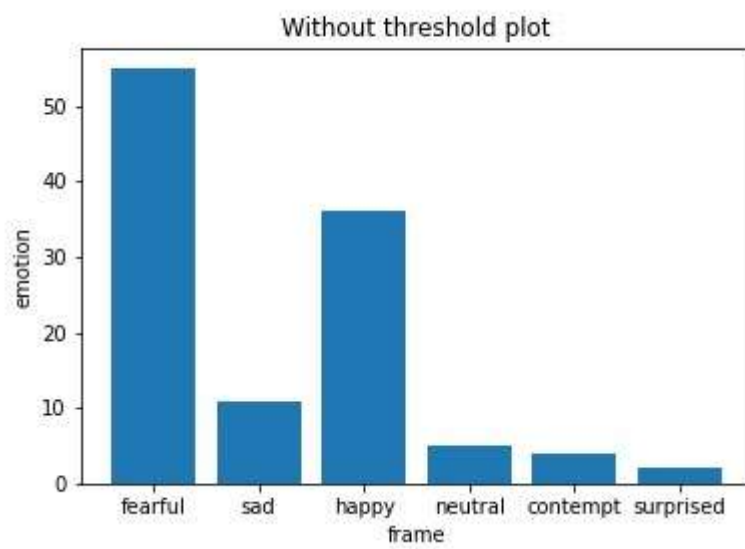
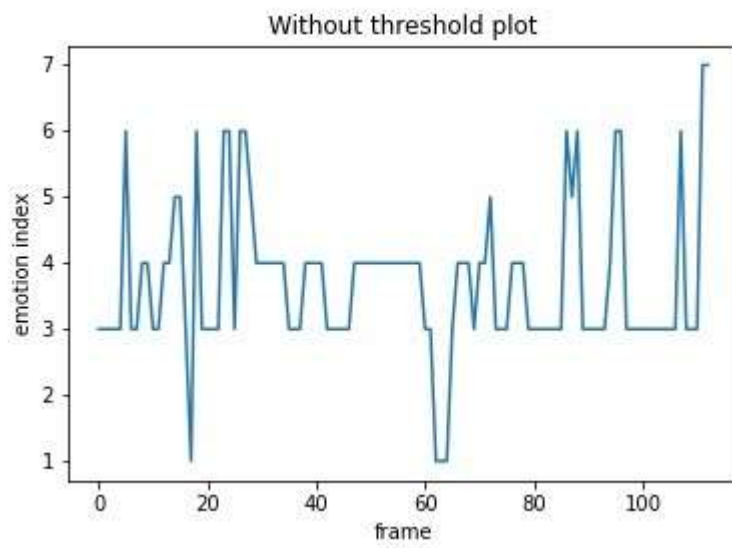
Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 272)	37264
dense_1 (Dense)	(None, 544)	148512
dense_2 (Dense)	(None, 272)	148240
dense_3 (Dense)	(None, 8)	2184
Total params: 336,200		
Trainable params: 336,200		
Non-trainable params: 0		

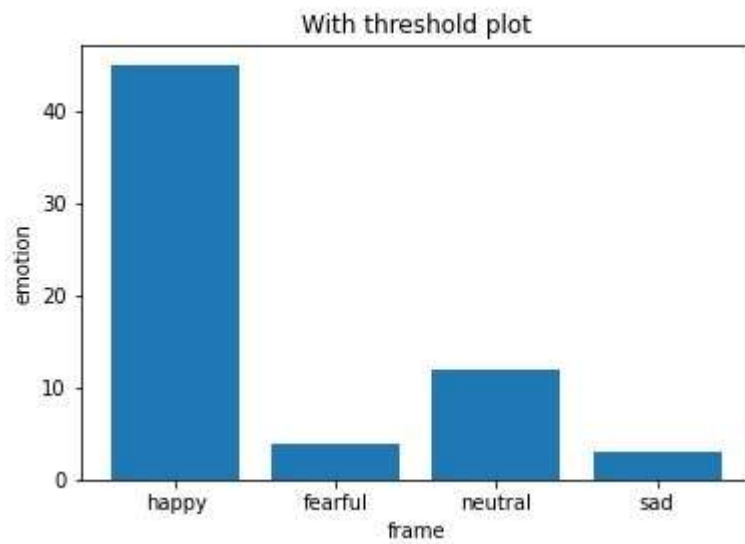
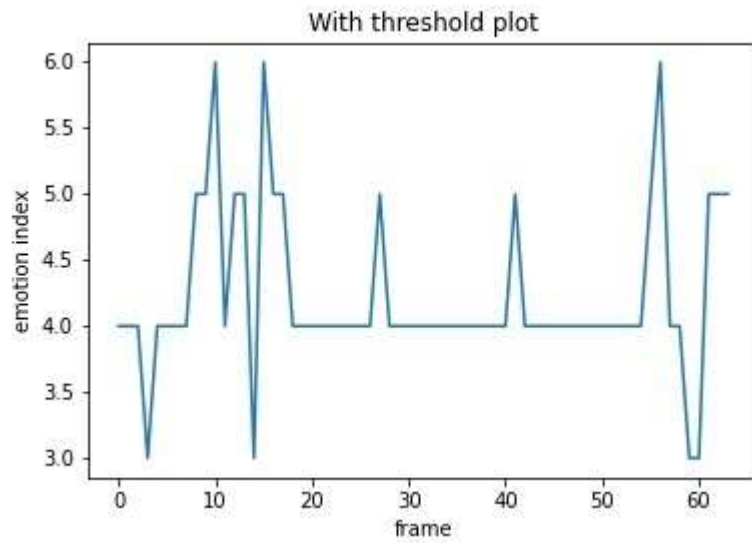
Improving accuracy for Real time processing

To improve the accuracy while performing real time processing, a threshold was set for the level of confidence for each of the eight emotions. The emotion is only displayed if the confidence level of that emotion is greater than its threshold value. If the emotion detected does not cross the threshold value then the emotion rendered in the previous frame is displayed.

- **Without threshold:**



- **With threshold:**



- Frame per second rate:

- On Laptop:

```
fps start
fps stop
```

```
[INFO] elapsed time: 12.74
[INFO] approx. FPS: 8.95
```

- Raspberry pi camera:

```
GST_ARGUS: Running with following settings:
Camera index = 0
Camera mode = 5 (live, from video card)
Output Stream W = 1280 H = 720
seconds to Run = 0
Frame Rate = 120.000005
GST_ARGUS: Setup Complete, Starting captures for 0 seconds
GST_ARGUS: Starting repeat capture requests.
```

- Fps recorded:

```
fps stop

fps recorded on Jetson nano
[INFO] elapsed time: 27.20
[INFO] approx. FPS: 4.19
```

Text Sentiment Analysis

LITERATURE REVIEW

Text sentiment analyzer is a tool that is used to predict the polarity of a sentence/passage with the help of various techniques present. There are 3 approaches to do text sentiment analysis

- **By building your own model**
 1. Using Machine learning techniques like: Linear regression, SVM, Naive Bayes
 2. Using BERT (Bidirectional Encoder Representations from Transformers)
- **By using a SaaS tool**
 1. Like: Monkeylearn & others
- **By using pre-trained models**
 1. textblob, vader, spacy, gensim

	content	textblob	textblob_bayes	nltk_vader
0	I've enjoyed and grown in my current role	25	65	51
1	I am an ambitious and driven individual. I thrive in a goal-oriented environment	12	92	48
2	What makes me unique is my ability to meet and exceed deadlines	38	59	32
3	While I highly valued my time at my previous company, there are no longer opportunities for growth that align with my career goals	0	3	73
4	I hated the job and the company. They were awful to work for.	-95	-60	-80
5	I do good work	70	4	44
6	I tend to lose my patience with incompetent people.	-35	-33	-70
7	I missed too much work.	20	-10	-30

Figure: Comparison among textblob default analyzer, vader, and textblob bayes

Mentioned above are some negative and positive interviewee responses to check how well these libraries can classify them as positive or negative and overall we find textblob_bayes yield more satisfying results. The numbers shown in the table are the polarity of each sentence where -100 means negative and +100 means positive.

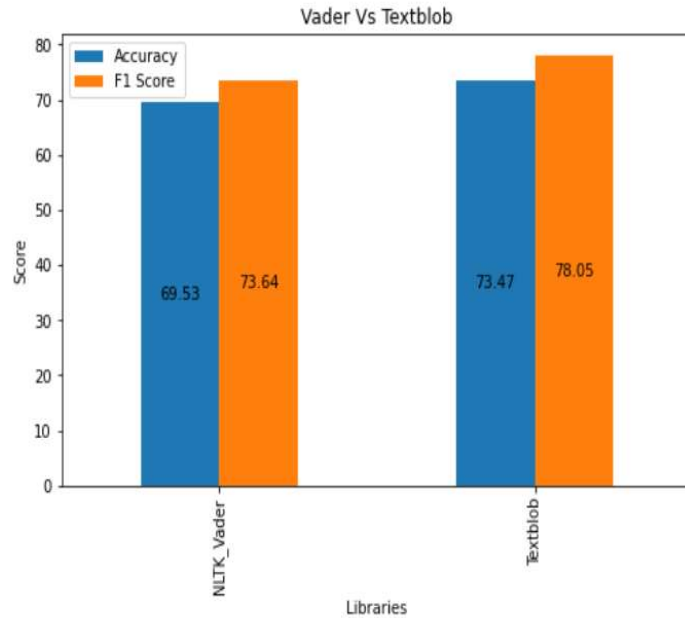


Figure: Bar graph displaying difference between Vader and Textblob

Given below are some of the most widely known speech recognition tools:

Link	Result
<u>A Benchmarking of IBM, Google and Wit Automatic Speech Recognition Systems</u>	This research paper differentiates among IBM, Google cloud speech, & Wit. Result: Google Cloud Speech dominates
<u>Which Automatic Transcription Service is the Most Accurate?</u>	Differentiating among various speech to text APIs available Result: 1st Google cloud speech & 2nd Temi by Rev.ai
<u>How Reliable is Speech-to-Text in 2021?</u>	An article that differentiates among different speech to text APIs. Result: 1st Temi by Rev.ai & 2nd Google cloud speech

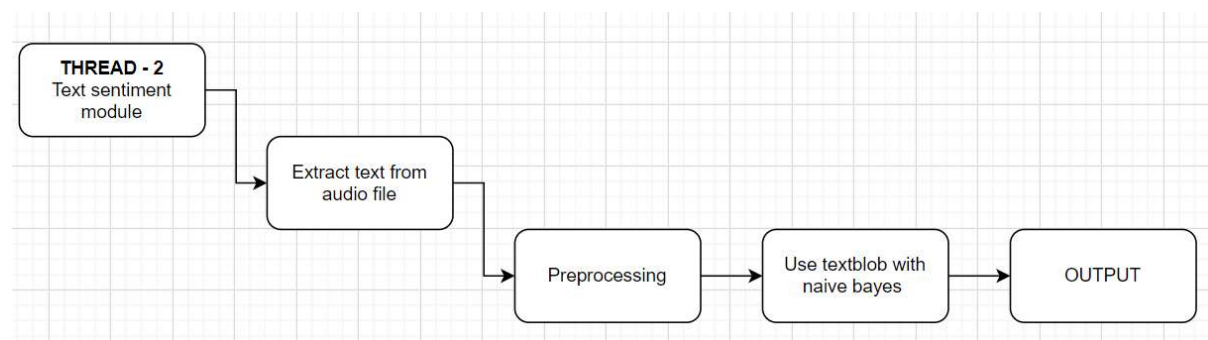
The text sentiment analysis algorithm with the help of **textblob library** allows us to determine whether the response of the speaker is positive, negative or neutral by calculating the average polarity over each word in a given text using a dictionary of adjectives and their hand-tagged scores. It uses a pattern library (a web mining module for Python) for that, which takes the individual word scores from sentiwordnet (lexical resource for opinion mining). Polarity lies between [-1,1], -1 indicates negative answer and +1 indicates positive answer. We have used this polarity scale to set a threshold which allows us to classify answers as either positive, negative & neutral i.e.

Polarity above 60% is classified as **positive**

Between 40-60% is classified as **neutral**

And below 40% is classified as **negative**

Working:



- **Pre-processing**
 - The text is extracted from the audio file using the **speech_recognition library** in python but the audio file being large (more than 1 min) can be an issue. So, the audio file is broken down into chunks and then bit by bit the text is extracted and then combined.
 - The extracted text will then be split/broken down into several sentences using the `split()` function. The delimiter considered is period(.)
 - After breaking into sentences a **punctuator model** is run on the sentences and if there are any new punctuations introduced by the **PM** then we further break it down into sentences.
- **Textblob**
 - Textblob calculates the average polarity over each word in a given text using a dictionary of adjectives and their hand-tagged scores. It uses a pattern library (a web mining module for Python) for this and takes the individual word scores from sentiwordnet (lexical resource for opinion mining).
 - Polarity of each sentence is calculated using textblob with naive bayes analyzer and the output will be shown like this:
 - Number of positive sentences in the passage: n
 - Number of negative sentences in the passage: m
 - Number of neutral sentences in the passage: l

- Total number of sentences in a passage: $n+m+1$
- **Overall positivity of the passage:** Sum of polarities above 60% / Total number of sentences in a passage
- **Overall neutrality of the passage:** Sum of polarities between 40% - 60% / Total number of sentences in a passage
- **Overall negativity of the passage:** Sum of polarities below 40% / Total number of sentences in a passage

Integration of the two models:

The integration process is basically combining the two modules explained above i.e. Facial emotion & text sentiment detection. The integration part is being done using **multi-threading** which helps us to run multiple function calls simultaneously i.e. one thread records the video using opencv & the other thread records the audio using pyaudio & the output of each of these threads will then be served as an input to the two modules implemented which will then predict emotions & analyze the polarity of the content obtained from the audio.

We calculated the fps rate for the multithreading process by: dividing the total number of frames with the elapsed time of the program & the fps recorded was about 4-5fps.

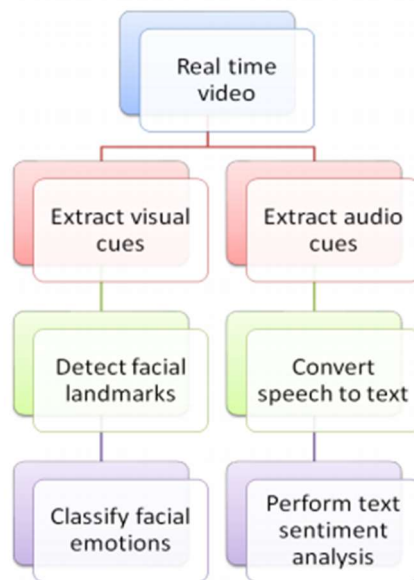


Figure: Flowchart of the process

Program output:

```

total frames 58
elapsed time 10.123109817504883
recorded fps 5.729464665068277
  
```

HARDWARE COMPONENT

NVIDIA Jetson Nano B01 & its components:

Jetson nano provides **Maxwell 128 core GPU**, emphasizing Deep Learning in its hardware design and software libraries. It is capable of running multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing.

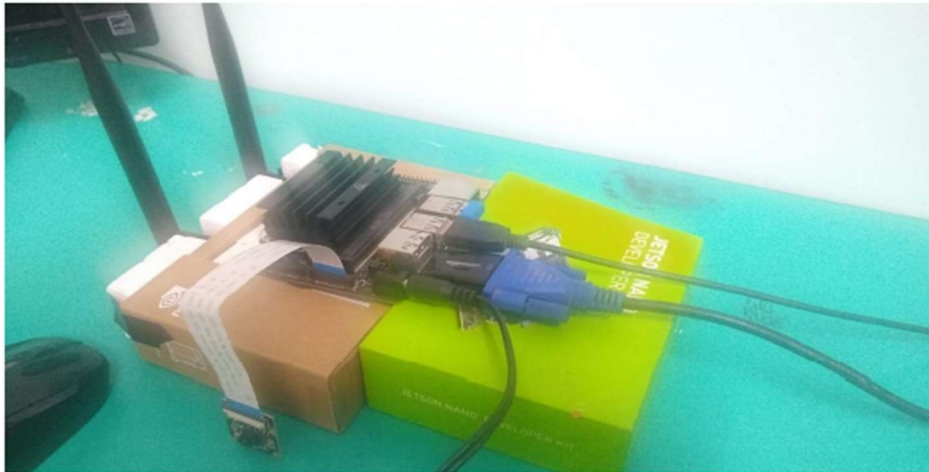


Figure: Jetson Nano setup

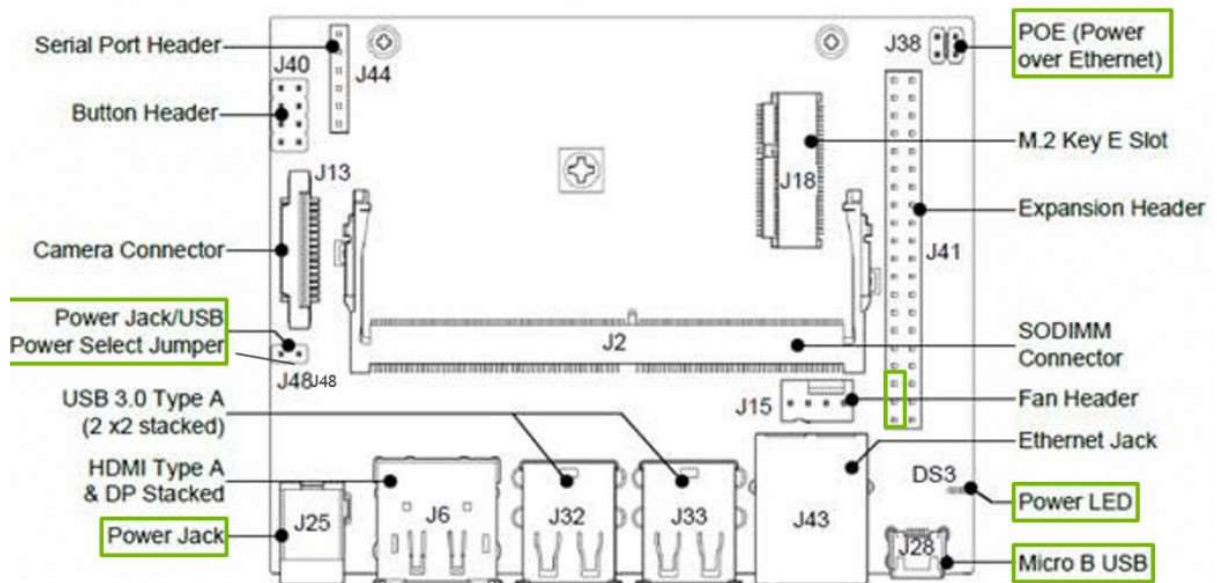


Figure: Jetson Nano hardware Schematic diagram

SOFTWARE COMPONENTS

All the software libraries that have been used in this project are mentioned below in the table:

Facial emotion recognition	Text sentiment analysis
➤ OpenCV	➤ Textblob
➤ matplotlib	➤ Speech_recognition
➤ Tensorflow	➤ Punctuator
➤ Dlib	➤ Pydub

SYSTEM CONSTRAINTS

Power requirement: Jetson Nano (Input: 5V 4 Amp)

Efficiency: The Jetson nano for real-time processing on Raspberry pi runs at the rate of about 20 fps

Feasibility: Currently the facial emotion recognition algorithm is constrained to detect frames that are showing relatively extreme emotions & the dlib library accurately **detects the facial landmarks at the angle of -30 to +30 degrees in any direction**. This helped us in deciding which images to select for the training dataset.

VALIDATION

In order to test the functioning of the overall system, we performed real time testing and gave a review on the movie Avengers: Age of Ultron. Video link is given below:

[Demonstration video.mp4](#)

CONCLUSION

The presented project is research on FER and analysing text for the sentiment , which allows us to know a way of sensing emotions that can be considered as mostly used AI and pattern analysis applications. The presented model can detect facial expressions of a person and analyse the sentiment of the text that is extracted from his audio. The proposed integrated system extracts video and audio simultaneously **with a frame rate of 4-5 fps**.The facial emotion detection system successfully detects facial expression of faces detected in **real time video with an accuracy of about 86.75%**. The **audio from the video** is successfully **extracted, converted to text, cleaned** and processed to determine if the attitude of the speaker in a given situation is **positive, negative or neutral**.

FUTURE PROSPECTS

This model can further be trained to improve its accuracy. And also the extracted audio from video can be used to perform speech emotion detection to recognise and improve the emotional aspects of speech.

REFERENCES

- [1] [Development of a Real-Time Emotion Recognition System Using Facial Expressions and EEG based on machine learning and deep neural network methods](#)
- [2] [Real time facial expression recognition in video using support vector machines](#)
- [3] [Real-time Mobile Facial Expression Recognition System](#)
- [4] [A fuzzy logic approach for real time facial recognition of facial emotions](#)
- [5] <https://learnopencv.com/facial-landmark-detection/>
- [6] <https://www.analyticsvidhya.com/blog/2019/08/how-to-remove-stopwords-text-normalization-nltk-spacy-gensim-python/>
- [7] <https://cloud.google.com/blog/products/gcp/toward-better-phone-call-and-video-transcription-with-new-cloud-speech-to-text>
- [8] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7256403/>
- [9] <https://stackoverflow.com/questions/14140495/how-to-capture-a-video-and-audio-in-python-from-a-camera-or-webcam>
- [10] <https://www.codeproject.com/Articles/5269453/Improving-NLTK-Sentiment-Analysis-with-Data-Annota>
- [11] <https://www.alphabold.com/sentiment-analysis-the-lexicon-based-approach/>
- [12] Lab manual 6
- [13] <https://www.programmersought.com/article/27703847966/>
- [14] <https://stackoverflow.com/questions/53945501/real-time-video-processing-using-multithreading-in-python>

Appendix A

```
#AudioVideo recording code
```

```
import cv2
import pyaudio
import wave
import threading
import time
import subprocess
import os
```

```
class VideoRecorder():
```

```

# Video class based on openCV
def __init__(self):

    self.fourcc = "MJPG"          # capture images (with no decrease i
n speed over time; testing is required)
    self.dim = (640,480)         # video formats and sizes also depen
d and vary according to the camera used
    self.video_filename = "Fer.avi"
    self.fps = 6
    self.cap = cv2.VideoCapture(0)
    self.open = True
    self.write = cv2.VideoWriter_fourcc(*self.fourcc)
    self.vid = cv2.VideoWriter(self.video_filename, self.write, sel
f.fps, self.dim)

    self.frame_counts = 1
        # fps should be the minimum constant rate at which
the camera can

    self.start_time = time.time()

# Video starts being recorded
def record(self):
    counter = 1
    timer_start = time.time()
    timer_current = 0

    while (self.open==True):
        ret, frame = self.cap.read()

        if ret:
            self.vid.write(frame)
            # print(str(counter) + " " + str(self.count) + " fr
ames written " + str(timer_current))
            self.frame_counts += 1
            counter += 1
            timer_current = time.time() - timer_start
            time.sleep(0.16)
            # gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            cv2.imshow('frame', frame)
            cv2.waitKey(1)

# Finishes the video recording therefore the thread too
def stop(self):

    if self.open==True:
        self.open=False

```

```

        self.vid.release()
        self.cap.release()
        cv2.destroyAllWindows()
    else:
        pass

# Launches the video recording function using a thread
def start(self):
    t1 = threading.Thread(target=self.record)
    t1.start()

class AudioRecorder():

    # Audio class based on pyAudio and Wave
    def __init__(self):

        self.open = True
        self.rate = 44100
        self.frames_per_buffer = 1024
        self.channels = 2
        self.format = pyaudio.paInt16
        self.audio_filename = "video 1.wav"
        self.audio = pyaudio.PyAudio()
        self.stream = self.audio.open(format=self.format,
                                      channels=self.channels,
                                      rate=self.rate,
                                      input=True,
                                      frames_per_buffer = self.frames_p
er_buffer)
        self.audio_frames = []

    # Audio starts being recorded
    def record(self):

        self.stream.start_stream()
        while (self.open == True):
            data = self.stream.read(self.frames_per_buffer)
            self.audio_frames.append(data)
            if self.open==False:
                break

    # Finishes the audio recording therefore the thread too
    def stop(self):

```

```

        if self.open==True:
            self.open = False
            self.stream.stop_stream()
            self.stream.close()
            self.audio.terminate()

            aud = wave.open(self.audio_filename, 'wb')
            aud.setnchannels(self.channels)
            aud.setsampwidth(self.audio.get_sample_size(self.format))
            aud.setframerate(self.rate)
            aud.writeframes(b''.join(self.audio_frames))
            aud.close()

        pass

# Launches the audio recording function using a thread
def start(self):
    t2 = threading.Thread(target=self.record)
    t2.start()

def start_AVrecording(filename):

    global t1
    global t2

    t1 = VideoRecorder()
    t2 = AudioRecorder()

    t2.start()
    t1.start()

    return filename

def start_video_recording(filename):

    global t1

    t1 = VideoRecorder()
    t1.start()

    return filename

```



```

def start_audio_recording(filename):

    global t2

    t2 = AudioRecorder()
    t2.start()

    return filename

def stop_AVrecording(filename):

    t2.stop()
    frame_counts = t1.frame_counts
    elapsed_time = time.time() - t1.start_time
    recorded_fps = frame_counts / elapsed_time
    print("total frames " + str(frame_counts))
    print("elapsed time " + str(elapsed_time))
    print("recorded fps " + str(recorded_fps))
    t1.stop()

    # Makes sure the threads have finished
    while threading.active_count() > 1:
        time.sleep(1)

# Required and wanted processing of final files
def file_manager(filename):

    local_path = os.getcwd()

    if os.path.exists(str(local_path) + "/temp_audio.wav"):
        os.remove(str(local_path) + "/temp_audio.wav")

    if os.path.exists(str(local_path) + "/temp_video.avi"):
        os.remove(str(local_path) + "/temp_video.avi")

    if os.path.exists(str(local_path) + "/temp_video2.avi"):
        os.remove(str(local_path) + "/temp_video2.avi")

    if os.path.exists(str(local_path) + "/" + filename + ".avi"):
        os.remove(str(local_path) + "/" + filename + ".avi")

filename = "Default_user"
file_manager(filename)

```

```
start_AVrecording(filename)

time.sleep(20)

stop_AVrecording(filename)
print("Done")
```

Appendix B

```
#Face emotion detection:

import dlib
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# initialize face and facial landmark detector
detector = dlib.get_frontal_face_detector()

# replace with proper path!!!!!!
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

#loading DNN
path_save = "./testsave4"
model_restore = tf.keras.models.load_model(
path_save)

model_restore.summary()

#text characteristics
window_name = 'Image'
font = cv2.FONT_HERSHEY_SIMPLEX
fontScale = 1
color = (0, 0, 255)
thickness = 2

#emotion detected dictionary
emotions = { 0:"angry" ,1:"contempt" ,2:"disgusted",3:"fearful", 4:"happy", 5:"neutral",6:"sad",7:"surprised"}
print(emotions)

#normalize and add to array function
def normalize(detected_face,shape,new_arr):
    i=1
```

```

arr = []
x_scale = -1*(shape.parts()[0].x - shape.parts()[33].x)
y_scale = shape.parts()[8].y - shape.parts()[33].y
for p in shape.parts():
#     detected_face = cv2.circle(detected_face, (p.x,p.y), 2, (0,0,2
55), -1)
    p=p-shape.parts()[33]
    x_new = p.x / x_scale
    y_new = p.y / y_scale
    arr = np.append(arr,x_new)
    arr = np.append(arr,y_new)
    i+=1
return arr

```

```

#finding emotion from output
def result(test_result,emotion_result,index_result):
    for r in test_result:
        c = ""
        if r[0]>99:
            j=0
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[1]>0.99:
            j=1
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[2]>0.99:
            j=2
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[3]>0.99:
            j=3
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[4]>0.85:
            j=4
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[5]>0.90:
            j=5
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "

```

```

    if r[6]>0.99:
        j=6
        index_result.append(j)
        emotion_result.append(emotions[j])
        c = c + emotions[j] + " "
    if r[7]>0.90:
        j=7
        index_result.append(j)
        emotion_result.append(emotions[j])
        c = c + emotions[j] + " "

    return emotion_result, index_result,c

from imutils.video import FPS
# vid = cv2.VideoCapture(0)
vid = cv2.VideoCapture('fer_video.mp4')

fps = FPS().start()

x = 0
analysis_arr = []
analysis_ind = []
prev_c = "unknown"
c=""
out = cv2.VideoWriter('output.mp4', -1, 20.0, (640,480))

while True:
    ret, frame = vid.read()
    print(x)
    if ret:
#         print(frame.shape)
        gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        faces = detector(gray, 0)
        detected_face = frame
        new_arr = []
#         print(faces)
        fps.update()

        for f in faces:
            shape = predictor(gray, f)
            pred = normalize(detected_face,shape,new_arr)
            new_arr.append(pred)
        q=0
        for f in faces:
            arr_x = np. reshape(new_arr[q], (1,136))
            index_result=[]

```

```

        emotion_result=[]
        test_result = model_restore.predict(arr_x)
#         print(test_result)
        emotion_result, index_result, c = result(test_result,emotion_result,index_result)
        if c=="":
            c=prev_c
        if len(index_result)!=0:
            analysis_arr.append(emotion_result[0])
            analysis_ind.append(index_result[0])
            detected_face = cv2.rectangle(detected_face, (f.tl_corner().x, f.tl_corner().y),
                                           (f.br_corner().x, f.br_corner().y), (
0,255,0), 3)
            frame = cv2.putText(frame, c, (f.tl_corner().x, f.tl_corner().y), font,
                                fontStyle, color, thickness, cv2.LINE_AA)
            q+=1

#         cv2.imwrite(f"Frames/Frame{x}.jpg",frame)
        out.write(frame)
        cv2.imshow('frame', frame)
        prev_c = c
        x += 1
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

out.release()
vid.release()

fps.stop()
print(x)
print("fps start")
print("fps stop\n")
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
print("\n")

cv2.destroyAllWindows()

# print(analysis_ind)
# print(analysis_arr)

```

Appendix C

```
#text sentiment analysis
```

```

from textblob import TextBlob
from textblob.classifiers import NaiveBayesClassifier
from textblob.sentiments import NaiveBayesAnalyzer
import nltk
from pydub import AudioSegment
import speech_recognition as sr
from os import path
from nltk import tokenize

nltk.download('movie_reviews')
nltk.download('punkt')
nltk.download('stopwords')

#Converting mp4 to wav format with 128k bitrate
src="debate1.mp4"

AudioSegment.converter = "C:/ffmpeg-4.4-full_build/bin/ffmpeg.exe"
AudioSegment.ffmpeg = "C:/ffmpeg-4.4-full_build/bin/ffmpeg.exe"
AudioSegment.ffprobe = "C:/ffmpeg-4.4-full_build/bin/ffprobe.exe"

sound = AudioSegment.from_file(file=src, format="mp4")
sound.export("recording.mp3", format="mp3", bitrate="128k")

# convert mp3 file to wav

sound = AudioSegment.from_mp3("recording.mp3")
sound.export("transcript.wav", format="wav")

##Code-----
# importing libraries
import speech_recognition as sr
import os
from pydub import AudioSegment
from pydub.silence import split_on_silence

# create a speech recognition object
r = sr.Recognizer()

# a function that splits the audio file into chunks
# and applies speech recognition
def get_large_audio_transcription(path):
    """
        Splitting the large audio file into chunks
        and apply speech recognition on each of these chunks
    """

```

```

# open the audio file using pydub
sound = AudioSegment.from_wav(path)
# split audio sound where silence is 700 miliseconds or more and ge
t chunks
chunks = split_on_silence(sound,
    # experiment with this value for your target audio file
    min_silence_len = 500,
    # adjust this per requirement
    silence_thresh = sound.dBFS-14,
    # keep the silence for 1 second, adjustable as well
    keep_silence=500,
)
folder_name = "audio-chunks"
# create a directory to store the audio chunks
if not os.path.isdir(folder_name):
    os.mkdir(folder_name)
whole_text = ""
# process each chunk
for i, audio_chunk in enumerate(chunks, start=1):
    # export audio chunk and save it in
    # the `folder_name` directory.
    chunk_filename = os.path.join(folder_name, f"chunk{i}.wav")
    audio_chunk.export(chunk_filename, format="wav")
    # recognize the chunk
    with sr.AudioFile(chunk_filename) as source:
        audio_listened = r.record(source)
        # try converting it to text
        try:
            text = r.recognize_google(audio_listened)
        except sr.UnknownValueError as e:
            print("Error:", str(e))
        else:
            text = f"{text.capitalize()} "
            #print(chunk_filename, ":", text)
            whole_text += text
# return the text for all chunks detected
return whole_text

path = "transcript.wav"
#print("\nFull text:", get_large_audio_transcription(path))
t=get_large_audio_transcription(path)
print(t)

sentence_break=[]
sentence_break=t.split('.')
print(sentence_break)

from punctuator import Punctuator

```

```

p = Punctuator('punctuator_model/Demo-Europarl-EN.pcl')
semi_final=[]
final=[]
for ele in sentence_break:
    if len(ele)>1:
        test=p.punctuate(ele)
        semi_final=test.split('.')
        for i in semi_final:
            if i!="":
                final.append(i)
                #pre-trained model 1
# #p1=Punctuator('punctuator_model/INTERSPEECH-T-
BRNN.pcl')          # pre-trained model 2
# t=p.punctuate(text)
# print(t)
print(final)

l=[]
b=[]
for i in range(0,len(final)):
    blob=TextBlob(final[i],analyzer=NaiveBayesAnalyzer())
    #print(blob.sentiment)
    l.append(blob.sentiment.p_pos)
    b.append(blob.sentiment.p_neg)

pos=0
neg=0
neu=0
pos_per=0
neg_per=0
neu_per=0
for i in l:
    if i>0.6:
        pos=pos+1
        pos_per=pos_per+i
    elif i>0.4 and i<0.6:
        neu=neu+1
        neu_per=neu_per+i
    elif i<0.4:
        neg=neg+1
        neg_per=neg_per+i
# print(l)
# print(len(final))

print("Number of positive sentences in the passage:",pos)
print("Number of negative sentences in the passage:",neg)
print("Number of neutral sentences in the passage:",neu)

```



```

print("Overall positivity of the passage:", round(pos_per/sum(1),2))
print("Overall negativity of the passage:", round(neg_per/sum(1),2))
print("Overall neutrality of the passage:", round(neu_per/sum(1),2))

chart=[]
chart.append(round(pos_per/sum(1),2))
chart.append(round(neu_per/sum(1),2))
chart.append(round(neg_per/sum(1),2))

# plt.pie(chart)
mylabels = ["Positive", "Neutral", "Negative"]
mycolors = ["green" , "yellow", "red"]
plt.pie(chart, labels = mylabels, colors = mycolors)
my_circle=plt.Circle( (0,0), 0.7, color='white')
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.show()

```

Appendix D

```

#Dataset to csv

import dlib
import cv2
import numpy as np

print("Dlib version: {}".format(dlib.__version__))
print("OpenCV version: {}".format(cv2.__version__))

# initialize face and facial landmark detector
detector = dlib.get_frontal_face_detector()

# replace with proper path!!!!!!
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat
")

import os
import csv
import glob

Classes=['anger', 'contempt', 'disgust', 'fear', 'happy', 'neutral', 'sad', 's
urprise']

x=0

for category in Classes:
    path = glob.glob(f"train/{category}/*.jpg")

```

```

for img in path:

    img_array=cv2.imread(img)
    img_gray = cv2.cvtColor(img_array, cv2.COLOR_RGB2GRAY)
    # plt.imshow(img_gray)
    # plt.show()

    #detect faces in image
    faces = detector(img_gray, 0)
    #print(len(faces),faces)
    if len(faces)!=0:
        detected_face = img_array

        for f in faces:
            # draw bounding box
            detected_face = cv2.rectangle(detected_face,
                (f.tl_corner().x, f.tl_corner().y),
#top left corner of the d
                (f.br_corner().x, f.br_corner().y),
#bottom right corner of t
                (0,255,0),3)

            landmark_arr = np.array([])
            # detect facial landmarks in a box
            shape = predictor(img_gray, f)

            i=1
            x_scale = max(shape.parts()[33].x - shape.parts()[0].x,
shape.parts()[16].x - shape.parts()[33].x)
            y_scale = shape.parts()[8].y -shape.parts()[33].y

            for p in shape.parts():
                detected_face = cv2.circle(detected_face, (p.x,p.y),
2, (0,0,255), -1)

                p=p-shape.parts()[33]
                x_new = p.x / x_scale
                y_new = p.y / y_scale
                landmark_arr = np.append(landmark_arr,x_new)
                landmark_arr = np.append(landmark_arr,y_new)
                i+=1

            print(x)
            x+=1
            landmark_arr=np.append(arr,Classes.index(category))
            print(landmark_arr)

        with open('train4.csv', 'a+' , newline='') as write_obj:

```

```
csv_writer = csv.writer(write_obj)
csv_writer.writerow(landmark_arr)
```

Appendix E

```
#Train DNN
```

```
import tensorflow as tf
```

```
featureDim = 136
```

```
classes = 8
```

```
model = tf.keras.Sequential(layers = (tf.keras.layers.Dense(272, input_
shape=(featureDim,), activation='sigmoid'),
    tf.keras.layers.Dense(544, activation='sigmoid'),
    tf.keras.layers.Dense(272, activation='sigmoid'),
    tf.keras.layers.Dense(classes, activation='sigmoid'))
)
```

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_1
ogits=True),
```

```
    optimizer='adam',
```

```
    metrics=['accuracy'])
```

```
model.summary()
```

```
def createData(pathToData, featureDim = 136, classes = 8):
```

```
    f = open(pathToData, "r")
```

```
    x = []
```

```
    y = []
```

```
    for line in f:
```

```
        parse = line.split(',')
```

```
        item_x = [float(d) for d in parse[:featureDim]]
```

```
        x.append(item_x)
```

```
        label = parse[-1]
```

```
        label = label[:3]
```

```
        y.append(int(float(label)))
```

```
    #     print(x)
```

```
    #return tf.convert_to_tensor(x, dtype=tf.float32), tf.convert_to_t
nsor(y, dtype=tf.float32)
```

```
    return x, y
```

```
train_x, train_y = createData("C:/Users/Dell/Downloads/Lab 6/Emotion Re
cognition Using DNN/train4.csv",
```

```
    featureDim = featureDim,
```

```
    classes = classes
```

```
)
```

```

print(len(train_x))

# import pandas as pd
# data = pd.read_csv("train1.csv")
# print(data.head())

#fit dataset
model.fit(x = train_x, y = train_y, batch_size = 64, shuffle = True, epochs = 1000)

#save model
path_save = "./testsave4"

tf.keras.models.save_model(
model,path_save, overwrite=True, include_optimizer=True, save_format=None , signatures=None, options=None)

#restore saved model
model_restore = tf.keras.models.load_model(
path_save)

model_restore.summary()

# load train dataset
test_x, test_y = createData("C:/Users/Dell/Downloads/Lab 6/Emotion Recognition Using DNN/test4.csv",
                             featureDim = featureDim,
                             classes = classes
                             )

#evaluate test accuracy
model.evaluate(test_x,test_y)

#plot confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

confusion_matrix = confusion_matrix(test_y , result)

plt.figure()
plt.imshow(confusion_matrix, interpolation='nearest', cmap=plt.cm.Blues
)

thresh = confusion_matrix.max() / 2.

```

```

for i in range(confusion_matrix.shape[0]):
    for j in range(confusion_matrix.shape[1]):
        plt.text(j, i, format(confusion_matrix[i, j]),
                 ha="center", va="center",
                 color="white" if confusion_matrix[i, j] == 0 or confusion_matrix[i, j] > thresh else "black")
plt.tight_layout()
plt.colorbar()

```

Appendix F

#Detect angle code:

```

import cv2
import numpy as np
import dlib
import time
import math

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
POINTS_NUM_LANDMARK = 68

# Get the biggest face
def _largest_face(dets):
    if len(dets) == 1:
        return 0

    face_areas = [ (det.right()-det.left())*(det.bottom()-det.top()) for det in dets]

    largest_area = face_areas[0]
    largest_index = 0
    for index in range(1, len(dets)):
        if face_areas[index] > largest_area :
            largest_index = index
            largest_area = face_areas[index]

    print("largest_face index is {} in {} faces".format(largest_index, len(dets)))

    return largest_index

# Extract the point coordinates needed for pose estimation from the detection results of dlib
def get_image_points_from_landmark_shape(landmark_shape):
    if landmark_shape.num_parts != POINTS_NUM_LANDMARK:

```

```

        print("ERROR:landmark_shape.num_parts-
{}".format(landmark_shape.num_parts))
        return -1, None

    #2D image points. If you change the image, you need to change vector
    image_points = np.array([
        (landmark_shape.part(30).x, landmark_shape.part(30).y),    # Nose tip
        (landmark_shape.part(8).x, landmark_shape.part(8).y),    # Chin
        (landmark_shape.part(36).x, landmark_shape.part(36).y),  # Left eye left corner
        (landmark_shape.part(45).x, landmark_shape.part(45).y),  # Right eye right corner
        (landmark_shape.part(48).x, landmark_shape.part(48).y),  # Left Mouth corner
        (landmark_shape.part(54).x, landmark_shape.part(54).y)   # Right mouth corner
    ], dtype="double")

    return 0, image_points

    # Use dlib to detect key points and return the coordinates of several
    points needed for pose estimation
    def get_image_points(img):

        #gray = cv2.cvtColor( img, cv2.COLOR_BGR2GRAY) # The picture is
        #adjusted to gray
        dets = detector( img, 0 )

        for f in dets:
            shape = predictor(img, f)
            q=0
            for f in dets:
                img = cv2.rectangle(img, (f.tl_corner().x, f.tl_corner().y), (f.br_corner().x, f.br_corner().y), (0,255,0), 3)

            q+=1

        if 0 == len( dets ):
            print( "ERROR: found no face" )
            return -1, None
        largest_index = _largest_face(dets)
        face_rectangle = dets[largest_index]

        landmark_shape = predictor(img, face_rectangle)

```

```

    return get_image_points_from_landmark_shape(landmark_shape)

# Get rotation vector and translation vector
def get_pose_estimation(img_size, image_points ):
    # 3D model points.
    model_points = np.array([
        (0.0, 0.0, 0.0),          # Nose tip
        (0.0, -330.0, -65.0),    # Chin
        (-225.0, 170.0, -
135.0),      # Left eye left corner
        (225.0, 170.0, -
135.0),      # Right eye right corne
        (-150.0, -150.0, -
125.0),     # Left Mouth corner
        (150.0, -150.0, -
125.0)     # Right mouth corner

    ])

    # Camera internals

    focal_length = img_size[1]
    center = (img_size[1]/2, img_size[0]/2)
    camera_matrix = np.array(
        [[focal_length, 0, center[0]],
         [0, focal_length, center[1]],
         [0, 0, 1]], dtype = "double"
    )

    print("Camera Matrix :{}".format(camera_matrix))

    dist_coeffs = np.zeros((4,1)) # Assuming no lens distortion
    (success, rotation_vector, translation_vector) = cv2.solvePnP(model
_points, image_points, camera_matrix, dist_coeffs, flags=cv2.SOLVEPNP_I
TERATIVE )

    print("Rotation Vector:\n {}".format(rotation_vector))
    print("Translation Vector:\n {}".format(translation_vector))
    return success, rotation_vector, translation_vector, camera_matrix,
    dist_coeffs

# Convert from rotation vector to Euler angle
def get_euler_angle(rotation_vector):
    # calculate rotation angles
    theta = cv2.norm(rotation_vector, cv2.NORM_L2)

    # transformed to quaterniond

```

```

w = math.cos(theta / 2)
x = math.sin(theta / 2)*rotation_vector[0][0] / theta
y = math.sin(theta / 2)*rotation_vector[1][0] / theta
z = math.sin(theta / 2)*rotation_vector[2][0] / theta

ysqr = y * y
# pitch (x-axis rotation)
t0 = 2.0 * (w * x + y * z)
t1 = 1.0 - 2.0 * (x * x + ysqr)
print('t0:{}, t1:{}'.format(t0, t1))
pitch = math.atan2(t0, t1)

# yaw (y-axis rotation)
t2 = 2.0 * (w * y - z * x)
if t2 > 1.0:
    t2 = 1.0
if t2 < -1.0:
    t2 = -1.0
yaw = math.asin(t2)

# roll (z-axis rotation)
t3 = 2.0 * (w * z + x * y)
t4 = 1.0 - 2.0 * (ysqr + z * z)
roll = math.atan2(t3, t4)

print('pitch:{}, yaw:{}, roll:{}'.format(pitch, yaw, roll))

# Unit conversion: convert radians to degrees
Y = int((pitch/math.pi)*180)
X = int((yaw/math.pi)*180)
Z = int((roll/math.pi)*180)

return 0, Y, X, Z

def get_pose_estimation_in_euler_angle(landmark_shape, im_szie):
    try:
        ret, image_points = get_image_points_from_landmark_shape(landmark_shape)
        if ret != 0:
            print('get_image_points failed')
            return -1, None, None, None

        ret, rotation_vector, translation_vector, camera_matrix, dist_coeffs = get_pose_estimation(im_szie, image_points)
        if ret != True:
            print('get_pose_estimation failed')
            return -1, None, None, None

```



```

ret, pitch, yaw, roll = get_euler_angle(rotation_vector)
if ret != 0:
    print('get_euler_angle failed')
    return -1, None, None, None

euler_angle_str = 'Y: {}, X: {}, Z: {}'.format(pitch, yaw, roll)
print(euler_angle_str)
return 0, pitch, yaw, roll

except Exception as e:
    print('get_pose_estimation_in_euler_angle exception: {}'.format(
e))

    return -1, None, None, None

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FPS, 10)
fourcc = cv2.VideoWriter_fourcc(*'XVID')
output_video = cv2.VideoWriter('output.mp4', fourcc, 10.0, (640, 480))
while (cap.isOpened()):
    start_time = time.time()

    # Read Image
    ret, im = cap.read()
    if ret != True:
        print('read frame failed')
        continue
    size = im.shape
    if size[0] > 700:
        h = size[0] / 3
        w = size[1] / 3
        im = cv2.resize( im, (int( w ), int( h )), interpolation=cv2.IN
TER_CUBIC )
        size = im.shape

    ret, image_points = get_image_points(im)
    if ret != 0:
        print('get_image_points failed')
        continue

    ret, rotation_vector, translation_vector, camera_matrix, dist_coeff
s = get_pose_estimation(size, image_points)
    if ret != True:
        print('get_pose_estimation failed')
        continue
    used_time = time.time() - start_time
    print("used_time: {} sec".format(round(used_time, 3)))

```

```

ret, pitch, yaw, roll = get_euler_angle(rotation_vector)
euler_angle_str = 'Y:{}, X:{}, Z:{}'.format(pitch, yaw, roll)
print(euler_angle_str)

# Project a 3D point (0, 0, 1000.0) onto the image plane.
# We use this to draw a line sticking out of the nose

(nose_end_point2D, jacobian) = cv2.projectPoints(np.array([(0.0, 0.0, 1000.0)]), rotation_vector, translation_vector, camera_matrix, dist_coeffs)

for p in image_points:
    cv2.circle(im, (int(p[0]), int(p[1])), 3, (0,0,255), -1)

p1 = ( int(image_points[0][0]), int(image_points[0][1]))
p2 = ( int(nose_end_point2D[0][0][0]), int(nose_end_point2D[0][0][1]))

cv2.line(im, p1, p2, (255,0,0), 2)

# Display image
#cv2.putText( im, str(rotation_vector), (0, 100), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1 )
cv2.putText( im, euler_angle_str, (0, 120), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1 )
cv2.imshow("Output", im)
output_video.write(im)
if cv2.waitKey(1) & 0xFF == ord('s'):
    break

output_video.release()
cap.release()
cv2.waitKey(0)
cv2.destroyAllWindows()

```

ENGR 498-07 Research in Artificial Intelligence and Deep Learning

Artificial Intelligence System for Emotion Recognition and Text Analytics

Team Members:

Reshu Agarwal
Namrata Chaudhari
Meghna Narwade

Advisor: Dr. Jafar Saniie

Summer 2021

Artificial Intelligence System for Emotion Recognition and Text Analytics

ABSTRACT :

Companies around the world are trying to harness the power of emotional intelligence to improve their business processes. Emotional analysis can help to gain an accurate understanding of customer response which can be used to improve an existing process, seize new opportunities, and reduce costs in any business facing customers. In this project, we propose an artificial intelligence based stand alone system which will allow us to classify and analyse facial expression in real time and perform sentiment analysis by examining the body of the text (extracted from audio) to understand the opinion expressed by it. This helps us provide a deeper understanding of how customers really feel at a given time. The proposed system uses a deep neural network (DNN) for classifying 8 basic emotions based on features extracted from facial expression and uses pretrained sentiment analysis tools to quantify text (extracted from audio) based on polarity.

INTRODUCTION :

The aim of this project is to build a stand alone system capable of classifying emotions from real time video and categorizing the text extracted from audio as positive, negative or neutral. This can be used by users to analyze and improve their behavioral skills and maintain a good conversation tone. It can be used by companies in the market research industry by employing behavioral methods that observe user's reaction while interacting with a brand or product along with the traditionally used review analysis. The proposed system extracts the audio and visual cues from real time audio and video respectively, and uses these extracted cues to perform facial expression recognition and text sentiment analysis. The facial expression recognition pipeline classifies emotions from the detected faces in the frame (of the video) using a deep neural network by extracting vectorized landmarks features from the detected faces. The text sentiment analysis pipeline uses pretrained sentiment analysis tools provided in various Pythonic NLP libraries.

RELATED WORKS:

Effective communication involves two components: Verbal cues and Non verbal cues. The proposed system covers the verbal aspect of communication by performing text sentiment analysis and non-verbal aspect of communication by analysing facial expressions.

Facial emotion detection system:

In recent years, advances in facial expression detection have accelerated, and more and more experts have been involved in the development of emotion recognition. The research of expression recognition in computer vision focuses on the feature extraction and feature classification. Feature extraction refers to extracting landmarks from faces that can be used for classification from input pictures or video streams. There are multiple methods for feature extraction from detected faces. The facial expression classification refers to the use of specific algorithms to identify the categories of facial expressions according to the extracted features. Commonly used methods of facial expression classification are Hidden Markov Model (HMM), Support Vector Machine (SVM), AdaBoost, and Artificial Neural Networks (ANN).

Techniques for facial emotion detection using landmark extraction :

Research Paper	Number of landmarks	Method of landmark detection	Dataset used	Classifier used	Accuracy
<u>Real time emotion recognition system using facial expression and EEG</u>	10	Manually placed through optical flow algorithm	Own database	CNN	93.02%
<u>Real time facial expression recognition in Video</u>	22	Manually placed using feature displacement approach	CK+ database	SVM	86.0%
<u>Real-time Mobile Facial Expression Recognition System</u>	77	Extracted using STASM library	CK+ database	SVM	85.8%

<u>A fuzzy logic approach for real time facial recognition of facial emotions</u>	68	Extracted using DLIB library	CK+ database	FURIA	83.2%
Our approach: Response sentiment analysis system.	68	Extracted using DLIB library	Images from CK+ database, JAFFE database, TFEID database, RaFD database	DNN	86.75%

Text Sentiment Analysis:

In the proposed system, text sentiment analysis is performed on the extracted real time audio which is converted to text. Speech to text conversion can be done using various available API's and python libraries.

The most popular speech to text conversion APIs include Google Cloud Speech, IBM and Rev.ai

Link	Result
<u>A Benchmarking of IBM, Google and Wit Automatic Speech Recognition Systems</u>	This research paper differentiates among IBM, Google cloud speech, & Wit. Result: Google Cloud Speech dominates
<u>Which Automatic Transcription Service is the Most Accurate?</u>	Differentiating among various speech to text APIs available Result: 1st Google cloud speech & 2nd Temi by Rev.ai
<u>How Reliable is Speech-to-Text in 2021?</u>	An article that differentiates among different speech to text APIs. Result: 1st Temi by Rev.ai & 2nd Google cloud speech

Sentiment analysis (opinion mining) is a text mining technique that uses machine learning and natural language processing (nlp) to automatically analyze text for the sentiment of the speaker (positive, negative, neutral). Text Sentiment analysis is normally implemented using 2 approaches:

1. Constructing supervised machine learning and deep learning models. Text sentiment can be classified using machine learning models like Support Vector Machine (SVM), Naive Bayes and Decision Tree.
2. Using unsupervised lexicon based approaches. Determining polarity of text using pretrained sentiment analysis tools from various Python NLP libraries (TextBlob, Vader)

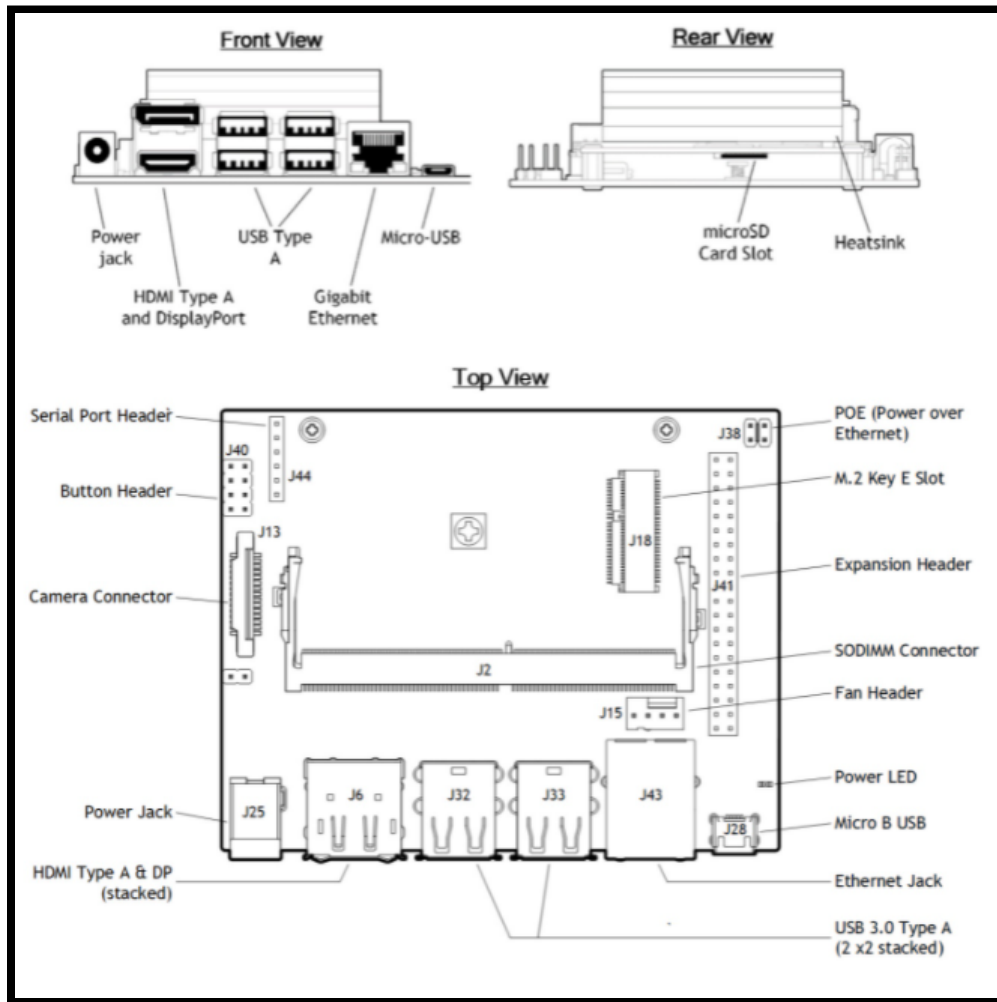
We have used an unsupervised lexicon based approach to implement text sentiment analysis.

SYSTEM COMPONENTS:

The proposed system can be implemented using a laptop PC. In addition we have used Nvidia's Jetson nano as a hardware component. Jetson nano is a compact, low voltage System on Chip (SoC) designed to carry out programmed instructions. It provides Maxwell 128 core GPU emphasized on Deep Learning in its hardware design and software libraries. It is capable of running multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing. The Jeston nano is powered using a 5W 4A power supply. The camera used is Raspberry Pi MIPI CSI which has a frame rate of about 90 fps.

The programming language used to code the system is python. Python is an open source language and has extensive support libraries which allow us to perform video processing, speech recognition and natural language processing (NLP) tasks.

Jetson Nano Specifications



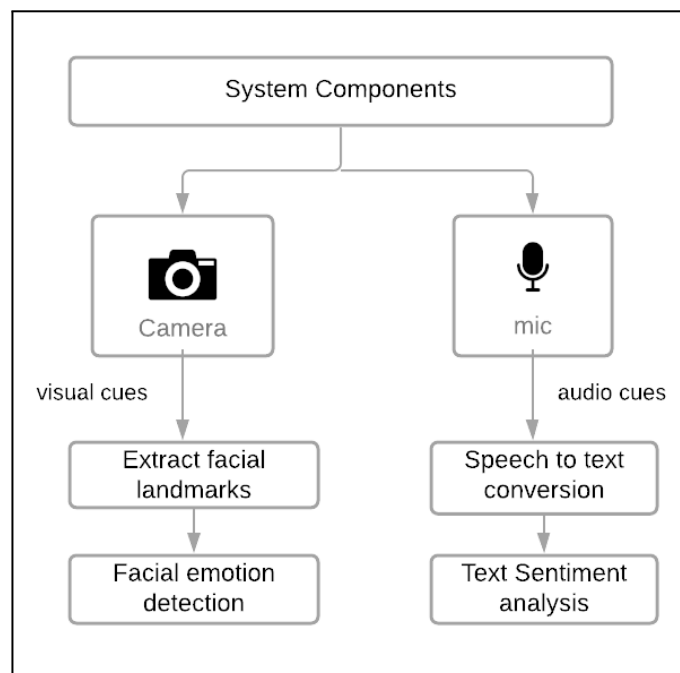
GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 4x 1080p @ 30 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 2x 4K @ 30 8x 1080p @ 30 18x 720p @ 30 (H.264/H.265)
Camera	1x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI 2.0 and eDP 1.4
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I ² C, I ² S, SPI, UART
Mechanical	100 mm x 80 mm x 29 mm

Python libraries used

Library	Use
OpenCV	Video Processing
Dlib	Face detection and landmark extraction
Tensorflow	Build and train Deep Neural Network
Pyaudio	To record audio
Speech Recognition	Speech to text conversion
Punctuator	Add punctuations to text
TextBlob	Simple API to perform basic NLP tasks

SYSTEM OVERVIEW :

The proposed system uses the camera to extract visual cues which are used to perform facial expression recognition and uses the mic to extract audio cues which are converted to text and used to perform text sentiment analysis.



We need to extract the generated audio and visual cues simultaneously from a real time scenario. This is being done using multi-threading which helps us to run multiple function calls simultaneously i.e. one thread records the video using opencv and the other thread records the audio using pyaudio and the output of each of these threads will then be served as an input to the two modules implemented which will then predict emotions and analyze the polarity of the content obtained from the audio.

The frame rate for the multithreading process is calculated by: dividing the total number of frames with the elapsed time of the program & the fps recorded was about 4-5fps.

FACIAL EMOTION DETECTION SYSTEM :

The facial emotion detection module is built from scratch to detect one of eight emotions: happiness, sadness, anger, surprise, fear, disgust and contempt, The visual cues are used to detect faces and extract 68 landmarks (features) which are then fed to the deep neural network (DNN) to classify emotion from the given frame.

Facial landmark Extraction :

Convolutional neural networks can be used to classify raw input images but performing feature landmark extraction allows us to achieve comparable results with a simpler neural network.

Facial landmark extraction is performed using the Dlib library in python. The extracted features are then fed as an input to the neural network. The Dlib library detects faces from the input image and uses the predictor function to place 68 landmarks on the detected faces. It uses Histogram of Oriented Gradients (HOG) for Object Detection with a linear classifier, an image pyramid, and sliding window detection scheme to detect faces in an image. Once the region of face is determined, facial landmarks will be detected using One Millisecond Face Alignment with an Ensemble of Regression Trees. The Dlib library accurately detects landmarks from the detected faces at an angle of -25 to +25 degrees in any direction. *(Code for checking angle: Appendix F)*

The coordinates of the 68 landmarks have a fixed orientation (shown in the figure below). The resultant landmarks are given in the form of an array.

Resultant array : = [(x0,y0) , (x1,y1), , (x67,y67)]



Extracting features from faces allows us to construct a simple neural network with less training data which will converge faster as compared to traditional CNNs.

Neural Networks perform best when the feature vector is scaled to a small range of values $[-1, 1]$. In order to optimize the gradient descent process normalize the facial landmarks and align them at the tip of the nose (x_{33}, y_{33}) . Vectorization of facial landmarks is achieved by putting tensors of 2-dimensional coordinates into a vector which is fed into the neural network.

Shifting the origin to the tip of the nose (x_{33}, y_{33}) :

For (x, y) in resultant array:

$$x = x - x_{33}$$

$$y = y - y_{33}$$

Normalizing the coordinates in range $[-1, 1]$:

scale height = y_8 // coordinate $(x_8, y_8) := (*, -1)$

scale width = $\max(|x_0|, |x_{16}|)$

For (x, y) in resultant array:

$$x = x / \text{scale width}$$

$$y = y / \text{scale height}$$

The normalized coordinates are stored in the form of a feature vector.

$[(x_0, y_0), (x_1, y_1), \dots, (x_{67}, y_{67})] \rightarrow [x_0, y_0, x_1, y_1, \dots, x_{67}, y_{67}]$

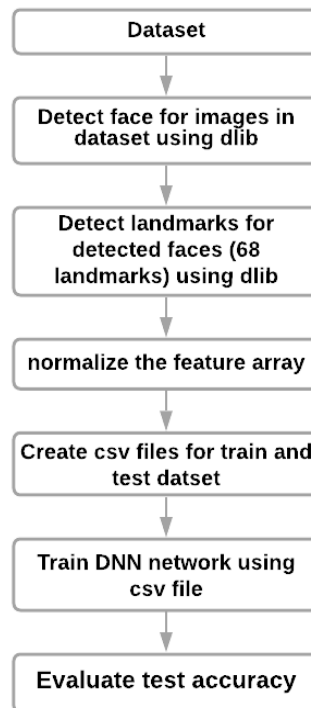
The result data can be stored in a CSV file with an integer indicating the emotion in the last column (label L) which can be used to train and test the neural network.

Building a Deep Neural Network (DNN):

The dataset was created using images from CK+ (Extended Cohn-Kanade dataset), JAFFE dataset, TFEID (Taiwanese Facial Expression Image Database), and RaFD(Radboud Faces Database) . The created dataset is composed of eight classes with a total of 3000 images divided into training and test sets. The vectorized facial landmarks of images from the dataset are stored in a CSV file along with an integer indicating the emotion. The test and train csv files are then used to train and evaluate the DNN.

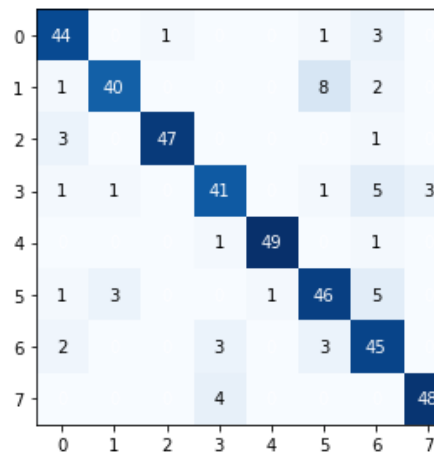
The model used in building the deep neural network is a sequential model with three hidden layers. The type of layers used is dense which implies that every neuron in the dense layer receives input from all neurons of the previous layer. The activation function used was a sigmoid. Adam optimizer allows the framework to adjust the step size depending on the loss. Accuracy obtained after testing the model: 86.75%

Implementation Flowchart



Model Summary:

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 272)	37264
dense_1 (Dense)	(None, 544)	148512
dense_2 (Dense)	(None, 272)	148240
dense_3 (Dense)	(None, 8)	2184
Total params: 336,200		
Trainable params: 336,200		
Non-trainable params: 0		

Confusion Matrix for the test set classification:

- 0: angry
- 1: contempt
- 2: disgust
- 3: fear
- 4: happiness
- 5: neutral
- 6: sadness
- 7: surprise

Real time facial emotion Detection

The system uses OpenCV, to read video frames either by using the feed from a camera connected to a computer or by reading a video file. We then perform face

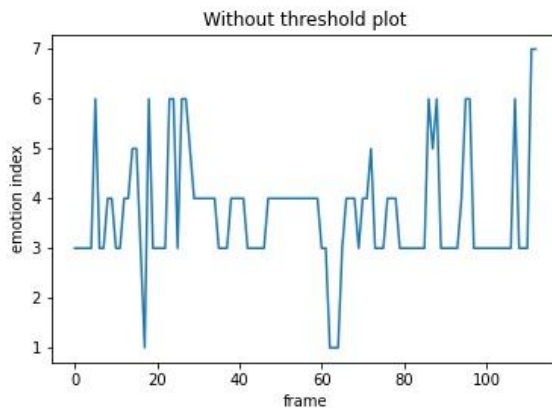
detection and facial landmark extraction on the frame and feed the normalized landmark coordinates into the DNN which classifies the emotion of the faces in the frame.

Since we use the sigmoid activation function in the neural network, the output of the DNN is an array in which each element represents the probability of (indexed) emotion occurring independent of other emotions. The sum of the array elements may not necessarily be 1 as sigmoid function doesn't treat emotions to be mutually exclusive.

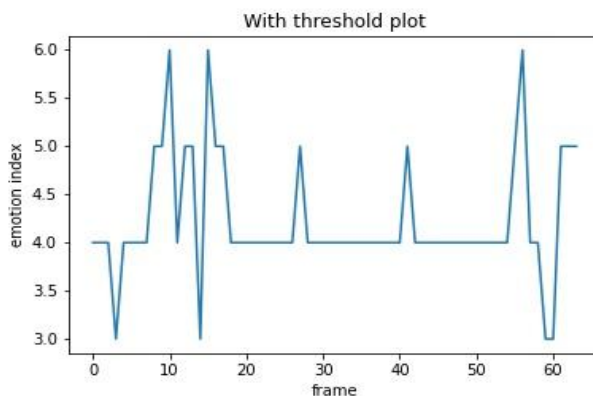
This allows us to improve the accuracy of our system while performing real time processing by setting a threshold for the level of confidence for each of the eight emotions. We only display the emotion if the confidence level of that emotion is greater than its threshold value. If the emotion detected does not cross the threshold value we display the emotion rendered in the previous frame.

The facial emotion detection of a video performed with and without threshold is shown below.

Without threshold:



With Threshold:



The frame rate achieved for real time face emotion detection is about 8.9 fps for laptop PC and 4.1 on Jetson Nano.

On Laptop :

```
fps start
fps stop

[INFO] elapsed time: 12.74
[INFO] approx. FPS: 8.95
```

On Jetson Nano :

```
fps stop

fps recorded on Jetson nano
[INFO] elapsed time: 27.20
[INFO] approx. FPS: 4.19
```

TEXT SENTIMENT ANALYSIS SYSTEM :

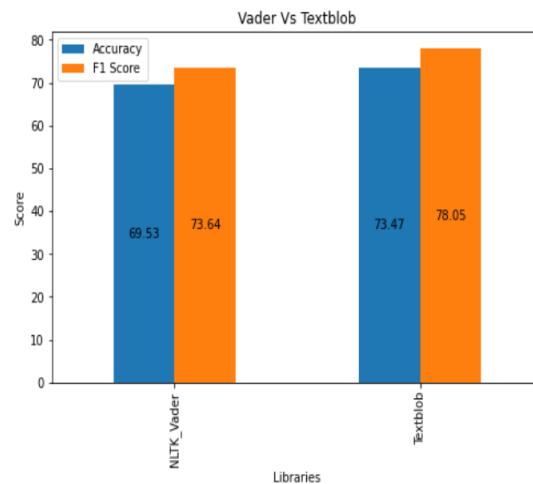
The system converts real time audio to text using the Speech Recognition library in python. We use the Pyaudio library to record audio from a mic. The recorded audio is broken down into chunks and processed bit by bit using the Recognizer function in the Speech recognition library which transcribes the audio. The transcribed audio is split into sentences before using the Punctuation Model adding the required punctuation to the text. This text is then used to perform text sentiment analysis .

The proposed system determines the polarity of text using pretrained sentiment analysis tools from various Python NLP libraries (TextBlob, Vader). The most widely used pretrained libraries for estimating polarity of text are TextBlob and Vader.

The following are some negative and positive interviewee responses to check how well these libraries can classify their polarity and overall we find TextBlob with Naive Bayes yields more satisfying results. The numbers shown in the table are the polarity of each sentence where -100 means negative and +100 means positive.

	content	textblob	textblob_bayes	nlTK_vader
0	I've enjoyed and grown in my current role	25	65	51
1	I am an ambitious and driven individual. I thrive in a goal-oriented environment	12	92	48
2	What makes me unique is my ability to meet and exceed deadlines	38	59	32
3	While I highly valued my time at my previous company, there are no longer opportunities for growth that align with my career goals	0	3	73
4	I hated the job and the company. They were awful to work for.	-95	-60	-80
5	I do good work	70	4	44
6	I tend to lose my patience with incompetent people.	-35	-33	-70
7	I missed too much work.	20	-10	-30

The accuracy of Textblob vs Vader was compared by testing these models on the IMDB dataset and the product review dataset. It can be seen that TextBlob has higher precision and F1 score for these datasets



The proposed system uses the TextBlob library with Naive Bayes Classifier to estimate the polarity of the text. TextBlob is a python library of Natural Language Processing (NLP) that uses the Natural Language ToolKit (NLTK) to perform its functions. NLTK is a library that provides easy access to many lexical resources and allows users to work with categorization, classification and many other tasks. It calculates average polarity and subjectivity over each word in a given text using a dictionary of adjectives and their hand-tagged scores. It actually uses a pattern library for that, which takes the individual word scores from sentiwordnet. The TextBlob with Naive Bayes calculates the sentiment score by NaiveBayesAnalyzer trained on a dataset of movie reviews. We use the polarity calculated by TextBlob to classify text as either positive, negative or neutral by

setting a threshold value. The polarity value lies in the range of $[-1, 1]$, where -1 indicates negativity and $+1$ indicates positivity.

Threshold values set to classify text into three classes:

Polarity above 60% is classified as Positive

Polarity between 40% and 60% is classified as Neutral

Polarity below 40% is classified as Negative

Analysis of a transcribed text passage is done as follows:

Number of positive sentences in the passage: x

Number of negative sentences in the passage: y

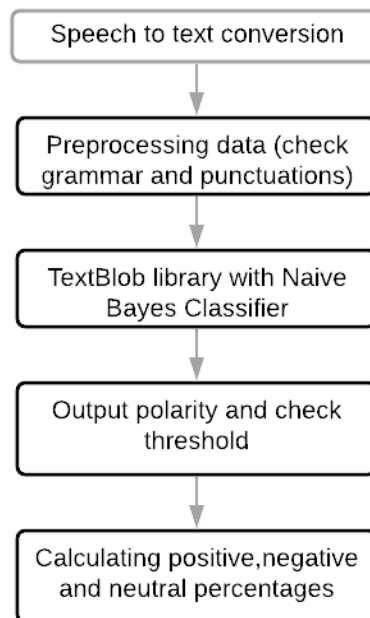
Number of neutral sentences in the passage: z

Total number of sentences in a passage: $x+y+z$

Overall positivity of the passage: Sum of polarities above 60% / Total number of sentences in a passage

Overall neutrality of the passage: Sum of polarities between 40% - 60% / Total number of sentences in a passage

Overall negativity of the passage: Sum of polarities below 40% / Total number of sentences in a passage



Text Sentiment Analysis Workflow

RESULTS AND DISCUSSION :

The integrated system extracts video and audio simultaneously with a frame rate of 4-5 fps. The facial emotion detection system successfully detects facial expression of faces detected in real time video with an accuracy of about 86.75%. The audio from the video is successfully extracted, converted to text, cleaned and processed to determine if the attitude of the speaker in a given situation is positive, negative or neutral.

The proposed system can be used in a wide sale of applications. It can be used to make the interview process bias free by analyzing the emotional expressions and answers of prospective candidates for its entry-level jobs. Candidates can also use this system analysing their own responses during a mock interview.It can be used to perform market research by analysing customers' response to a particular advertising scheme. If customized this system can be used for the interrogation process.

The results and applications are used in the video attached.

<https://drive.google.com/file/d/1wnGr-dIYQGUqjDZS2CVY2-WS850fUCvO/view?usp=sharing>

CONCLUSION :

The project is research on face expression recognition and analysing text for the sentiment , which allows us to know a way of sensing emotions that can be considered as mostly used AI and pattern analysis applications. To summarize, we have developed a system that can perform emotion detection and text sentiment analysis in real time.

FUTURE WORK :

The system can be further improved by covering more aspects of communication skills like using the extracted audio from video to perform speech emotion detection to recognize the emotional aspects of speech irrespective of the semantic contents. The accuracy of the facial emotion detection and text sentiment analysis system can be further improved to make the system more feasible and accurate.

REFERENCES :

[1] Dlib Library python:

<https://pypi.org/project/dlib/>

[2] Textblob Library python:

<https://pypi.org/project/textblob/>

[3] OpenCV:

<https://pypi.org/project/opencv-python/>

[4]Speech Recognition library python:

<https://pypi.org/project/SpeechRecognition/>

[5] Recording Audio and Video together code:

<https://stackoverflow.com/questions/14140495/how-to-capture-a-video-and-audio-in-python-from-a-camera-or-webcam>

[6] Facial emotion recognition dataset images:

<https://github.com/spenceryee/CS229>

[7] Angle detection for landmarks:

<https://www.programmingsought.com/article/27703847966/>

[8] Related works in facial emotion detection:

- [Development of a Real-Time Emotion Recognition System Using Facial Expressions and EEG based on machine learning and deep neural network methods](#)
- [Real time facial expression recognition in video using support vector machines](#)
- [Real-time Mobile Facial Expression Recognition System](#)
- [A fuzzy logic approach for real time facial recognition of facial emotions](#)

CODE:

Appendix A: Extracting audio and visual cues

```
#AudioVideo recording code

import cv2
import pyaudio
import wave
import threading
import time
import subprocess
import os

class VideoRecorder():

    # Video class based on openCV
    def __init__(self):

        self.fourcc = "MJPG"          # capture images (with no dec
rease in speed over time; testing is required)
        self.dim = (640,480)         # video formats and sizes als
o depend and vary according to the camera used
        self.video_filename = "Fer.avi"
        self.fps = 6
        self.cap = cv2.VideoCapture(0)
        self.open = True
        self.write = cv2.VideoWriter_fourcc(*self.fourcc)
        self.vid = cv2.VideoWriter(self.video_filename, self.wri
te, self.fps, self.dim)

        self.frame_counts = 1
            # fps should be the minimum constant rate at
which the camera can

        self.start_time = time.time()

    # Video starts being recorded
    def record(self):
        counter = 1
        timer_start = time.time()
        timer_current = 0
```

```

        while (self.open==True):
            ret, frame = self.cap.read()

            if ret:
                self.vid.write(frame)
                print(str(counter) + " " + str(self.count)
+ " frames written " + str(timer_current))
                self.frame_counts += 1
                counter += 1
                timer_current = time.time() - timer_start
                time.sleep(0.16)
            #
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2G
RAY)

            cv2.imshow('frame', frame)
            cv2.waitKey(1)

# Finishes the video recording therefore the thread too
def stop(self):

    if self.open==True:
        self.open=False
        self.vid.release()
        self.cap.release()
        cv2.destroyAllWindows()
    else:
        pass

# Launches the video recording function using a thread

def start(self):
    t1 = threading.Thread(target=self.record)
    t1.start()

class AudioRecorder():

# Audio class based on pyAudio and Wave
def __init__(self):

    self.open = True
    self.rate = 44100
    self.frames_per_buffer = 1024
    self.channels = 2

```

```

self.format = pyaudio.paInt16
self.audio_filename = "video 1.wav"
self.audio = pyaudio.PyAudio()
self.stream = self.audio.open(format=self.format,
                               channels=self.channels,
                               rate=self.rate,
                               input=True,
                               frames_per_buffer = self.f
rames_per_buffer)
    self.audio_frames = []

# Audio starts being recorded
def record(self):

    self.stream.start_stream()
    while (self.open == True):
        data = self.stream.read(self.frames_per_buffer)
        self.audio_frames.append(data)
        if self.open==False:
            break

# Finishes the audio recording therefore the thread too
def stop(self):

    if self.open==True:
        self.open = False
        self.stream.stop_stream()
        self.stream.close()
        self.audio.terminate()

        aud = wave.open(self.audio_filename, 'wb')
        aud.setnchannels(self.channels)
        aud.setsampwidth(self.audio.get_sample_size(self.for
mat))

        aud.setframerate(self.rate)
        aud.writeframes(b''.join(self.audio_frames))
        aud.close()

    pass

# Launches the audio recording function using a thread
def start(self):
    t2 = threading.Thread(target=self.record)
    t2.start()

```

```
def start_AVrecording(filename):

    global t1
    global t2

    t1 = VideoRecorder()
    t2 = AudioRecorder()

    t2.start()
    t1.start()

    return filename

def start_video_recording(filename):

    global t1

    t1 = VideoRecorder()
    t1.start()

    return filename

def start_audio_recording(filename):

    global t2

    t2 = AudioRecorder()
    t2.start()

    return filename

def stop_AVrecording(filename):

    t2.stop()
    frame_counts = t1.frame_counts
    elapsed_time = time.time() - t1.start_time
    recorded_fps = frame_counts / elapsed_time
    print("total frames " + str(frame_counts))
    print("elapsed time " + str(elapsed_time))
```

```

print("recorded fps " + str(recorded_fps))
t1.stop()

# Makes sure the threads have finished
while threading.active_count() > 1:
    time.sleep(1)

# Required and wanted processing of final files
def file_manager(filename):

    local_path = os.getcwd()

    if os.path.exists(str(local_path) + "/temp_audio.wav"):
        os.remove(str(local_path) + "/temp_audio.wav")

    if os.path.exists(str(local_path) + "/temp_video.avi"):
        os.remove(str(local_path) + "/temp_video.avi")

    if os.path.exists(str(local_path) + "/temp_video2.avi"):
        os.remove(str(local_path) + "/temp_video2.avi")

    if os.path.exists(str(local_path) + "/" + filename + ".avi"):
:
        os.remove(str(local_path) + "/" + filename + ".avi")

filename = "Default_user"
file_manager(filename)

start_AVrecording(filename)

time.sleep(20)

stop_AVrecording(filename)
print("Done")

```


Appendix B: Real time face emotion detection

```
#Face emotion detection:

import dlib
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# initialize face and facial landmark detector
detector = dlib.get_frontal_face_detector()

# replace with proper path!!!!!!
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

#loading DNN
path_save = "./testsave4"
model_restore = tf.keras.models.load_model(path_save)

model_restore.summary()

#text characteristics
window_name = 'Image'
font = cv2.FONT_HERSHEY_SIMPLEX
fontScale = 1
color = (0, 0, 255)
thickness = 2

#emotion detected dictionary
emotions = { 0:"angry" ,1:"contempt" ,2:"disgusted",3:"fearful",
  4:"happy", 5:"neutral",6:"sad",7:"surprised"}
print(emotions)

#normalize and add to array function
def normalize(detected_face,shape,new_arr):
    i=1
    arr = []
    x_scale =-1*(shape.parts()[0].x - shape.parts()[33].x)
    y_scale = shape.parts()[8].y -shape.parts()[33].y
    for p in shape.parts():
#         detected_face = cv2.circle(detected_face,(p.x,p.y), 2,
(0,0,255), -1)
        p=p-shape.parts()[33]
```

```
x_new = p.x / x_scale
y_new = p.y / y_scale
arr = np.append(arr,x_new)
arr = np.append(arr,y_new)
i+=1
return arr
```

```
#finding emotion from output
```

```
def result(test_result,emotion_result,index_result):
    for r in test_result:
        c = ""
        if r[0]>99:
            j=0
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[1]>0.99:
            j=1
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[2]>0.99:
            j=2
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[3]>0.99:
            j=3
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[4]>0.85:
            j=4
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[5]>0.90:
            j=5
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[6]>0.99:
            j=6
            index_result.append(j)
            emotion_result.append(emotions[j])
```

```

        c = c + emotions[j] + " "
    if r[7]>0.90:
        j=7
        index_result.append(j)
        emotion_result.append(emotions[j])
        c = c + emotions[j] + " "

    return emotion_result, index_result,c

from imutils.video import FPS
# vid = cv2.VideoCapture(0)
vid = cv2.VideoCapture('fer_video.mp4')

fps = FPS().start()

x = 0
analysis_arr = []
analysis_ind = []
prev_c = "unknown"
c=""
out = cv2.VideoWriter('output.mp4', -1, 20.0, (640,480))

while True:
    ret, frame = vid.read()
    print(x)
    if ret:
#         print(frame.shape)
        gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        faces = detector(gray, 0)
        detected_face = frame
        new_arr = []
#         print(faces)
        fps.update()

        for f in faces:
            shape = predictor(gray, f)
            pred = normalize(detected_face, shape, new_arr)
            new_arr.append(pred)
        q=0
        for f in faces:
            arr_x = np.reshape(new_arr[q], (1,136))
            index_result=[]
            emotion_result=[]

```

```

        test_result = model_restore.predict(arr_x)
#         print(test_result)
        emotion_result, index_result, c = result(test_result
,emotion_result,index_result)
        if c=="":
            c=prev_c
        if len(index_result)!=0:
            analysis_arr.append(emotion_result[0])
            analysis_ind.append(index_result[0])
            detected_face = cv2.rectangle(detected_face, (f.tl_c
orner().x, f.tl_corner().y),
                                          (f.br_corner().x, f.br_corner(
).y), (0,255,0), 3)
            frame = cv2.putText(frame, c, (f.tl_corner().x, f.t
l_corner().y), font,
                                fontStyle, color, thickness, cv2.LINE_AA)
            q+=1

#         cv2.imwrite(f"Frames/Frame{x}.jpg", frame)
        out.write(frame)
        cv2.imshow('frame', frame)
        prev_c = c
        x += 1
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

out.release()
vid.release()

fps.stop()
print(x)
print("fps start")
print("fps stop\n")
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
print("\n")

cv2.destroyAllWindows()

# print(analysis_ind)
# print(analysis_arr)

```

Appendix C: Text Sentiment Analysis

```
#text sentiment analysis

from textblob import TextBlob
from textblob.classifiers import NaiveBayesClassifier
from textblob.sentiments import NaiveBayesAnalyzer
import nltk
from pydub import AudioSegment
import speech_recognition as sr
from os import path
from nltk import tokenize

nltk.download('movie_reviews')
nltk.download('punkt')
nltk.download('stopwords')

#Converting mp4 to wav format with 128k bitrate
src="debatel.mp4"

AudioSegment.converter = "C:/ffmpeg-4.4-full_build/bin/ffmpeg.exe"
AudioSegment.ffmpeg = "C:/ffmpeg-4.4-full_build/bin/ffmpeg.exe"
AudioSegment.ffprobe = "C:/ffmpeg-4.4-full_build/bin/ffprobe.exe"

sound = AudioSegment.from_file(file=src, format="mp4")
sound.export("recording.mp3", format="mp3", bitrate="128k")

# convert mp3 file to wav

sound = AudioSegment.from_mp3("recording.mp3")
sound.export("transcript.wav", format="wav")

##Code-----
# importing libraries
import speech_recognition as sr
import os
from pydub import AudioSegment
from pydub.silence import split_on_silence

# create a speech recognition object
r = sr.Recognizer()
```

```

# a function that splits the audio file into chunks
# and applies speech recognition
def get_large_audio_transcription(path):
    """
    Splitting the large audio file into chunks
    and apply speech recognition on each of these chunks
    """
    # open the audio file using pydub
    sound = AudioSegment.from_wav(path)
    # split audio sound where silence is 700 miliseconds or more
    and get chunks
    chunks = split_on_silence(sound,
        # experiment with this value for your target audio file
        min_silence_len = 500,
        # adjust this per requirement
        silence_thresh = sound.dBFS-14,
        # keep the silence for 1 second, adjustable as well
        keep_silence=500,
    )
    folder_name = "audio-chunks"
    # create a directory to store the audio chunks
    if not os.path.isdir(folder_name):
        os.mkdir(folder_name)
    whole_text = ""
    # process each chunk
    for i, audio_chunk in enumerate(chunks, start=1):
        # export audio chunk and save it in
        # the `folder_name` directory.
        chunk_filename = os.path.join(folder_name, f"chunk{i}.wav")
        audio_chunk.export(chunk_filename, format="wav")
        # recognize the chunk
        with sr.AudioFile(chunk_filename) as source:
            audio_listened = r.record(source)
            # try converting it to text
            try:
                text = r.recognize_google(audio_listened)
            except sr.UnknownValueError as e:
                print("Error:", str(e))
            else:
                text = f"{text.capitalize()} "
                #print(chunk_filename, ":", text)
                whole_text += text
    # return the text for all chunks detected
    return whole_text

```

```

path = "transcript.wav"
#print("\nFull text:", get_large_audio_transcription(path))
t=get_large_audio_transcription(path)
print(t)

```

```

sentence_break=[]
sentence_break=t.split('.')
print(sentence_break)

```

```

from punctuator import Punctuator
p = Punctuator('punctuator_model/Demo-Europarl-EN.pcl')
semi_final=[]
final=[]
for ele in sentence_break:
    if len(ele)>1:
        test=p.punctuate(ele)
        semi_final=test.split('.')
        for i in semi_final:
            if i!="":
                final.append(i)
                #pre-trained model 1
# #p1=Punctuator('punctuator_model/INTERSPEECH-T-BRNN.pcl')
# pre-trained model 2
# t=p.punctuate(text)
# print(t)
print(final)

```

```

l=[]
b=[]
for i in range(0,len(final)):
    blob=TextBlob(final[i],analyzer=NaiveBayesAnalyzer())
    #print(blob.sentiment)
    l.append(blob.sentiment.p_pos)
    b.append(blob.sentiment.p_neg)

```

```

pos=0
neg=0
neu=0
pos_per=0
neg_per=0
neu_per=0
for i in l:
    if i>0.6:

```

```

        pos=pos+1
        pos_per=pos_per+i
    elif i>0.4 and i<0.6:
        neu=neu+1
        neu_per=neu_per+i
    elif i<0.4:
        neg=neg+1
        neg_per=neg_per+i
# print(l)
# print(len(final))

print("Number of positive sentences in the passage:",pos)
print("Number of negative sentences in the passage:",neg)
print("Number of neutral sentences in the passage:",neu)

print("Overall positivity of the passage:",round(pos_per/sum(l),
2))
print("Overall negativity of the passage:",round(neg_per/sum(l),
2))
print("Overall neutrality of the passage:",round(neu_per/sum(l),
2))

chart=[]
chart.append(round(pos_per/sum(l),2))
chart.append(round(neu_per/sum(l),2))
chart.append(round(neg_per/sum(l),2))

# plt.pie(chart)
mylabels = ["Positive", "Neutral", "Negative"]
mycolors = ["green", "yellow", "red"]
plt.pie(chart, labels = mylabels, colors = mycolors)
my_circle=plt.Circle( (0,0), 0.7, color='white')
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.show()

```


Appendix D: Training and Test csv files

```
#Dataset to csv

import dlib
import cv2
import numpy as np

print("Dlib version: {}".format(dlib.__version__))
print("OpenCV version: {}".format(cv2.__version__))

# initialize face and facial landmark detector
detector = dlib.get_frontal_face_detector()

# replace with proper path!!!!!!
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

import os
import csv
import glob

Classes=['anger', 'contempt', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

x=0

for category in Classes:
    path = glob.glob(f"train/{category}/*.jpg")

    for img in path:

        img_array=cv2.imread(img)
        img_gray = cv2.cvtColor(img_array, cv2.COLOR_RGB2GRAY)
        # plt.imshow(img_gray)
        # plt.show()

        #detect faces in image
        faces = detector(img_gray, 0)
        #print(len(faces),faces)
        if len(faces)!=0:
            detected_face = img_array

            for f in faces:
                # draw bounding box
```

```

        detected_face = cv2.rectangle(detected_face,
            (f.tl_corner().x, f.tl_corner().y),
#top left corner of the d
            (f.br_corner().x, f.br_corner().y),
#bottom right corner of t
            (0,255,0),3)

        landmark_arr = np.array([])
        # detect facial landmarks in a box
        shape = predictor(img_gray, f)

        i=1
        x_scale = max(shape.parts()[33].x - shape.parts(
) [0].x, shape.parts()[16].x - shape.parts()[33].x)
        y_scale = shape.parts()[8].y -shape.parts()[33].
y

        for p in shape.parts():
            detected_face = cv2.circle(detected_face, (p.
x,p.y), 2, (0,0,255), -1)
            p=p-shape.parts()[33]
            x_new = p.x / x_scale
            y_new = p.y / y_scale
            landmark_arr = np.append(landmark_arr,x_new)
            landmark_arr = np.append(landmark_arr,y_new)
            i+=1

        print(x)
        x+=1
        landmark_arr=np.append(arr,Classes.index(category))
        print(landmark_arr)

        with open('train4.csv', 'a+' , newline='') as write_
obj:

            csv_writer = csv.writer(write_obj)
            csv_writer.writerow(landmark_arr)

```

Appendix E: training the Deep Neural Network

```
#Train DNN

import tensorflow as tf

featureDim = 136
classes = 8

model = tf.keras.Sequential(layers = (tf.keras.layers.Dense(272,
    input_shape=(featureDim,), activation='sigmoid'),
    tf.keras.layers.Dense(544, activation='sigmoid'),
    tf.keras.layers.Dense(272, activation='sigmoid'),
    tf.keras.layers.Dense(classes, activation='sigmoid')))
)

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy
    (from_logits=True),
    optimizer='adam',
    metrics=['accuracy'])
model.summary()

def createData(pathToData, featureDim = 136, classes = 8):
    f = open(pathToData, "r")
    x = []
    y = []
    for line in f:
        parse = line.split(',')
        item_x = [float(d) for d in parse[:featureDim]]
        x.append(item_x)
        label = parse[-1]
        label = label[:3]
        y.append(int(float(label)))
    #     print(x)
    #return tf.convert_to_tensor(x, dtype=tf.float32), tf.conver
t_to_tensor(y, dtype=tf.float32)
    return x, y

train_x, train_y = createData("C:/Users/Namrata
Chaudhari/Downloads/Lab 6/Emotion Recognition Using DNN/train4.c
sv",

                                featureDim = featureDim,
                                classes = classes
```

```

)

print(len(train_x))

# import pandas as pd
# data = pd.read_csv("train1.csv")
# print(data.head())

#fit dataset
model.fit(x = train_x, y = train_y, batch_size = 64, shuffle = True, epochs = 1000)

#save model
path_save = "./testsave4"

tf.keras.models.save_model(
model,path_save, overwrite=True, include_optimizer=True, save_format=None , signatures=None, options=None)

#restore saved model
model_restore = tf.keras.models.load_model(
path_save)

model_restore.summary()

# load train dataset
test_x, test_y = createData("C:/Users/Namrata
Chaudhari/Downloads/Lab 6/Emotion Recognition Using DNN/test4.csv",
                             featureDim = featureDim,
                             classes = classes
)

#evaluate test accuracy
model.evaluate(test_x,test_y)

#plot confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

confusion_matrix = confusion_matrix(test_y , result)

plt.figure()

```

```
plt.imshow(confusion_matrix, interpolation='nearest', cmap=plt.cm.Blues)

thresh = confusion_matrix.max() / 2.
for i in range(confusion_matrix.shape[0]):
    for j in range(confusion_matrix.shape[1]):
        plt.text(j, i, format(confusion_matrix[i, j]),
                 ha="center", va="center",
                 color="white" if confusion_matrix[i, j] == 0 or
                 confusion_matrix[i, j] > thresh else "black")
plt.tight_layout()
plt.colorbar()
```

Appendix F: Checking angles for landmark detection

```
#Detect angle code:

import cv2
import numpy as np
import dlib
import time
import math

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
POINTS_NUM_LANDMARK = 68

# Get the biggest face
def _largest_face(dets):
    if len(dets) == 1:
        return 0

    face_areas = [ (det.right()-det.left())*(det.bottom()-det.top()) for det in dets]

    largest_area = face_areas[0]
    largest_index = 0
    for index in range(1, len(dets)):
        if face_areas[index] > largest_area :
            largest_index = index
            largest_area = face_areas[index]

    print("largest_face index is {} in {} faces".format(largest_index, len(dets)))

    return largest_index

# Extract the point coordinates needed for pose estimation from the detection results of dlib
def get_image_points_from_landmark_shape(landmark_shape):
    if landmark_shape.num_parts != POINTS_NUM_LANDMARK:
        print("ERROR:landmark_shape.num_parts-{}".format(landmark_shape.num_parts))
        return -1, None
```

```

    #2D image points. If you change the image, you need to change
    e vector
    image_points = np.array([
(landmark_shape.part(30).x, landmark_shape.part(30).y),
# Nose tip
(landmark_shape.part(8).x, landmark_shape.part(8).y),
# Chin
(landmark_shape.part(36).x, landmark_shape.part(36).y),
# Left eye left corner
(landmark_shape.part(45).x, landmark_shape.part(45).y),
# Right eye right corner
(landmark_shape.part(48).x, landmark_shape.part(48).y),
# Left Mouth corner
(landmark_shape.part(54).x, landmark_shape.part(54).y)
# Right mouth corner
    ], dtype="double")

    return 0, image_points

# Use dlib to detect key points and return the coordinates of several
# points needed for pose estimation
def get_image_points(img):

    #gray = cv2.cvtColor( img, cv2.COLOR_BGR2GRAY) # The picture is
    # adjusted to gray
    dets = detector( img, 0 )

    for f in dets:
        shape = predictor(img, f)
        q=0
        for f in dets:
            img = cv2.rectangle(img, (f.tl_corner().x, f.tl_corner().y),
            (f.br_corner().x, f.br_corner().y), (0,255,0), 3)

            q+=1

    if 0 == len( dets ):
        print( "ERROR: found no face" )
        return -1, None
    largest_index = _largest_face(dets)
    face_rectangle = dets[largest_index]

    landmark_shape = predictor(img, face_rectangle)

```

```

    return get_image_points_from_landmark_shape(landmark_shape)

# Get rotation vector and translation vector

def get_pose_estimation(img_size, image_points ):
    # 3D model points.
    model_points = np.array([
        (0.0, 0.0, 0.0),          # Nose tip
        (0.0, -330.0, -65.0),     # Chin
        (-225.0, 170.0, -135.0),  # Left eye left corner
        (225.0, 170.0, -135.0),   # Right eye right corner
        (-150.0, -150.0, -125.0), # Left Mouth corner
        (150.0, -150.0, -125.0)  # Right mouth corner

    ])

    # Camera internals

    focal_length = img_size[1]
    center = (img_size[1]/2, img_size[0]/2)
    camera_matrix = np.array(
        [[focal_length, 0, center[0]],
         [0, focal_length, center[1]],
         [0, 0, 1]], dtype = "double"
    )

    print("Camera Matrix :{}".format(camera_matrix))

    dist_coeffs = np.zeros((4,1)) # Assuming no lens distortion
    (success, rotation_vector, translation_vector) = cv2.solvePn
P(model_points, image_points, camera_matrix, dist_coeffs, flags=
cv2.SOLVEPNP_ITERATIVE )

    print("Rotation Vector:\n {}".format(rotation_vector))
    print("Translation Vector:\n {}".format(translation_vector))
    return success, rotation_vector, translation_vector, camera_
matrix, dist_coeffs

# Convert from rotation vector to Euler angle
def get_euler_angle(rotation_vector):
    # calculate rotation angles
    theta = cv2.norm(rotation_vector, cv2.NORM_L2)

    # transformed to quaterniond
    w = math.cos(theta / 2)

```



```

x = math.sin(theta / 2)*rotation_vector[0][0] / theta
y = math.sin(theta / 2)*rotation_vector[1][0] / theta
z = math.sin(theta / 2)*rotation_vector[2][0] / theta

ysqr = y * y
# pitch (x-axis rotation)
t0 = 2.0 * (w * x + y * z)
t1 = 1.0 - 2.0 * (x * x + ysqr)
print('t0:{}, t1:{}'.format(t0, t1))
pitch = math.atan2(t0, t1)

# yaw (y-axis rotation)
t2 = 2.0 * (w * y - z * x)
if t2 > 1.0:
    t2 = 1.0
if t2 < -1.0:
    t2 = -1.0
yaw = math.asin(t2)

# roll (z-axis rotation)
t3 = 2.0 * (w * z + x * y)
t4 = 1.0 - 2.0 * (ysqr + z * z)
roll = math.atan2(t3, t4)

print('pitch:{}, yaw:{}, roll:{}'.format(pitch, yaw, roll))

# Unit conversion: convert radians to degrees
Y = int((pitch/math.pi)*180)
X = int((yaw/math.pi)*180)
Z = int((roll/math.pi)*180)

return 0, Y, X, Z

def get_pose_estimation_in_euler_angle(landmark_shape, im_szie):
    try:
        ret, image_points = get_image_points_from_landmark_shape
(landmark_shape)
        if ret != 0:
            print('get_image_points failed')
            return -1, None, None, None

        ret, rotation_vector, translation_vector, camera_matrix,
dist_coeffs = get_pose_estimation(im_szie, image_points)
        if ret != True:
            print('get_pose_estimation failed')
            return -1, None, None, None

```

```

ret, pitch, yaw, roll = get_euler_angle(rotation_vector)
if ret != 0:
    print('get_euler_angle failed')
    return -1, None, None, None

euler_angle_str = 'Y:{}, X:{}, Z:{}'.format(pitch, yaw,
roll)

print(euler_angle_str)
return 0, pitch, yaw, roll

except Exception as e:
    print('get_pose_estimation_in_euler_angle exception:{}'.
format(e))
    return -1, None, None, None

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FPS, 10)
fourcc = cv2.VideoWriter_fourcc(*'XVID')
output_video = cv2.VideoWriter('output.mp4', fourcc, 10.0, (640,
480))
while (cap.isOpened()):
    start_time = time.time()

    # Read Image
    ret, im = cap.read()
    if ret != True:
        print('read frame failed')
        continue
    size = im.shape
    if size[0] > 700:
        h = size[0] / 3
        w = size[1] / 3
        im = cv2.resize(im, (int( w ), int( h )), interpolation
=cv2.INTER_CUBIC )
        size = im.shape

    ret, image_points = get_image_points(im)
    if ret != 0:
        print('get_image_point failed')
        continue

    ret, rotation_vector, translation_vector, camera_matrix, dis
t_coeffs = get_pose_estimation(size, image_points)
    if ret != True:

```

```

        print('get_pose_estimation failed')
        continue
used_time = time.time() - start_time
print("used_time:{} sec".format(round(used_time, 3)))

ret, pitch, yaw, roll = get_euler_angle(rotation_vector)
euler_angle_str = 'Y:{}, X:{}, Z:{}'.format(pitch, yaw, roll
)

print(euler_angle_str)

# Project a 3D point (0, 0, 1000.0) onto the image plane.
# We use this to draw a line sticking out of the nose

(nose_end_point2D, jacobian) = cv2.projectPoints(np.array([(
0.0, 0.0, 1000.0)]), rotation_vector, translation_vector, camera
_matrix, dist_coeffs)

for p in image_points:
    cv2.circle(im, (int(p[0]), int(p[1])), 3, (0,0,255), -1)

p1 = ( int(image_points[0][0]), int(image_points[0][1]))
p2 = ( int(nose_end_point2D[0][0][0]), int(nose_end_point2D[
0][0][1]))

cv2.line(im, p1, p2, (255,0,0), 2)

# Display image
#cv2.putText( im, str(rotation_vector), (0, 100), cv2.FONT_H
ERSHEY_PLAIN, 1, (0, 0, 255), 1 )
cv2.putText( im, euler_angle_str, (0, 120), cv2.FONT_HERSHEY
_PLAIN, 1, (0, 0, 255), 1 )
cv2.imshow("Output", im)
output_video.write(im)
if cv2.waitKey(1) & 0xFF == ord('s'):
    break

output_video.release()
cap.release()
cv2.waitKey(0)
cv2.destroyAllWindows()

```



Internship Report

Nermati Shravan Reddy

Internship Report

Of

Nermati Shravan Reddy

Btech/10663/18

BACHELOR OF TECHNOLOGY

Civil Engineering

Department of Civil and Environmental Engineering

Birla Institute of Technology

Mesra, Ranchi

2018-2022



On

Big Data Computing

Under the mentorship of

Ioan Raicu

Immersive Summer Research Experience

Illinois Institute of Technology, Chicago

(Duration 7th of June, 2021 to 31st of July 2021)



ILLINOIS TECH

Offer Letter

8/12/2021

BIT Webmail Mail - You've Been Admitted to your Illinois Tech Research Opportunity!



NERMATI SHRAVAN REDDY <btech10663.18@bitmesra.ac.in>

You've Been Admitted to your Illinois Tech Research Opportunity!

1 message

Illinois Tech Graduate Admission <grad.admission@iit.edu>
Reply-To: Illinois Tech Graduate Admission <grad.admission@iit.edu>
To: btech10663.18@bitmesra.ac.in

Sun, May 9, 2021 at 12:36 AM



Dear Nermati Shraavan,

Congratulations! You have been admitted to your Research Program

You have been admitted to Illinois Tech to perform research with Professor **Ioan Raicu** for the following opportunity: **COMP 495-304 Big Data Computing**. This research session will be occurring **online** from **6/7/21** to **7/31/21**

The price for the 3-credit research course is \$4,842; however, as a visiting research scholar, you will receive a scholarship of **\$2000**. If you indicated you are living on campus, you will receive an additional email with information about housing and costs.

To secure your spot in the research opportunity, you must submit your \$500 deposit (select "Summer Research" from the drop-down menu) by using the log in information below and following these instructions: [Submit your deposit](#).

- Campus Wide ID Number (CWID): **A20498221**
- Username/U-ID: **nreddy4**
- Password: MMDDXXXX (MMDD is the 2-digit month and day you were born and XXXX is the last 4 digits of your CWID)
- Illinois Tech Email Address: nreddy4@hawk.iit.edu

*Once you sign in, you will be asked to create a new password, and you will be required to create security questions. Please remember your log-on details. **If your name is misspelled, if your username is misspelled, or if you have trouble logging in to myIIT, please explain by email to supportdesk@iit.edu.*

Remember that spots are filled on a first-come, first-served basis. Therefore, submitting your deposit as soon as possible will reserve your spot in this opportunity. If you applied to more than one opportunity, please note that the opportunity for which you enroll will be the opportunity in which you participate.

Once again, congratulations on your admission for research at Illinois Tech. We look forward to welcoming you to our community.

Sincerely,
The Elevate Team

Campus Wide ID Number (CWID): **A20498221**

<https://mail.google.com/mail/u/2?ik=01a316b087&view=pt&search=all&permthid=thread-f%3A1699218238875164165&simpl=msg-f%3A1699218...> 1/2

Certificate

Upon the recommendation of the faculty of
the

**College of Computing
of Illinois Institute of Technology**

Nermati Shravan Reddy

is recognized as achieving
Summer Undergraduate Computing Research
Immersion Program Certificate of Participation

with all the rights, privileges, and honors thereunto appertaining.
Awarded at Chicago, in the State of Illinois of the United States of America
July 31, 2021

ILLINOIS TECH

College of Computing



Lance Fortnow, Dean
College of Computing

Acknowledgement

I am grateful and would like to express my gratitude, and I am fortunate to have had the kind association and mentorship of Professor Ioan Raicu.

Their exemplary guidance, constant encouragement, and support were so helpful for me to learn and made my experience a wonderful one.

I would also like to thank Ms. Mary Dawson for helping me with the process and making it feel seamless.

I would like to extend my gratefulness to Mr. Vishal H Shah, Associate Dean (Alumni and International Relations), and Mr. Utpal Baul Dean (Alumni and International Relations), for being there during the entire application process and helping me through it.

I would like to thank the Institute and BITMAA-NA for helping with the expenses of the program.

I also like to extend my warm gratitude and regards to everyone who helped me during my internship.

Regards,

Nermati Shravan Reddy

(Btech/10663/18)

Table of Contents

S. No.	Particulars	Page No.
1	Offer Letter	2
2	Grade Card	3
3	Acknowledgement	4
4	Project Information	6

5	About	7-8
6	Execution Part-1	9-25
7	Execution Part-2	26-48
8	Execution Part-3	49-60

Project Information

Research Topic: Big Data Computing

Mentor: Ioan Raicu

University: Illinois Institute of Technology

Brief Description of the flow of the project:

The project consisted of three parts. The first part dealt with the machine (setting up of virtual machine) and varied configurations of it. The second part concentrated on finding the best feasible approach for sorting large amounts of data using various methods of Java and using multithreading. The third part of the project focused on finding efficient and agile techniques for sorting possible in python.

Big Data Computing

The definition of big data is data that contains greater variety, arriving in increasing volumes and with more velocity. With the digitalization and increase in the usage of the internet abundance of data is generated every millisecond.

In simple words, big data is larger, more complex data sets, especially from new data sources. These data sets are so large and will overwhelm the traditional software, but are very important to address many problems which could not be tackled otherwise.

The three Vs of big data:

Volume: With big data, one has to process high volumes of low-density, unstructured data. This can be data of unknown value, such as Instagram data feeds, Twitter data, or sensor-enabled equipment. Depending on the size of organizations, this data might be tens of terabytes of data to hundreds of petabytes.

Velocity: Velocity is the rate at which data is received and acted on or computed. Generally, the highest velocity of data streams directly into memory versus being written to disk. Some products operate in real-time or near real-time and will require real-time evaluation and action.

Variety: Variety refers to the many types of data that are available. Traditional data types were structured and fit neatly in a database. With the rise of big data, data comes in new unstructured data types. Unstructured and semi-structured data types, such as text, audio, and video, require additional pre-processing to derive meaning and support metadata.

The History of Big Data:

Although the concept of big data itself is relatively new, the origins of large data sets go back to the 1960s and '70s when the world of data was just getting started with the first data centres and the development of the relational database.

Around 2005, people began to realize just how much data users generated through Facebook, YouTube, and other online services. Hadoop (an open-source framework created specifically to store and analyse big data sets) was developed that same year. NoSQL also began to gain popularity during this time.

The development of open-source frameworks, such as Hadoop (and more recently, Spark) was essential for the growth of big data because they make big data easier to work with and cheaper to store. In the years since then, the volume of big data has skyrocketed. Users are still generating huge amounts of data—but it's not just humans who are doing it.

With the advent of the Internet of Things (IoT), more objects and devices are connected to the internet, gathering data on customer usage patterns and product performance. The emergence of machine learning has produced still more data.

While big data has come far, its usefulness is only just beginning. Cloud computing has expanded big data possibilities even further. The cloud offers truly elastic scalability, where developers can simply spin up ad hoc clusters to test a subset of data.

For understanding and working on Big Data a detailed and thorough understanding of computer systems is very important and it is also very important to understand parallelism and finding the perfect algorithm. Hence major part of this project will run around computer systems, parallelism and finding the best algorithm.

Execution Part-1

The first part of the project deals with setting up a virtual machine and trying different configurations.

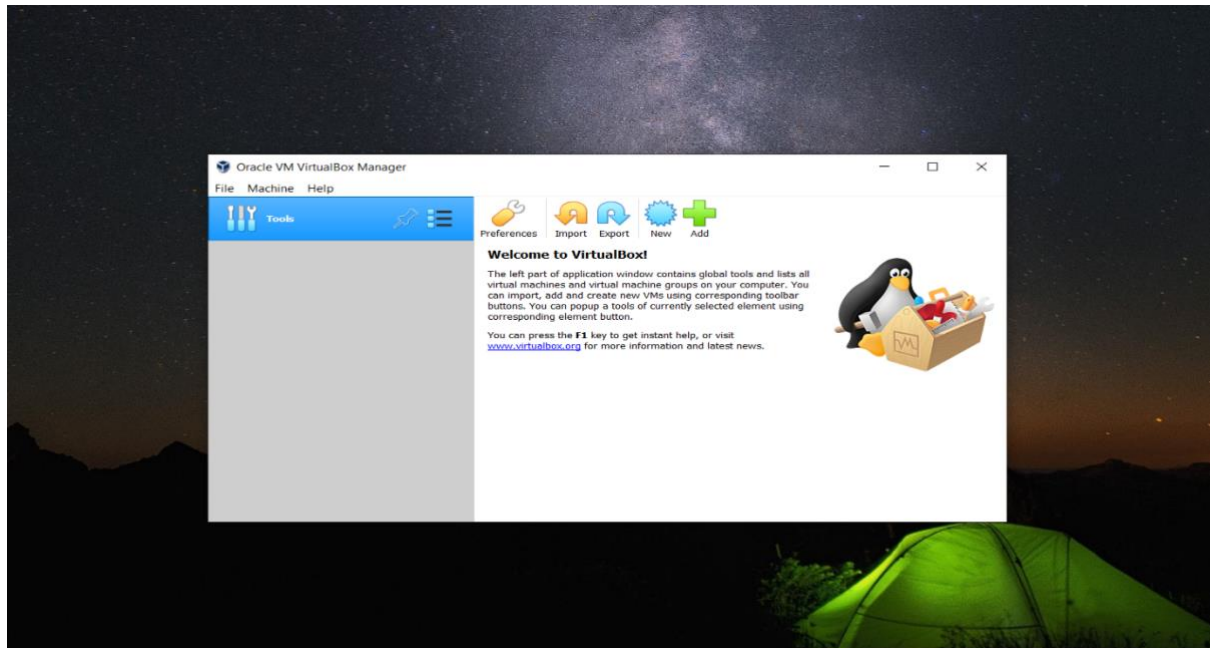
Setting up of Virtual Machine:

The first step of setting up a Virtual Machine is to download and install a VirtualBox Manager, which I downloaded from <https://www.virtualbox.org/wiki/Downloads> and choosing the appropriate host machine operating system, in my case it is windows host.

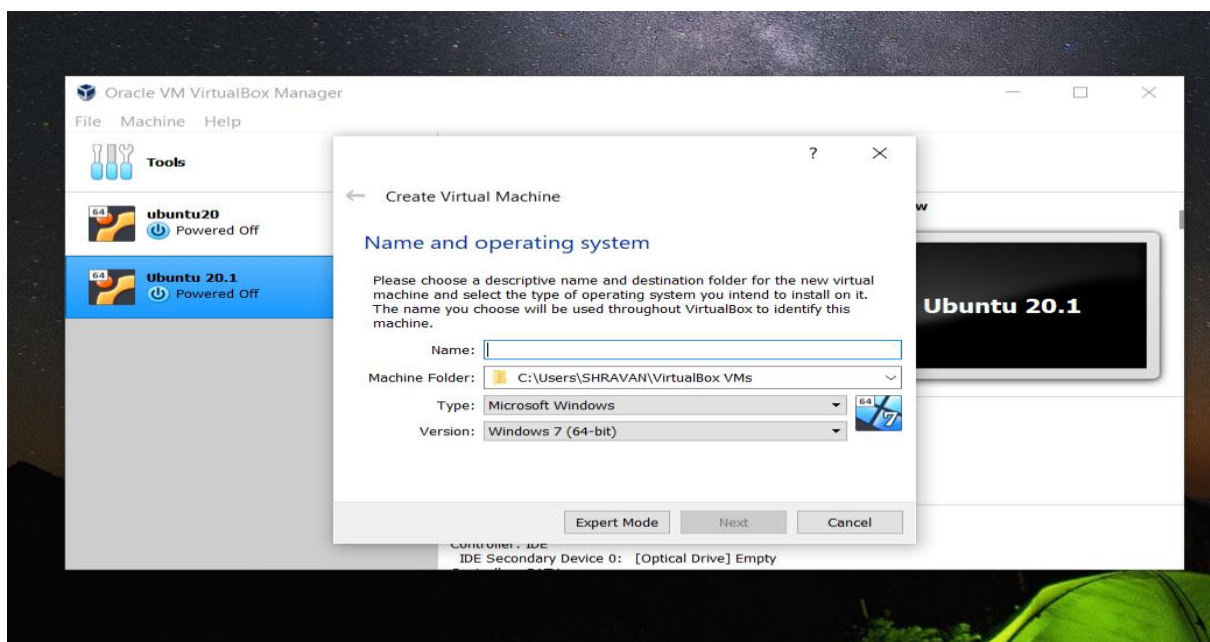


The screenshot shows the 'Download VirtualBox' page on the Oracle VM VirtualBox website. On the left, there is a navigation menu with links: About, Screenshots, Downloads, Documentation, End-user docs, Technical docs, Contribute, and Community. The main content area features the VirtualBox logo and the heading 'Download VirtualBox'. Below this, it states: 'Here you will find links to VirtualBox binaries and its source code.' The page is divided into sections: 'VirtualBox binaries' with a license agreement and version information (6.0 and 5.2); 'VirtualBox 6.1.22 platform packages' with a list of supported operating systems (Windows, OS X, Linux, Solaris, Solaris 11 IPS); and 'VirtualBox 6.1.22 Oracle VM VirtualBox Extension Pack' with a link to download for all supported platforms. A note mentions upgrading the guest additions.

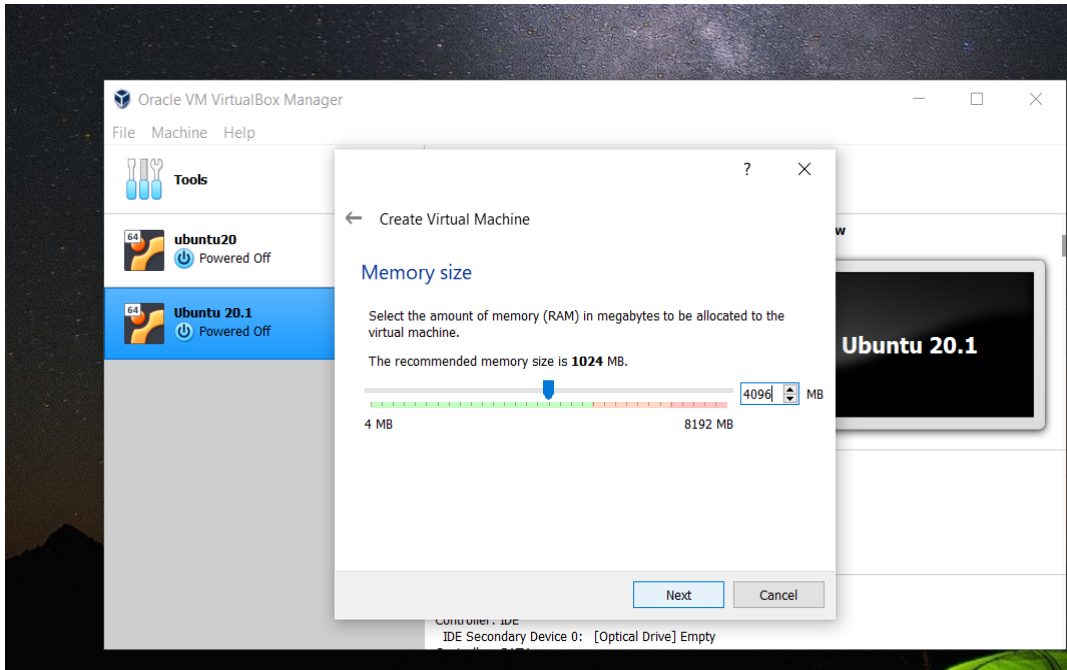
Upon downloading, it can be installed by running it. Once installation is done the following VirtualBox Manager can be launched.



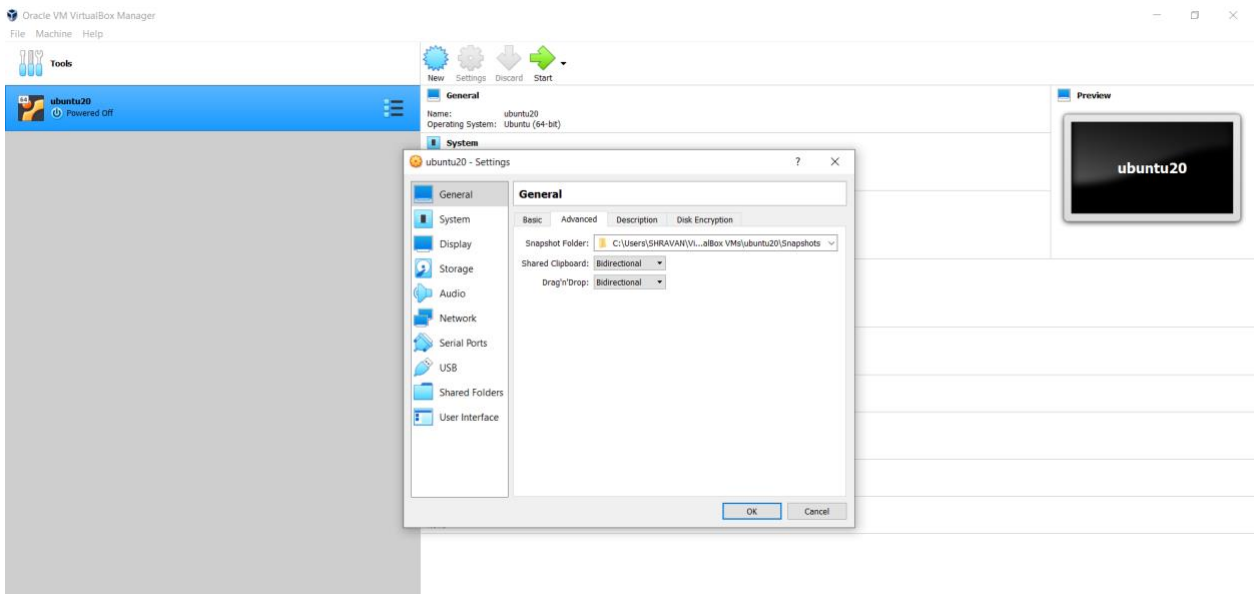
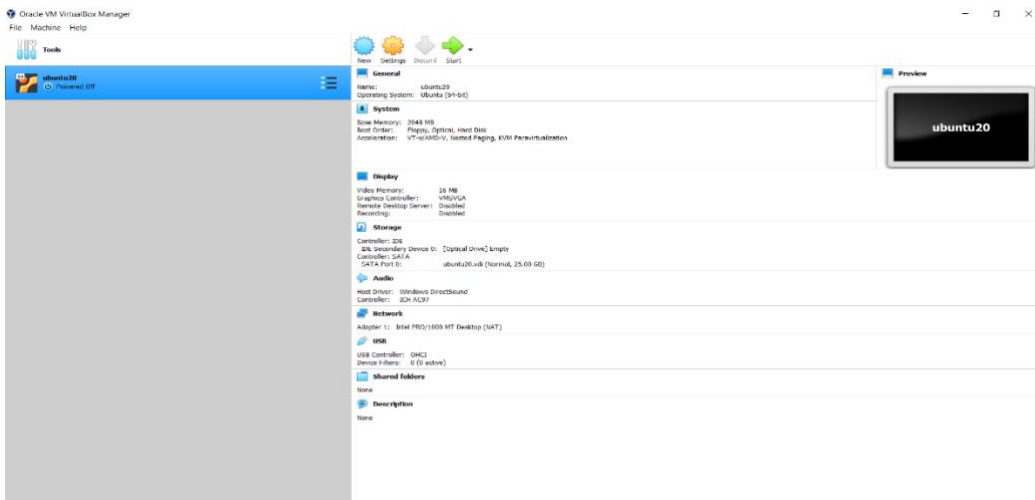
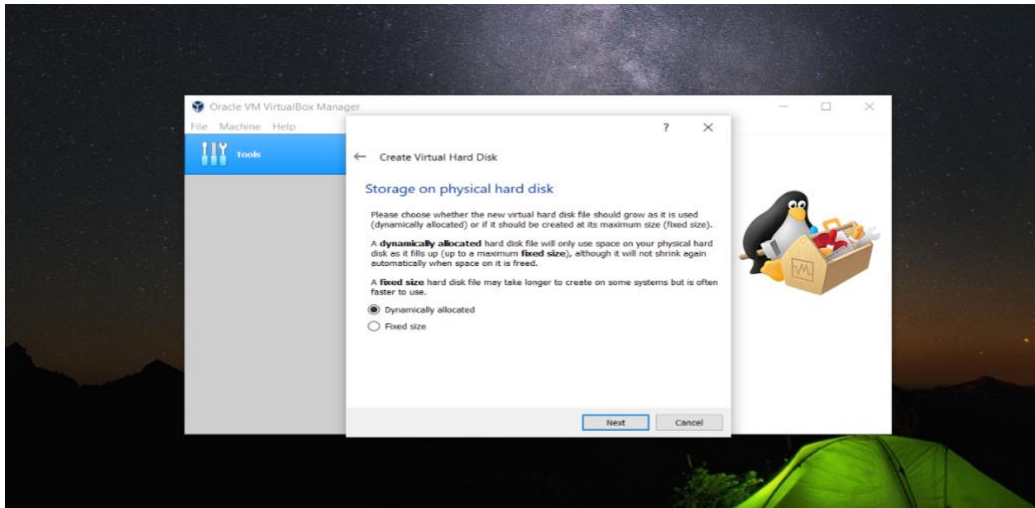
Now for the creation of a new virtual machine, we need to click on the new and a name, folder, type, version must be selected. In my case I have chosen the name as Ubuntu20, folder as default, type as Linux and version as Ubuntu 64-bit (as we will be working on a Linux machine for the rest of the project)



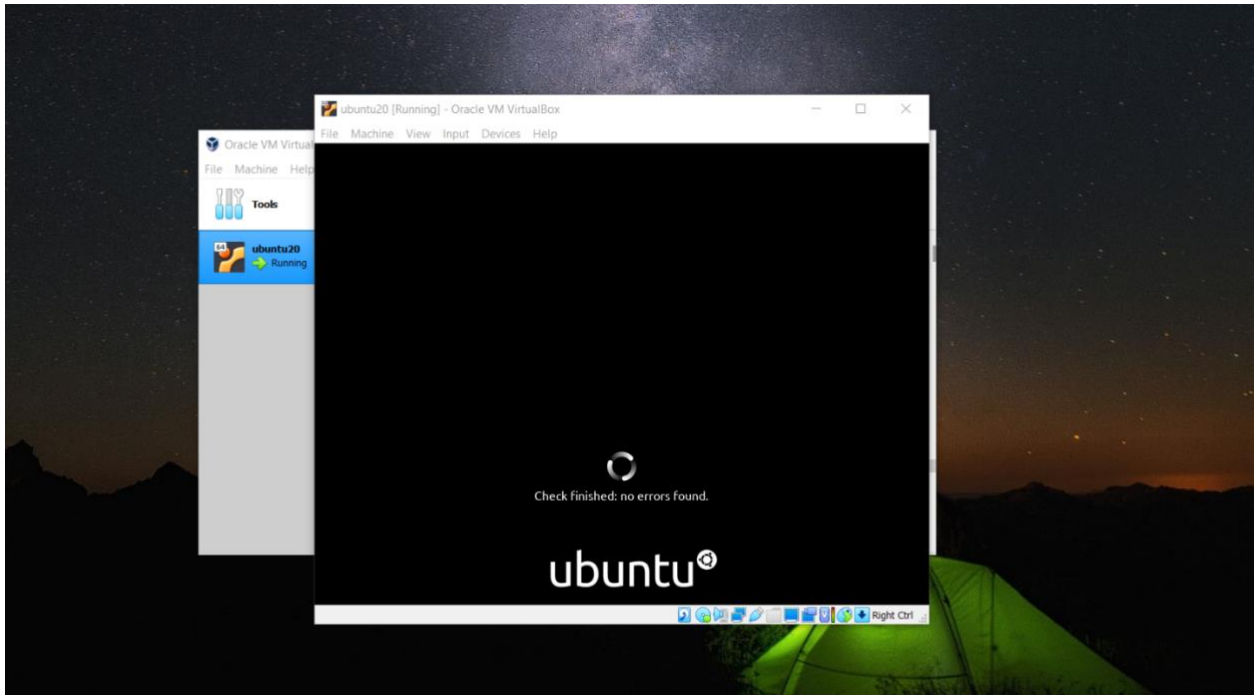
Then the memory size must be chosen in MB.



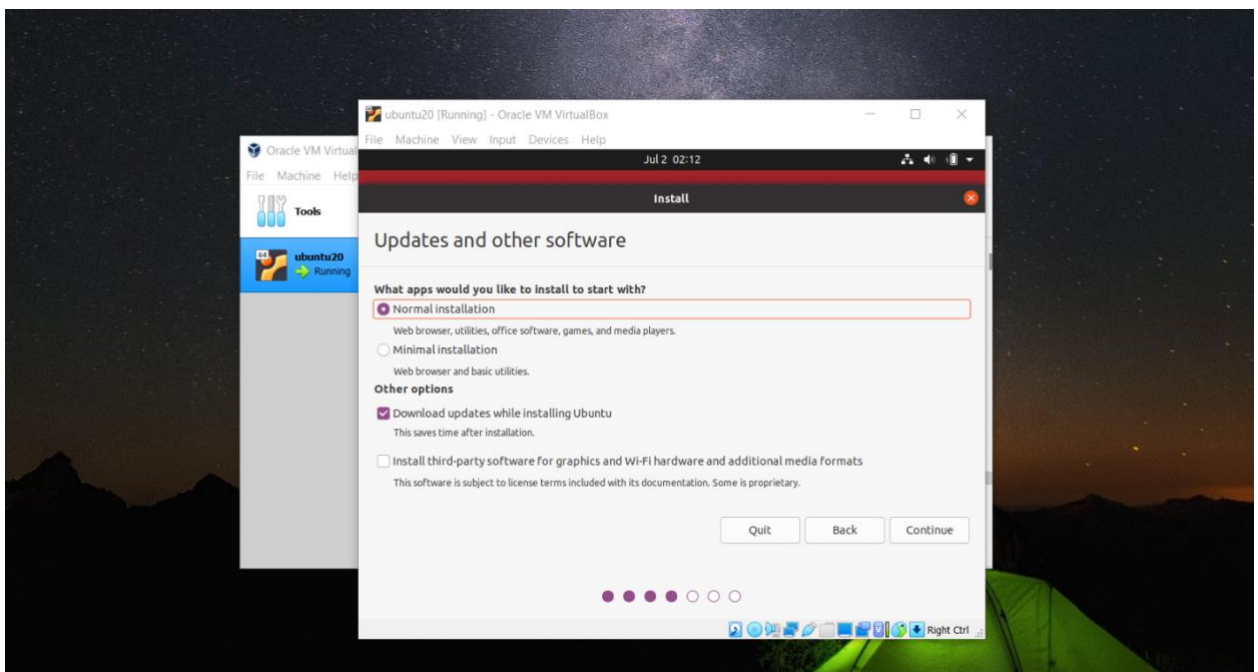
And the next few steps are to be followed as in the screenshots attached.

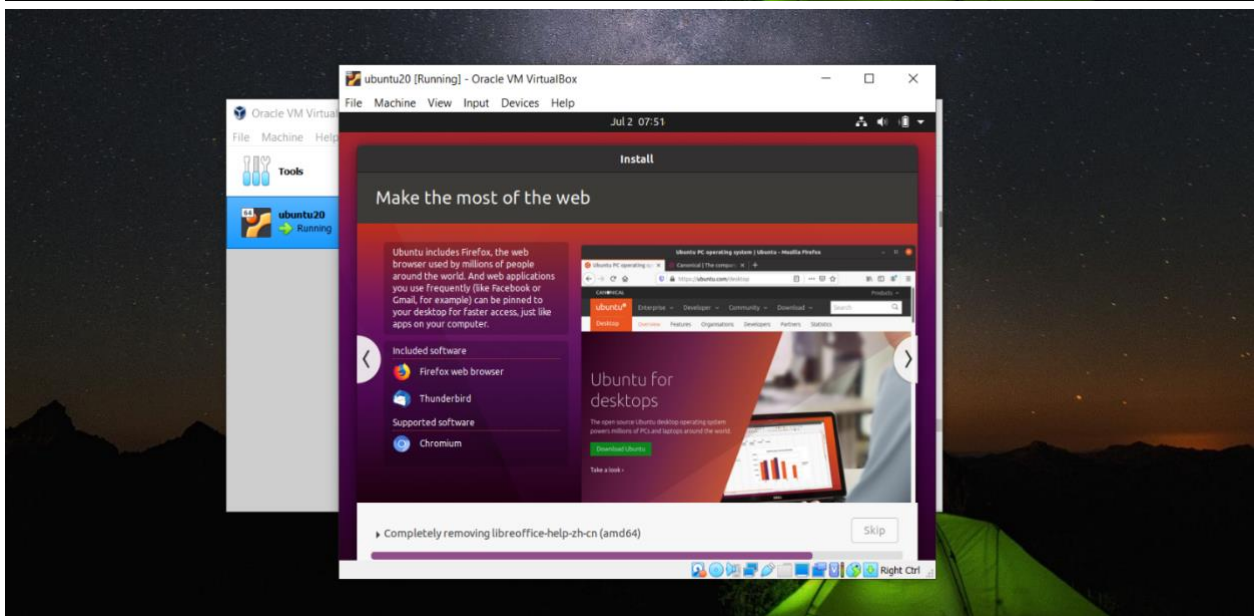
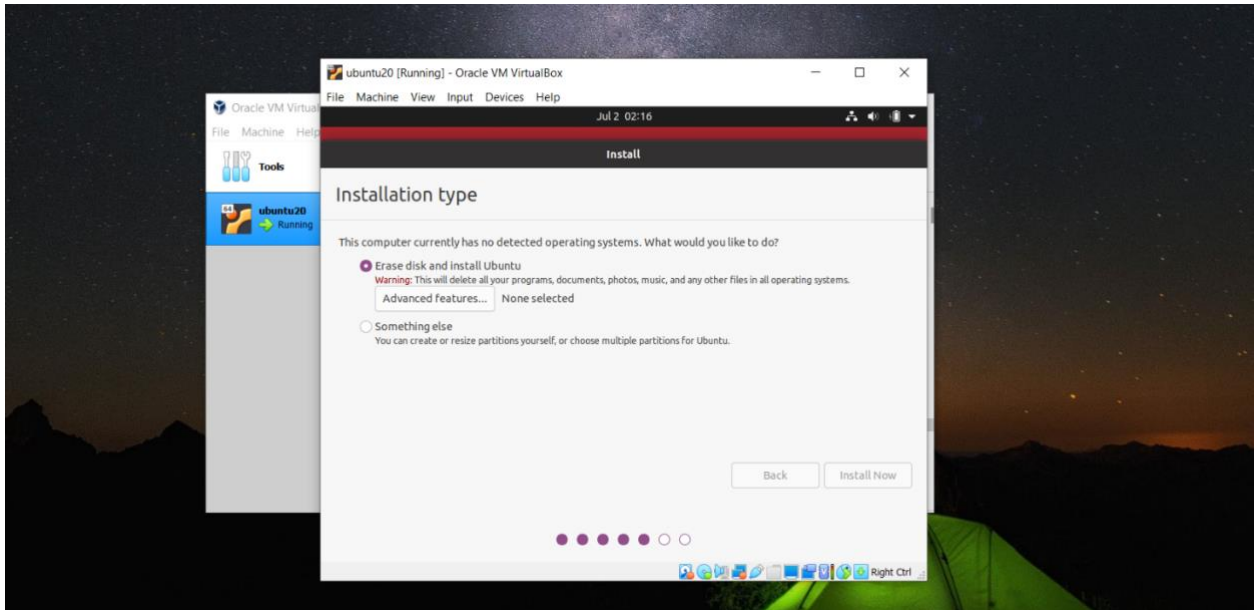


Once the setting up and configuring a virtual machine is done we can launch the machine.



Now upon launching the virtual machine we will need to install the Linux machine which is done as shown in the pictures below.





The Virtual Machine is set up, formatted as per the above screenshots and then the VM is started.

Now for setting up the public private keys, we need to download the public private keys to our local host machine and copy the public key into the VM.

Upon copying into the VM proper permissions must be set.

Then the public private keys are set and now the system can be logged into without actually entering the password everytime.

Command Prompt

```
C:\Users\SHRAVAN>ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\SHRAVAN\.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\SHRAVAN\.ssh/id_rsa.
Your public key has been saved in C:\Users\SHRAVAN\.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:BAAdj7x9N1ekbWRw1useX1xd4oVZ212xLrzNfPibN7eg shravan@LAPTOP-89JCQV90
The key's randomart image is:
+---[RSA 4096]-----+
|
|  =.  oO0|
| - =  .*0|
|  o  .+=0=|
|  o o. +=|=|
| S . . . **|
|  . . =.=|
|  .  o=0|
|  . *+|
|  .E.o|
+-----[SHA256]-----+

C:\Users\SHRAVAN>
```

Command Prompt

```
C:\Users\SHRAVAN>sshdir
Volume in drive C is Windows
Volume Serial Number is FC91-C721

Directory of C:\Users\SHRAVAN\.ssh

03-07-2021 17:09 <DIR> .
03-07-2021 17:09 <DIR> ..
03-07-2021 17:09          3,389 id_rsa
03-07-2021 17:09          750 id_rsa.pub
03-07-2021 17:09          614 known_hosts
03-07-2021 17:07          3 File(s)      4,753 bytes
                2 Dir(s)  427,556,663,296 bytes free

C:\Users\SHRAVAN\.ssh>scp id_rsa.pub shravan@LinuxVM:/home/shravan/.ssh/uploaded_key.pub
shravan@LinuxVM's password:
id_rsa.pub
100% 750 259.7KB/s 00:00

C:\Users\SHRAVAN\.ssh>
```

```
shravan@LinuxVM: ~/ssh
shravan@LinuxVM:~$ cd .ssh
shravan@LinuxVM:~/ssh$ ls
uploaded_key.pub
shravan@LinuxVM:~/ssh$
```

```
shravan@LinuxVM:~/ssh
shravan@LinuxVM:~/ssh$
shravan@LinuxVM:~/ssh$ cat ~/.ssh/uploade_key.pub >> ~/.ssh/authorized_keys
shravan@LinuxVM:~/ssh$ cat ~/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDfXeOCbefgRMZvIu000E0BBbSwu0F/NjZeRFTNOg4b0RooH+Ue8LRZYiyX0mJ8pRtpmcJ4mSwHcCLrMApOCbG0PIgCIPpVacChqMUUEVagGGHGAZdHnddX7PZ1+puUk393c23Snfejaqe/UZng1+LIxbzCRjackKd1P0qKEZ5j8JnZC+/t0GYcZePd3JCHK+MqEK89xFLV7r6cL1SqqyD68NHpASwM9eDkbFKp+UvF4hYn6kbj/KW58nzBZw2m4u0qXgn1Fv6KzVqrpIbUTfYgRhh8PAscV8QmLqqdJ3F+HzCBjQ88LV9+MqClhvkJZps+17gdrnXCnbJF3YLGng72H6MG150QKBtApKdH+R1ZjV1HuzAw1dNb71FiVv5mw8mJyVAbuo+HRGp6QmasmoxvyinuacbnBeKNEj8B1S1AcdbuK1KtDRmzr0qrg18CT8uH10PrBYiQ1IvFhb9Sku561W7Wt0bqHEsPLuLCTDQ1Iux3FS+xsQCtKS3VqB4gG1L1bQh3VPPM60H1Pm3zCX7dpzQdfsgVIX92w0iff1Rfj1uFv71Kiyt1x7eAe4ejpah0n1s1LkEVhKJD+XtGfQok+4w6ZUfcp3fDgGEZj8Rmt0EhH+M7+Los66739rWPhrxpXQeyRKEfgrKGeegD1DFOMTrQAA== shravan@LAPTOP-891CQ90
shravan@LinuxVM:~/ssh$
```

```
shravan@LinuxVM: ~/ssh
shravan@LinuxVM:~/ssh$ chmod 700 ~/ssh/
shravan@LinuxVM:~/ssh$ chmod 600 ~/ssh/*
shravan@LinuxVM:~/ssh$
```

```
shravan@LinuxVM: ~/ssh
shravan@LinuxVM:~/ssh$ sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.back
[sudo] password for shravan:
shravan@LinuxVM:~/ssh$
```

```
shravan@LinuxVM: ~
C:\Users\SHRAVAM\.ssh>ssh shravan@LinuxVM
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.8.0-59-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

236 updates can be installed immediately.
92 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Sat Jul 3 17:07:05 2021 from 2401:4900:27c8:c592:65d2:d3f8:99f6:d1b0
shravan@LinuxVM:~$
```

```
Command Prompt
C:\Users\SHRAVAM\.ssh>dir
Volume in drive C is Windows
Volume Serial Number is FC91-C721

Directory of C:\Users\SHRAVAM\.ssh

03-07-2021 17:09 <DIR>          .
03-07-2021 17:09 <DIR>          ..
03-07-2021 17:09                3,389 id_rsa
03-07-2021 17:09                750 id_rsa.pub
03-07-2021 17:07                614 known_hosts
                3 File(s)      4,753 bytes
                2 Dir(s)    427,556,663,296 bytes free

C:\Users\SHRAVAM\.ssh>scp id_rsa.pub shravan@LinuxVM:/home/shravan/.ssh/uploaded_key.pub
shravan@linuxvm's password:
id_rsa.pub                               100% 750 259.7KB/s 00:00

C:\Users\SHRAVAM\.ssh>
```

This brings us to the end of setting up a virtual Linux machine.

Now we come to the next section of the first part, where we are required to analyse different configuratons.

1. Processor Count:

In general, increasing the number of processors increases the speedup of the machine, but it reaches a plateau and may even decrease upon increasing after a certain number of processors. Amdahl's law is often used in parallel computing to predict the theoretical speedup when using multiple processors. So, in our case we have chosen 2.

2. Acceleration:

None: This explicitly **turns off exposing** any and all paravirtualization interface.

Legacy: The legacy option is chosen for VMs that were created with the older VirtualBox versions and will pick a paravirtualization interface while starting the VM with VirtualBox 5.0 and newer.

Minimal: Announces the presence of a virtualized environment. Additionally, reports the TSC and APIC frequency to the guest operating system. **This provider is mandatory for running any Mac OS X guests.**

Hyper-V: This presents a Microsoft Hyper-V hypervisor interface which is accepted by Windows 7 and newer operating systems. VirtualBox's implementation at present supports Para virtualized clocks, APIC frequency reporting, guest debugging, guest crash reporting and relaxed timer checks. **This provider is recommended for Windows guests.**

KVM: This presents a Linux KVM hypervisor interface which is accepted by Linux kernels starting with version 2.6.25. VirtualBox's implementation currently supports Para virtualized clocks and SMP spinlocks. **This provider is recommended for Linux guests.**

3. Storage Devices:

IDE:

IDE is short for Integrated Drive Electronics.

IDE is an interface standard for connecting storages devices like HDD, SSD, and CD/DVD drives to the computer.

Adding/removing components while the computer is running is not supported.

IDE's speed of data transfers ranges from 100 MB/s to 133 MB/s.

IDE drives are slower than SATA drives

It is a parallel connection.

SATA:

SATA is short for Serial Advanced Technology Attachment.

SATA is a computer bus interface or standard hardware interface connecting HDD, SSD, and CD/DVD drives to the computer.

Adding/removing components while the computer is running is supported.

SATA speed of data transfer ranges from 150 MB/s for SATA I and 300 MB/s for SATA II.

SATA drives are faster than IDE drives.

It is a serial connection.

NVMe:

NVME is short for Non-Volatile Memory Express.

NVME is an interface protocol built especially for SSD.

IDE's speed of data transfer peaked up to 550MB/s.

NNMe drivers are faster than SATA and IDE.

4. Network Adaptors:

NAT:

VM can connect to the host.

The host cannot connect to the VM.

VM can connect to the internet using the host network.

A VM cannot connect to another VM in the same network as they have the same IP address.

Other computers on the host network cannot connect to the VM.

Bridged Network:

VM can connect to the host.

The host can connect to the VM.

VM can connect to the external network unless it is on a VPN.

A VM can connect to another VM in the same network.

Other computers on the host network can connect to the VM.

Internal Network:

VM cannot connect to the host.

The host cannot connect to the VM.

VM cannot connect to the external network unless it is on a VPN.

A VM can connect to another VM in the same network.

Other computers on the host network cannot connect to the VM.

This is useful in cases where you want to isolate your test environment.

Host-Only:

VM cannot connect to the host.

The host can connect to the VM.

VM cannot connect to the external network unless it is on a VPN.

A VM can connect to another VM in the same network.

Other computers on the host network can connect to the VM.

5. USB Configuration:

USB 1.1

Data transfer rates defined in the specification are as Low Speed 1.5 Mbits/sec and Full Speed 12 Mbits/sec.

The maximum length of each cable section is 5 meters.

Power 500 mA (limited to 100mA during start-up).

USB 2.0:

There are only some minor variations from USB 1.1 to the USB 2.0 specification.

In a way, 2.0 specification is a superset of 1.1 with the major functional difference is the addition of a High Speed 480 Mbits/sec data transfer mode.

USB 3.0:

It is backward compatible with 2.0.

Power 900mA

It has a SuperSpeed >4.8 Gbits/sec data transfer mode and 400MBytes/sec after protocol overheads.

In the last and final section of the part one we generate data using Linux and compare the time taken to generate, sort using simplest forms of sorting techniques in Linux and Python.

Generation of Dataset:

```
#!/usr/bin/bash

#Taking input arguments i.e., dataset name and number of records

args=("$@")

#checking if user have entered two arguments

if [ $# -ne 2 ]

then

    echo "You need to enter both the dataset name and num of records"

    exit

fi

integer="${args[0]}.int"

string="${args[0]}.str"

:>${args[0]}

#Generation a entered number of random numbers and ascii strings

(shuf -i 0-4294967295 -n ${args[1]})> $integer

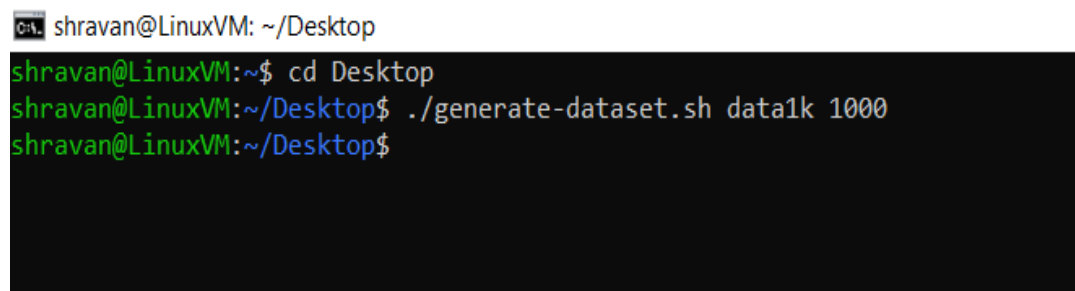
base64 -i /dev/urandom | fold -w 95 | head -n ${args[1]} > $string

#pasting the generated random integers and strings as datasets into the file

paste $integer $string > ${args[0]}
```

To run the above code a file must be created with .sh extension say, generate-dataset.sh and to run the file the file path must be given as the command and the dataset name and size of the dataset should be given as the program expects you to give them as arguments.

The screenshot below shows how it is run.



```
shravan@LinuxVM: ~/Desktop
shravan@LinuxVM:~$ cd Desktop
shravan@LinuxVM:~/Desktop$ ./generate-dataset.sh data1k 1000
shravan@LinuxVM:~/Desktop$
```

Sorting Using Bash:

```
#!/usr/bin/bash

filename=$1

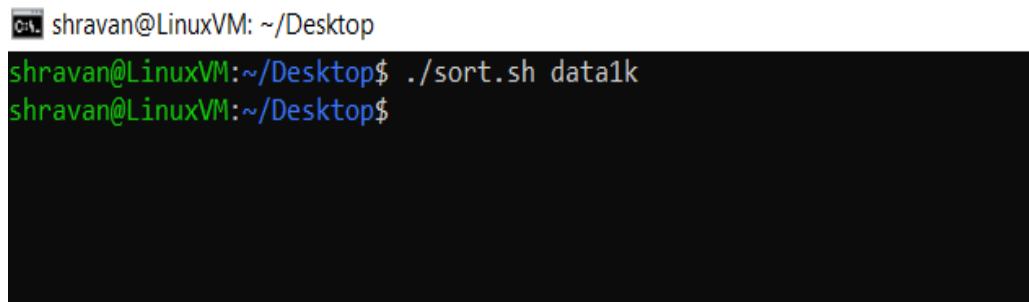
if [ $# -ne 1 ]
then
    echo "You need to enter the dataset name to sort"
    exit
fi

sorted_file="$filename.sorted"

sort -bk 1,1rn $filename > $sorted_file
```

To run the above code a file must be created with .sh extension say, sort.sh and to run the file the file path must be given as the command and the dataset name should be given as the program expects you to give it as an argument.

The screenshot below shows how it is run.

A terminal window screenshot showing a user named shravan@LinuxVM in the directory ~/Desktop. The user runs the command ./sort.sh data1k, and the terminal shows the command being executed and the prompt returning to the user.

```
shravan@LinuxVM: ~/Desktop
shravan@LinuxVM:~/Desktop$ ./sort.sh data1k
shravan@LinuxVM:~/Desktop$
```

Sorting using Python:

```
#!/usr/bin/bash

import time

import os.path

st=time.time()

fname=input()

sorted_lines=None

with open(fname, 'r') as file_:

    lines=file_.readlines()

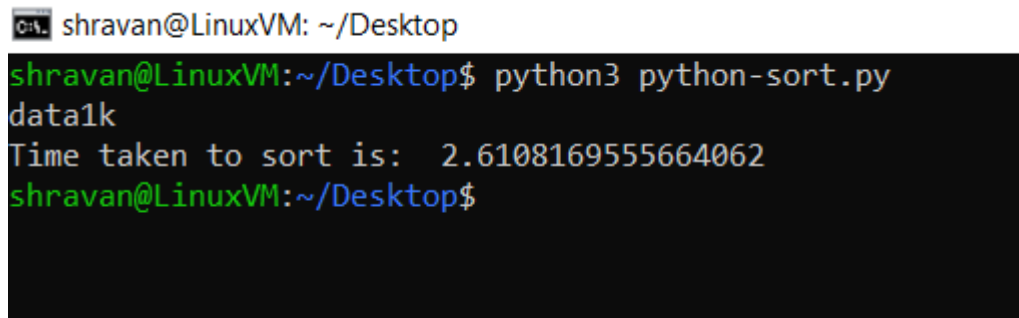
    sorted_lines=sorted(lines, key=lambda x: int(x.split()[0]))
```

```
with open(fname, 'w') as file_:
    for e in sorted_lines:
        file_.write(e)

print("Time taken to sort is: ", time.time()-st)
```

To run the above code a file must be created with .py extension say, python-sort.py and to run the program we need to type python3 file path as the command. Upon pressing enter the name of the must be given as input and enter must be pressed.

The screenshot below shows how it is run.



```
shravan@LinuxVM: ~/Desktop
shravan@LinuxVM:~/Desktop$ python3 python-sort.py
data1k
Time taken to sort is: 2.6108169555664062
shravan@LinuxVM:~/Desktop$
```

Sorting Using C:

```
#include<stdio.h>

#include<string.h>

#include<time.h>

int main()
{
    char fileName[25];

    printf("Enter the name of the file to sort:\n");

    scanf("%s", fileName);

    clock_t startTime = clock();

    FILE* f = fopen(fileName, "r");

    printf("\n");

    long i = 0;

    long n=0;

    char strings[1000][95], tempstr[95];
```

```

long integer[1000];

while (!feof(f))
{
    fscanf(f, "%ld %s\n", &integer[i], &strings[i][0]);
    i++;
}

n=i+1;

fclose(f);

for(int i = n - 2; i >= 0; i--)
{
    for(int j = 0; j <= i; j++)
    {
        if(integer[j] > integer[j+1])
        {
            long temp = integer[j];
            integer[j] = integer[j+1];
            integer[j+1] = temp;
            //handles swapping
            strcpy(tempstr, strings[j]);
            strcpy(strings[j], strings[j+1]);
            strcpy(strings[j+1], tempstr);
        }
    }
}

FILE* of = fopen("sorted_C.txt", "w+");

for(int i = 1; i < n; i++)

    fprintf(of, "%ld %s\n", integer[i], strings[i]);

```

```

    clock_t endTime = clock();

    double totalTime = (double)(endTime - startTime)/CLOCKS_PER_SEC;

    fclose(of);

    printf("Time is %f seconds\n", totalTime);

    return 0;
}

```

To run the above code a file must be created with .c extension say, c-sort.c and to compile we need to type the command gcc filename.c -o prog, then the prog file can be run using the file path of the prog file. Upon pressing enter the name of the must be given as input and enter must be pressed.

The screenshot below shows how it is run.

```

shravan@LinuxVM: ~/Desktop
shravan@LinuxVM:~/Desktop$ gcc c-sort.c -o prog
shravan@LinuxVM:~/Desktop$ ./prog
Enter the name of the file to sort:
data10

Time is 0.000140 seconds
shravan@LinuxVM:~/Desktop$

```

Plotting and comparing different methods:

```

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
d={"Data Size":[1000, 100000, 10000000], "Creation":[0.03, 0.217, 20.50], "Sort":[0.01, 0.09, 13.7
5], "Sort-Python":[4.38, 5.51, 58.49]}
df=pd.DataFrame(d)
plt.subplot(3, 1, 1)
plt.plot(df["Data Size"], df["Creation"], 'o-')
plt.title('Creation')
plt.xlabel('Size of Data')
plt.ylabel('Time')

plt.subplot(3, 1, 2)
plt.plot(df["Data Size"], df["Sort"], '-.')
plt.xlabel('Size of Data')
plt.ylabel('Time')
plt.title('Bash Sort')

plt.subplot(3, 1, 3)
plt.plot(df["Data Size"], df["Sort-Python"], '-.')

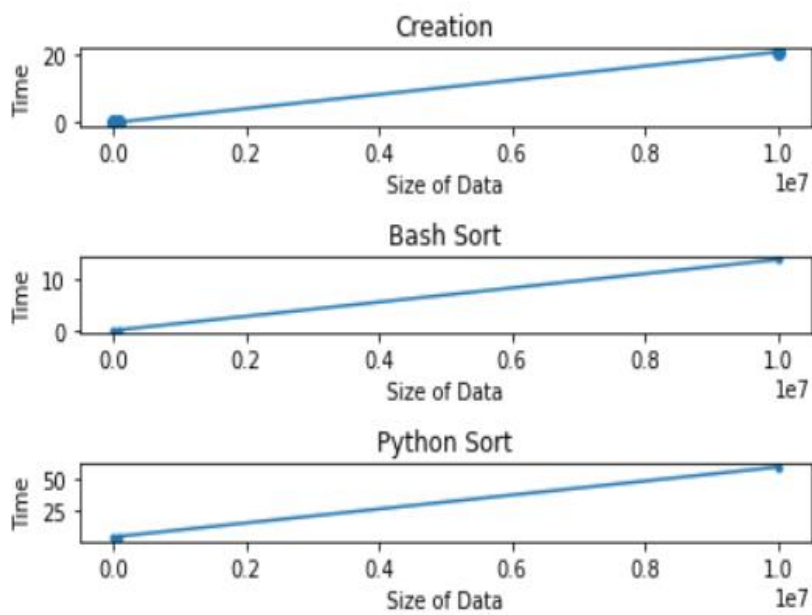
```

```
plt.xlabel('Size of Data')
plt.ylabel('Time')
plt.title('Python Sort')
```

```
plt.tight_layout()
plt.show()
```

To run the above code a file must be created with .py extension say, plots.py and to run the program we need to type python3 file path as the command.

The screenshot below shows the plots.



Execution Part-2

The second part of the projects concentrates on generating data using gensort and sort it with various levels of multithreading.

Version 1:

This is a simple JAVA sorting using the bubble sort algorithm and reading is done by scanning one line at a time to the memory.

```
import java.util.*;
```

```
import java.io.*;
```

```
class first
```

```
{
```

```
    public static void swap(int i1, int i2, String[] array)
```

```
    {
```

```
        String temp=array[i1];
```

```
        array[i1]=array[i2];
```

```
        array[i2]=temp;
```

```
    }
```

```
    public static boolean compare(String s1, String s2)
```

```
    {
```

```
        int i=0;
```

```
        while(i<10)
```

```
        {
```

```
            if(s1.charAt(i)>s2.charAt(i))
```

```
            {
```

```
                //System.out.println("After compare returning True");
```

```
                //System.out.println(s1+ "===" + s2);
```

```
                return true;
```

```
        }  
        i++;  
  
    }  
    //System.out.println("After compare returning false");  
    //System.out.println(s1+ "==" + s2);  
    return false;  
}
```

```
public static String[] bubbleSort(String[] array, int num_lines)  
{  
    for(int i=0; i<num_lines-1; i++)  
    {  
        for(int j=0; j<num_lines-i-1; j++)  
        {  
            if(compare(array[j], array[j+1]))  
                swap(j, j+1, array);  
        }  
    }  
    return array;  
}
```

```
public static void main(String args[])  
{  
    Scanner sc=new Scanner(System.in);  
    System.out.println("Enter the name of the file");  
    String fname;
```



```

int num_lines=0;

List<String> lines_t=new ArrayList<String>();

fname=sc.nextLine();

try
{
    File my_file=new File(fname);

    Scanner reader= new Scanner(my_file);

    while (reader.hasNextLine())
    {
        String data=reader.nextLine();

        lines_t.add(data);
    }
}

catch (FileNotFoundException e)
{
    System.out.println("Error has occured");
}

String[] lines=lines_t.toArray(new String[lines_t.size()]);

num_lines=lines_t.size();

String[] sorted_lines=bubbleSort(lines, num_lines);

String outFileName=fname.substring(0, fname.length()-4)+"sortedv1.txt";

try
{
    FileWriter f=new FileWriter(outFileName);

    //String newLine= System.getProperty("line.seperator");

```

```

for(int t=0; t<num_lines; t++)
{
    f.write(sorted_lines[t] + "\n");
}
f.close();
}
catch( IOException e)
{
    System.out.println("Some Error has occurred");
}
}
}

```

Version 2:

This is a simple JAVA sorting using the bubble sort algorithm and reading is done by scanning a chunk of lines at a time to the memory.

```

import java.util.*;
import java.io.*;
class second
{
public static void swap(int i1, int i2, String[] array)
{
    String temp=array[i1];
    array[i1]=array[i2];
    array[i2]=temp;
}
}

```

```

public static boolean compare(String s1, String s2)
{
    int i=0;
    while(i<10)
    {
        if(s1.charAt(i)>s2.charAt(i))
        {
            return true;
        }
        i++;
    }
    return false;
}

```

```

public static String[] bubbleSort(String[] array, int num_lines)
{
    for(int i=0; i<num_lines-1; i++)
    {
        for(int j=0; j<num_lines-i-1; j++)
        {
            if(compare(array[j], array[j+1]))
                swap(j, j+1, array);
        }
    }
    return array;
}

```

```

public static void main(String args[])
{
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter the name of the file");
    String fname;
    fname=sc.nextLine();
    List<String> lines_t=new ArrayList<String>();

    try
    {
        InputStream my_file=new FileInputStream(fname);
        InputStreamReader reader= new InputStreamReader(my_file);
        try
        {
            while (true)
            {
                int size=1000;
                char[] buffer=new char[size];
                StringBuffer strbuf=new StringBuffer();
                int read= reader.read(buffer, 0, size);
                if( read==-1)
                {
                    break;
                }

                strbuf.append(buffer,0, read);
                int temp = 0;

```

```

        while(temp < 10){
            lines_t.add(strbuf.substring(0 + temp*100, 99+ temp*100));
            temp++;

        }
    }
}
catch (IOException e)
{
    System.out.println("Error has occurred");
}
}
catch (FileNotFoundException e)
{
    System.out.println("Error has occurred");
}

String[] lines = lines_t.toArray(new String[lines_t.size()]);
int num_lines;
num_lines=lines_t.size();
String[] sorted_lines=bubbleSort(lines, num_lines);
String outFileName=fname.substring(0, fname.length()-4)+"sorted";
try
{
    FileWriter f=new FileWriter(outFileName);
    //String newLine= System.getProperty("line.seperator");
    for(int t=0; t<num_lines; t++)
    {

```

```

        f.write(sorted_lines[t] + "\n");
    }
    f.close();
}
catch( IOException e)
{
    System.out.println("Some Error has occured");
}
}
}
}

```

Version 3:

This version is sorting using merge sort algorithm and reading is done by scanning a chunk of lines at a time to the memory.

```

import java.util.*;
import java.io.*;
class third
{
    public static void swap(int i1, int i2, String[] array)
    {
        String temp=array[i1];
        array[i1]=array[i2];
        array[i2]=temp;
    }

    public static boolean compare(String s1, String s2)
    {
        int i=0;

```

```
while(i<10)
{
    if(s1.charAt(i)>s2.charAt(i))
    {
        return true;
    }
    i++;
}
return false;
}
```

```
public static String[] bubbleSort(String[] array, int num_lines)
{
    for(int i=0; i<num_lines-1; i++)
    {
        for(int j=0; j<num_lines-i-1; j++)
        {
            if(compare(array[j], array[j+1]))
                swap(j, j+1, array);
        }
    }
    return array;
}
```

```
public static void process_chunk(String fname, int i, int chunk_size)
{
    try
    {
        byte[] b=new byte[1000];
```

```

char[] buff=new char[1000];

try
{
    RandomAccessFile file = new RandomAccessFile(fname, "r");
    file.seek(i*chunk_size);
    file.read(b,0, 1000);
    int tt=0;
    for(byte t: b)
    {
        buff[tt]=(char)t;
        tt++;
    }
}
catch (IOException e)
{
    System.out.println("Error");
}

String[] lines =new String[10];
int num_lines=chunk_size/100;
for(int temp=0;temp<num_lines;temp++)
{
    String s_temp="";
    for(int j=0+temp*100;j<100+(temp*100)-1;j++)
    {
        s_temp+=buff[j];
    }
    lines[temp]=s_temp;
}

```



```

    }
String[] sorted_lines=bubbleSort(lines, num_lines);
String outFile_name=fname.substring(0, fname.length()-4)+"sortedv3_"+ i+ ".txt";

    try
    {
        FileWriter f=new FileWriter(outFile_name);
        try
        {
            for(int t=0; t<num_lines; t++)
            {
                f.write(sorted_lines[t] +"\n");
            }
            f.close();
        }
        catch( IOException e)
        {
            System.out.println("Some Error has occurred");
        }
    }
    catch (FileNotFoundException e)
    {
        System.out.println("Some Error" + e);
    }
}

catch (Exception e)
{

```

```

        System.out.println(e);
    }
}
public static String smallest(String[] latest_element, int chunk_num, File[] files_array, int n)
{
    String current_smallest_element=latest_element[0];
    int index=0;
    for(int i=0;i<chunk_num;i++)
    {
        if (compare(current_smallest_element, latest_element[i]))
        {
            current_smallest_element=latest_element[i];
            index=i;
        }
    }
    try
    {
        Scanner reader= new Scanner(files_array[index]);
        if (reader.hasNextLine())
        {
            latest_element[index]=reader.nextLine();
        }
        else
        {
            latest_element[index]=null;
            n--;
        }
    }
}

```

```

        catch (FileNotFoundException e)
        {
            System.out.println("Error");
        }

        return current_smallest_element;
    }
    public static void merge_chunks(String sorted_fname, int chunk_num)
    {
        File[] files_array=new File[chunk_num];
        for(int i=0; i<chunk_num;i++)
            files_array[i]=new File(sorted_fname+"_"+i+".txt");

        String[] latest_element=new String[chunk_num];
        try
        {
            for(int i=0;i<chunk_num;i++)
            {
                Scanner reader= new Scanner(files_array[i]);
                if (reader.hasNextLine())
                {
                    latest_element[i]=reader.nextLine();
                }
            }
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
    }

```

```

    }
    int n=chunk_num;
    try
    {
        while(n>0)
        {
            FileWriter f=new FileWriter(sorted_fname);
            f.write(smallest(latest_element, chunk_num, files_array,n) +"\n");
            f.close();
        }
    }
    catch( IOException e)
    {
        System.out.println("Some Error has occurred");
    }
}

```

```

public static void main(String args[])
{
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter the name of the file");
    String fname;
    fname=sc.nextLine();
    int file_length = 0;
    File file = new File(fname);
    if (file.exists())
    {

```

```

        file_length = (int) file.length();
    }

    int chunk_num = file_length/1000;
    for (int i = 0; i< chunk_num; i++)
    {
        process_chunk(fname, i, 1000);
    }

    String sorted_fname=fname.substring(0, fname.length()-4)+"sortedv3";
    merge_chunks(sorted_fname, chunk_num);
}
}

```

Version 4:

This version is sorting using merge sort algorithm and reading is done by scanning a chunk of lines at a time to the memory and using multithreading.

```
import java.util.*;
```

```
import java.io.*;
```

```
class wThreads implements Runnable
```

```

{
    private int i;

    private int chunk_size;

    private String fname;

    private int n;

    public wThreads(String fname, int i , int chunk_size, int chunk_num)
    {
        this.fname=fname;

        this.i=i;
    }
}

```

```

        this.chunk_size=chunk_size;

        this.n=chunk_num;
    }
    public void run()
    {
        byte[] b=new byte[chunk_size];
        char[] buff=new char[chunk_size];

        try
        {
            RandomAccessFile file = new RandomAccessFile(fname, "r");

            try
            {
                file.seek(i*chunk_size);
                file.read(b,0, chunk_size);

                int tt=0;
                for(byte t: b)
                {
                    buff[tt]=(char)t;
                    tt++;
                }
            }
            catch (IOException e)
            {
                System.out.println(e);
            }
        }
        catch (FileNotFoundException e)
        {

```

```

        System.out.println(e);
    }
    String[] lines =new String[10];
    int num_lines=chunk_size/100;
    for(int temp=0;temp<num_lines;temp++)
    {
        String s_temp="";
        for(int j=0+temp*100;j<100+(temp*100)-1;j++)
        {
            s_temp+=buff[j];
        }
        lines[temp]=s_temp;
    }
    String[] sorted_lines=bubbleSort(lines, num_lines);
    String outFileName=fname.substring(0, fname.length()-4)+"sorted_"+ i+ ".txt";

    try
    {
        FileWriter f=new FileWriter(outFileName);
        try
        {
            for(int t=0; t<num_lines; t++)
            {
                f.write(sorted_lines[t] +"\n");
            }
            f.close();
        }
    }

```

```

        catch( IOException e)
        {
            System.out.println("Some Error has occurred");
        }
    }
    catch (IOException e)
    {
        System.out.println("Some Error" + e);
    }
String sorted_fname=fname.substring(0, fname.length()-4)+"sortedv4";
    merge_chunks(sorted_fname);
}

```

```

public static void swap(int i1, int i2, String[] array)
{
    String temp=array[i1];
    array[i1]=array[i2];
    array[i2]=temp;
}

```

```

public static boolean compare(String s1, String s2)
{
    int i=0;
    while(i<10)
    {
        if(s1.charAt(i)>s2.charAt(i))
        {

```



```

        return true;
    }
    i++;
}
return false;
}

public static String[] bubbleSort(String[] array, int num_lines)
{
    for(int i=0; i<num_lines-1; i++)
    {
        for(int j=0; j<num_lines-i-1; j++)
        {
            if(compare(array[j], array[j+1]))
                swap(j, j+1, array);
        }
    }
    return array;
}

public static String smallest(String[] latest_element, int chunk_num, File[] files_array, int n)
{
    String current_smallest_element=latest_element[0];
    int index=0;
    for(int i=0; i<chunk_num; i++)
    {
        if (compare(current_smallest_element, latest_element[i]))
        {
            current_smallest_element=latest_element[i];
        }
    }
}

```

```

        index=i;
    }
}
try
{
    Scanner reader= new Scanner(files_array[index]);
    if (reader.hasNextLine())
    {
        latest_element[index]=reader.nextLine();
    }
    else
    {
        latest_element[index]=null;
        n--;
    }
}
catch (FileNotFoundException e)
{
    System.out.println("Error");
}

return current_smallest_element;
}

```

```

public void merge_chunks(String sorted_fname)
{
    int chunk_num=n;

```

```

File[] files_array=new File[chunk_num];

for(int i=0; i<chunk_num;i++)
    files_array[i]=new File(fname.substring(0, fname.length()-4)+"sorted_" + i+
".txt");

String[] latest_element=new String[chunk_num];

try
{
    for(int i=0;i<chunk_num;i++)
    {
        Scanner reader= new Scanner(files_array[i]);
        if (reader.hasNextLine())
        {
            latest_element[i]=reader.nextLine();
        }
    }
}
catch (IOException e)
{
    System.out.println(e);
}

try
{
    while(n>0)
    {
        FileWriter f=new FileWriter(sorted_fname);
        f.write(smallest(latest_element, chunk_num, files_array,n) +"\n");
        f.close();
    }
}

```

```

        System.out.println(n);
    }
}
catch( IOException e)
{
    System.out.println("Some Error has occurred");
}
}
}
class v4
{
    public static void main(String [] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the name of the file");
        String fname;
        fname=sc.nextLine();
        int file_length = 0;
        File file = new File(fname);
        if (file.exists())
        {
            file_length = (int) file.length();
        }
        int chunk_size=200;
        int max_threads=8;
        int chunk_num = file_length/(chunk_size);
        int cur_chunk=0;
        Thread[] myThreads= new Thread[max_threads];
    }
}

```

```

while(cur_chunk<chunk_num)
{
    for(int ti=0; ti<max_threads && cur_chunk<chunk_num; ti++)
    {
        myThreads[ti]=new Thread(new wThreads(fname, cur_chunk, chunk_size,
chunk_num));

        myThreads[ti].start();

        cur_chunk++;

    }
    for(int ti=0; ti<max_threads && cur_chunk<chunk_num; ti++)
    {
        try
        {
            myThreads[ti].join();
        }
        catch(InterruptedException e)
        {
            System.out.println("Error" + e);
        }
    }
}

String sorted_fname=fname.substring(0, fname.length()-4)+"sortedv4";
}
}

```

There is a significant decrease in time for each version starting from version 1 to version 4.

Execution Part 3

In the third and final part of the project we work on the various sorting techniques in python including those in pandas, numpy, tensorflow and spark and plot a graph for various sizes of data sets.

```
#!/usr/bin/bash

import numpy as np

import pandas as pd

import numpy.random as r

import time

import os.path

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

import torch

method=[]

time_data=[]

size=int(input("Enter the size of the data set: "))

a=np.random.randint(low=1, high=size+1, size=(size))

l=a.tolist()

st=time.time()

sorted_=sorted(l)

t=time.time()-st

print("Time taken to Vanilla Sort copy: ", t)

method.append("Vanilla-copy")
```

```
time_data.append(t)
```

```
st=time.time()
```

```
l.sort()
```

```
t=time.time()-st
```

```
print("Time taken to Vanilla Sort in-place: ", t)
```

```
method.append("Vanilla-in-place")
```

```
time_data.append(t)
```

```
print("\n")
```

```
a=np.random.randint(low=1, high=size+1, size=(size))
```

```
st=time.time()
```

```
sorted_l=np.sort(a)
```

```
t=time.time()-st
```

```
print("Time taken to Numpy sort copy is: ", t)
```

```
method.append("Numpy-copy")
```

```
time_data.append(t)
```

```
st=time.time()
```

```
sorted_l=a.sort()
```

```
t=time.time()-st
```

```
print("Time taken to Numpy sort in-place is: ", t)
method.append("Numpy-in-place")
time_data.append(t)

print("\n")

a=np.random.randint(low=1, high=size+1, size=(size))

st=time.time()
sorted_l=np.sort(a, kind='quicksort')
t=time.time()-st
print("Time taken to Numpy Quick sort copy is: ", t)
method.append("Numpy-Quick-copy")
time_data.append(t)

st=time.time()
sorted_l=a.sort(kind='quicksort')
t=time.time()-st
print("Time taken to Numpy Quick sort in-place is: ", t)
method.append("Vanilla-Quick-In-place")
time_data.append(t)

print("\n")
```



```
a=np.random.randint(low=1, high=size+1, size=(size))
```

```
st=time.time()
```

```
sorted_l=np.sort(a, kind='mergesort')
```

```
t=time.time()-st
```

```
print("Time taken to Numpy Merge sort copy is: ", t)
```

```
method.append("Numpy-Merge-copy")
```

```
time_data.append(t)
```

```
st=time.time()
```

```
sorted_l=a.sort(kind='mergesort')
```

```
t=time.time()-st
```

```
print("Time taken to Numpy Merge sort in-place is: ", t)
```

```
method.append("Numpy-Merge-in-place")
```

```
time_data.append(t)
```

```
print("\n")
```

```
a=np.random.randint(low=1, high=size+1, size=(size))
```

```
st=time.time()
```

```
sorted_l=np.sort(a,kind='heapsort')
t=time.time()-st
print("Time taken to Numpy Heap sort copy is: ", t)
method.append("Numpy-Heap-copy")
time_data.append(t)
```

```
st=time.time()
sorted_l=a.sort(kind='heapsort')
t=time.time()-st
print("Time taken to Numpy Heap sort in-place is: ", t)
method.append("Numpy-Heap-in-place")
time_data.append(t)
```

```
print("\n")
```

```
a=np.random.randint(low=1, high=size+1, size=(size))
d=pd.DataFrame(a, columns=["Integers"])
```

```
st=time.time()
sorted_d=d.sort_values(by="Integers")
t=time.time()-st
print("Time taken to Pandas sort copy is: ", t)
method.append("pandas-copy")
time_data.append(t)
```

```
st=time.time()
sorted_d=d.sort_values(by="Integers", inplace=True)
t=time.time()-st
print("Time taken to Pandas sort in-place is: ", t)
method.append("pandas-in-place")
time_data.append(t)
```

```
print("\n")
```

```
a=np.random.randint(low=1, high=size+1, size=(size))
d=pd.DataFrame(a, columns=["Integers"])
```

```
st=time.time()
sorted_d=d.sort_values(by="Integers", kind='quicksort' )
t=time.time()-st
print("Time taken to Pandas quick sort copy is: ", t)
method.append("pandas-Quick-copy")
time_data.append(t)
```

```
st=time.time()
sorted_d=d.sort_values(by="Integers", inplace=True, kind='quicksort')
t=time.time()-st
```

```
print("Time taken to Pandas quick sort in-place is: ", t)
method.append("pandas-Quick-in-place")
time_data.append(t)
```

```
print("\n")
```

```
a=np.random.randint(low=1, high=size+1, size=(size))
d=pd.DataFrame(a, columns=["Integers"])
```

```
st=time.time()
sorted_d=d.sort_values(by="Integers", kind='heapsort' )
t=time.time()-st
print("Time taken to Pandas heap sort copy is: ", t)
method.append("pandas-Heap-copy")
time_data.append(t)
```

```
st=time.time()
sorted_d=d.sort_values(by="Integers", inplace=True, kind='heapsort')
t=time.time()-st
print("Time taken to Pandas heap sort in-place is: ", t)
method.append("pandas-heap-in-place")
time_data.append(t)
```

```
print("\n")
```

```
a=np.random.randint(low=1, high=size+1, size=(size))
```

```
d=pd.DataFrame(a, columns=["Integers"])
```

```
st=time.time()
```

```
sorted_d=d.sort_values(by="Integers", kind='mergesort' )
```

```
t=time.time()-st
```

```
print("Time taken to Pandas merge sort copy is: ", t)
```

```
method.append("pandas-merge-copy")
```

```
time_data.append(t)
```

```
st=time.time()
```

```
sorted_d=d.sort_values(by="Integers", inplace=True, kind='mergesort')
```

```
t=time.time()-st
```

```
print("Time taken to Pandas merge sort in-place is: ", t)
```

```
method.append("pandas-merge-in-place")
```

```
time_data.append(t)
```

```
print("\n")
```

```
a=np.random.randint(low=1, high=size+1, size=(size))
```

```
value_1 = tf.convert_to_tensor(a, dtype=tf.int64)
method.append("Tensor flow")
```

```
st=time.time()
tf.sort(value_1)
t=time.time()-st
print("Time taken to sort using Tensor: ", t)
time_data.append(t)
```

```
print("\n")
```

```
a=np.random.randint(low=1, high=size+1, size=(size))
value_1 = tf.convert_to_tensor(a, dtype=tf.int64)
with tf.device('/GPU:0'):
    st=time.time()
    tf.sort(value_1)
    t=time.time()-st
    print("TF with GPU", t)
    method.append("Tensor_flow GPU")
    time_data.append(t)
```

```
with tf.device('/CPU:0'):
    st=time.time()
    tf.sort(value_1)
    t=time.time()-st
```

```

print("TF with CPU", t)

method.append("Tensor_flow CPU")

time_data.append(t)

a=np.random.randint(low=1, high=size+1, size=(size))

value_1=torch.from_numpy(a)

st=time.time()

torch.sort(value_1)

t=time.time()-st

print("Torch CPU", t)

method.append("Torch CPU")

time_data.append(t)

gpu_tensor=value_1.cuda()

st=time.time()

torch.sort(gpu_tensor)

t=time.time()-st

print("Torch GPU", t)

method.append("Torch GPU")

time_data.append(t)

df=pd.DataFrame({"Names":method, "Time":time_data})

f = plt.figure()

f.set_figwidth(15)

f.set_figheight(5)

plt.bar(df["Names"], df["Time"], width=0.5)

plt.xticks(

```

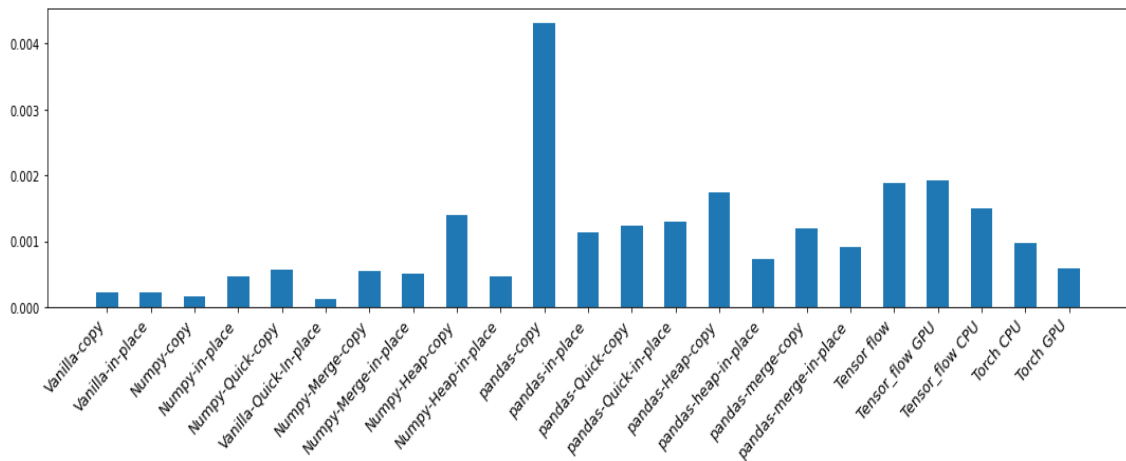
```

rotation=45,
horizontalalignment='right',
fontweight='light',
fontsize='large'
)
plt.tight_layout()
plt.show()

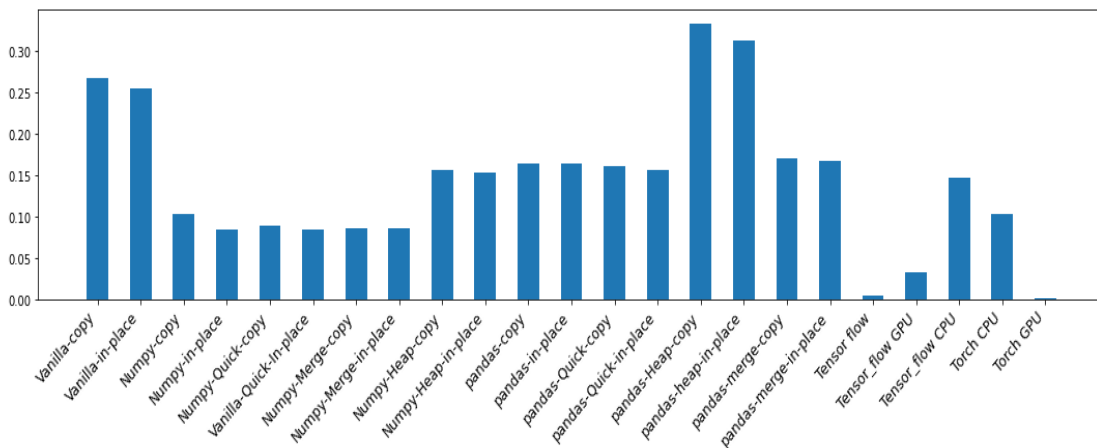
```

The above up on running on data sets of different sizes ranging from 1000 to 100000000, gives the following plots.

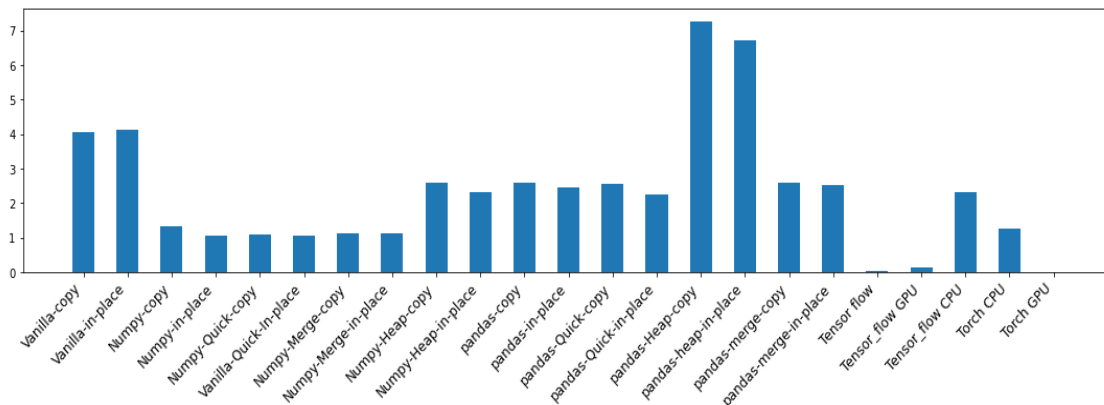
Data set of 1000 elements(1k):



Data set of 100000 elements(1M):



Data set of 1000000 elements(10M):



Data set of 10000000 elements(100M):

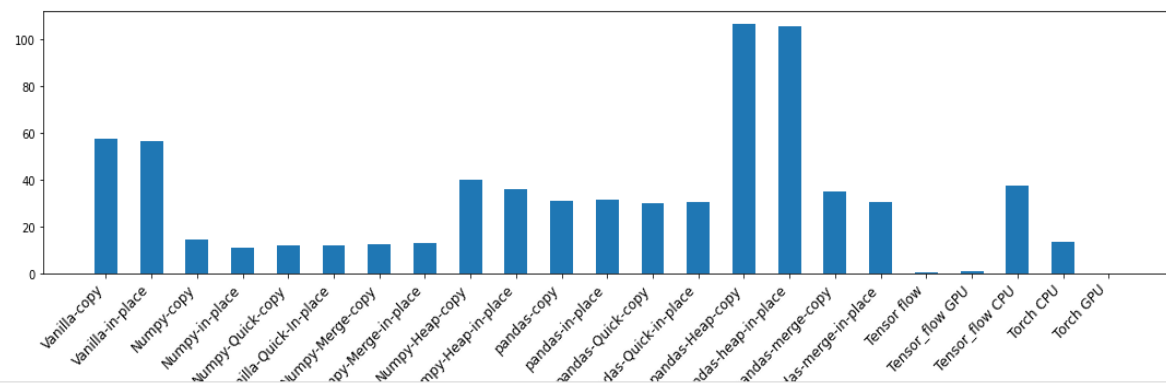
Time taken to sort using Tensor: 0.36566734313964844

TF with GPU 0.9038710594177246

TF with CPU 37.63810658454895

Torch CPU 13.466159343719482

Torch GPU 0.01686263084411621



It can be hence concluded that sorting using Torch-GPU, TensorFlow are some of the best available sorting techniques.

Thank you



MaViSS AI

Artificial Intelligence based COVID19
Norms Machine Vision Surveillance
System

Arpan Kundu

(Team Members:

Arpan Kundu, Ritika Nigam)

Advisor: Dr. Jafar Saniie

Summer 2021



CONTENTS

CONTENTS	1
BACKGROUND	2
ABSTRACT	2
INTRODUCTION	3
SYSTEM MODULES	4
SYSTEM WORKFLOW	10
HARDWARE IMPLEMENTATION	11
SOFTWARE IMPLEMENTATION	13
RESULTS AND DISCUSSION	15
REAL-TIME CONSTRAINTS AND FEASIBILITY	18
SECURITY ISSUES	19
CONCLUSION	19
FUTURE WORK	20
REFERENCES	20
APPENDICES	21



BACKGROUND

The ongoing COVID-19 coronavirus outbreak has caused a global disaster with its deadly spreading. Due to the absence of effective remedial agents and the shortage of immunizations against the virus, population vulnerability increases. Though vaccines have been developed by various nations, but as suggested by the **World Health Organization (WHO)**, vaccines rarely protect 100% of the recipients and vaccinated individuals still run the risk of contracting the disease. Consequently, all additional precautions against the epidemic should be carefully considered.



ABSTRACT

Manually monitoring whether all the necessary precautions are being followed by the people in public areas is tedious, inefficient and often inaccurate. Thus, this brings out the aim of our project - an **automated machine vision surveillance system for real-time** monitoring of COVID-19 norms which is **cost effective, accurate, feasible** and **secure** and overcomes the real time challenges faced during manual monitoring of norms.

Our proposed system -

- Detects and tracks humans for monitoring social distancing and keeps track of the human count for crowd management,
- Detects face mask and keeps track of face mask usage, and
- Sends alerts in real-time directly to the (monitoring authority's) smartphone whenever the norms are breached.

In view of these alerts, security personnel can take relevant actions. Therefore, our proposed automated surveillance system surpasses several limitations of the manual monitoring systems.

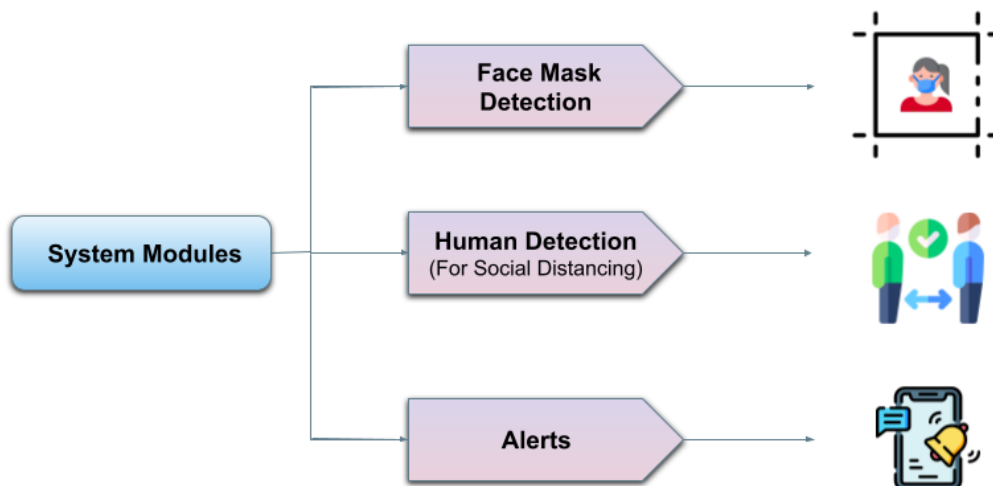
INTRODUCTION

Since the outbreak of the pandemic, many systems have been developed to monitor different aspects of COVID-19 norms like face mask usage and social distancing, but most of them have addressed only specific aspects of the norms and do not have a holistic approach. Moreover, these systems mainly focus on visualizing the system output but do not have any alerts component to keep track of violations in norms and notify the monitoring user about the same.

Our proposed system has a more holistic approach to address these shortcomings and offers a complete platform for monitoring the COVID-19 norms, as well as sends alerts in realtime directly to the monitoring user's smartphone using an instant messaging service like Telegram.

This solution is comprehensive, feasible and fast as well as secure to use.

Our proposed system is composed of three modules:



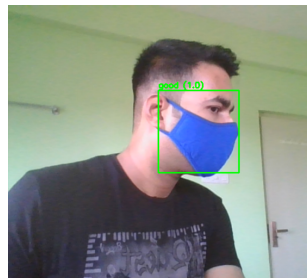
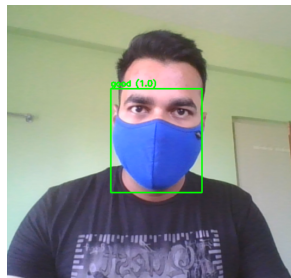
SYSTEM MODULES

- Face Mask Detection

The face mask detection module detects people's faces, checks their face mask usage and classifies it into one of the following three categories:

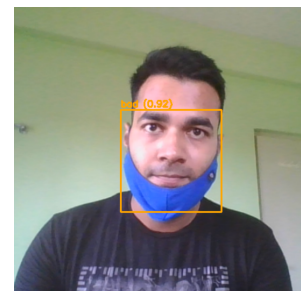
GOOD

Enclosed within a green bounding box with a 'good' remark, it represents that the person is **properly masked**.



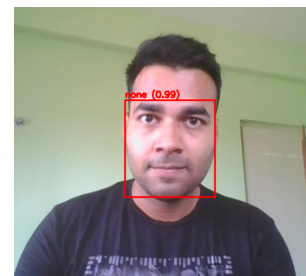
BAD

Enclosed within an orange bounding box with a 'bad' remark, it represents that the person is **improperly masked**, i.e. although he/she is wearing a mask, it is not covering his/her nose or mouth properly.



NONE

Enclosed within a red bounding box with a 'none' remark, it represents that the person is **not masked**.



This module uses the **mask-YOLOv4-tiny** model which is a neural network based on the [darknet](#) framework and [YOLOv4](#) architecture. This neural network is trained by [cansik](#) and the dataset for this pre-trained network (consisting of 678 images of people with and without masks) is provided by [VictorLin000](#).

This model was trained on a 1080TI for about 2 hours over 6000 iterations with a batch size of 64 and 16 subdivisions. The weights were trained on an image size of 416x416. Complete details of training can be found [here](#).



Fig. Sample test run of Face Mask Detection module

The code for the face mask detection module is attached in [Appendix A](#). We tested this module on our PCs and the Jetson Nano and achieved the following performance figures:

Model	FPS Range (Jetson Nano)		FPS Range (PC)
	CPU	GPU	CPU
mask-YOLOv4-tiny	1.37-1.77	3-6	6.5-8.5

Table. Face Mask Detection module performance figures

The model this module is built on, i.e. YOLOv4-tiny has a mean Average Precision (mAP) of 40.2%. Precision is defined as:

$$Precision = TP / (TP + FP)$$

Where, TP is the number of true positives, i.e. the number of correct predictions and FP is the number of false positives, i.e. the number of incorrect predictions. mean Average Precision (mAP) is the average of all the average precisions of the classes in the given dataset. Thus, this figure is a measure of the accuracy of detection of the given detection algorithm.

The frame rate is measured in frames per second (FPS). The hardware configurations of the CPU and GPU of the Jetson Nano and the PC used in the above test runs are mentioned in the [HARDWARE IMPLEMENTATION](#) section.

- **Human Detection**

The human detection module detects humans keeping a human count (for crowd management), calculates the distance between each pair of detected people (for social distancing) and then classifies each person into one of the following three colors of bounding boxes:

GREEN

Represents that the person is at a safe distance* from others (**No Violations**).



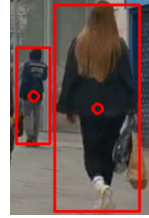
YELLOW

Represents that the person is at the minimum safe distance** from others, but not at a safe distance* from others (**Abnormal Violation**).



RED

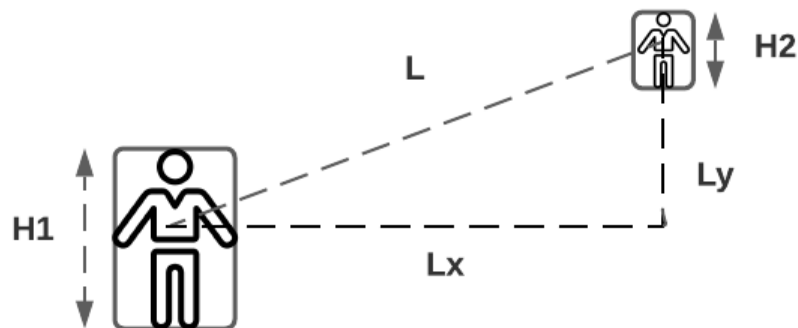
Represents that the person is not at the minimum safe distance** from others (**Serious Violation**).



**Minimum safe distance : 1 metre (as set by WHO)
*Safe distance : 2 metres (as set by several countries)

This module uses the **YOLOv3-608** model which is a neural network based on the [darknet](#) framework and [YOLOv3](#) architecture. This pre-trained neural network was trained on the COCO (Common Objects in Context) dataset and can detect 80 different classes of objects. In our case, it is used to detect humans.

DISTANCE CALCULATION



Firstly, the Euclidean distance **L** is calculated between the two persons (in pixels).

$$L = \sqrt{(Lx)^2 + (Ly)^2}$$

Next, this distance is calibrated into metres by multiplying it with the calibration factor **k**.

$$k = ((\frac{1}{H1} + \frac{1}{H2})/2) * H$$

Where,

H1 and H2 are the heights of the two persons (in pixels) respectively and H is the average height of a person (in centimeters), which we have assumed to be 170 cm.

Finally, the calibrated distance D (in cm). Is obtained as:

$$D = k * L$$

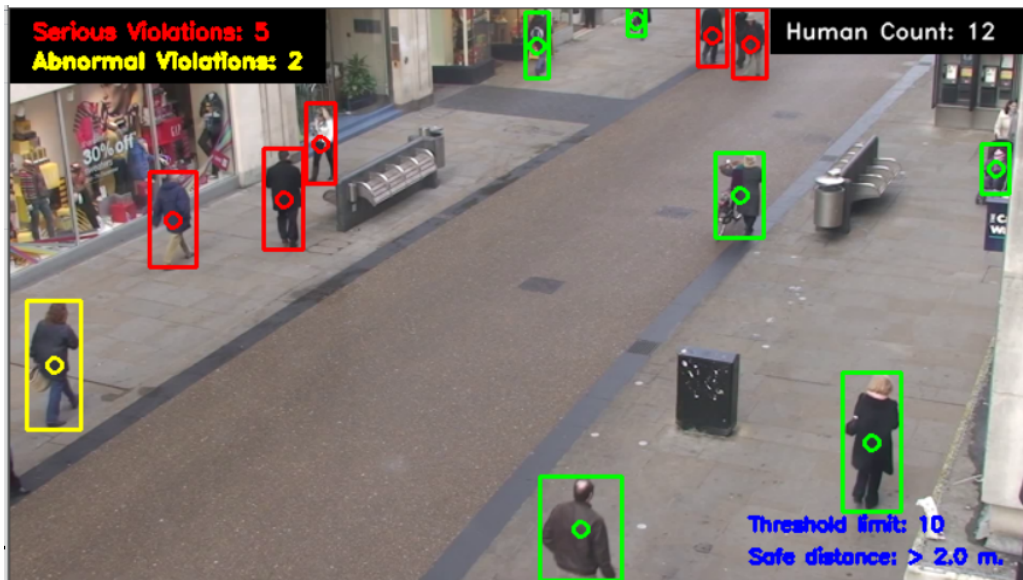


Fig. Sample test run of Human Detection module

The code for the human detection module is attached in [Appendix B](#). We tested this module on our PCs and the Jetson Nano and achieved the following performance figures:

Model	FPS Range (Jetson Nano)		FPS Range (PC)
	CPU	GPU	CPU
YOLOv3-608	0.21	0.71-0.79	1.5-2

Table. Human Detection module performance figures

The model this system is built on, i.e. YOLOv3 has a mean Average Precision (mAP) of 57.9%.

The frame rate is measured in frames per second (FPS). The hardware configurations of the CPU and GPU of the Jetson Nano and the PC used in the above test runs are mentioned in the [HARDWARE IMPLEMENTATION](#) section.

- **Alerts**

The alerts module sends alert messages in realtime to the monitoring user through a Telegram bot, whenever a COVID-19 norms violation (in social distancing or face mask usage) is detected by the system.

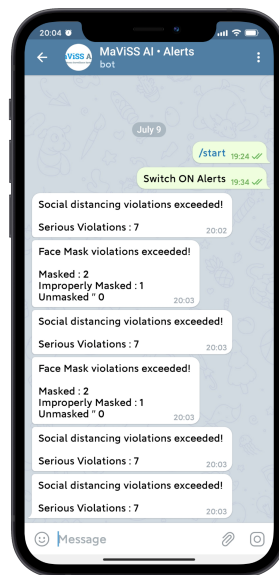


Fig. Telegram bot sending alert messages

This module consists of a trigger function containing a personalized alert message for different norms violations cases along with count statistics. This function sends a GET request (containing the alert message) to the Telegram bot's server, which in turn conveys the alert message to the monitoring user.

The code for the alerts module is attached in [Appendix C](#).



SYSTEM WORKFLOW

Our system workflow has six main steps or phases:

1. **Video** : Video is captured from a source like an IP Camera, CCTV or Webcam. Frames are extracted from this video source.
2. **Preprocessing** : Preprocessing is done on these received frames and they are resized for model inference.
3. **Model Inference** : Model Inference is done by using the YOLO architecture neural network for state-of-the-art, real-time humans and face mask detection.
4. **Calibration** : Calibration involves computing parameters like social distancing & face mask metrics, validating it with the norms and identifying violations.
5. **Output** : Output is generated in real-time to the monitoring user, displaying the social distancing metrics, color coded bounding boxes for persons detection & tracking and face mask usage, and information regarding any violations.
6. **Alerts** : Alerts are sent directly to the monitoring user's smartphone in real-time through a Telegram bot.

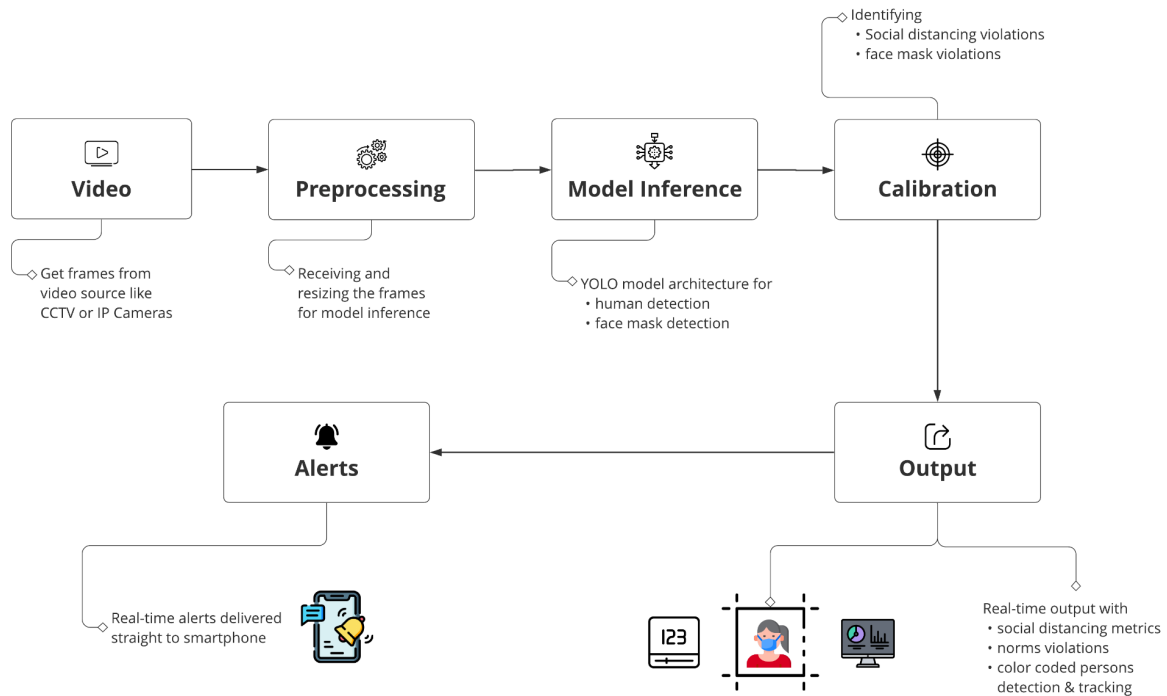


Fig. System Workflow

HARDWARE IMPLEMENTATION

The hardware components used in this project are summarized in the table below:

Item Description	Units	Configuration
Nvidia Jetson Nano Development Board	1	128 Cuda Cores, Cortex®-A57, 4 GB 64-Bit LPDDR4
IMX 219-77 Camera	1	8 MP, 3280 x 2464 resolution, 77° FOV
Personal Computer (used for additional testing)*	1	Intel Core i5 8th Gen @ 1.6 GHz, 8 GB 64-Bit DDR4
External Monitor*	1	-

USB Wireless Keyboard	1	-
USB Wireless Mouse	1	-
MicroSD Card (with adapter)	1	32 GB UHS-1 (minimum)
USB WiFi Adapter (or Ethernet cable for connecting Jetson Nano to Internet)	1	-
HDMI Cable	1	-
AC/DC Power Supply	1	5V, 4A

Table. Hardware Components

Complete setup cost : 150-180 USD (* items not included in this estimate).

HARDWARE SPECIFICATIONS

- Nvidia Jetson Nano Developer Kit

GPU 128-core Maxwell
CPU Quad-core ARM A57 @ 1.43 GHz
Memory 4 GB 64-bit LPDDR4 25.6 GB/s
Storage microSD (not included)
Video Encode 4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode 4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30
(H.264/H.265)
Camera 2x MIPI CSI-2 DPHY lanes
Connectivity Gigabit Ethernet, M.2 Key E
Display HDMI and display port
USB 4x USB 3.0, USB 2.0 Micro-B
Others GPIO, I2C, I2S, SPI, UART
Mechanical 69 mm x 45 mm, 260-pin edge connector

Complete Description at [Nvidia Jetson Nano Developer Kit](#).

CIRCUIT DIAGRAM

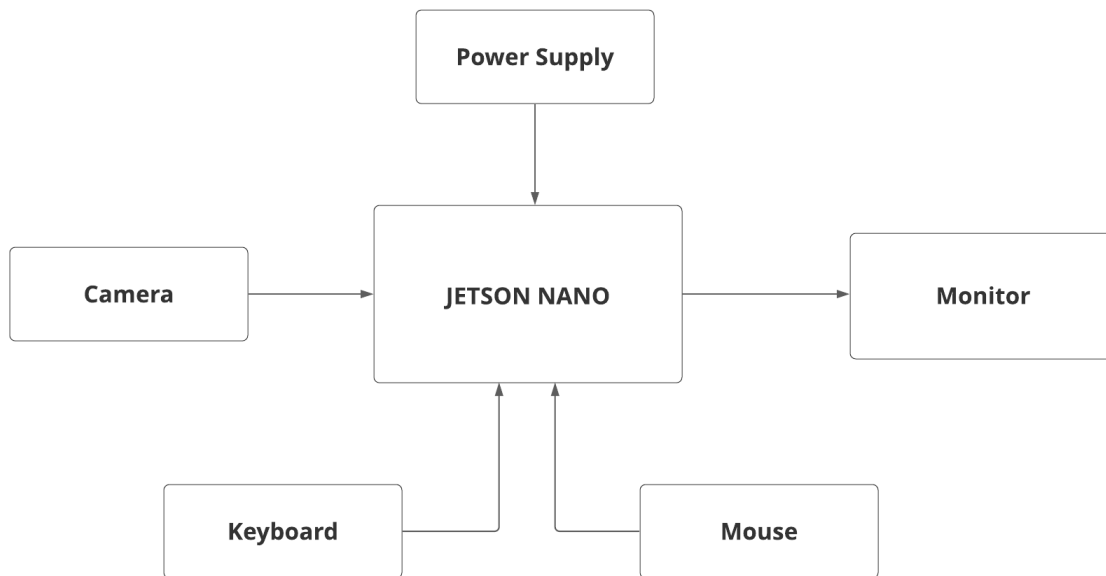


Fig. Block Circuit Diagram of System Setup

SOFTWARE IMPLEMENTATION

- PROGRAMMING LANGUAGE 

The project is coded in the **Python 3.7** programming language. Python is a simple, consistent and open source programming language, offering concise and readable code. The simplicity of syntax allows developers to focus on solving the system problem rather than the technical nuances of the language.

Additionally, Python also offers a range of frameworks and libraries that enable developers to solve common programming tasks. It has a rich technology stack specially for artificial intelligence and machine learning, some of which we have implemented in our project.

Apart from these, Python is also platform independent allowing developers to implement things on one machine and use them on another machine without any (or with only minimal) changes. Lastly, it has a huge open source developer community that enables budding developers to learn and get support all through their project or programming journey.

All of these factors make Python an ideal choice for building our project.

- **LIBRARIES**

→ OpenCV and imutils



The OpenCV computer vision library is the main library used in our project. This project uses **OpenCV 4.5.2** and **imutils** for the video and image processing tasks. Also, the **OpenCV DNN (Deep Neural Networks)** module is used to facilitate the deep learning inference on the videos/live streams we are processing. This module is compatible with the YOLO (You Only Look Once) architecture that forms our main detection model.

The OpenCV 4.5.2 version we're using was compiled with **CUDA backend support**, in order to utilize the GPU capabilities of the Nvidia Jetson Nano. The details of the compilation are summarized below:

```
jtop Nano (Developer Kit Version) - JC: Inactive - MAXN
NVIDIA Jetson Nano (Developer Kit Version) - Jetpack 4.5.1 [L4T 32.5.1]
- Up Time: 0 days 0:25:23 Version: 3.1.0
- Jetpack: 4.5.1 [L4T 32.5.1] Author: Raffaello Bonghi
- Board: e-mail: raffaello@rnext.it
  * Type: Nano (Developer Kit Version)
  * SOC Family: tegra210 ID: 33
  * Module: P3448-0000 Board: P3449-0000
  * Code Name: porg
  * Cuda ARCH: 5.3
  * Serial Number: 1420521018379
  * Board ids: 3448
- Libraries:
  * CUDA: 10.2.89 - Hostname: ritika-desktop
  * OpenCV: 4.5.2 compiled CUDA: YES * Interfaces:
  * TensorRT: 7.1.3.0 * wlan0: 192.168.29.228
  * VPI: ii libnvvp1 1.0.15 arm64 NVIDIA Vision Programming Interface
  * VisionWorks: 1.6.0.501
  * Vulkan: 1.2.70
  * cuDNN: 8.0.0.180
1ALL 2GPU 3CPU 4MEM 5CTRL 6INFO Quit Raffaello Bonghi
```

Fig. OpenCV with CUDA support

→ SciPy and Numpy



SciPy and Numpy libraries were used to facilitate various scientific calculations and calibrations (e.g. distance calculation and calibration) used in our project.

→ Urllib and Requests

The urllib and requests libraries were used in the alerts module of our project for sending GET requests to the Telegram bot's server containing the alert messages.

- **DETECTION MODEL** 

The main detection model used in our project (for human and face mask detection) is built around the YOLO architecture, based on the darknet framework. You Only Look Once (YOLO) is a state-of-the-art, realtime object detection system. Darknet is an open source neural network framework written in C and CUDA, known for its speed and support for CPU and GPU computation.

The human detection module (for social distancing) of our project uses the YOLOv3-608 model and the face mask detection module uses the mask-YOLOv4-tiny model for performing their respective detection tasks. More information on this is provided in the SYSTEM MODULES section above.



RESULTS AND DISCUSSION

We tested our system on the Nvidia Jetson Nano as well as our PCs, and pre-recorded as well as live stream video sources were used.

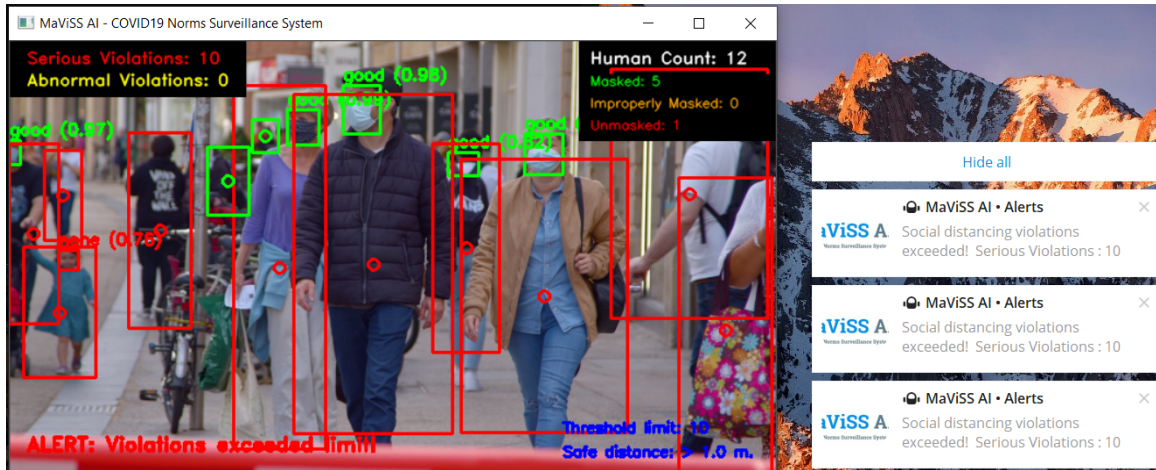


Fig. Sample test run (1)



Fig. Sample test run (2)

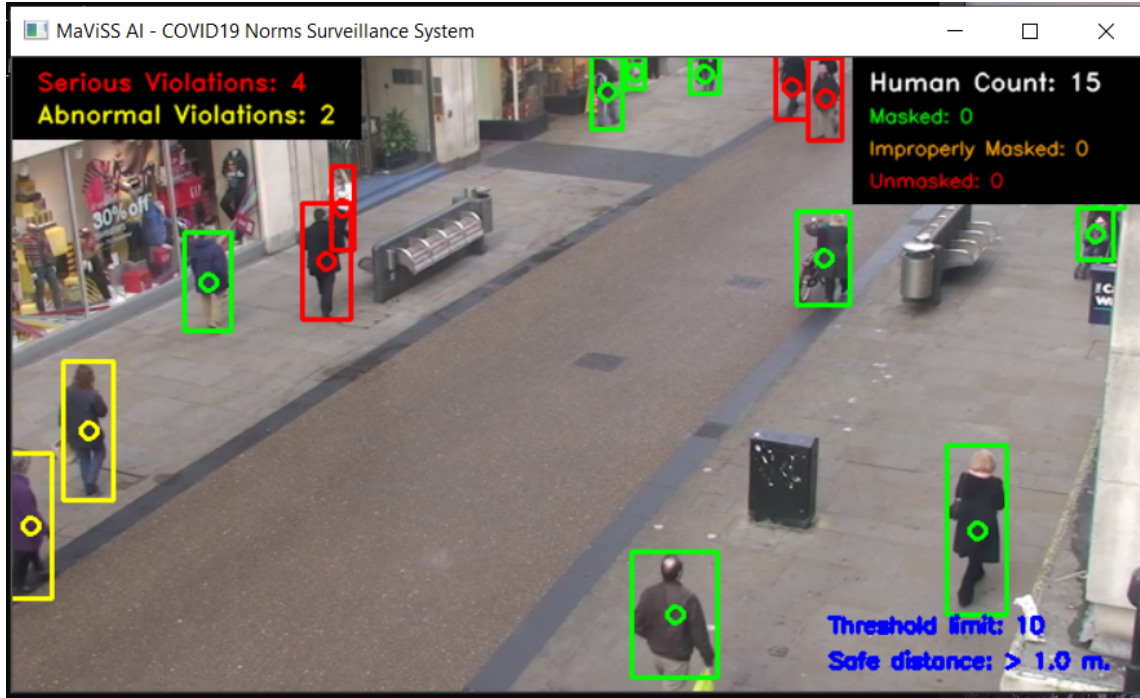


Fig. Sample test run (3)

The results obtained from these test runs are shown in our demonstration video, which can be found here - [Demonstration Video](#).

The performance figures of the system on the Jetson Nano and our PCs are as follows:

Model	FPS Range (Jetson Nano)		FPS Range (PC)
	CPU	GPU	CPU
MaViSS AI (YOLOv3-608 + mask-YOLOv4-tiny)	0.15-0.2	0.65-0.83	1.5-2

Table. System test runs performance figures

The frame rate is measured in frames per second (FPS). The hardware configurations of the CPU and GPU of the Jetson Nano and the PC used in the above test runs are mentioned in the [HARDWARE IMPLEMENTATION](#) section.

From the above performance figures we infer that our system, **MaViSS AI** utilizes the **powerful GPU of Jetson Nano with CUDA backend** to improve its performance by approximately **4 times better** than that achieved on CPU and runs at a frame rate of **0.65-0.83 FPS**.



REAL-TIME CONSTRAINTS AND FEASIBILITY

Our proposed system has major applications in tracking of COVID19 norms in busy areas like malls, streets, offices, stations, airports, etc. Such applications demand realtime performance and notifications of violations, so that immediate and swift action can be taken in response. Thus to boost the performance of the system, the GPU capabilities of the Jetson Nano have been utilised which results in an improvement of about **4 times** in the performance. However, the performance can be further improved by using more optimized detection algorithms and higher end hardware, as stated in the future work section below.

The system can be used feasibly in tracking COVID19 norms in busy outdoor areas as well as indoor environments. Moreover, the alerts feature enables the monitoring user to get personalized, realtime alert messages directly to their phones just like an instant messaging app's messages. The system, however, can be made more insightful and feasible by adding a dashboard to the output window containing different metrics of norms tracking and a better user interface.



SECURITY ISSUES

The system's workflow in terms of storing and sending tracking data to monitoring user's smartphone is secure. But, the process can be made more efficient and secure by shifting this information storage and transmission process to the cloud. A cloud infrastructure would allow for more storage of the system's generated data and also make the transmission of data as alert messages more secure. Moreover, the entire system can be mounted on the cloud, enabling the users to use the system (independent of location or device) as a Software-as-a-Service (SaaS) by means of a web application of the same.



CONCLUSION

To summarize, our proposed system **MaViSS AI** enables the user to monitor social distancing norms and face mask usage in the scene captured by the surveillance camera and any norm breach is reported directly to the user as an alert message. Thus, our system surpasses several limitations of the manual monitoring systems and provides an **efficient** and **accurate** way of monitoring and reporting breaches in COVID19 norms.

Through the course of this project we encountered many challenges and learned several new concepts and technologies related to setting up the Jetson Nano for our project, compiling OpenCV with CUDA support so as to utilize the Jetson Nano's GPU, exploring different libraries for our project and working on improving the performance and accuracy of our system. We kept our work pace steady from the beginning, sticking strictly to our timeline. Finally, we were able to develop a completely functional system capable of **monitoring and reporting breaches in COVID19 norms in various busy places to the monitoring user in realtime.**



FUTURE WORK

- System output can be made more insightful by collecting and storing the monitoring data generated, and crunching the same using different Machine Learning techniques to deliver more insightful results to the user.
- Distant faces that are not detected by the Face Mask Detection module can be handled by using advanced cameras with zoom feature and adjusting the same as per the scene to obtain frames with detectable faces.
- System performance can be improved by using higher end hardware and more optimized detection algorithms.
- Distance calculation can be made more accurate by using depth and aspect information.
- System can be made more insightful and have broader applications in curbing COVID19 spread by including a body temperature detection module using a thermal camera.



REFERENCES

- [Neuralet's smart social distancing](#)
- [J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection." 2016 IEEE Conference on Computer Vision and Pattern Recognition \(CVPR\), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.](#)
- YOLO v3 (for human detection) - <https://pjreddie.com/darknet/yolo/>
- Mask YOLO v4 tiny (for face mask detection) - <https://github.com/cansik/yolo-mask-detection>
- YOLOv4 architecture - <https://arxiv.org/abs/2004.10934>

- [M. M. Rahman, M. M. H. Manik, M. M. Islam, S. Mahmud and J. -H. Kim, "An Automated System to Limit COVID-19 Using Facial Mask Detection in Smart City Network," 2020 IEEE International IOT, Electronics and Mechatronics Conference \(IEMTRONICS\), 2020, pp. 1-5, doi: 10.1109/IEMTRONICS51293.2020.9216386.](#)
- [Ansari, M., Singh, D.K. Monitoring social distancing through human detection for preventing/reducing COVID spread. Int. j. inf. tecnol. 13, 1255-1264 \(2021\). <https://doi.org/10.1007/s41870-021-00658-2>](#)
- Jetson Nano - <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit>

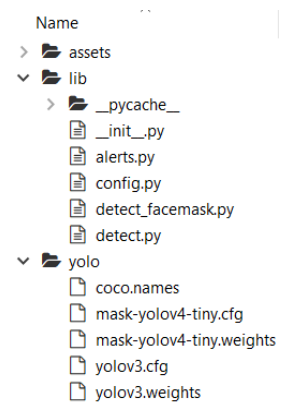


APPENDICES

FILES & FOLDERS STRUCTURE

The system folder contains 1 file - **main.py** and 3 folders - **assets**, **lib** and **yolo**.

- assets contains pre-recorded videos for testing purpose
- lib contains 5 files - **_init_.py**, **detect_facemask.py**, **detect.py**, **alerts.py** and **config.py**
- yolo contains 5 files - **coco.names**, **mask-yolov4-tiny.cfg**, **mask-yolov4-tiny.weights**, **yolov3.cfg** and **yolov3.weights**
- main.py is the main execution script integrating all the different modules of the system



APPENDIX A - detect_facemask.py

```
#####/Face Mask Detection
module\#####

# importing necessary Libraries
```



```

import time
import cv2
import numpy as np
from lib.config import Use_GPU

class DETECT_FACEMASK:

    def __init__(self, config, model, labels, size=416,
confidence=0.5, threshold=0.3):
        self.confidence = confidence
        self.threshold = threshold
        self.size = size

        self.labels = labels
        self.net = cv2.dnn.readNetFromDarknet(config, model)
        # checking if there's GPU usage
        if Use_GPU:
            # set CUDA as the preferable backend and target
            print("")
            print("[INFO] Looking for GPU")
            self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
            self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

    def inference_from_file(self, file):
        mat = cv2.imread(file)
        return self.inference(mat)

    def inference(self, image):
        ih, iw = image.shape[:2]

        ln = self.net.getLayerNames()
        ln = [ln[i[0] - 1] for i in self.net.getUnconnectedOutLayers()]

        blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (self.size,
self.size),
            swapRB=True, crop=False)
        self.net.setInput(blob)
        start = time.time()

```

```

layerOutputs = self.net.forward(ln)
end = time.time()
inference_time = end - start

boxes = []
confidences = []
classIDs = []

for output in layerOutputs:
    # Looping over each of the detections
    for detection in output:
        # extracting the class ID and confidence (i.e.,
probability) of
        # the current object detection
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]
        # filtering out weak predictions by ensuring the
detected
        # probability is greater than the minimum
probability
        if confidence > self.confidence:
            # scaling the bounding box coordinates back relative
to the
            # size of the image, keeping in mind that YOLO
actually
            # returns the center (x, y)-coordinates of the
bounding
            # box followed by the boxes' width and height
            box = detection[0:4] * np.array([iw, ih, iw, ih])
            (centerX, centerY, width, height) =
box.astype("int")
            # using the center (x, y)-coordinates to derive the
top and
            # and left corner of the bounding box
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))
            # updating our list of bounding box coordinates,

```

```

confidences,
    # and class IDs
    boxes.append([x, y, int(width), int(height)])
    confidences.append(float(confidence))
    classIDs.append(classID)

idxs = cv2.dnn.NMSBoxes(boxes, confidences, self.confidence,
self.threshold)

results = []
if len(idxs) > 0:
    for i in idxs.flatten():
        # extracting the bounding box coordinates
        x, y = (boxes[i][0], boxes[i][1])
        w, h = (boxes[i][2], boxes[i][3])
        id = classIDs[i]
        confidence = confidences[i]

        results.append((id, self.labels[id], confidence, x,
y, w, h))

return iw, ih, inference_time, results

```

APPENDIX B - detect.py

```

#=====/Human Detection
Module\=====#

# importing necessary libraries
from lib.config import NMS_Threshold, Min_Prob, Human_Counter
import numpy as np
import cv2

# defining the detect_humans function
def detect_humans(frame, net, layer_names, human_idx = 0):
    # extracting the dimensions of the frame and

```

```

    # initializing the results list
(H, W) = frame.shape[:2]
results = []

# constructing a blob from the input frame and performing a
forward
# pass of the YOLO object detector
# gives us the bounding boxes and associated probabilities
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
    swapRB = True, crop = False)
net.setInput(blob)
layerOutputs = net.forward(layer_names)

# initializing the lists of detected bounding boxes,
# centroids and confidences
boxes = []
centroids = []
probabilities = []

# iterating through the layer outputs
for output in layerOutputs:
    # iterating through each of the detections
    for detection in output:
        # extracting the class ID and object detection
probability
        scores = detection[5:]
        classID = np.argmax(scores)
        probability = scores[classID]

        # filtering detections by:-
        # (1) ensuring that a human was detected and
        # (2) that the minimum probability criteria was satisfied
        if classID == human_idx and probability > Min_Prob:
            # scaling the bounding box coordinates back
relative to
            # the size of the image, as YOLO returns
            # the center (x, y) coordinates of the bounding box
            # followed by the width and height

```

```

        box = detection[0:4] * np.array([W, H, W, H])
        (centerX, centerY, width, height) =
box.astype("int")

        # using the center (x, y) coordinates to find the
        # top-left corner coordinates
        x = int(centerX - (width / 2))
        y = int(centerY - (height / 2))

        # updating the list of bounding box coordinates,
        # centroids and confidences
        boxes.append([x, y, int(width), int(height)])
        centroids.append((centerX, centerY))
        probabilities.append(float(probability))

# applying non-maxima suppression (NMS) to suppress weaker,
# overlapping bounding boxes
idxs = cv2.dnn.NMSBoxes(boxes, probabilities, Min_Prob,
NMS_Threshold)

# calculating the total humans in frame
if Human_Counter:
    human_count = "Human Count: {}".format(len(idxs))
    cv2.rectangle(frame, (520, 0), (700, 30), (0, 0, 0), -1)
    cv2.putText(frame, human_count, (530, 20),
cv2.FONT_HERSHEY_DUPLEX, 0.50, (255, 255, 255), 1, cv2.LINE_AA)

# ensuring at least one detection exists
if len(idxs) > 0:
    # iterating through the indexes
    for i in idxs.flatten():
        # extracting the bounding box coordinates
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])

        # updating the results list to contain
        # detection probability, bounding box coordinates and
centroid

```

```

        res = (probabilities[i], (x, y, x + w, y + h),
centroids[i])
        results.append(res)

# returning the list of results
return results

```

APPENDIX C - alerts.py

```

#=====/Alerts
Module\=====#
# importing required libraries
import urllib, requests
from lib.config import chat_id, token

# this script initiates the Telegram alert trigger function

# defining the trigger function
def trigger(arr, typ):
    # setting the alert messages for:
    # (1) social distancing violations
    message1 = 'Social distancing violations exceeded!\n\nSerious
Violations :
    {}'.format(arr[0])
    # (2) face mask usage violations
    message2 = 'Face Mask violations exceeded!\n\nMasked : {}
\nImproperly
    Masked : {} \nUnmasked : {}'.format(arr[1], arr[2],
arr[3])

    # sending GET requests to the Telegram bot's server
    # for each case of violation
    if typ == 1:
        url =
'https://api.telegram.org/bot%/sendMessage?chat_id=%s&text=%s' %
        (token, chat_id, urllib.parse.quote_plus(message1))

```

```

_ = requests.get(url, timeout=10)

if typ == 2:
url =
'https://api.telegram.org/bot%s/sendMessage?chat_id=%s&text=%s' %
(token, chat_id, urllib.parse.quote_plus(message2))
_ = requests.get(url, timeout=10)

```

APPENDIX D - config.py

```

#=====\Configuration
Script/=====#
# base path to YOLO directory
YOLO_PATH = "yolo"

# minimum object detection probability
Min_Prob = 0.3

# minimum threshold for non-maxima suppression
NMS_Threshold = 0.3

# to count number of people in frame (True/False)
Human_Counter = True

# set the threshold value for violations
Violations_Threshold = 10

# set the ip camera url (e.g. url =
'http://192.168.43.39:4747/video')
# set url = 0 for webcam
url = 0

#-----|TELEGRAM
ALERTS/-----#
# toggle telegram alert feature (True/False)
Alert = False

```

```

# telegram bot's chat ID and token
chat_id = ''
token = ''
#-----#
-----#

# toggle GPU usage for computations (True/False)
# CPU used by default
Use_GPU = True

# set minimum safe distance between 2 people (in cm.)
MAX_DISTANCE = 200 # (i.e. safe distance)
MIN_DISTANCE = 100 # (1.e. minimum safe distance)

# set average height of a person (in cm.)
avg_height = 170

```

APPENDIX E - main.py

```

#=====/Main Execution
Script\=====#
# importing necessary libraries
from lib import config
from lib.detect_facemask import DETECT_FACEMASK
from lib.detect import detect_humans
from lib.alerts import trigger
from imutils.video import FPS
from scipy.spatial import distance as dist
import numpy as np
import argparse, imutils, cv2, os, time

#-----/ARGUMENTS
PARSING/-----#
# argument parser to parse command line arguments
ap = argparse.ArgumentParser()

```



```

ap.add_argument("-i", "--input", type=str, default="",
                help="path to (optional) input video file")

ap.add_argument("-o", "--output", type=str, default="",
                help="path to (optional) output video file")

ap.add_argument("-d", "--display", type=int, default=1,
                help="whether or not output frame should be displayed")

args = vars(ap.parse_args())
#-----
----- #

# Loading YOLO facemask detector classes & object
classes = ["good", "bad", "none"]
detect_facemask = DETECT_FACEMASK("yolo/mask-yolov4-tiny.cfg",
                                  "yolo/mask-yolov4-tiny.weights", classes)

# initializing facemask detector size & confidence
detect_facemask.size = 416
detect_facemask.confidence = 0.5

# facemask detector component colors
colors = [(0, 255, 0), (0, 165, 255), (0, 0, 255)]

# Loading the COCO class labels
labelsPath = os.path.sep.join([config.YOLO_PATH, "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# deriving the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([config.YOLO_PATH, "yolov3.weights"])
configPath = os.path.sep.join([config.YOLO_PATH, "yolov3.cfg"])

# Loading the YOLO object detector trained on COCO dataset (80
classes)
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

```

```

# checking if there's GPU usage
if config.Use_GPU:
    # set CUDA as the preferable backend and target
    print("")
    print("[INFO] Looking for GPU")
    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

# determining only the *output* Layer names that we need from YOLO
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# if a video path was not supplied
# creating a reference with source as the camera
if not args.get("input", False):
    print("[INFO] Starting the live stream..")
    vs = cv2.VideoCapture(config.url)
    time.sleep(1.0)

# else, creating a reference with source as the video file
else:
    print("[INFO] Starting the video..")
    vs = cv2.VideoCapture(args["input"])

writer = None

# starting the FPS counter
fps = FPS().start()

# iterating through the frames from the video stream
while True:
    # reading the next frame from the file
    (grabbed, frame) = vs.read()
    # if the frame was not grabbed, then we have reached the end of
    the stream
    if not grabbed:
        break

```

```

# resizing the frame
frame = imutils.resize(frame, width=700)

# calling detect_facemask function to detect face & masks usage
in frames
width, height, inference_time, fm_results =
detect_facemask.inference(frame)

# counter for mask usage
masked = 0
improper_masked = 0
unmasked = 0

# Looping through facemask detector results
for detection in fm_results:
id, name, confidence, x, y, w, h = detection
cx = x + (w / 2)
cy = y + (h / 2)

# updating counters
if id == 0:
masked = masked + 1
if id == 1:
improper_masked = improper_masked + 1
if id == 2:
unmasked = unmasked + 1

# drawing a bounding box rectangle and label on the image
color_fm = colors[id]
cv2.rectangle(frame, (x, y), (x + w, y + h), color_fm, 2)
text_fm = "%s (%s)" % (name, round(confidence, 2))
cv2.putText(frame, text_fm, (x, y - 5),
cv2.FONT_HERSHEY_SIMPLEX,
0.5, color_fm, 2)

# formatting counters text
masked_text = "Masked: {}".format(masked)
improper_masked_text = "Improperly Masked:

```

```

{}".format(improper_masked)
    unmasked_text = "Unmasked: {}".format(unmasked)

    # calling detect_humans function to detect only humans in the
frames
    results = detect_humans(frame, net, ln,
human_idx=LABELS.index("person"))

    # initializing the set of indexes that violate the max/min
social distance limits
    serious = set()
    abnormal = set()

    # ensuring there are at least two people detections (required
in
    # order to compute our pairwise distance maps)
    if len(results) >= 2:
        # extracting all centroids from the results and computing the
# Euclidean distances between all pairs of centroids
        centroids = np.array([r[2] for r in results])
        # extracting heights of all detected bounding boxes
        pixel_heights = np.array([r[1][3]-r[1][1] for r in results])
        D = dist.cdist(centroids, centroids, metric="euclidean")

        # Loop over the upper triangular of the distance matrix
        for i in range(0, D.shape[0]):
            for j in range(i + 1, D.shape[1]):
                # calibrating the pixel distance to centimeters
                calib_factor = (1/pixel_heights[i] +
1/pixel_heights[j]) / 2 * config.avg_height
                D[i, j] = D[i, j] * calib_factor
                # check to see if the distance between any two
# centroid pairs is less than the configured number
of pixels

                if D[i, j] < config.MIN_DISTANCE:
                    # update our violation set with the indexes of the
centroid pairs
                    serious.add(i)

```

```

        serious.add(j)
        # update our abnormal set if the centroid distance
is below max distance limit
        if (D[i, j] < config.MAX_DISTANCE) and not serious:
            abnormal.add(i)
            abnormal.add(j)

# iterating through the results
for (i, (prob, bbox, centroid)) in enumerate(results):
    # extracting the bounding box and centroid coordinates, and
    # initializing the color of the annotation
    (startX, startY, endX, endY) = bbox
    (cX, cY) = centroid
    color = (0, 255, 0)

    # if the index pair exists within the violation/abnormal sets,
then update the color
    if i in serious:
        color = (0, 0, 255)
    elif i in abnormal:
        color = (0, 255, 255) #orange = (0, 165, 255)

# drawing:-
# (1) a bounding box around the person and
# (2) the centroid coordinates of the person
cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
cv2.circle(frame, (cX, cY), 5, color, 2)

# drawing some of the parameters
Safe_Distance = "Safe distance: > {}
m.".format(config.MIN_DISTANCE/100)
cv2.putText(frame, Safe_Distance, (505, frame.shape[0] - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 0, 0), 2)
Violations_Threshold = "Threshold limit:
{}".format(config.Violations_Threshold)
cv2.putText(frame, Violations_Threshold, (505, frame.shape[0] -
37),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 0, 0), 2)

```

```

    # drawing the total number of social distancing violations on
the output frame
    cv2.rectangle(frame, (0, 0), (215, 50), (0, 0, 0), -1)

    text = "Serious Violations: {}".format(len(serious))
    cv2.putText(frame, text, (15, 20), cv2.FONT_HERSHEY_DUPLEX,
0.50, (0, 0, 255), 1, cv2.LINE_AA)

    text1 = "Abnormal Violations: {}".format(len(abnormal))
    cv2.putText(frame, text1, (15, 40), cv2.FONT_HERSHEY_DUPLEX,
0.50, (0, 255, 255), 1, cv2.LINE_AA)

    # displaying counters on screen
    cv2.rectangle(frame, (520, 30), (700, 90), (0, 0, 0), -1)
    cv2.putText(frame, masked_text, (530, 40),
cv2.FONT_HERSHEY_SIMPLEX, 0.40, (0, 255, 0), 1, cv2.LINE_AA)
    cv2.putText(frame, improper_masked_text, (530, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.40, (0, 165, 255), 1, cv2.LINE_AA)
    cv2.putText(frame, unmasked_text, (530, 80),
cv2.FONT_HERSHEY_SIMPLEX, 0.40, (0, 0, 255), 1, cv2.LINE_AA)

#----- |Alert
function|-----#
    # alerts info array (to be passed to alerts module)
    arr = [len(serious), masked, improper_masked, unmasked]

    if len(serious) >= config.Violations_Threshold:
        cv2.putText(frame, "ALERT: Violations exceeded limit!", (15,
frame.shape[0] - 20),
            cv2.FONT_HERSHEY_DUPLEX, 0.60, (0, 0, 255), 2)
    if config.Alert:
        print("")
        print('[ALERT] Sending social distancing alert...')
        trigger(arr, 1)
        print('[ALERT] Alert sent')

    if unmasked > 3 or improper_masked > 0:

```

```

    cv2.putText(frame, "ALERT: Face Mask Violation!", (15,
frame.shape[0] - 40),
                cv2.FONT_HERSHEY_DUPLEX, 0.60, (0, 165, 255), 2)
    if config.Alert:
        print("")
        print('[ALERT] Sending face mask usage alert...')
        trigger(arr, 2)
        print('[ALERT] Alert sent')
#-----#
# checking to see if the output frame should be displayed
if args["display"] > 0:
    # displaying the output frame
    cv2.imshow("MaViSS AI - COVID19 Norms Surveillance System",
frame)
    key = cv2.waitKey(1) & 0xFF

    # breaking loop if 'ESC' key is pressed
    if key == 27:
        break
    # updating the FPS counter
    fps.update()

    # if an output video file path has been supplied and the video
    # writer has not been initialized, doing so now
    if args["output"] != "" and writer is None:
        # initializing the video writer
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 25,
                                (frame.shape[1], frame.shape[0]), True)

        # if the video writer is not None, writing the frame to the
        output video file
        if writer is not None:
            writer.write(frame)

# stopping the timer and displaying FPS information
fps.stop()

```

```
print("=====")
print("[INFO] Elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] Approx. FPS: {:.2f}".format(fps.fps()))

# closing any open windows
cv2.destroyAllWindows()
```


Topic - Effects of hypobaric hypoxia on the functions of our Brain

Mentored by – Prof. Abhinav Bhushan

- **Area – Human Health (Engineering)**
- **Unmet needs in the topic –**
 - Persistent immune dysregulation during exploration missions
 - Combined immune-suppressive effects of spaceflight environmental factors when witnessed along with hypoxia is a cause of grave concern
 - VIIP syndrome (visual impairment / intracranial pressure)
 - Exercise countermeasures
 - Sensorimotor performance due to hypoxia
 - Acute mountain sickness
 - Cardiovascular degenerative effects
 - Oxidative stress
- **What has been done in the past?**
 - NASA gains the capability for efficient EVA with low DCS risk, but it also accrues the human health and performance risks associated with the addition of hypobaric hypoxia to the spaceflight environment
 - Research addressing some of the unmet needs but no concrete solution as to how to combat the need in the ISS and other manned missions.
- **Answering the “Why” question**
 - High-altitude cerebral edema is associated with increased ICP (intracranial pressure)
 - AMS (Acute Mountain sickness) appears to be strongly associated with increased optic nerve sheath diameter, reflecting increased ICP.
 - Increased optic nerve sheath diameter has been found to correlate positively with ICP based on the fact that the subarachnoid cerebrospinal fluid (CSF) compartment communicates with the peri optic CSF space.
 - With the addition of microgravity-induced intracranial hypertension, it is likely that astronauts would develop greater increases in ICP in an 8.2/34 environment than in 14.7/21.
- **Normobaric Vs Hypobaric hypoxia-**
 - **Normobaric Hypoxia-** This may be realized by a decrease in oxygen fraction (FO₂), without a change in PB. The effects are either not observed or observed at a very low intensity when we talk about Normobaric hypoxia
 - **Hypobaric Hypoxia-** This may be realized by a decrease in barometric pressure (PB) leading to hypobaric hypoxia (HH). Intravascular bubble formation,

FINAL REPORT BY SHARANYA JHA (BIT MESRA, INDIA)

mismatched ventilation and perfusion, and altered gas density or fluid permeability through the alveolar epithelium observed in HH

- **How does it affect the brain?**
 - Ocular and cerebral adaptations to microgravity and oxygen deprivation associated with long-duration spaceflight
 - Ophthalmic changes consisting of disc oedema, posterior globe flattening, choroidal folds, cotton wool spots, nerve fibre layer thickening, and decreased near vision and hyperopic shifts
 - Psychomotor impairment (including incoordination and tremors)
 - Concentration, confusion, memory loss, flexibility, working memory, and drowsiness
 - Postural control influenced by HH, and exacerbated by anteroposterior plane with eyes wide open
- **Secondary Factors that exacerbate HH-** High levels of carbon dioxide in spacecraft cabins, Heavy resistive exercise, Anthropomorphic changes due to microgravity, High sodium diet. Additionally, an enzymatic polymorphism in the 1-carbon metabolism cycle has recently been identified as a factor associated with the observed vision changes, but is not clear if this is causative
- **Approaches**
 - Non-Medicinal Approach: Visual acuity tests, High resolution retinal imagery, Visual field assessment, and detailed imagery of intracranial ophthalmic structures, Inflight diagnostic ultrasound can show intra-orbital changes such as globe flattening and optic nerve sheath distension over time. NASA's approach- Developing an in-flight OCT (Optical Coherence Tomography) capability that could provide early recognition of microgravity-induced eye and optic nerve changes
 - Lower body negative pressure to safely reduce intracranial pressure: Weightlessness prevents the normal cerebral volume and pressure 'unloading' associated with upright postures on Earth, which may be part of the cerebral and ocular pathophysiology
 - Usage of NGEN and QUR - Administration of flavonoid showed neuronal protection and prevented the accumulation of ubiquitin and lesser caspase-3 activation. Mounting evidence suggest that mitochondrial electron chain acts as an oxygen sensor, releasing reactive oxygen species in response to hypoxia stress. NGEN and QUR are known antioxidant compounds and therefore, were effective against the treatment of deficiency of oxygen caused due to a pressure reduction.
 - Dual-Task Approach - The Transit Food System will deliver a food system during the transit and the initial stay on the lunar or planetary surface. The Lunar/ Planetary Surface Food System will provide the crew with the proper nutrition during the long-duration surface stay. These two food systems are intrinsically

FINAL REPORT BY SHARANYA JHA (BIT MESRA, INDIA)

different. The Transit Food System has to operate in microgravity, and the Lunar/Planetary Surface Food System must operate in partial gravity, allowing for more flexibility and more Earth-like operations.

- Hydroponic System - Through hydroponic growth of fruits and vegetables, tomatoes can be cultivated which comes under the category of ready to eat salad crops. They can thus be the source of NGEN and QUR (flavonoids) that can act as HH reducers in the body.
- **Summary of the Current NASA Food System** - Space Shuttle, International Space Station- All the food supporting these programs is processed to achieve shelf stability. These processed foods are designed to provide crewmembers with a variety of menu options that are ready to eat or that require only minimal preparation, such as adding water to or reheating foods. **NASA's Advanced Food Technology (AFT) project team** is investigating the possibility of a partially **bioregenerative food system** on the Martian surface. Fresh fruits and vegetables and possibly other commodities can be grown hydroponically in environmentally controlled chambers.
- **Overall Solutions**
 - Lower body negative pressure to safely reduce intracranial pressure
 - Usage of NGEN and QUR
 - Cultivation of ready to eat salad crops through a bio generative system in a hydroponic environment
- **References**
 - NASA Exploration Atmospheres Working Group, "Recommendations for exploration spacecraft internal atmospheres: The final report of the NASA exploration atmospheres working group. NASA Technical Publication NASA/TP-2010-216134," NASA, Johnson Space Center, 2010
 - P. D. Campbell, "Recommendations for Exploration Spacecraft Internal Atmospheres: The Final Report of the NASA Exploration Atmospheres Working Group," NASA, Johnson Space Center, Houston TX, January 2006.
 - G. P. Millet and V. Pialoux, "Point:counterpoint: hypobaric hypoxia induces / does not induce different responses from normobaric hypoxia," J. Appl. Physiol., vol. 112, pp. 1783-84, 2012.
 - R. Mounier and J. V. Brugniaux, "Point:counterpoint: hypobaric hypoxia induces / does not induce different responses from normobaric hypoxia," J Appl Physiol, vol. 112, pp. 1784-86, 2012.
 - Naringenin and quercetin reverse the effect of hypobaric hypoxia and elicit neuroprotective response in the murine model Aditi Sarkara , M. Sonia Angelinea , Kushi Ananda , Rashmi K Ambastaa , Pravir Kumara,b,c,n
 - Mission to Mars: Food Production and Processing for the Final Frontier Michele H. Perchonok,¹ Maya R. Cooper,² and Patricia M. Catauro²

Reconfigurable Hardware Design For Signal Processing Applications

Shuvam Adhikary
Birla Institute of Technology

The analysis and evaluation of an ultrasonic data extracted from non-destructive testing applications is quite a difficult task and arduous. This research provides the technique to analyse and improve ultrasonic signals on the basis of their flaw(detect). Split Spectrum Processing and Post Processing Techniques such as Minimization and Averaging are used to process the ultrasonic signal and greatly improve its flaw-to-clutter ratio(FCR). The algorithm can also be embedded onto a field programmable gate array(FPGA) for the real time evaluation of the ultrasonic data.

I. INTRODUCTION

The most efficient and economical approach for determining flaws in structures or materials such as bridges, buildings is through **non-destructive testing**. Ultrasonic signal plays a major role when it comes to non-destructive evaluation of materials. However, the clutter echoes resulting from the microstructure of materials pose a serious problem in the detection of the flaw in the ultrasonic scan. The **A-scan** is a one dimensional data which contains information of clutter echoes and flaw echoes. Due to the randomness of the clutter echoes, the clutter echoes often mask the flaw and since both the clutter echoes and the flaw span over the same frequency range, it becomes difficult to decorrelate the clutter and improve the flaw visibility. However, it is possible to achieve clutter decorrelation by frequency diversification i.e. by obtaining a set of frequency diverse signals(multiple channels).

This project presents the method to improve the **flaw visibility** of the ultrasonic images containing the flaws through split-spectrum processing. Post-processing methods such as minimization and averaging are used to improve the flaw visibility. The parameter flaw-to-clutter ratio(FCR) serves as the criterion to check the improvement of the flaw visibility. Moreover, in future, the algorithm can also be embedded onto a field programmable gate array(FPGA) for the real time evaluation of the ultrasonic data by designing an HLS code/Verilog code from the corresponding MATLAB code and burning the code onto the FPGA platform.

II. SPLIT SPECTRUM PROCESSING

Sub-band decomposition also known as split spectrum processing is an effective technique for obtaining the frequency-diverse signals. However, the ultrasonic data has to undergo through the various stages of split-spectrum processing before providing the final result with an improved flaw visibility. After the reception of the echo from the sample under test, it passes through an analog-to-digital converter. The A-scan then passes through the **Fast Fourier Transform(FFT)** block, where

the A-scan in time domain is converted into its corresponding frequency spectrum. The frequency spectrum is then divided into various **sub-frequency bands** by the sub-band filters present in the next component. **Inverse Fast Fourier Transform** is applied to each sub-frequency band to generate their corresponding time domain signals. These signals from each frequency band are then normalized before passing them onto the post-processor block. The post-processor block includes the order-statistics filters such as minimization and averaging. These order-statistic filters help to improve the flaw visibility by improving the flaw-to-clutter ratio.

The flaw is more dominant in the **low frequency** region rather than the high frequency region where it gets suppressed. Hence, the sub-band filters are concentrated more onto the low frequency region of the frequency spectrum to obtain maximum information about the flaw. The efficient extraction of information about the flaw highly depends upon the number of channels used to filter the frequency spectrum. More the number of channels, more is the possibility to isolate the flaw echo from the undesired noise. This project presents the results obtained using 8 channels. Again, the maximization of FCR highly depends on the selection of the **size of the filters** and the **degree of overlap** between the channels. Hence, the proper selection of both the size of the filter and the degree of overlap becomes a major task.

The post-processor combines all the **normalized** signals coming from each channel after the inverse fast fourier transform to reconstruct the original time-domain signal but with an improved FCR. There are various order statistics filters such as Minimization, Averaging, Median, Polarity checker, Geometric Mean etc. However, it is found that the FCR is greatly improved when minimization and averaging are employed as the post-processors. Hence, this project presents the results obtained through the post-processors Minimization and Average. There are certainly limitations to these processors. When there are a number of channels exhibiting null observations i.e. the clutter information is more dominant and the flaw echo information is almost negligible, these processors tend to suppress the flaw

echo information to the extent that the information is almost negligible while the clutter information is enhanced greatly which is quite opposite to the desired result.

The mathematical expressions for these post-processors are as follows:

I. Minimization:

$$\phi_{\min}(n) = \min[|z_j(n)|, \quad j = 1, 2, \dots, k]$$

II. Averaging:

$$\phi_{\text{av}}(n) = \frac{1}{k} \sum_{j=1}^k |z_j(n)|,$$

The performance of these post-processors is calculated and compared using the parameter Flow-to-clutter ratio. The FCR is the logarithmic ratio of the maximum flow echo amplitude to the maximum clutter echo amplitude. The mathematical expression for the FCR is given by:

$$\text{FCR} = 20 \times \log_{10}(F/C)$$

where F is the maximum amplitude of the flaw echo and C is the maximum amplitude of the clutter echo.

III. RESULTS

An algorithm was designed implementing the following processes:

1. Importing and reading the A-scan data in MATLAB.
2. Applying **FFT** onto the A-scan data.
3. Splitting the frequency spectrum obtained into **8 sub-frequency bands** with variable window size and degree of overlap.
4. Applying **IFFT** to all the 8 channels.
5. Normalizing the data of all the 8 channels after the inverse FFT.
6. Applying the post-processors **minimization** and **average** to reconstruct the original time domain signal with the improved FCR.

The algorithm was implemented on 14 experimental(real) ultrasonic A-scans. The results for minimization and averaging were plotted along with the 8 decomposed channels. The FCR was calculated for the original data and also after the minimization and averaging steps. The FCR obtained from these post-processors was then compared with the original FCR to evaluate the improvement in the flaw visibility.

The following plots give information about the original experimental(real) ultrasonic A-scan data, the 8 channels or the sub-frequency bands, the time domain signal after minimization and the time domain signal after averaging for all the 14 experimental(real) ultrasonic A-scan data.

I. Newscan1

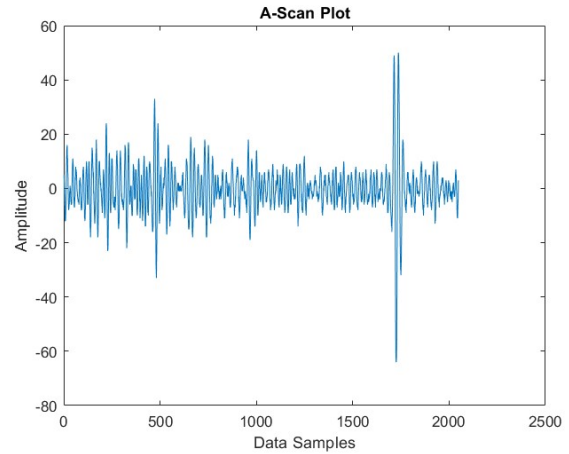


Fig.1.1 Experimental A-Scan plot for Newscan1

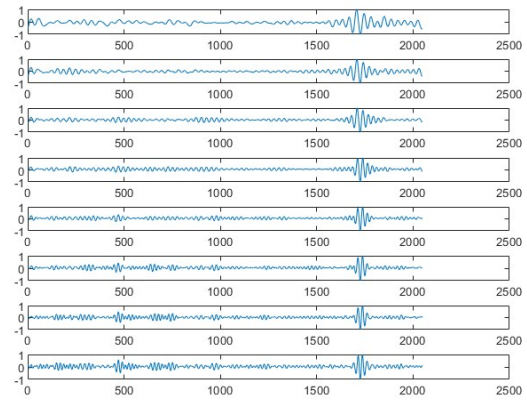


Fig.1.2 8 Observation channels for Newscan1

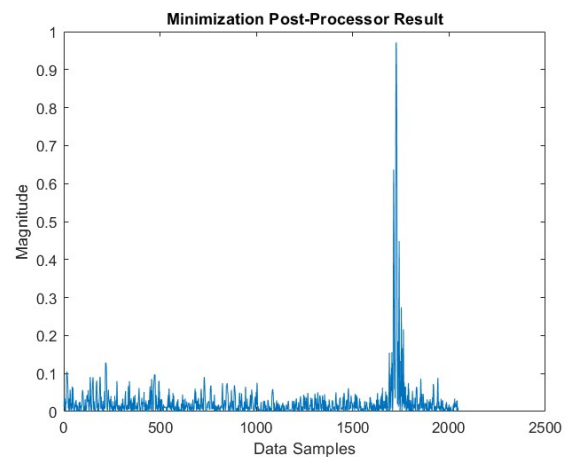


Fig.1.3 Minimization post-processor result for Newscan1

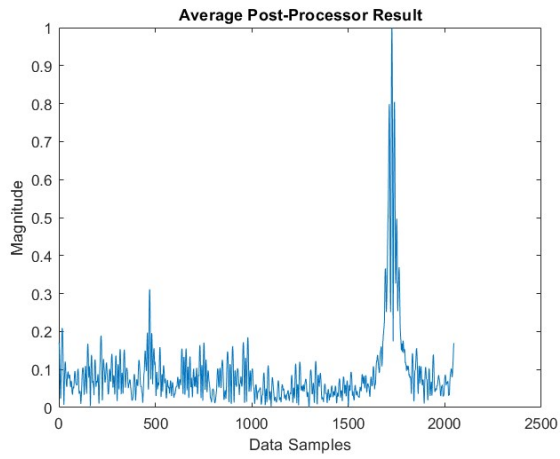


Fig.1.4 Average post-processor result for Newscan1

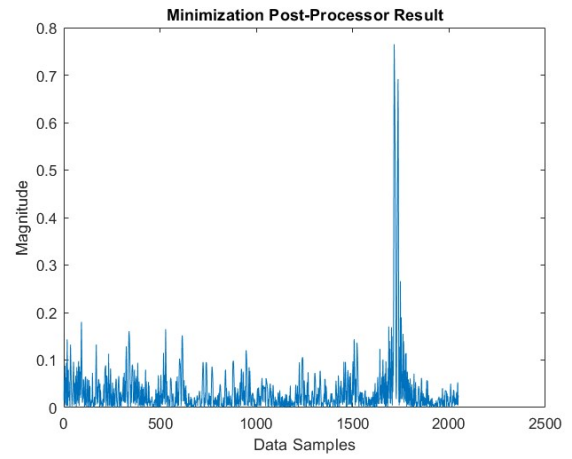


Fig.2.3 Minimization post-processor result for Newscan2

II. Newscan2

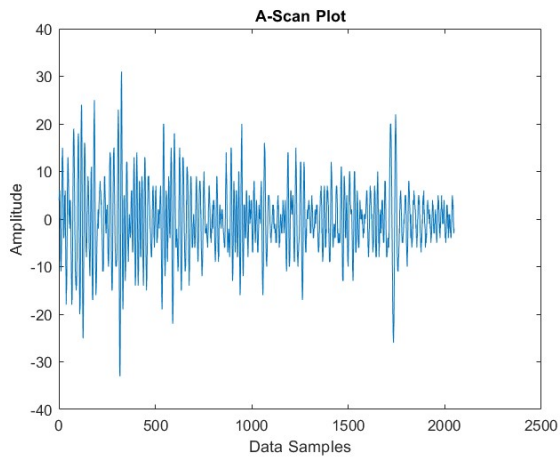


Fig.2.1 Experimental A-Scan plot for Newscan2

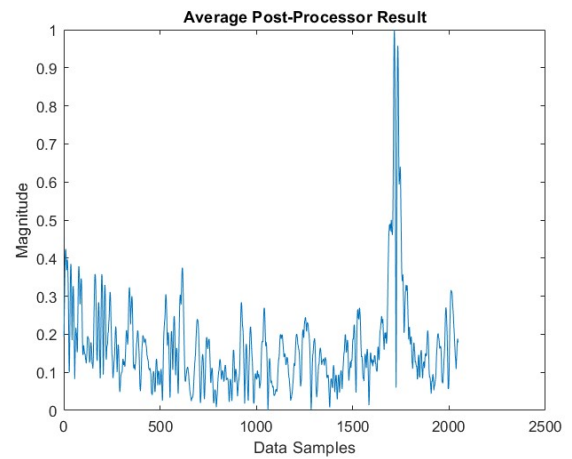


Fig.2.4 Average post-processor result for Newscan2

III. Newscan3

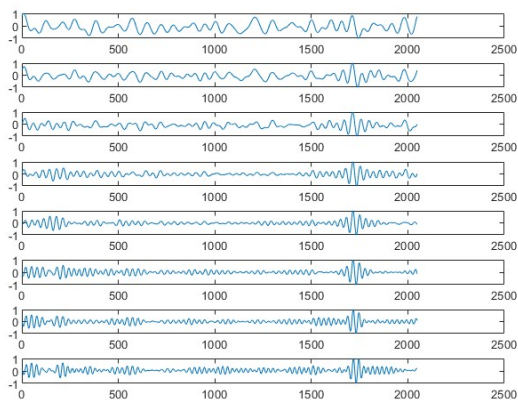


Fig.2.2 8 Observation channels for Newscan2

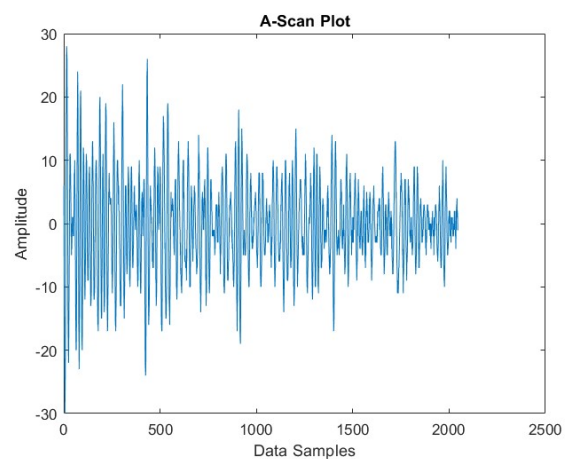


Fig.3.1 Experimental A-Scan plot for Newscan3

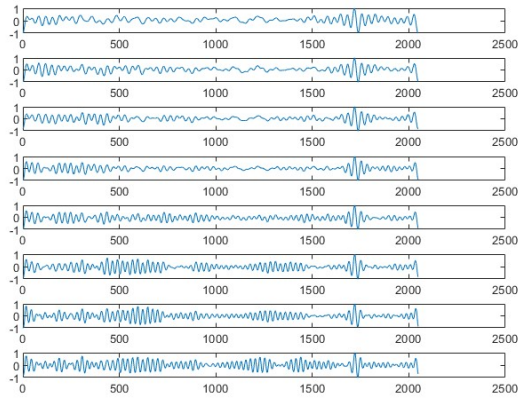


Fig.3.2 8 Observation channels for Newscaan3

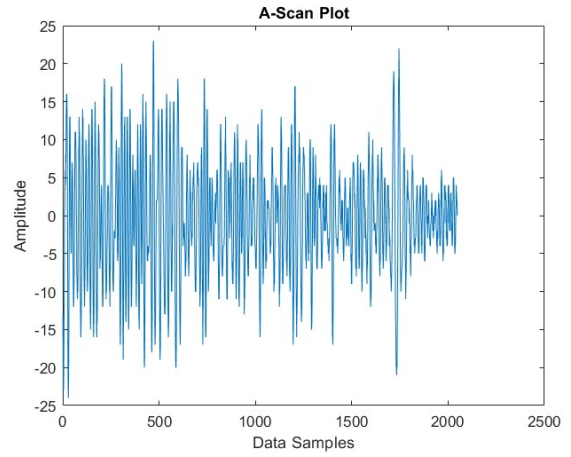


Fig.4.1 Experimental A-Scan plot for Newscaan4

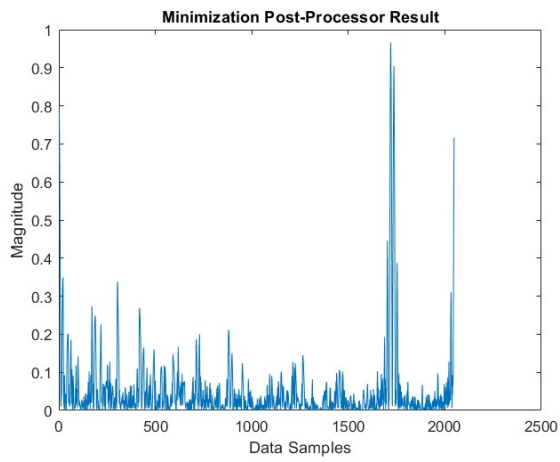


Fig.3.3 Minimization post-processor result for Newscaan3

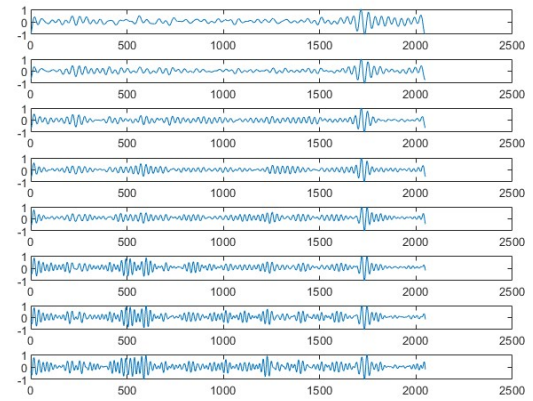


Fig.4.2 8 Observation channels for Newscaan4

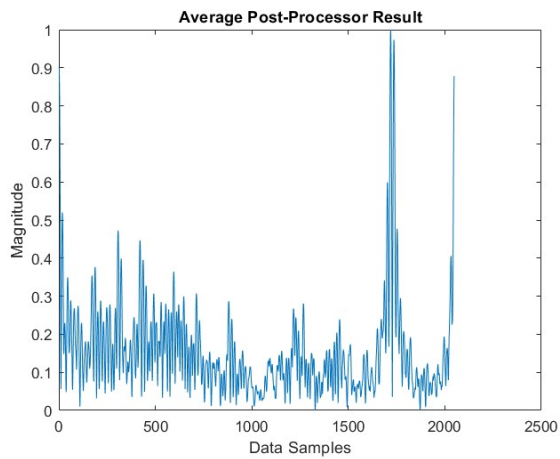


Fig.3.4 Average post-processor result for Newscaan3

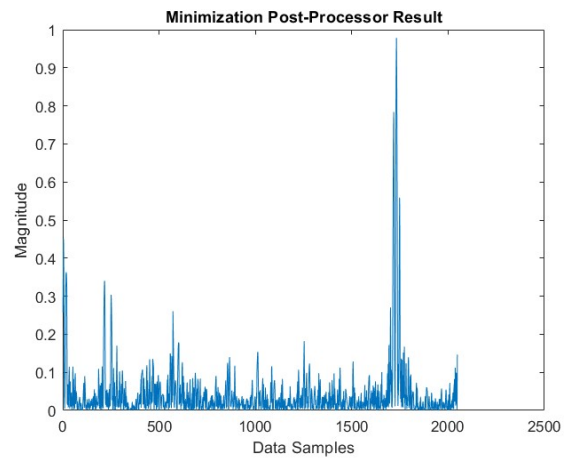


Fig.4.3 Minimization post-processor result for Newscaan4

IV. Newscaan4

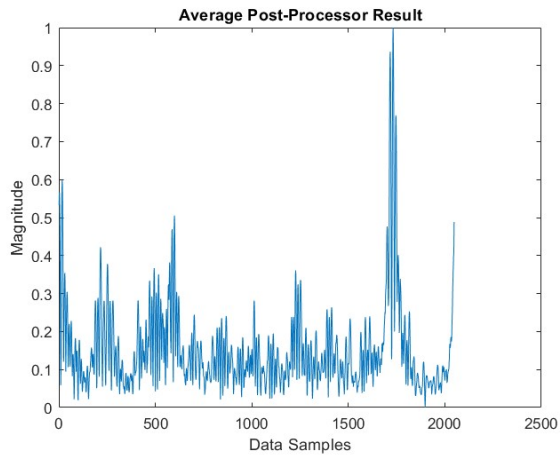


Fig.4.4 Average post-processor result for Newscan4

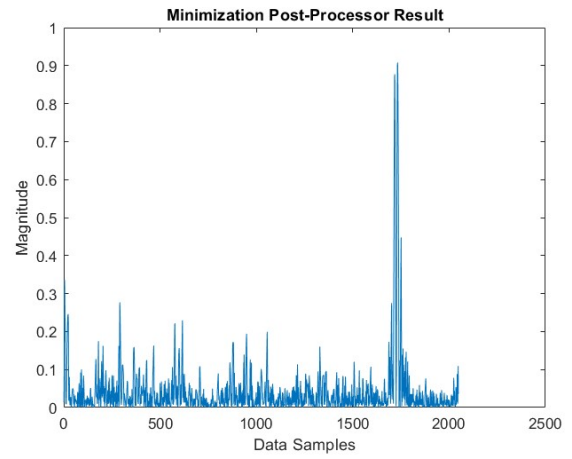


Fig.5.3 Minimization post-processor result for Newscan5

V. Newscan5

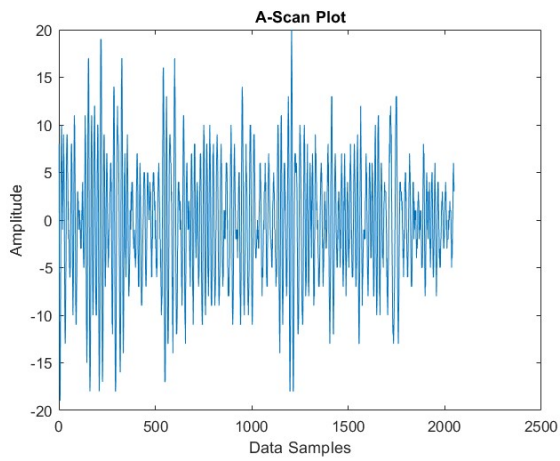


Fig.5.1 Experimental A-Scan plot for Newscan5

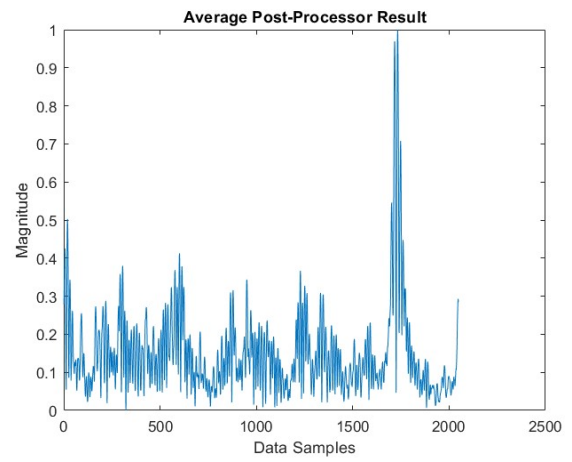


Fig.5.4 Average post-processor result for Newscan5

VI. Newscan6

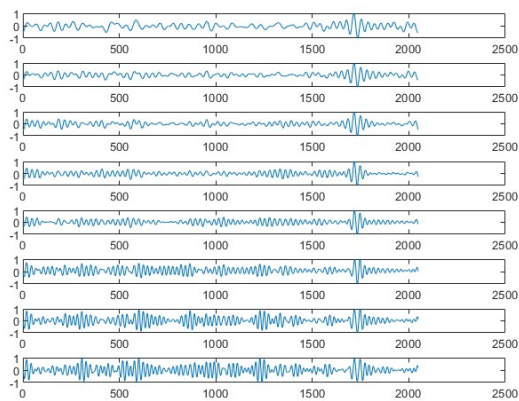


Fig.5.2 8 Observation channels for Newscan5

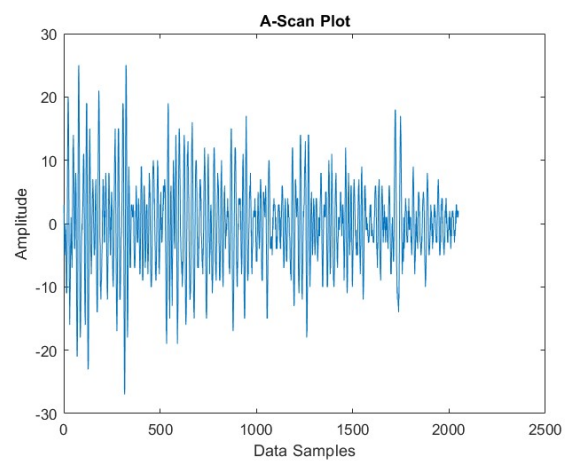


Fig.6.1 Experimental A-Scan plot for Newscan6

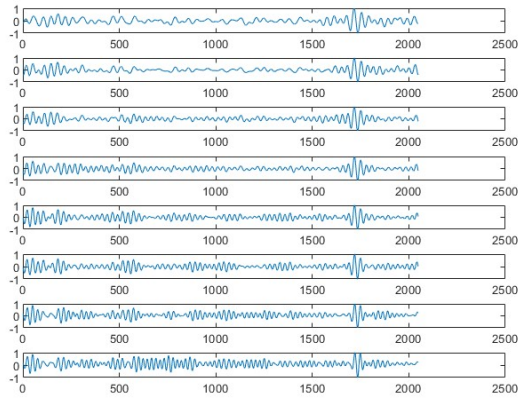


Fig.6.2 8 Observation channels for Newsca6

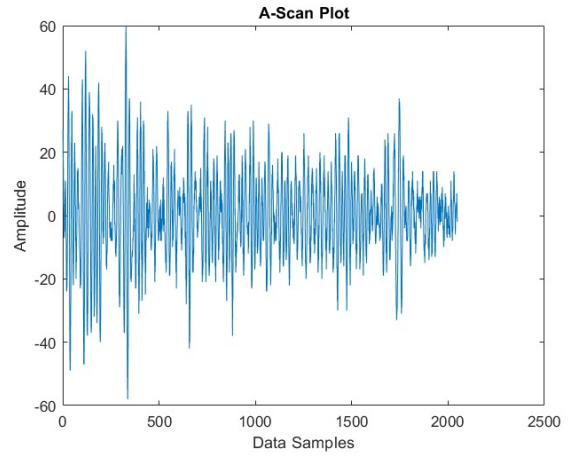


Fig.7.1 Experimental A-Scan plot for Newsca7

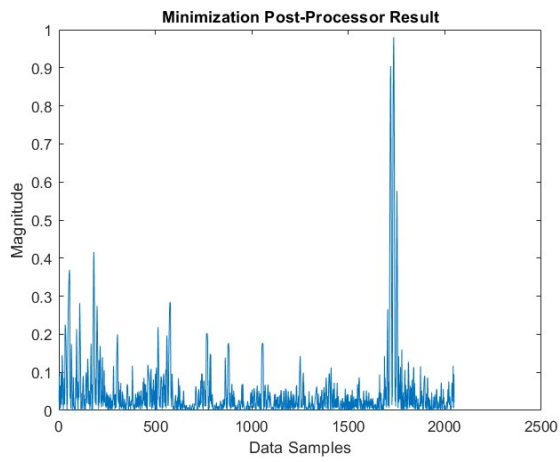


Fig.6.3 Minimization post-processor result for Newsca6

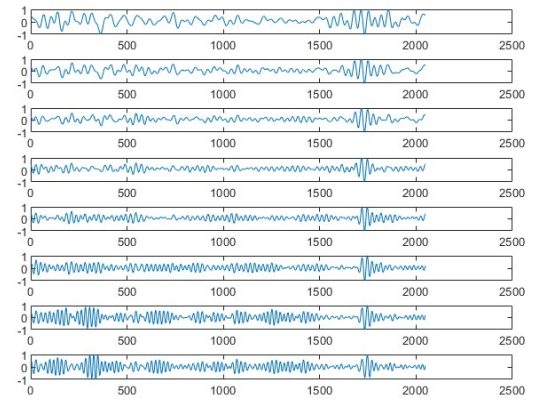


Fig.7.2 8 Observation channels for Newsca7

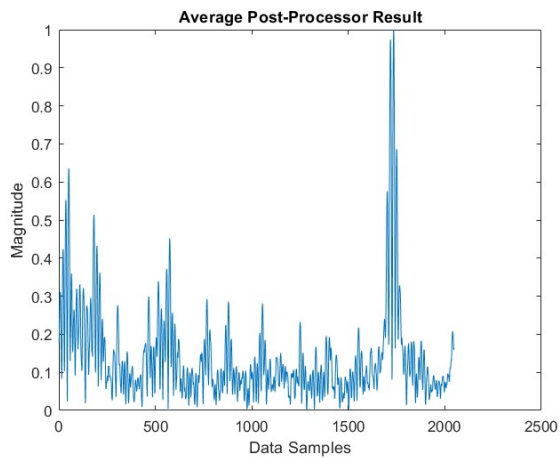


Fig.6.4 Average post-processor result for Newsca6

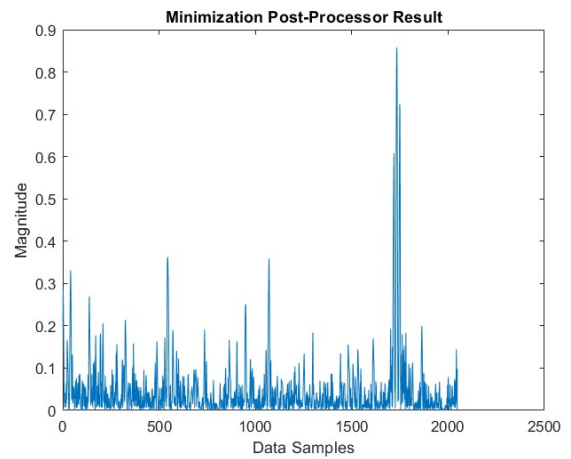


Fig.7.3 Minimization post-processor result for Newsca7

VII. Newsca7

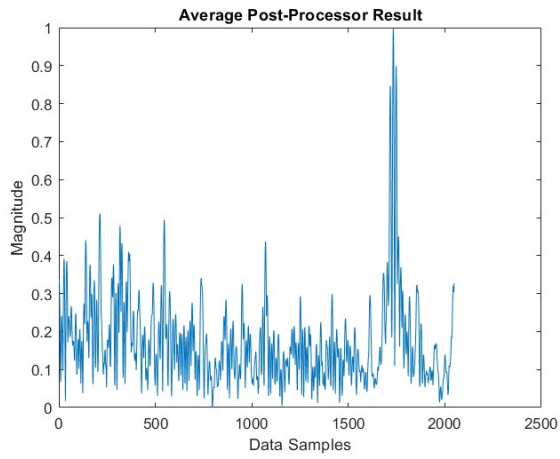


Fig.7.4 Average post-processor result for Newscan7

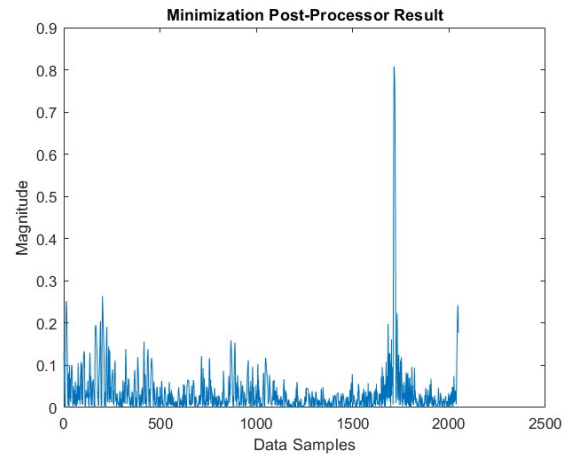


Fig.8.3 Minimization post-processor result for Newscan8

VIII. Newscan8

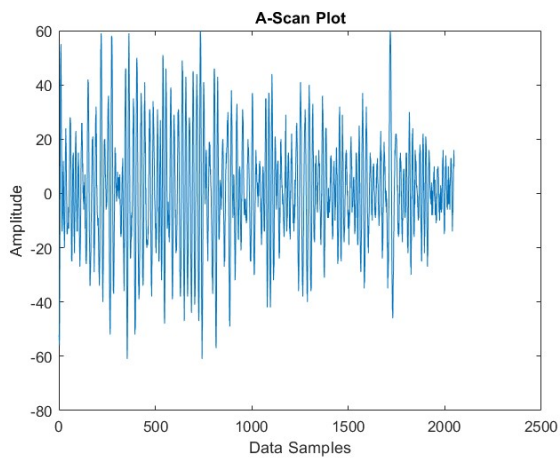


Fig.8.1 Experimental A-Scan plot for Newscan8

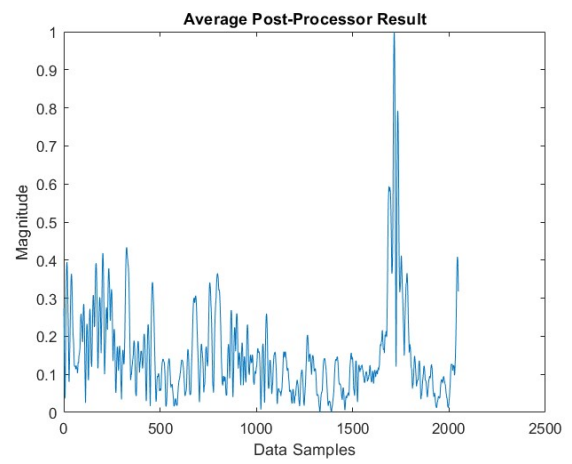


Fig.8.4 Average post-processor result for Newscan8

IX. Newscan9

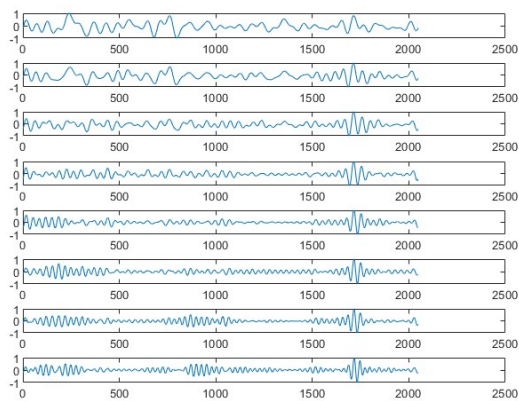


Fig.8.2 8 Observation channels for Newscan8

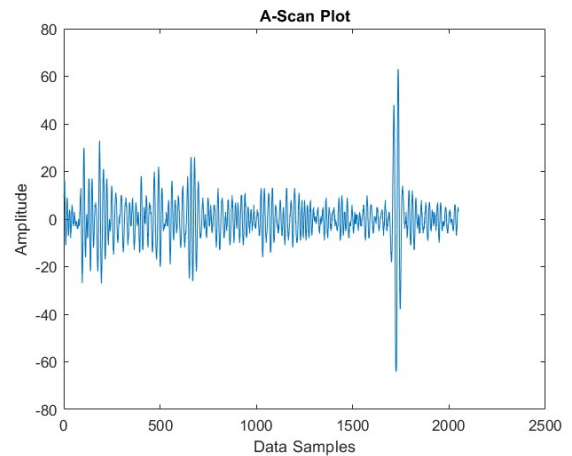


Fig.9.1 Experimental A-Scan plot for Newscan9

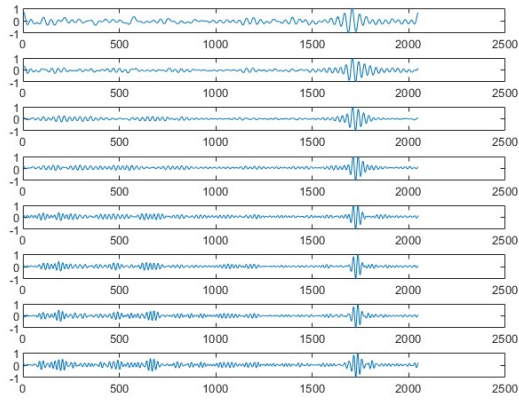


Fig.9.2 8 Observation channels for News9

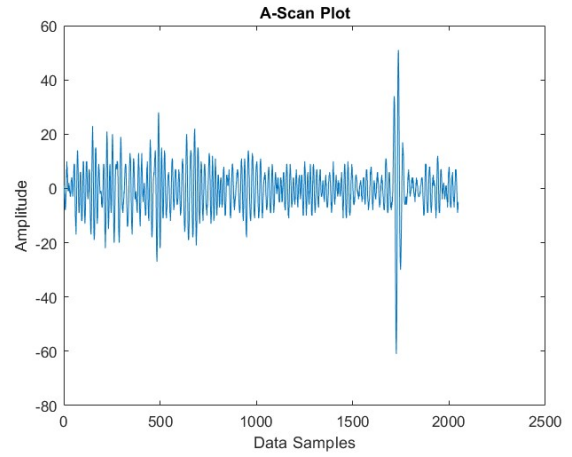


Fig.10.1 Experimental A-Scan plot for News10

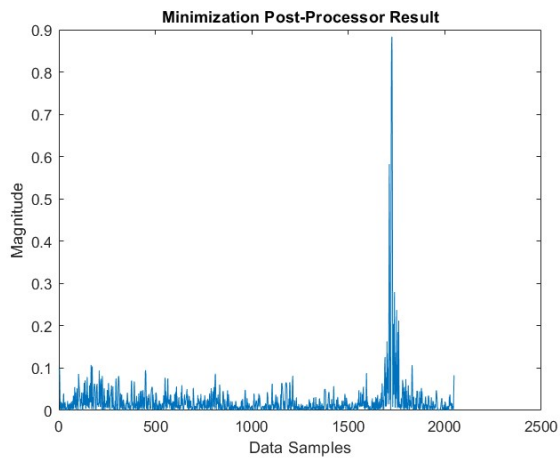


Fig.9.3 Minimization post-processor result for News9

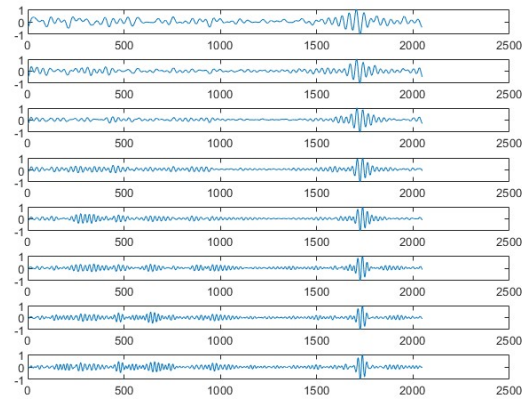


Fig.10.2 8 Observation channels for News10

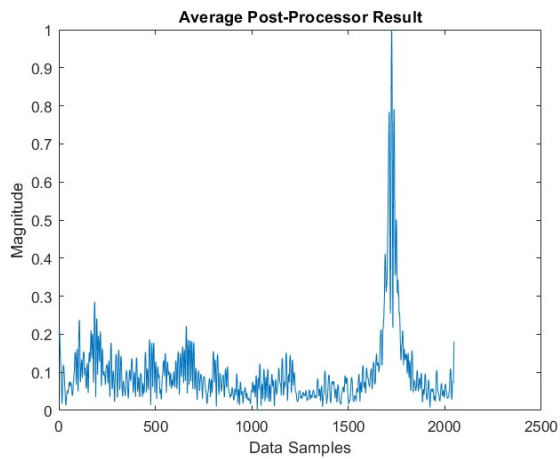


Fig.9.4 Average post-processor result for News9

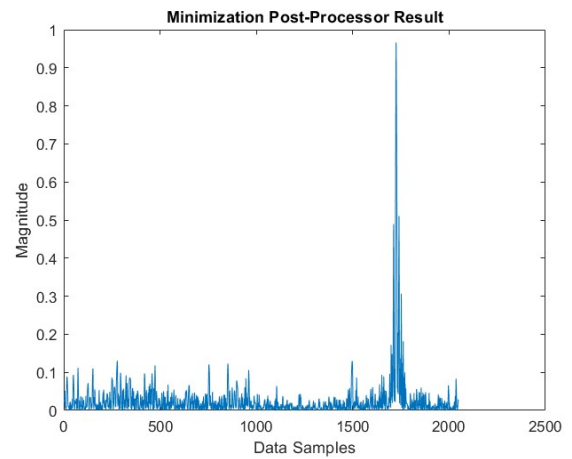


Fig.10.3 Minimization post-processor result for News10

X. News10

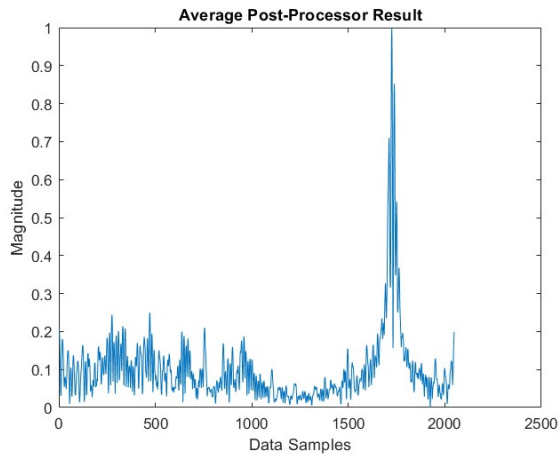


Fig.10.4 Average post-processor result for Newscan10

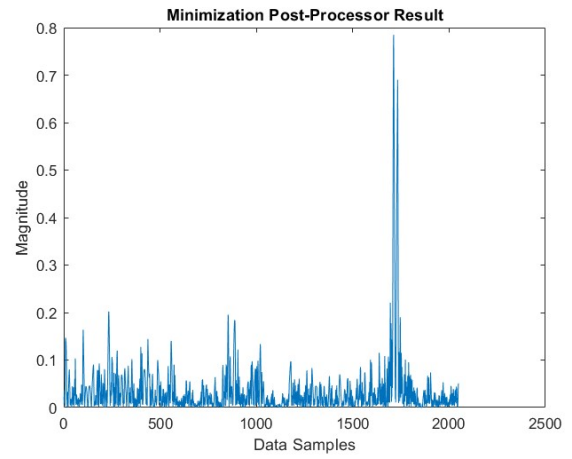


Fig.11.3 Minimization post-processor result for Newscan11

XI. Newscan11

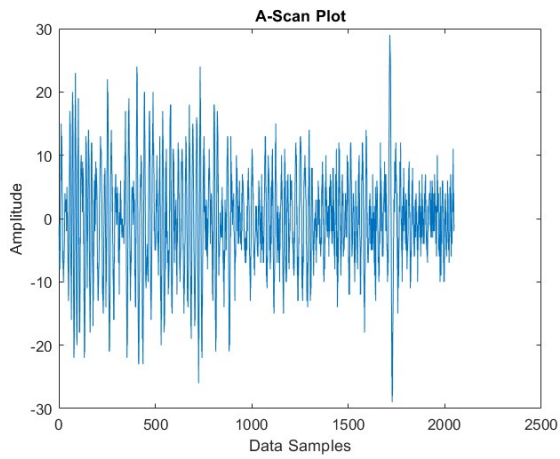


Fig.11.1 Experimental A-Scan plot for Newscan11

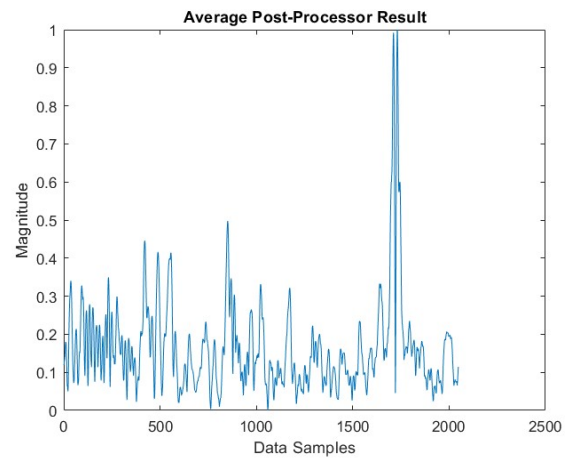


Fig.11.4 Average post-processor result for Newscan11

XII. Newscan12

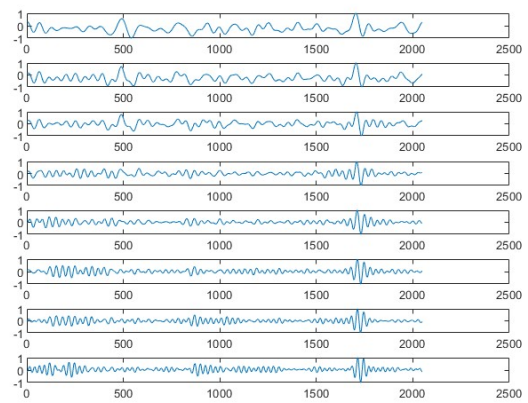


Fig.11.2 8 Observation channels for Newscan11

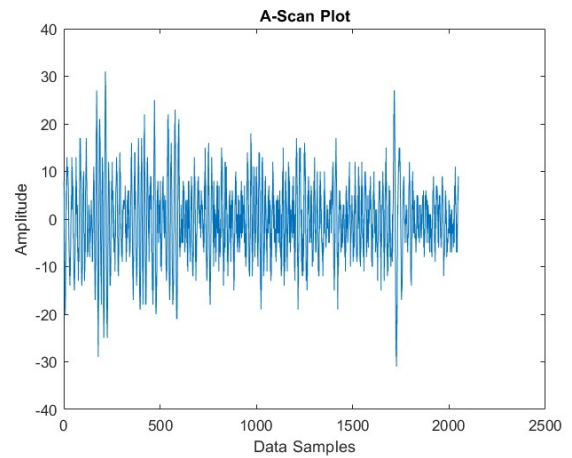


Fig.12.1 Experimental A-Scan plot for Newscan12

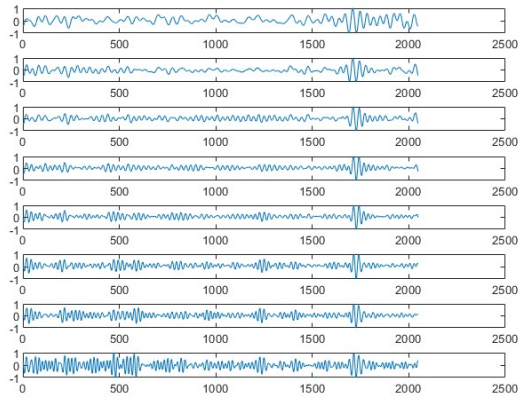


Fig.12.2 8 Observation channels for Newsan12

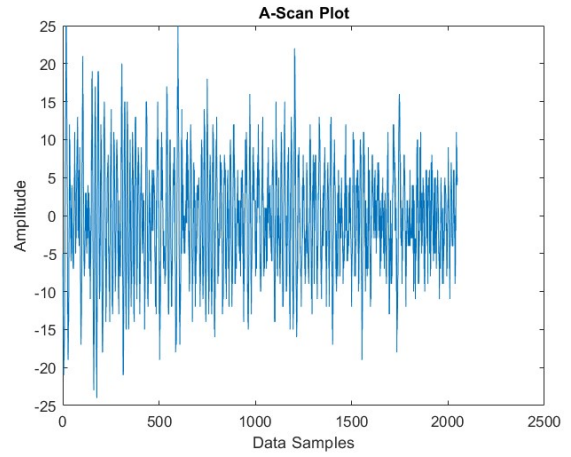


Fig.13.1 Experimental A-Scan plot for Newsan13

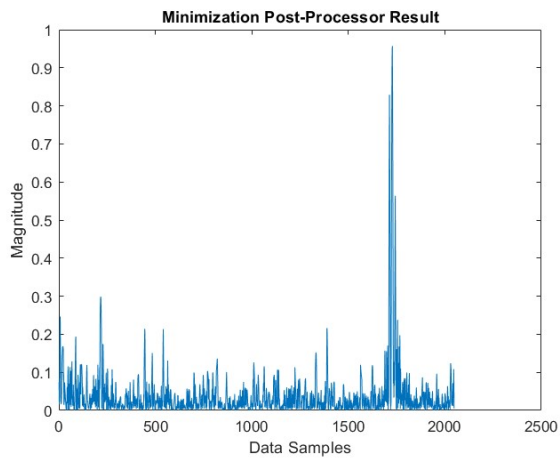


Fig.12.3 Minimization post-processor result for Newsan12

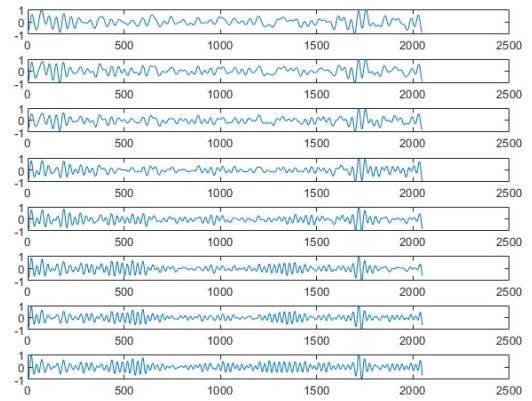


Fig.13.2 8 Observation channels for Newsan13

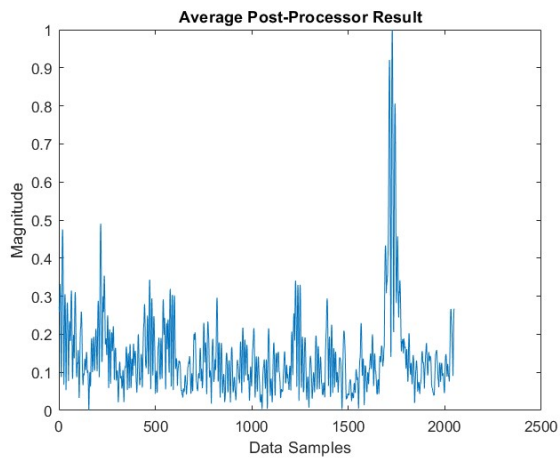


Fig.12.4 Average post-processor result for Newsan12

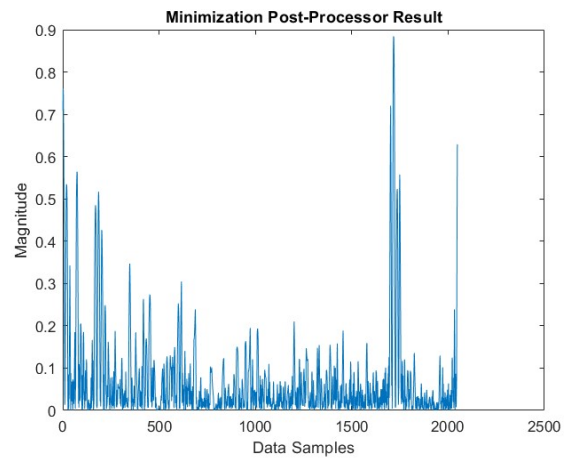


Fig.13.3 Minimization post-processor result for Newsan13

XIII. Newsan13

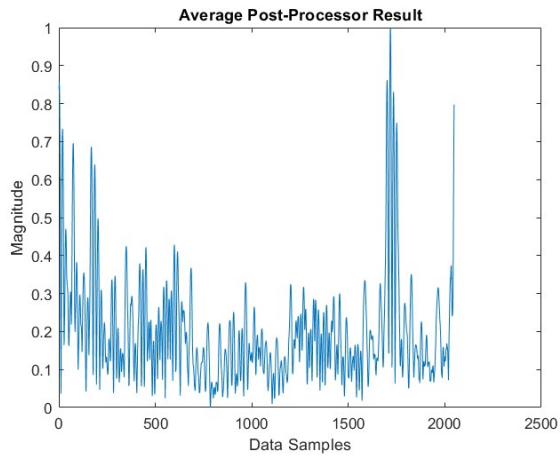


Fig.13.4 Average post-processor result for Newscan13

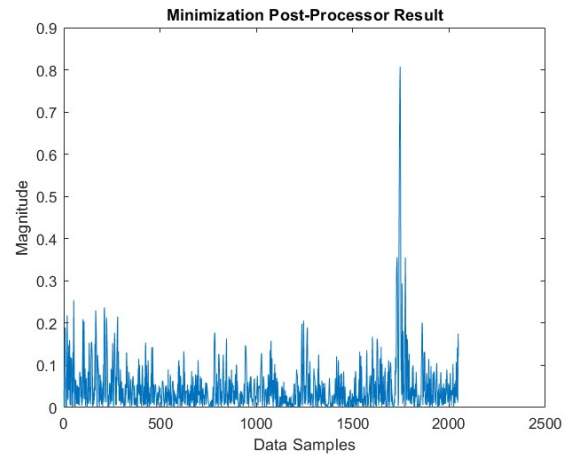


Fig.14.3 Minimization post-processor result for Newscan14

XIV. Newscan14

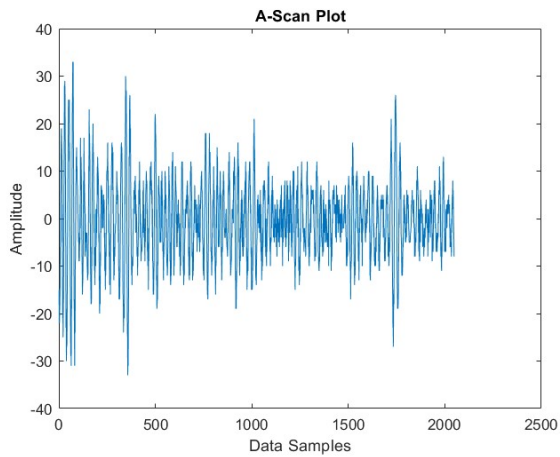


Fig.14.1 Experimental A-Scan plot for Newscan14

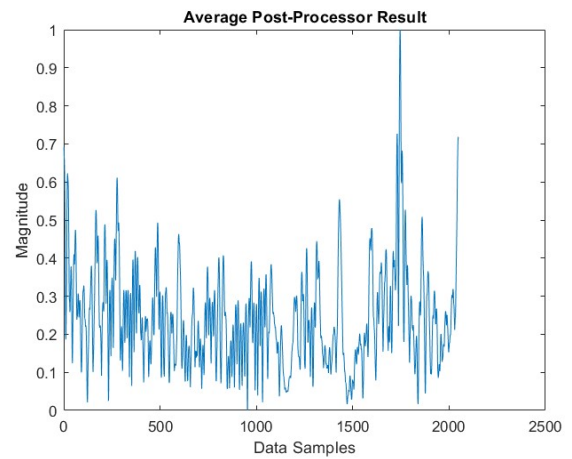


Fig.14.4 Average post-processor result for Newscan14

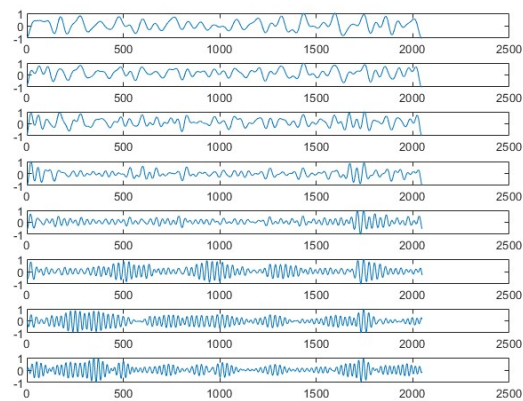


Fig.14.2 8 Observation channels for Newscan14

Table 1. gives the values of the original FCR, FCR after minimization, the improvement in the FCR and also the window size and the degree of overlap for each experimental(real) ultrasonic A-scan data.

Newscan	FCR Original	FCR Min	FCR Improved	Window Size	Gap
1	5.7533	17.5633	11.81	60	10
2	-2.0708	12.5489	14.6197	40	7
3	-7.2636	1.7421	9.0057	60	4
4	-0.7558	6.7358	7.4916	60	8
5	-3.7417	8.6275	12.3692	60	7
6	-3.5218	7.4467	10.9685	60	5
7	-4.1990	7.4858	11.6848	60	7
8	-0.1436	9.7097	9.8533	40	6
9	5.7533	18.3009	12.5476	60	11
10	6.7634	17.3859	10.6225	60	9
11	0.9485	11.7854	10.8369	40	6
12	0	10.1034	10.1034	60	9
13	-2.8534	1.3033	4.1567	60	4
14	-1.7430	10.0454	11.7884	40	8

Table 2. gives the values of the original FCR, FCR after averaging, the improvement in the FCR and also the window size and the degree of overlap for each experimental(real) ultrasonic A-scan data.

Newsan	FCR_Original	FCR_Avg	FCR_Improved	Window_Size	Gap
1	5.7533	10.1378	4.3845	60	10
2	-2.0708	7.4609	9.5317	40	7
3	-7.2636	0.9049	8.1685	60	4
4	-0.7558	4.4715	5.2273	60	8
5	-3.7417	5.9777	9.7194	60	7
6	-3.5218	3.9363	7.4581	60	5
7	-4.1990	5.8432	10.0422	60	7
8	-0.1436	7.2634	7.407	40	6
9	5.7533	10.9165	5.1632	60	11
10	6.7634	12.0558	5.2924	60	9
11	0.9485	6.0616	5.1131	40	6
12	0	6.1863	6.1863	60	9
13	-2.8534	1.3367	4.1901	60	4
14	-1.7430	2.8719	4.6149	40	8

IV. DISCUSSION

It can be seen from the results obtained that the experimental(real) ultrasonic A-scans having a quite good FCR originally such as the scans 1, 9 10 have been tremendously improved. Also, the experimental(real) ultrasonic A-scans which do not have quite a good FCR originally and the flaw is very poorly visible such as the scans 3, 5, 6, 7 have also been greatly improved in terms of their FCR and visibility.

Future work would include:

- I. Implementing a **Neural Networks** Post-Processor for the best results.
- II. Embedding the corresponding HLS code/Verilog code onto an FPGA platform for real time evaluation of the ultrasonic data.
- III. Implementing both FPGA and ARM together.

Implementing the algorithm onto the FPGA platform would require the following blocks:

- I. A block which captures the incoming data from the ADC and the transducer.
- II. A block which implements the split spectrum processing and contains the components for FFT, windowing, IFFT, normalization and the post processor.
- III. A block for communication with the host PC.
- IV. A Finite State Machine for controlling the process flow of the split-spectrum processing and data acquisition and transmission.

V. The output may be visualised on a LCD or softwares such as GTKWave.

V. CONCLUSION

Both minimization and averaging post processors produce quite satisfying results and improve the flaw visibility to a great extent. However, the minimization post-processor is quite better than the averaging post-processor in isolating the flaw and greatly maximizing its visibility while suppressing all the clutter echo information.

Table 3. provides a direct comparison between the improvement in the FCR of minimization and average post-processors.

Newsan	FCR_Original	FCR_Improved(Min)	FCR_Improved(Average)
1	5.7533	11.81	4.3845
2	-2.0708	14.6197	9.5317
3	-7.2636	9.0057	8.1685
4	-0.7558	7.4916	5.2273
5	-3.7417	12.3692	9.7194
6	-3.5218	10.9685	7.4581
7	-4.1990	11.6848	10.0422
8	-0.1436	9.8533	7.407
9	5.7533	12.5476	5.1632
10	6.7634	10.6225	5.2924
11	0.9485	10.8369	5.1131
12	0	10.1034	6.1863
13	-2.8534	4.1567	4.1901
14	-1.7430	11.7884	4.6149

VI. ACKNOWLEDGEMENT

I would like to extend my sincere thanks to Dr. Erdal Oruklu for giving me this opportunity to work under him on this research. I thank him for the constant support and guidance throughout the research . I would also like to thank him for all the knowledge he gave me about the processing of ultrasonic signals. I would also like to acknowledge ISRE undergraduate research programme for giving me the opportunity to work under Dr. Erdal Oruklu.

VII. REFERENCES

- I. Jafar Saniee, Fellow, IEEE, Erdal Oruklu, Senior Member, IEEE, and Sungjoon Yoon System-on-Chip Design for Ultrasonic Target Detection Using Split-Spectrum Processing and Neural Networks *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 58, no. 7, July 2011

ENGR 498-07 Research in Artificial Intelligence and Deep Learning

Artificial Intelligence System for Emotion Recognition and Text Analytics

Team Members:

Reshu Agarwal
Namrata Chaudhari
Meghna Narwade

Advisor: Dr. Jafar Saniie

Summer 2021

Artificial Intelligence System for Emotion Recognition and Text Analytics

ABSTRACT :

Companies around the world are trying to harness the power of emotional intelligence to improve their business processes. Emotional analysis can help to gain an accurate understanding of customer response which can be used to improve an existing process, seize new opportunities, and reduce costs in any business facing customers. In this project, we propose an artificial intelligence based stand alone system which will allow us to classify and analyse facial expression in real time and perform sentiment analysis by examining the body of the text (extracted from audio) to understand the opinion expressed by it. This helps us provide a deeper understanding of how customers really feel at a given time. The proposed system uses a deep neural network (DNN) for classifying 8 basic emotions based on features extracted from facial expression and uses pretrained sentiment analysis tools to quantify text (extracted from audio) based on polarity.

INTRODUCTION :

The aim of this project is to build a stand alone system capable of classifying emotions from real time video and categorizing the text extracted from audio as positive, negative or neutral. This can be used by users to analyze and improve their behavioral skills and maintain a good conversation tone. It can be used by companies in the market research industry by employing behavioral methods that observe user's reaction while interacting with a brand or product along with the traditionally used review analysis. The proposed system extracts the audio and visual cues from real time audio and video respectively, and uses these extracted cues to perform facial expression recognition and text sentiment analysis. The facial expression recognition pipeline classifies emotions from the detected faces in the frame (of the video) using a deep neural network by extracting vectorized landmarks features from the detected faces. The text sentiment analysis pipeline uses pretrained sentiment analysis tools provided in various Pythonic NLP libraries.

RELATED WORKS:

Effective communication involves two components: Verbal cues and Non verbal cues. The proposed system covers the verbal aspect of communication by performing text sentiment analysis and non-verbal aspect of communication by analysing facial expressions.

Facial emotion detection system:

In recent years, advances in facial expression detection have accelerated, and more and more experts have been involved in the development of emotion recognition. The research of expression recognition in computer vision focuses on the feature extraction and feature classification. Feature extraction refers to extracting landmarks from faces that can be used for classification from input pictures or video streams. There are multiple methods for feature extraction from detected faces. The facial expression classification refers to the use of specific algorithms to identify the categories of facial expressions according to the extracted features. Commonly used methods of facial expression classification are Hidden Markov Model (HMM), Support Vector Machine (SVM), AdaBoost, and Artificial Neural Networks (ANN).

Techniques for facial emotion detection using landmark extraction :

Research Paper	Number of landmarks	Method of landmark detection	Dataset used	Classifier used	Accuracy
<u>Real time emotion recognition system using facial expression and EEG</u>	10	Manually placed through optical flow algorithm	Own database	CNN	93.02%
<u>Real time facial expression recognition in Video</u>	22	Manually placed using feature displacement approach	CK+ database	SVM	86.0%
<u>Real-time Mobile Facial Expression Recognition System</u>	77	Extracted using STASM library	CK+ database	SVM	85.8%

<u>A fuzzy logic approach for real time facial recognition of facial emotions</u>	68	Extracted using DLIB library	CK+ database	FURIA	83.2%
Our approach: Response sentiment analysis system.	68	Extracted using DLIB library	Images from CK+ database, JAFFE database, TFEID database, RaFD database	DNN	86.75%

Text Sentiment Analysis:

In the proposed system, text sentiment analysis is performed on the extracted real time audio which is converted to text. Speech to text conversion can be done using various available API's and python libraries.

The most popular speech to text conversion APIs include Google Cloud Speech, IBM and Rev.ai

Link	Result
<u>A Benchmarking of IBM, Google and Wit Automatic Speech Recognition Systems</u>	This research paper differentiates among IBM, Google cloud speech, & Wit. Result: Google Cloud Speech dominates
<u>Which Automatic Transcription Service is the Most Accurate?</u>	Differentiating among various speech to text APIs available Result: 1st Google cloud speech & 2nd Temi by Rev.ai
<u>How Reliable is Speech-to-Text in 2021?</u>	An article that differentiates among different speech to text APIs. Result: 1st Temi by Rev.ai & 2nd Google cloud speech

Sentiment analysis (opinion mining) is a text mining technique that uses machine learning and natural language processing (nlp) to automatically analyze text for the sentiment of the speaker (positive, negative, neutral). Text Sentiment analysis is normally implemented using 2 approaches:

1. Constructing supervised machine learning and deep learning models. Text sentiment can be classified using machine learning models like Support Vector Machine (SVM), Naive Bayes and Decision Tree.
2. Using unsupervised lexicon based approaches. Determining polarity of text using pretrained sentiment analysis tools from various Python NLP libraries (TextBlob, Vader)

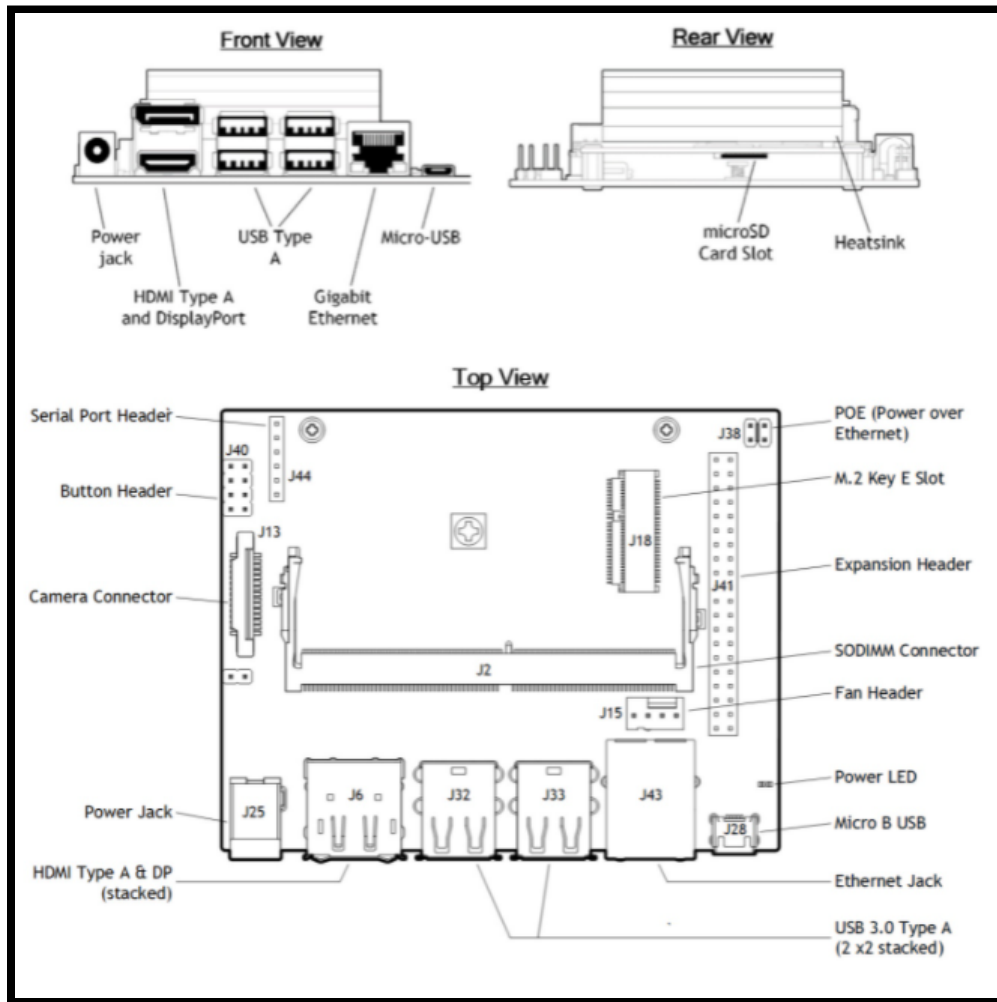
We have used an unsupervised lexicon based approach to implement text sentiment analysis.

SYSTEM COMPONENTS:

The proposed system can be implemented using a laptop PC. In addition we have used Nvidia's Jetson nano as a hardware component. Jetson nano is a compact, low voltage System on Chip (SoC) designed to carry out programmed instructions. It provides Maxwell 128 core GPU emphasized on Deep Learning in its hardware design and software libraries. It is capable of running multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing. The Jeston nano is powered using a 5W 4A power supply. The camera used is Raspberry Pi MIPI CSI which has a frame rate of about 90 fps.

The programming language used to code the system is python. Python is an open source language and has extensive support libraries which allow us to perform video processing, speech recognition and natural language processing (NLP) tasks.

Jetson Nano Specifications



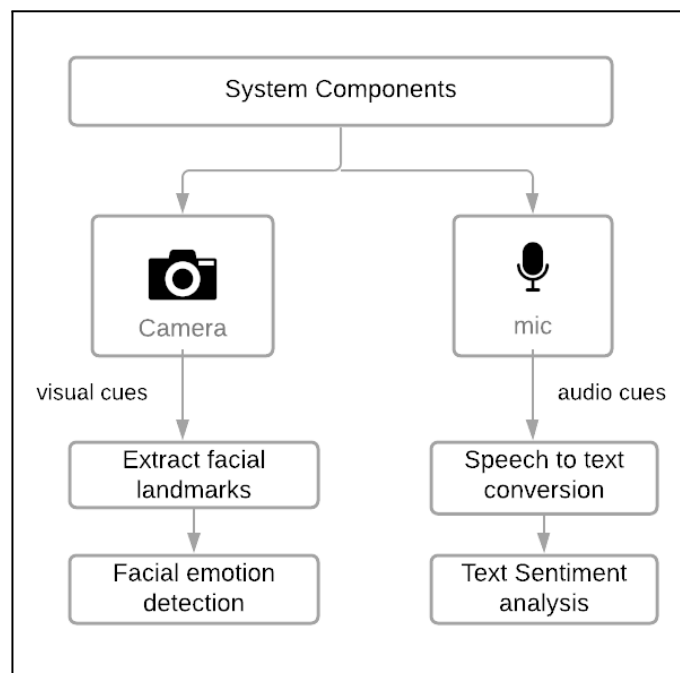
GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 4x 1080p @ 30 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 2x 4K @ 30 8x 1080p @ 30 18x 720p @ 30 (H.264/H.265)
Camera	1x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI 2.0 and eDP 1.4
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I ² C, I ² S, SPI, UART
Mechanical	100 mm x 80 mm x 29 mm

Python libraries used

Library	Use
OpenCV	Video Processing
Dlib	Face detection and landmark extraction
Tensorflow	Build and train Deep Neural Network
Pyaudio	To record audio
Speech Recognition	Speech to text conversion
Punctuator	Add punctuations to text
TextBlob	Simple API to perform basic NLP tasks

SYSTEM OVERVIEW :

The proposed system uses the camera to extract visual cues which are used to perform facial expression recognition and uses the mic to extract audio cues which are converted to text and used to perform text sentiment analysis.



We need to extract the generated audio and visual cues simultaneously from a real time scenario. This is being done using multi-threading which helps us to run multiple function calls simultaneously i.e. one thread records the video using opencv and the other thread records the audio using pyaudio and the output of each of these threads will then be served as an input to the two modules implemented which will then predict emotions and analyze the polarity of the content obtained from the audio.

The frame rate for the multithreading process is calculated by: dividing the total number of frames with the elapsed time of the program & the fps recorded was about 4-5fps.

FACIAL EMOTION DETECTION SYSTEM :

The facial emotion detection module is built from scratch to detect one of eight emotions: happiness, sadness, anger, surprise, fear, disgust and contempt, The visual cues are used to detect faces and extract 68 landmarks (features) which are then fed to the deep neural network (DNN) to classify emotion from the given frame.

Facial landmark Extraction :

Convolutional neural networks can be used to classify raw input images but performing feature landmark extraction allows us to achieve comparable results with a simpler neural network.

Facial landmark extraction is performed using the Dlib library in python. The extracted features are then fed as an input to the neural network. The Dlib library detects faces from the input image and uses the predictor function to place 68 landmarks on the detected faces. It uses Histogram of Oriented Gradients (HOG) for Object Detection with a linear classifier, an image pyramid, and sliding window detection scheme to detect faces in an image. Once the region of face is determined, facial landmarks will be detected using One Millisecond Face Alignment with an Ensemble of Regression Trees. The Dlib library accurately detects landmarks from the detected faces at an angle of -25 to +25 degrees in any direction. (*Code for checking angle: Appendix F*)

The coordinates of the 68 landmarks have a fixed orientation (shown in the figure below). The resultant landmarks are given in the form of an array.

Resultant array : = [(x0,y0) , (x1,y1), , (x67,y67)]



Extracting features from faces allows us to construct a simple neural network with less training data which will converge faster as compared to traditional CNNs.

Neural Networks perform best when the feature vector is scaled to a small range of values $[-1, 1]$. In order to optimize the gradient descent process normalize the facial landmarks and align them at the tip of the nose (x_{33}, y_{33}) . Vectorization of facial landmarks is achieved by putting tensors of 2-dimensional coordinates into a vector which is fed into the neural network.

Shifting the origin to the tip of the nose (x_{33}, y_{33}) :

For (x, y) in resultant array:

$$x = x - x_{33}$$

$$y = y - y_{33}$$

Normalizing the coordinates in range $[-1, 1]$:

scale height = y_8 // coordinate $(x_8, y_8) := (*, -1)$

scale width = $\max(|x_0|, |x_{16}|)$

For (x, y) in resultant array:

$$x = x / \text{scale width}$$

$$y = y / \text{scale height}$$

The normalized coordinates are stored in the form of a feature vector.

$[(x_0, y_0), (x_1, y_1), \dots, (x_{67}, y_{67})] \rightarrow [x_0, y_0, x_1, y_1, \dots, x_{67}, y_{67}]$

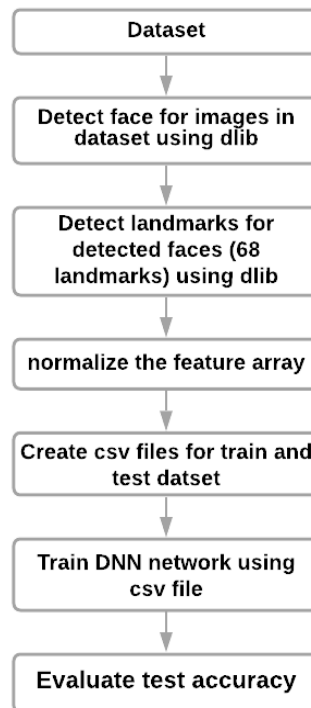
The result data can be stored in a CSV file with an integer indicating the emotion in the last column (label L) which can be used to train and test the neural network.

Building a Deep Neural Network (DNN):

The dataset was created using images from CK+ (Extended Cohn-Kanade dataset), JAFFE dataset, TFEID (Taiwanese Facial Expression Image Database), and RaFD (Radboud Faces Database). The created dataset is composed of eight classes with a total of 3000 images divided into training and test sets. The vectorized facial landmarks of images from the dataset are stored in a CSV file along with an integer indicating the emotion. The test and train csv files are then used to train and evaluate the DNN.

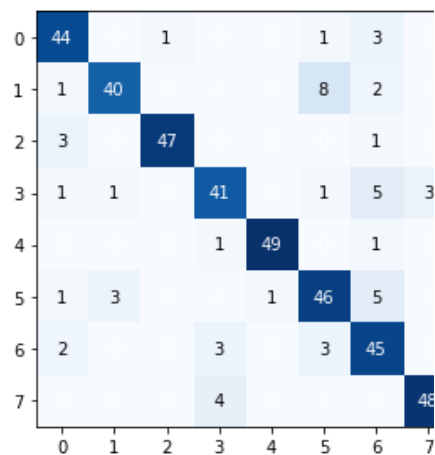
The model used in building the deep neural network is a sequential model with three hidden layers. The type of layers used is dense which implies that every neuron in the dense layer receives input from all neurons of the previous layer. The activation function used was a sigmoid. Adam optimizer allows the framework to adjust the step size depending on the loss. Accuracy obtained after testing the model: 86.75%

Implementation Flowchart



Model Summary:

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 272)	37264
dense_1 (Dense)	(None, 544)	148512
dense_2 (Dense)	(None, 272)	148240
dense_3 (Dense)	(None, 8)	2184
Total params: 336,200		
Trainable params: 336,200		
Non-trainable params: 0		

Confusion Matrix for the test set classification:

- 0: angry
- 1: contempt
- 2: disgust
- 3: fear
- 4: happiness
- 5: neutral
- 6: sadness
- 7: surprise

Real time facial emotion Detection

The system uses OpenCV, to read video frames either by using the feed from a camera connected to a computer or by reading a video file. We then perform face

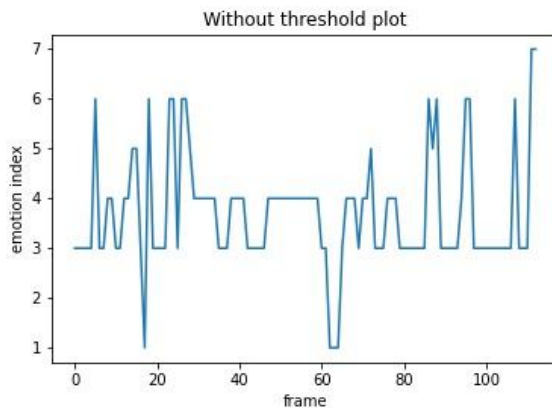
detection and facial landmark extraction on the frame and feed the normalized landmark coordinates into the DNN which classifies the emotion of the faces in the frame.

Since we use the sigmoid activation function in the neural network, the output of the DNN is an array in which each element represents the probability of (indexed) emotion occurring independent of other emotions. The sum of the array elements may not necessarily be 1 as sigmoid function doesn't treat emotions to be mutually exclusive.

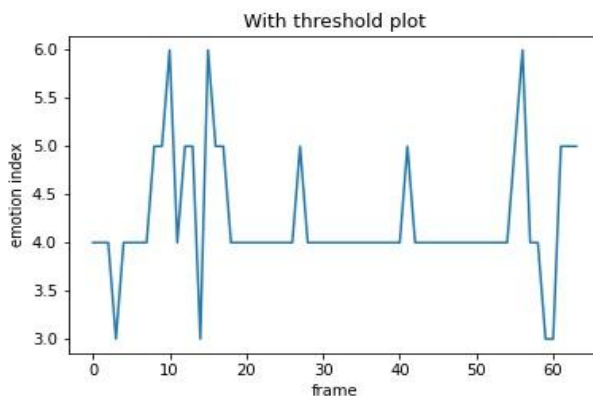
This allows us to improve the accuracy of our system while performing real time processing by setting a threshold for the level of confidence for each of the eight emotions. We only display the emotion if the confidence level of that emotion is greater than its threshold value. If the emotion detected does not cross the threshold value we display the emotion rendered in the previous frame.

The facial emotion detection of a video performed with and without threshold is shown below.

Without threshold:



With Threshold:



The frame rate achieved for real time face emotion detection is about 8.9 fps for laptop PC and 4.1 on Jetson Nano.

On Laptop :

```
fps start
fps stop

[INFO] elapsed time: 12.74
[INFO] approx. FPS: 8.95
```

On Jetson Nano :

```
fps stop

fps recorded on Jetson nano
[INFO] elapsed time: 27.20
[INFO] approx. FPS: 4.19
```

TEXT SENTIMENT ANALYSIS SYSTEM :

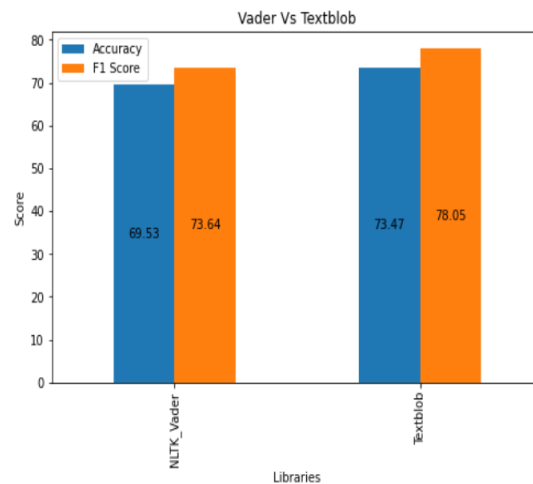
The system converts real time audio to text using the Speech Recognition library in python. We use the Pyaudio library to record audio from a mic. The recorded audio is broken down into chunks and processed bit by bit using the Recognizer function in the Speech recognition library which transcribes the audio. The transcribed audio is split into sentences before using the Punctuation Model adding the required punctuation to the text. This text is then used to perform text sentiment analysis .

The proposed system determines the polarity of text using pretrained sentiment analysis tools from various Python NLP libraries (TextBlob, Vader). The most widely used pretrained libraries for estimating polarity of text are TextBlob and Vader.

The following are some negative and positive interviewee responses to check how well these libraries can classify their polarity and overall we find TextBlob with Naive Bayes yields more satisfying results. The numbers shown in the table are the polarity of each sentence where -100 means negative and +100 means positive.

	content	textblob	textblob_bayes	nlTK_vader
0	I've enjoyed and grown in my current role	25	65	51
1	I am an ambitious and driven individual. I thrive in a goal-oriented environment	12	92	48
2	What makes me unique is my ability to meet and exceed deadlines	38	59	32
3	While I highly valued my time at my previous company, there are no longer opportunities for growth that align with my career goals	0	3	73
4	I hated the job and the company. They were awful to work for.	-95	-60	-80
5	I do good work	70	4	44
6	I tend to lose my patience with incompetent people.	-35	-33	-70
7	I missed too much work.	20	-10	-30

The accuracy of Textblob vs Vader was compared by testing these models on the IMDB dataset and the product review dataset. It can be seen that TextBlob has higher precision and F1 score for these datasets



The proposed system uses the TextBlob library with Naive Bayes Classifier to estimate the polarity of the text. TextBlob is a python library of Natural Language Processing (NLP) that uses the Natural Language ToolKit (NLTK) to perform its functions. NLTK is a library that provides easy access to many lexical resources and allows users to work with categorization, classification and many other tasks. It calculates average polarity and subjectivity over each word in a given text using a dictionary of adjectives and their hand-tagged scores. It actually uses a pattern library for that, which takes the individual word scores from sentiwordnet. The TextBlob with Naive Bayes calculates the sentiment score by NaiveBayesAnalyzer trained on a dataset of movie reviews. We use the polarity calculated by TextBlob to classify text as either positive, negative or neutral by

setting a threshold value. The polarity value lies in the range of $[-1, 1]$, where -1 indicates negativity and $+1$ indicates positivity.

Threshold values set to classify text into three classes:

Polarity above 60% is classified as Positive

Polarity between 40% and 60% is classified as Neutral

Polarity below 40% is classified as Negative

Analysis of a transcribed text passage is done as follows:

Number of positive sentences in the passage: x

Number of negative sentences in the passage: y

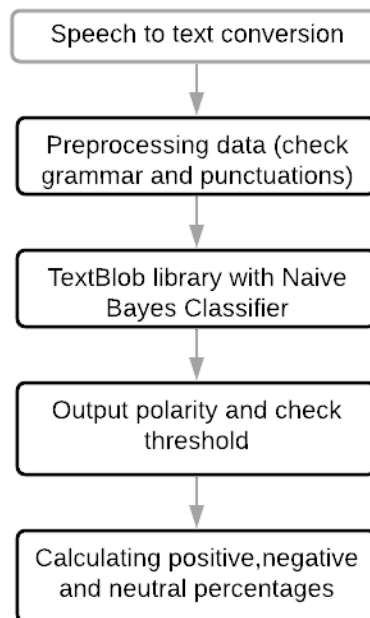
Number of neutral sentences in the passage: z

Total number of sentences in a passage: $x+y+z$

Overall positivity of the passage: Sum of polarities above 60% / Total number of sentences in a passage

Overall neutrality of the passage: Sum of polarities between 40% - 60% / Total number of sentences in a passage

Overall negativity of the passage: Sum of polarities below 40% / Total number of sentences in a passage



Text Sentiment Analysis Workflow

RESULTS AND DISCUSSION :

The integrated system extracts video and audio simultaneously with a frame rate of 4-5 fps. The facial emotion detection system successfully detects facial expression of faces detected in real time video with an accuracy of about 86.75%. The audio from the video is successfully extracted, converted to text, cleaned and processed to determine if the attitude of the speaker in a given situation is positive, negative or neutral.

The proposed system can be used in a wide sale of applications. It can be used to make the interview process bias free by analyzing the emotional expressions and answers of prospective candidates for its entry-level jobs. Candidates can also use this system analysing their own responses during a mock interview.It can be used to perform market research by analysing customers' response to a particular advertising scheme. If customized this system can be used for the interrogation process.

The results and applications are used in the video attached.

<https://drive.google.com/file/d/1wnGr-dIYQGUqjDZS2CVY2-WS850fUCvO/view?usp=sharing>

CONCLUSION :

The project is research on face expression recognition and analysing text for the sentiment , which allows us to know a way of sensing emotions that can be considered as mostly used AI and pattern analysis applications. To summarize, we have developed a system that can perform emotion detection and text sentiment analysis in real time.

FUTURE WORK :

The system can be further improved by covering more aspects of communication skills like using the extracted audio from video to perform speech emotion detection to recognize the emotional aspects of speech irrespective of the semantic contents. The accuracy of the facial emotion detection and text sentiment analysis system can be further improved to make the system more feasible and accurate.

REFERENCES :

[1] Dlib Library python:

<https://pypi.org/project/dlib/>

[2] Textblob Library python:

<https://pypi.org/project/textblob/>

[3] OpenCV:

<https://pypi.org/project/opencv-python/>

[4]Speech Recognition library python:

<https://pypi.org/project/SpeechRecognition/>

[5] Recording Audio and Video together code:

<https://stackoverflow.com/questions/14140495/how-to-capture-a-video-and-audio-in-python-from-a-camera-or-webcam>

[6] Facial emotion recognition dataset images:

<https://github.com/spenceryee/CS229>

[7] Angle detection for landmarks:

<https://www.programmingsought.com/article/27703847966/>

[8] Related works in facial emotion detection:

- [Development of a Real-Time Emotion Recognition System Using Facial Expressions and EEG based on machine learning and deep neural network methods](#)
- [Real time facial expression recognition in video using support vector machines](#)
- [Real-time Mobile Facial Expression Recognition System](#)
- [A fuzzy logic approach for real time facial recognition of facial emotions](#)

CODE:

Appendix A: Extracting audio and visual cues

```
#AudioVideo recording code

import cv2
import pyaudio
import wave
import threading
import time
import subprocess
import os

class VideoRecorder():

    # Video class based on openCV
    def __init__(self):

        self.fourcc = "MJPG"          # capture images (with no dec
rease in speed over time; testing is required)
        self.dim = (640,480)        # video formats and sizes als
o depend and vary according to the camera used
        self.video_filename = "Fer.avi"
        self.fps = 6
        self.cap = cv2.VideoCapture(0)
        self.open = True
        self.write = cv2.VideoWriter_fourcc(*self.fourcc)
        self.vid = cv2.VideoWriter(self.video_filename, self.wri
te, self.fps, self.dim)

        self.frame_counts = 1
            # fps should be the minimum constant rate at
which the camera can

        self.start_time = time.time()

    # Video starts being recorded
    def record(self):
        counter = 1
        timer_start = time.time()
        timer_current = 0
```

```

        while (self.open==True):
            ret, frame = self.cap.read()

            if ret:
                self.vid.write(frame)
                print(str(counter) + " " + str(self.count)
+ " frames written " + str(timer_current))
                self.frame_counts += 1
                counter += 1
                timer_current = time.time() - timer_start
                time.sleep(0.16)
            #
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2G
RAY)

            cv2.imshow('frame', frame)
            cv2.waitKey(1)

# Finishes the video recording therefore the thread too
def stop(self):

    if self.open==True:
        self.open=False
        self.vid.release()
        self.cap.release()
        cv2.destroyAllWindows()
    else:
        pass

# Launches the video recording function using a thread

def start(self):
    t1 = threading.Thread(target=self.record)
    t1.start()

class AudioRecorder():

# Audio class based on pyAudio and Wave
def __init__(self):

    self.open = True
    self.rate = 44100
    self.frames_per_buffer = 1024
    self.channels = 2

```

```

self.format = pyaudio.paInt16
self.audio_filename = "video 1.wav"
self.audio = pyaudio.PyAudio()
self.stream = self.audio.open(format=self.format,
                               channels=self.channels,
                               rate=self.rate,
                               input=True,
                               frames_per_buffer = self.f
rames_per_buffer)
    self.audio_frames = []

# Audio starts being recorded
def record(self):

    self.stream.start_stream()
    while (self.open == True):
        data = self.stream.read(self.frames_per_buffer)
        self.audio_frames.append(data)
        if self.open==False:
            break

# Finishes the audio recording therefore the thread too
def stop(self):

    if self.open==True:
        self.open = False
        self.stream.stop_stream()
        self.stream.close()
        self.audio.terminate()

        aud = wave.open(self.audio_filename, 'wb')
        aud.setnchannels(self.channels)
        aud.setsampwidth(self.audio.get_sample_size(self.for
mat))

        aud.setframerate(self.rate)
        aud.writeframes(b''.join(self.audio_frames))
        aud.close()

    pass

# Launches the audio recording function using a thread
def start(self):
    t2 = threading.Thread(target=self.record)
    t2.start()

```

```
def start_AVrecording(filename):

    global t1
    global t2

    t1 = VideoRecorder()
    t2 = AudioRecorder()

    t2.start()
    t1.start()

    return filename

def start_video_recording(filename):

    global t1

    t1 = VideoRecorder()
    t1.start()

    return filename

def start_audio_recording(filename):

    global t2

    t2 = AudioRecorder()
    t2.start()

    return filename

def stop_AVrecording(filename):

    t2.stop()
    frame_counts = t1.frame_counts
    elapsed_time = time.time() - t1.start_time
    recorded_fps = frame_counts / elapsed_time
    print("total frames " + str(frame_counts))
    print("elapsed time " + str(elapsed_time))
```

```

print("recorded fps " + str(recorded_fps))
t1.stop()

# Makes sure the threads have finished
while threading.active_count() > 1:
    time.sleep(1)

# Required and wanted processing of final files
def file_manager(filename):

    local_path = os.getcwd()

    if os.path.exists(str(local_path) + "/temp_audio.wav"):
        os.remove(str(local_path) + "/temp_audio.wav")

    if os.path.exists(str(local_path) + "/temp_video.avi"):
        os.remove(str(local_path) + "/temp_video.avi")

    if os.path.exists(str(local_path) + "/temp_video2.avi"):
        os.remove(str(local_path) + "/temp_video2.avi")

    if os.path.exists(str(local_path) + "/" + filename + ".avi"):
:
        os.remove(str(local_path) + "/" + filename + ".avi")

filename = "Default_user"
file_manager(filename)

start_AVrecording(filename)

time.sleep(20)

stop_AVrecording(filename)
print("Done")

```

Appendix B: Real time face emotion detection

```
#Face emotion detection:

import dlib
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# initialize face and facial landmark detector
detector = dlib.get_frontal_face_detector()

# replace with proper path!!!!!!
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

#loading DNN
path_save = "./testsave4"
model_restore = tf.keras.models.load_model(path_save)

model_restore.summary()

#text characteristics
window_name = 'Image'
font = cv2.FONT_HERSHEY_SIMPLEX
fontScale = 1
color = (0, 0, 255)
thickness = 2

#emotion detected dictionary
emotions = { 0:"angry" ,1:"contempt" ,2:"disgusted",3:"fearful",
  4:"happy", 5:"neutral",6:"sad",7:"surprised"}
print(emotions)

#normalize and add to array function
def normalize(detected_face,shape,new_arr):
    i=1
    arr = []
    x_scale =-1*(shape.parts()[0].x - shape.parts()[33].x)
    y_scale = shape.parts()[8].y -shape.parts()[33].y
    for p in shape.parts():
        # detected_face = cv2.circle(detected_face,(p.x,p.y), 2,
(0,0,255), -1)
        p=p-shape.parts()[33]
```

```
x_new = p.x / x_scale
y_new = p.y / y_scale
arr = np.append(arr,x_new)
arr = np.append(arr,y_new)
i+=1
return arr
```

```
#finding emotion from output
```

```
def result(test_result,emotion_result,index_result):
    for r in test_result:
        c = ""
        if r[0]>99:
            j=0
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[1]>0.99:
            j=1
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[2]>0.99:
            j=2
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[3]>0.99:
            j=3
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[4]>0.85:
            j=4
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[5]>0.90:
            j=5
            index_result.append(j)
            emotion_result.append(emotions[j])
            c = c + emotions[j] + " "
        if r[6]>0.99:
            j=6
            index_result.append(j)
            emotion_result.append(emotions[j])
```

```

        c = c + emotions[j] + " "
    if r[7]>0.90:
        j=7
        index_result.append(j)
        emotion_result.append(emotions[j])
        c = c + emotions[j] + " "

    return emotion_result, index_result,c

from imutils.video import FPS
# vid = cv2.VideoCapture(0)
vid = cv2.VideoCapture('fer_video.mp4')

fps = FPS().start()

x = 0
analysis_arr = []
analysis_ind = []
prev_c = "unknown"
c=""
out = cv2.VideoWriter('output.mp4', -1, 20.0, (640,480))

while True:
    ret, frame = vid.read()
    print(x)
    if ret:
#         print(frame.shape)
        gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        faces = detector(gray, 0)
        detected_face = frame
        new_arr = []
#         print(faces)
        fps.update()

        for f in faces:
            shape = predictor(gray, f)
            pred = normalize(detected_face,shape,new_arr)
            new_arr.append(pred)
        q=0
        for f in faces:
            arr_x = np.reshape(new_arr[q], (1,136))
            index_result=[]
            emotion_result=[]

```



```

        test_result = model_restore.predict(arr_x)
#         print(test_result)
        emotion_result, index_result, c = result(test_result
,emotion_result,index_result)
        if c=="":
            c=prev_c
        if len(index_result)!=0:
            analysis_arr.append(emotion_result[0])
            analysis_ind.append(index_result[0])
            detected_face = cv2.rectangle(detected_face, (f.tl_c
orner().x, f.tl_corner().y),
                                          (f.br_corner().x, f.br_corner(
).y), (0,255,0), 3)
            frame = cv2.putText(frame, c, (f.tl_corner().x, f.t
l_corner().y), font,
                                fontStyle, color, thickness, cv2.LINE_AA)
            q+=1

#         cv2.imwrite(f"Frames/Frame{x}.jpg", frame)
        out.write(frame)
        cv2.imshow('frame', frame)
        prev_c = c
        x += 1
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

out.release()
vid.release()

fps.stop()
print(x)
print("fps start")
print("fps stop\n")
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
print("\n")

cv2.destroyAllWindows()

# print(analysis_ind)
# print(analysis_arr)

```

Appendix C: Text Sentiment Analysis

```
#text sentiment analysis

from textblob import TextBlob
from textblob.classifiers import NaiveBayesClassifier
from textblob.sentiments import NaiveBayesAnalyzer
import nltk
from pydub import AudioSegment
import speech_recognition as sr
from os import path
from nltk import tokenize

nltk.download('movie_reviews')
nltk.download('punkt')
nltk.download('stopwords')

#Converting mp4 to wav format with 128k bitrate
src="debatel.mp4"

AudioSegment.converter = "C:/ffmpeg-4.4-full_build/bin/ffmpeg.exe"
AudioSegment.ffmpeg = "C:/ffmpeg-4.4-full_build/bin/ffmpeg.exe"
AudioSegment.ffprobe = "C:/ffmpeg-4.4-full_build/bin/ffprobe.exe"

sound = AudioSegment.from_file(file=src, format="mp4")
sound.export("recording.mp3", format="mp3", bitrate="128k")

# convert mp3 file to wav

sound = AudioSegment.from_mp3("recording.mp3")
sound.export("transcript.wav", format="wav")

##Code-----
# importing libraries
import speech_recognition as sr
import os
from pydub import AudioSegment
from pydub.silence import split_on_silence

# create a speech recognition object
r = sr.Recognizer()
```

```

# a function that splits the audio file into chunks
# and applies speech recognition
def get_large_audio_transcription(path):
    """
    Splitting the large audio file into chunks
    and apply speech recognition on each of these chunks
    """
    # open the audio file using pydub
    sound = AudioSegment.from_wav(path)
    # split audio sound where silence is 700 miliseconds or more
    and get chunks
    chunks = split_on_silence(sound,
        # experiment with this value for your target audio file
        min_silence_len = 500,
        # adjust this per requirement
        silence_thresh = sound.dBFS-14,
        # keep the silence for 1 second, adjustable as well
        keep_silence=500,
    )
    folder_name = "audio-chunks"
    # create a directory to store the audio chunks
    if not os.path.isdir(folder_name):
        os.mkdir(folder_name)
    whole_text = ""
    # process each chunk
    for i, audio_chunk in enumerate(chunks, start=1):
        # export audio chunk and save it in
        # the `folder_name` directory.
        chunk_filename = os.path.join(folder_name, f"chunk{i}.wav")
        audio_chunk.export(chunk_filename, format="wav")
        # recognize the chunk
        with sr.AudioFile(chunk_filename) as source:
            audio_listened = r.record(source)
            # try converting it to text
            try:
                text = r.recognize_google(audio_listened)
            except sr.UnknownValueError as e:
                print("Error:", str(e))
            else:
                text = f"{text.capitalize()} "
                #print(chunk_filename, ":", text)
                whole_text += text
    # return the text for all chunks detected
    return whole_text

```

```

path = "transcript.wav"
#print("\nFull text:", get_large_audio_transcription(path))
t=get_large_audio_transcription(path)
print(t)

```

```

sentence_break=[]
sentence_break=t.split('.')
print(sentence_break)

```

```

from punctuator import Punctuator
p = Punctuator('punctuator_model/Demo-Europarl-EN.pcl')
semi_final=[]
final=[]
for ele in sentence_break:
    if len(ele)>1:
        test=p.punctuate(ele)
        semi_final=test.split('.')
        for i in semi_final:
            if i!="":
                final.append(i)
                #pre-trained model 1
# #p1=Punctuator('punctuator_model/INTERSPEECH-T-BRNN.pcl')
# pre-trained model 2
# t=p.punctuate(text)
# print(t)
print(final)

```

```

l=[]
b=[]
for i in range(0,len(final)):
    blob=TextBlob(final[i],analyzer=NaiveBayesAnalyzer())
    #print(blob.sentiment)
    l.append(blob.sentiment.p_pos)
    b.append(blob.sentiment.p_neg)

```

```

pos=0
neg=0
neu=0
pos_per=0
neg_per=0
neu_per=0
for i in l:
    if i>0.6:

```

```

        pos=pos+1
        pos_per=pos_per+i
    elif i>0.4 and i<0.6:
        neu=neu+1
        neu_per=neu_per+i
    elif i<0.4:
        neg=neg+1
        neg_per=neg_per+i
# print(l)
# print(len(final))

print("Number of positive sentences in the passage:",pos)
print("Number of negative sentences in the passage:",neg)
print("Number of neutral sentences in the passage:",neu)

print("Overall positivity of the passage:",round(pos_per/sum(l),
2))
print("Overall negativity of the passage:",round(neg_per/sum(l),
2))
print("Overall neutrality of the passage:",round(neu_per/sum(l),
2))

chart=[]
chart.append(round(pos_per/sum(l),2))
chart.append(round(neu_per/sum(l),2))
chart.append(round(neg_per/sum(l),2))

# plt.pie(chart)
mylabels = ["Positive", "Neutral", "Negative"]
mycolors = ["green", "yellow", "red"]
plt.pie(chart, labels = mylabels, colors = mycolors)
my_circle=plt.Circle( (0,0), 0.7, color='white')
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.show()

```

Appendix D: Training and Test csv files

```
#Dataset to csv

import dlib
import cv2
import numpy as np

print("Dlib version: {}".format(dlib.__version__))
print("OpenCV version: {}".format(cv2.__version__))

# initialize face and facial landmark detector
detector = dlib.get_frontal_face_detector()

# replace with proper path!!!!!!
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

import os
import csv
import glob

Classes=['anger', 'contempt', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

x=0

for category in Classes:
    path = glob.glob(f"train/{category}/*.jpg")

    for img in path:

        img_array=cv2.imread(img)
        img_gray = cv2.cvtColor(img_array, cv2.COLOR_RGB2GRAY)
        # plt.imshow(img_gray)
        # plt.show()

        #detect faces in image
        faces = detector(img_gray, 0)
        #print(len(faces),faces)
        if len(faces)!=0:
            detected_face = img_array

            for f in faces:
                # draw bounding box
```

```

        detected_face = cv2.rectangle(detected_face,
            (f.tl_corner().x, f.tl_corner().y),
#top left corner of the d
            (f.br_corner().x, f.br_corner().y),
#bottom right corner of t
            (0,255,0),3)

        landmark_arr = np.array([])
        # detect facial landmarks in a box
        shape = predictor(img_gray, f)

        i=1
        x_scale = max(shape.parts()[33].x - shape.parts(
) [0].x, shape.parts()[16].x - shape.parts()[33].x)
        y_scale = shape.parts()[8].y - shape.parts()[33].
y

        for p in shape.parts():
            detected_face = cv2.circle(detected_face, (p.
x,p.y), 2, (0,0,255), -1)
            p=p-shape.parts()[33]
            x_new = p.x / x_scale
            y_new = p.y / y_scale
            landmark_arr = np.append(landmark_arr,x_new)
            landmark_arr = np.append(landmark_arr,y_new)
            i+=1

        print(x)
        x+=1
        landmark_arr=np.append(arr,Classes.index(category))
        print(landmark_arr)

        with open('train4.csv', 'a+' , newline='') as write_
obj:

            csv_writer = csv.writer(write_obj)
            csv_writer.writerow(landmark_arr)

```

Appendix E: training the Deep Neural Network

```
#Train DNN

import tensorflow as tf

featureDim = 136
classes = 8

model = tf.keras.Sequential(layers = (tf.keras.layers.Dense(272,
    input_shape=(featureDim,), activation='sigmoid'),
    tf.keras.layers.Dense(544, activation='sigmoid'),
    tf.keras.layers.Dense(272, activation='sigmoid'),
    tf.keras.layers.Dense(classes, activation='sigmoid')))
)

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy
    (from_logits=True),
    optimizer='adam',
    metrics=['accuracy'])
model.summary()

def createData(pathToData, featureDim = 136, classes = 8):
    f = open(pathToData, "r")
    x = []
    y = []
    for line in f:
        parse = line.split(',')
        item_x = [float(d) for d in parse[:featureDim]]
        x.append(item_x)
        label = parse[-1]
        label = label[:3]
        y.append(int(float(label)))
    #     print(x)
    #return tf.convert_to_tensor(x, dtype=tf.float32), tf.conver
t_to_tensor(y, dtype=tf.float32)
    return x, y

train_x, train_y = createData("C:/Users/Namrata
Chaudhari/Downloads/Lab 6/Emotion Recognition Using DNN/train4.c
sv",

                                featureDim = featureDim,
                                classes = classes
```



```

)

print(len(train_x))

# import pandas as pd
# data = pd.read_csv("train1.csv")
# print(data.head())

#fit dataset
model.fit(x = train_x, y = train_y, batch_size = 64, shuffle = True, epochs = 1000)

#save model
path_save = "./testsave4"

tf.keras.models.save_model(
model,path_save, overwrite=True, include_optimizer=True, save_format=None , signatures=None, options=None)

#restore saved model
model_restore = tf.keras.models.load_model(
path_save)

model_restore.summary()

# load train dataset
test_x, test_y = createData("C:/Users/Namrata
Chaudhari/Downloads/Lab 6/Emotion Recognition Using DNN/test4.csv",
                             featureDim = featureDim,
                             classes = classes
)

#evaluate test accuracy
model.evaluate(test_x,test_y)

#plot confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

confusion_matrix = confusion_matrix(test_y , result)

plt.figure()

```

```
plt.imshow(confusion_matrix, interpolation='nearest', cmap=plt.cm.Blues)

thresh = confusion_matrix.max() / 2.
for i in range(confusion_matrix.shape[0]):
    for j in range(confusion_matrix.shape[1]):
        plt.text(j, i, format(confusion_matrix[i, j]),
                 ha="center", va="center",
                 color="white" if confusion_matrix[i, j] == 0 or
                 confusion_matrix[i, j] > thresh else "black")
plt.tight_layout()
plt.colorbar()
```

Appendix F: Checking angles for landmark detection

```
#Detect angle code:

import cv2
import numpy as np
import dlib
import time
import math

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
POINTS_NUM_LANDMARK = 68

# Get the biggest face
def _largest_face(dets):
    if len(dets) == 1:
        return 0

    face_areas = [ (det.right()-det.left())*(det.bottom()-det.top()) for det in dets]

    largest_area = face_areas[0]
    largest_index = 0
    for index in range(1, len(dets)):
        if face_areas[index] > largest_area :
            largest_index = index
            largest_area = face_areas[index]

    print("largest_face index is {} in {} faces".format(largest_index, len(dets)))

    return largest_index

# Extract the point coordinates needed for pose estimation from the detection results of dlib
def get_image_points_from_landmark_shape(landmark_shape):
    if landmark_shape.num_parts != POINTS_NUM_LANDMARK:
        print("ERROR:landmark_shape.num_parts-{}".format(landmark_shape.num_parts))
        return -1, None
```

```

    #2D image points. If you change the image, you need to change
    e vector
    image_points = np.array([
(landmark_shape.part(30).x, landmark_shape.part(30).y),
# Nose tip
(landmark_shape.part(8).x, landmark_shape.part(8).y),
# Chin
(landmark_shape.part(36).x, landmark_shape.part(36).y),
# Left eye left corner
(landmark_shape.part(45).x, landmark_shape.part(45).y),
# Right eye right corner
(landmark_shape.part(48).x, landmark_shape.part(48).y),
# Left Mouth corner
(landmark_shape.part(54).x, landmark_shape.part(54).y)
# Right mouth corner
    ], dtype="double")

    return 0, image_points

# Use dlib to detect key points and return the coordinates of several
points needed for pose estimation
def get_image_points(img):

    #gray = cv2.cvtColor( img, cv2.COLOR_BGR2GRAY) # The picture is
adjusted to gray
    dets = detector( img, 0 )

    for f in dets:
        shape = predictor(img, f)
        q=0
        for f in dets:
            img = cv2.rectangle(img, (f.tl_corner().x, f.tl_corner().y),
(f.br_corner().x, f.br_corner().y), (0,255,0), 3)

            q+=1

    if 0 == len( dets ):
        print( "ERROR: found no face" )
        return -1, None
    largest_index = _largest_face(dets)
    face_rectangle = dets[largest_index]

    landmark_shape = predictor(img, face_rectangle)

```

```

    return get_image_points_from_landmark_shape(landmark_shape)

# Get rotation vector and translation vector

def get_pose_estimation(img_size, image_points ):
    # 3D model points.
    model_points = np.array([
        (0.0, 0.0, 0.0),          # Nose tip
        (0.0, -330.0, -65.0),    # Chin
        (-225.0, 170.0, -135.0), # Left eye left corner
        (225.0, 170.0, -135.0),  # Right eye right corner
        (-150.0, -150.0, -125.0), # Left Mouth corner
        (150.0, -150.0, -125.0)  # Right mouth corner

    ])

    # Camera internals

    focal_length = img_size[1]
    center = (img_size[1]/2, img_size[0]/2)
    camera_matrix = np.array(
        [[focal_length, 0, center[0]],
         [0, focal_length, center[1]],
         [0, 0, 1]], dtype = "double"
    )

    print("Camera Matrix :{}".format(camera_matrix))

    dist_coeffs = np.zeros((4,1)) # Assuming no lens distortion
    (success, rotation_vector, translation_vector) = cv2.solvePn
P(model_points, image_points, camera_matrix, dist_coeffs, flags=
cv2.SOLVEPNP_ITERATIVE )

    print("Rotation Vector:\n {}".format(rotation_vector))
    print("Translation Vector:\n {}".format(translation_vector))
    return success, rotation_vector, translation_vector, camera_
matrix, dist_coeffs

# Convert from rotation vector to Euler angle
def get_euler_angle(rotation_vector):
    # calculate rotation angles
    theta = cv2.norm(rotation_vector, cv2.NORM_L2)

    # transformed to quaterniond
    w = math.cos(theta / 2)

```

```

x = math.sin(theta / 2)*rotation_vector[0][0] / theta
y = math.sin(theta / 2)*rotation_vector[1][0] / theta
z = math.sin(theta / 2)*rotation_vector[2][0] / theta

ysqr = y * y
# pitch (x-axis rotation)
t0 = 2.0 * (w * x + y * z)
t1 = 1.0 - 2.0 * (x * x + ysqr)
print('t0:{}, t1:{}'.format(t0, t1))
pitch = math.atan2(t0, t1)

# yaw (y-axis rotation)
t2 = 2.0 * (w * y - z * x)
if t2 > 1.0:
    t2 = 1.0
if t2 < -1.0:
    t2 = -1.0
yaw = math.asin(t2)

# roll (z-axis rotation)
t3 = 2.0 * (w * z + x * y)
t4 = 1.0 - 2.0 * (ysqr + z * z)
roll = math.atan2(t3, t4)

print('pitch:{}, yaw:{}, roll:{}'.format(pitch, yaw, roll))

# Unit conversion: convert radians to degrees
Y = int((pitch/math.pi)*180)
X = int((yaw/math.pi)*180)
Z = int((roll/math.pi)*180)

return 0, Y, X, Z

def get_pose_estimation_in_euler_angle(landmark_shape, im_szie):
    try:
        ret, image_points = get_image_points_from_landmark_shape
(landmark_shape)
        if ret != 0:
            print('get_image_points failed')
            return -1, None, None, None

        ret, rotation_vector, translation_vector, camera_matrix,
dist_coeffs = get_pose_estimation(im_szie, image_points)
        if ret != True:
            print('get_pose_estimation failed')
            return -1, None, None, None

```

```

ret, pitch, yaw, roll = get_euler_angle(rotation_vector)
if ret != 0:
    print('get_euler_angle failed')
    return -1, None, None, None

euler_angle_str = 'Y:{}, X:{}, Z:{}'.format(pitch, yaw,
roll)

print(euler_angle_str)
return 0, pitch, yaw, roll

except Exception as e:
    print('get_pose_estimation_in_euler_angle exception:{}'.
format(e))
    return -1, None, None, None

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FPS, 10)
fourcc = cv2.VideoWriter_fourcc(*'XVID')
output_video = cv2.VideoWriter('output.mp4', fourcc, 10.0, (640,
480))
while (cap.isOpened()):
    start_time = time.time()

    # Read Image
    ret, im = cap.read()
    if ret != True:
        print('read frame failed')
        continue
    size = im.shape
    if size[0] > 700:
        h = size[0] / 3
        w = size[1] / 3
        im = cv2.resize(im, (int( w ), int( h )), interpolation
=cv2.INTER_CUBIC )
        size = im.shape

    ret, image_points = get_image_points(im)
    if ret != 0:
        print('get_image_point failed')
        continue

    ret, rotation_vector, translation_vector, camera_matrix, dis
t_coeffs = get_pose_estimation(size, image_points)
    if ret != True:

```

```

        print('get_pose_estimation failed')
        continue
used_time = time.time() - start_time
print("used_time:{} sec".format(round(used_time, 3)))

ret, pitch, yaw, roll = get_euler_angle(rotation_vector)
euler_angle_str = 'Y:{}, X:{}, Z:{}'.format(pitch, yaw, roll
)
print(euler_angle_str)

# Project a 3D point (0, 0, 1000.0) onto the image plane.
# We use this to draw a line sticking out of the nose

(nose_end_point2D, jacobian) = cv2.projectPoints(np.array([(
0.0, 0.0, 1000.0)]), rotation_vector, translation_vector, camera
_matrix, dist_coeffs)

for p in image_points:
    cv2.circle(im, (int(p[0]), int(p[1])), 3, (0,0,255), -1)

p1 = ( int(image_points[0][0]), int(image_points[0][1]))
p2 = ( int(nose_end_point2D[0][0][0]), int(nose_end_point2D[
0][0][1]))

cv2.line(im, p1, p2, (255,0,0), 2)

# Display image
#cv2.putText( im, str(rotation_vector), (0, 100), cv2.FONT_H
ERSHEY_PLAIN, 1, (0, 0, 255), 1 )
cv2.putText( im, euler_angle_str, (0, 120), cv2.FONT_HERSHEY
_PLAIN, 1, (0, 0, 255), 1 )
cv2.imshow("Output", im)
output_video.write(im)
if cv2.waitKey(1) & 0xFF == ord('s'):
    break

output_video.release()
cap.release()
cv2.waitKey(0)
cv2.destroyAllWindows()

```