

**MACHINE LEARNING APPROACH TO DETECT &
ANNOTATE EYE DISEASES USING RETINAL IMAGES**

2023-162

Status Document

Kariyawasam K.G.P.C

IT20172978

B.Sc. (Hons) Degree in Information Technology
Specializing in Software Engineering

Department of Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

May 2023

Table of Contents

1. Project progress.....	3
1.1 Datasets.....	3
1.2 Age-Related Macular Degeneration Classification Model.....	4
2. Project View.....	6
3. Updated Gantt chart.....	8
4. Screenshots of chats and calls of MS Teams.....	9

1. Project progress

1.1 Datasets

Retinal OCT Images (optical coherence tomography)

84,495 images, 4 categories

Data Card Code (174) Discussion (8)

About Dataset

Context
[http://www.cell.com/cell/fulltext/S0092-8674\(18\)30154-5](http://www.cell.com/cell/fulltext/S0092-8674(18)30154-5)

Retinal optical coherence tomography (OCT) is an imaging technique used to capture high-resolution cross sections of the retinas of living patients. Approximately 30 million OCT scans are performed each year, and the analysis and interpretation of these images takes up a significant amount of time (Swanson and Fujimoto, 2017).

Usability 7.50

License CC BY-NC-SA 4.0

Expected update frequency Not specified

Tags

Health Biology
Image Eyes and Vision
Medicine

Figure 01: Kaggle Dataset

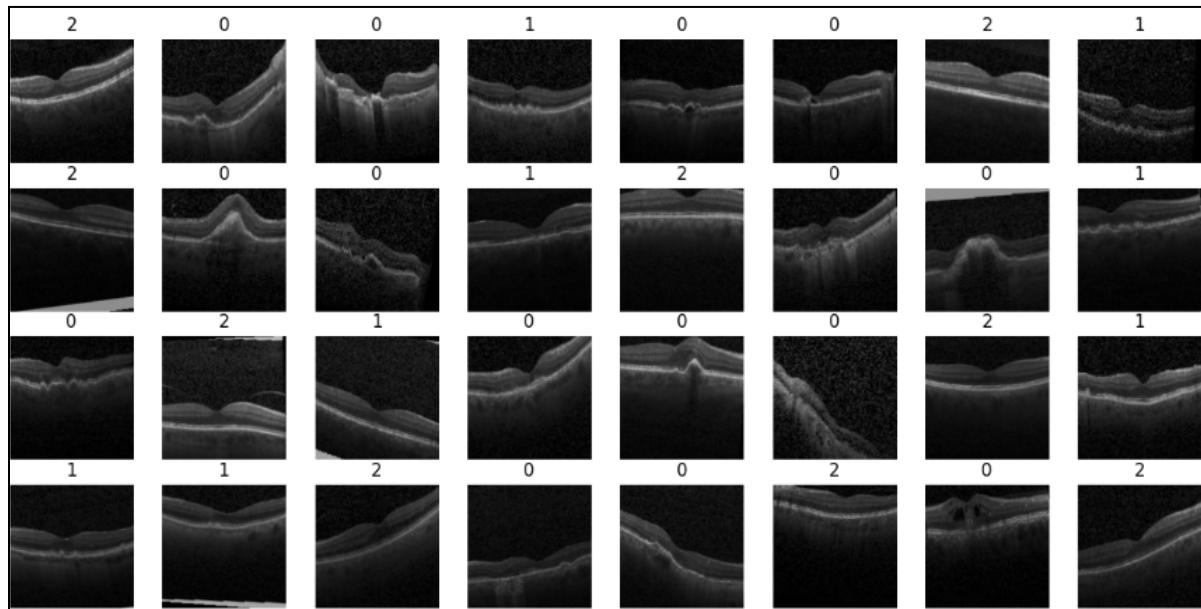


Figure 02: Sample Data

1.2 Age-Related Macular Degeneration Classification Model

This research aimed to develop a machine-learning model for classifying dry and wet age-related macular degeneration (AMD) from OCT images. The dataset consisted of 72,136 scans categorized into three classes: choroidal neovascularization (CNV), Drusen, and Normal retina. The images were processed to a standardized size of 150x150 pixels and normalized to improve consistency and quality. Data augmentation techniques were applied to increase the diversity of the dataset. The model architecture was based on the VGG16 model, with customized layers for AMD classification. The model was trained using the Adam optimizer and evaluated using accuracy, precision, recall, and F1-score metrics. Comparative analysis and interpretability techniques, such as activation maps, were used to assess the model's performance and understand its decision-making process.

```
▼ Import Libraries

import numpy as np
import pandas as pd
from PIL import Image
import cv2
import os
import glob
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from tensorflow.keras.applications.vgg16 import VGG16
from keras.layers import Activation,Dense, Dropout, Flatten, Conv2D, ReLU
from keras.models import Model
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.optimizers import SGD, Adam
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, precision_recall_curve, f1_score
import itertools

▼ Preprocessing

[ ] #Calculate class weight for the imbalanced data
total = 72136

count_cnv = 37205
count_drusen = 8616
count_normal = 26315

cnv_weight = (1/count_cnv) * (total/3)
drusen_weight = (1/count_drusen) * (total/3)
norm_weight = (1/count_normal) * (total/3)

class_weight = {0 : cnv_weight, 1 : drusen_weight, 2: norm_weight}
print(class_weight)

labels = ['CNV', 'DRUSEN', 'NORMAL']

def load(path):
    listed_paths = []
    for i in labels:
        base_path = os.path.join(path, i)
        imgs = os.listdir(base_path)
        for img in imgs:
            img_path = base_path + '/' + img
            listed_paths.append(img_path)
    return listed_paths

train_paths = load(train_path)
val_paths = load(val_path)
test_paths = load(test_path)

{0: 0.6462930609685078, 1: 2.790776849272671, 2: 0.9137500791690417}
```

Figure 03: AMD Classification Model Implementation

```
▼ Data augmentation

# Train Data

train_datagen = ImageDataGenerator(
    rescale= 1./255, # rescale pixel values to [0,1]
    zoom_range= (0.73, 0.9), # randomly zoom images
    horizontal_flip= True, # randomly flip images horizontally
    rotation_range= 10, # randomly rotate images by up to 20 degrees
    width_shift_range= 0.1, # randomly shift images horizontally by up to 10%
    fill_mode= 'constant', # fill any empty pixels with constant
    height_shift_range= 0.1, # randomly shift images vertically by up to 10%
    brightness_range= (0.55, 0.9), # modify the brightness
    shear_range=0.2, # randomly shear images by up to 20%
)

train_generator = train_datagen.flow_from_directory(
    train_path, #data to read
    target_size= (150, 150), #resized
    color_mode= 'rgb', #full colored images
    batch_size= 64,
    class_mode= 'categorical', #for multiclass classification
    shuffle= True, #for the over fitting problem
    seed= 1337
)
Found 72136 images belonging to 3 classes.

#Validation Data

val_datagen = ImageDataGenerator(
    rescale= 1./255, # rescale pixel values to [0,1]
    zoom_range= (0.73, 0.9), # randomly zoom images
    horizontal_flip= True, # randomly flip images horizontally
    rotation_range= 10, # randomly rotate images by up to 20 degrees
    width_shift_range= 0.1, # randomly shift images horizontally by up to 10%
    fill_mode= 'constant', # fill any empty pixels with constant
    shear_range=0.2, # randomly shear images by up to 20%
```

Figure 04: AMD Classification Model Implementation

▼ Build the Model VGG16

```
[ ] # pre-trained VGG16 model

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
trainable_layers = len(base_model.layers[1:-5])
for layer in base_model.layers[trainable_layers:]:
    layer.trainable = False # Freeze the weights of all layers except for the last 5
for layer in base_model.layers[trainable_layers:]:
    layer.trainable = True

[ ] # Add new classification layers on top of the pre-trained layers
num_classes = 3 # cnv, drusen, and normal.
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x) # Add a dropout layer with a rate of 0.5
predictions = Dense(num_classes, activation='softmax')(x)

❶ # Create the final model
model_3 = Model(inputs=base_model.input, outputs=predictions)
# Compile the model with Adam optimizer
opt = Adam(lr=0.0001)
model_3.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
# Define the callbacks
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.00001)
early_stop = EarlyStopping(monitor='val_loss', patience=5)
# Train the model
num_epochs = 20
history_3 = model_3.fit(train_generator,
                        epochs = num_epochs,
                        steps_per_epoch = 500,
                        validation_data = val_generator,
                        validation_steps = len(val_generator),
                        callbacks = [reduce_lr, early_stop])

Epoch 1/20
500/500 [=====] - 348s 695ms/step - loss: 0.2179 - accuracy: 0.9230 - val_loss: 0.0513 - val_accuracy: 1.0000 - lr: 1.0000e-04
Epoch 2/20
500/500 [=====] - 341s 682ms/step - loss: 0.1854 - accuracy: 0.9347 - val_loss: 0.0311 - val_accuracy: 1.0000 - lr: 1.0000e-04
Epoch 3/20
500/500 [=====] - 338s 676ms/step - loss: 0.1743 - accuracy: 0.9397 - val_loss: 0.0548 - val_accuracy: 0.9583 - lr: 1.0000e-04
Epoch 4/20
500/500 [=====] - 337s 675ms/step - loss: 0.1637 - accuracy: 0.9427 - val_loss: 0.0370 - val_accuracy: 1.0000 - lr: 1.0000e-04
Epoch 5/20
500/500 [=====] - 328s 656ms/step - loss: 0.1526 - accuracy: 0.9471 - val_loss: 0.0443 - val_accuracy: 1.0000 - lr: 1.0000e-04
Epoch 6/20
500/500 [=====] - 326s 652ms/step - loss: 0.1303 - accuracy: 0.9521 - val_loss: 0.0215 - val_accuracy: 1.0000 - lr: 2.0000e-05
Epoch 7/20
500/500 [=====] - 326s 652ms/step - loss: 0.1302 - accuracy: 0.9541 - val_loss: 0.0243 - val_accuracy: 1.0000 - lr: 2.0000e-05
Epoch 8/20
500/500 [=====] - 325s 649ms/step - loss: 0.1233 - accuracy: 0.9560 - val_loss: 0.0487 - val_accuracy: 1.0000 - lr: 2.0000e-05
Epoch 9/20
500/500 [=====] - 324s 648ms/step - loss: 0.1222 - accuracy: 0.9577 - val_loss: 0.0204 - val_accuracy: 1.0000 - lr: 2.0000e-05
Epoch 10/20
500/500 [=====] - 323s 646ms/step - loss: 0.1196 - accuracy: 0.9581 - val_loss: 0.0197 - val_accuracy: 1.0000 - lr: 2.0000e-05
Epoch 11/20
500/500 [=====] - 325s 650ms/step - loss: 0.1206 - accuracy: 0.9584 - val_loss: 0.0202 - val_accuracy: 1.0000 - lr: 2.0000e-05
Epoch 12/20
500/500 [=====] - 323s 645ms/step - loss: 0.1158 - accuracy: 0.9584 - val_loss: 0.0372 - val_accuracy: 1.0000 - lr: 2.0000e-05
Epoch 13/20
500/500 [=====] - 364s 727ms/step - loss: 0.1117 - accuracy: 0.9610 - val_loss: 0.0197 - val_accuracy: 1.0000 - lr: 2.0000e-05
Epoch 14/20
500/500 [=====] - 330s 659ms/step - loss: 0.1091 - accuracy: 0.9619 - val_loss: 0.0163 - val_accuracy: 1.0000 - lr: 1.0000e-05
```

Figure 05: AMD Classification Model Implementation

2. Project View

The screenshot shows the Planner - Board View for a project titled "23-162_DETECT_AND_ANNOТА...". The interface is divided into four columns, each representing a team member: Nilaksha, Praveen, Rasanga, and Chamod. Each column contains a list of tasks with their due dates and status indicators (green checkmark for completed, red exclamation mark for in progress, blue question mark for pending). A summary at the bottom of each column indicates the number of completed tasks.

Team Member	Task Description	Due Date	Status
Nilaksha	Testing the Application	07/23	Completed
	Integrate the Trained Model	07/22	In Progress
	Implement Functionality of Application using React Native	07/14	Pending
	Implement UI using React Native	07/10	In Progress
	Design Mobile Application Interfaces	07/09	Pending
	Train the Model	07/17	In Progress
	Design and Implement the Model	07/16	Pending
	Testing the Application	07/23	Completed
Praveen	Integrate the Trained Model	07/22	In Progress
	Implement Functionality of Application using React Native	07/14	Pending
	Implement UI using React Native	07/10	In Progress
	Design Mobile Application Interfaces	07/09	Pending
	Train the Model	07/17	In Progress
	Design and Implement the Model	07/16	Pending
	Testing the Application	07/23	Completed
	Integrate the Trained Model	07/22	In Progress
Rasanga	Implement Functionality of Application using React Native	07/14	Pending
	Implement UI using React Native	07/10	In Progress
	Design Mobile Application Interfaces	07/10	Pending
	Train the Model	07/17	In Progress
	Design and Implement the Model	07/16	Pending
	Testing the Application	07/23	Completed
	Integrate the Trained Model	07/22	In Progress
	Implement Functionality of Application using React Native	07/14	Pending
Chamod	Implement Functionality of Application using React Native	07/14	Pending
	Implement UI using React Native	07/10	In Progress
	Design Mobile Application Interfaces	07/12	Pending
	Train the Model	07/17	In Progress
	Design and Implement the Model	07/16	Pending
	Testing the Application	07/23	Completed
	Integrate the Trained Model	07/22	In Progress
	Implement Functionality of Application using React Native	07/14	Pending

Figure 06: Planner – Board View

The screenshot shows the Planner - Chart View for the same project. It includes several charts and a detailed task list:

- Status:** A donut chart showing 22 tasks left, with segments for Not started (12), In progress (10), Late (0), and Completed (22).
- Bucket:** A bar chart showing the distribution of tasks across four buckets: Nilaksha, Praveen, Rasanga, and Chamod. The y-axis represents the count of tasks (0 to 12), and the x-axis represents the buckets.
- Priority:** A bar chart showing the distribution of tasks by priority level: Urgent, Important, and Medium (Low).
- Members:** A chart showing the assignment of tasks to team members: Nilaksha, Praveen, Rasanga, and Chamod. The x-axis represents the count of tasks (0 to 12), and the y-axis represents the team members.
- Tasks:** A detailed list of tasks for Nilaksha, including their descriptions, due dates, and current status.

Figure 07: Planner – Chart View

The screenshot shows the Microsoft Planner interface in 'Schedule' view for the month of July 2023. The calendar grid displays tasks assigned to team members. The tasks are color-coded by assignee:

- Nilaksha**: Design Mobile App., Implement UI using ...
- Praveen**: Implement Function...
- Rasanga**: Design and imple..., Train the Model
- Chamod**: Testing the Application
- Completed**: Integrate the Trained...

The sidebar on the left shows the team structure with 'General' selected. The top navigation bar includes 'Search', 'Meet', 'General', 'Posts', 'Files', 'Planner', 'Activity', 'Chat', 'Teams', 'Assignments', 'Calendar', 'Calls', and 'Files'. The right sidebar features 'Unscheduled tasks' and a 'Add task' button.

Figure 08: Planner – Schedule View

3. Updated Gantt chart

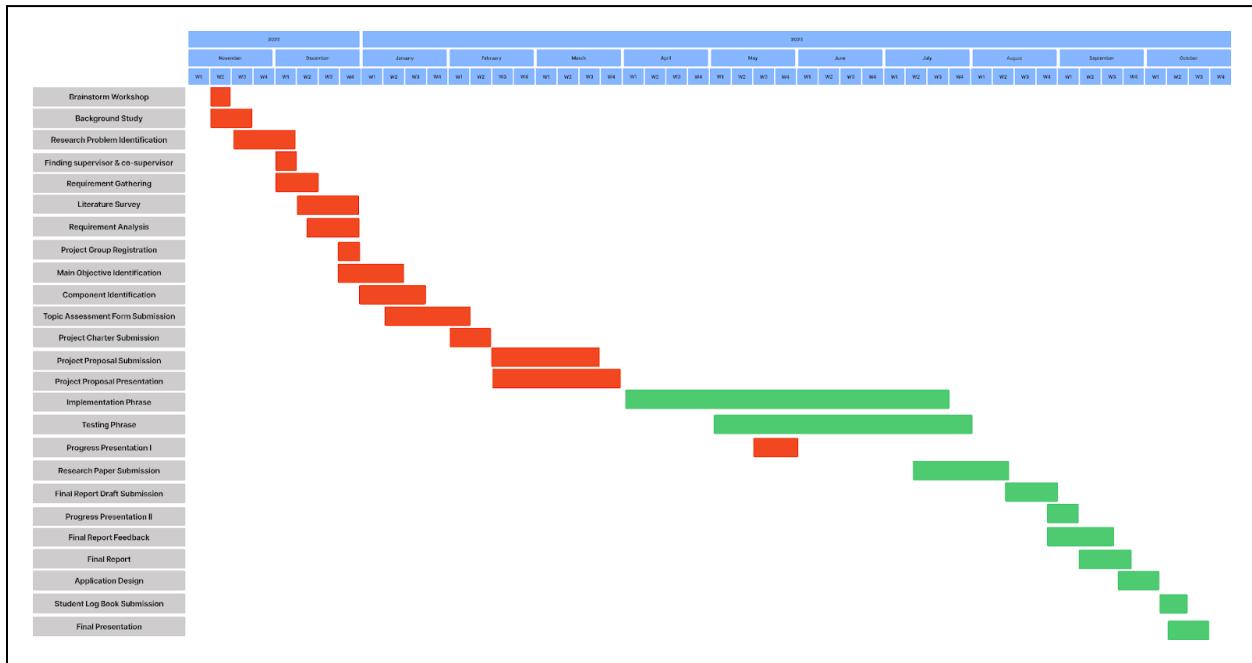


Figure 09: Gantt Chart

4. Screenshots of chats and calls of MS Teams

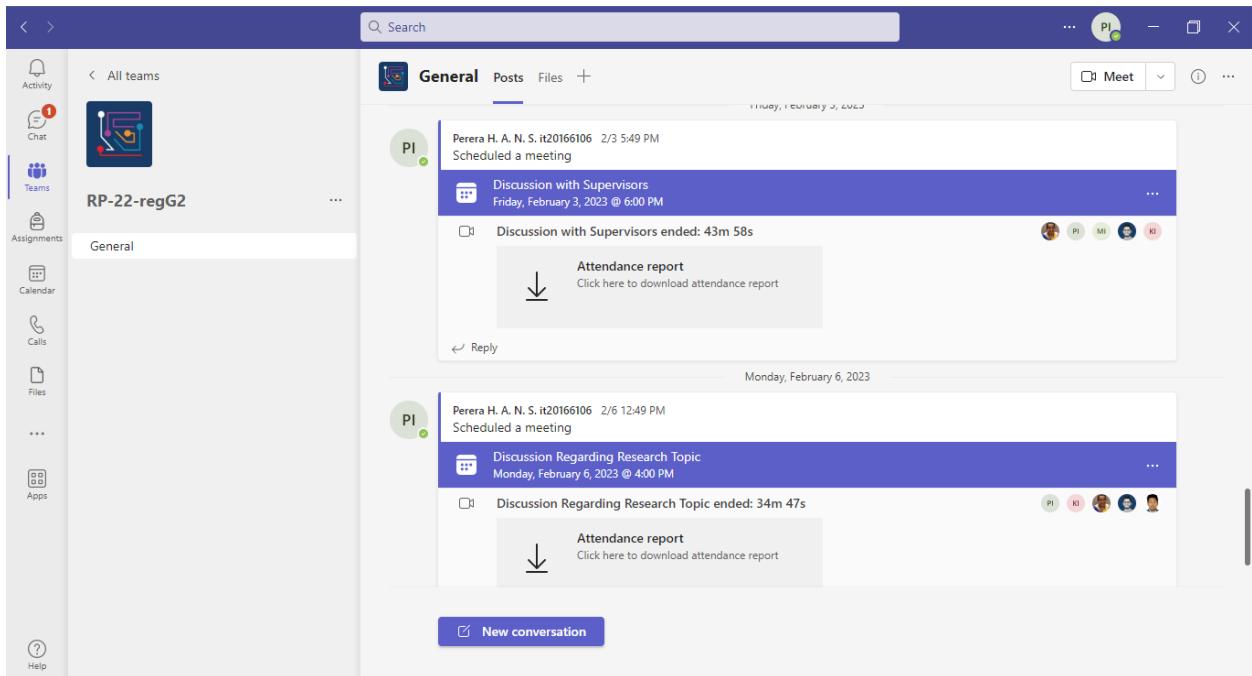


Figure 10: Screenshots of MS Teams Chats

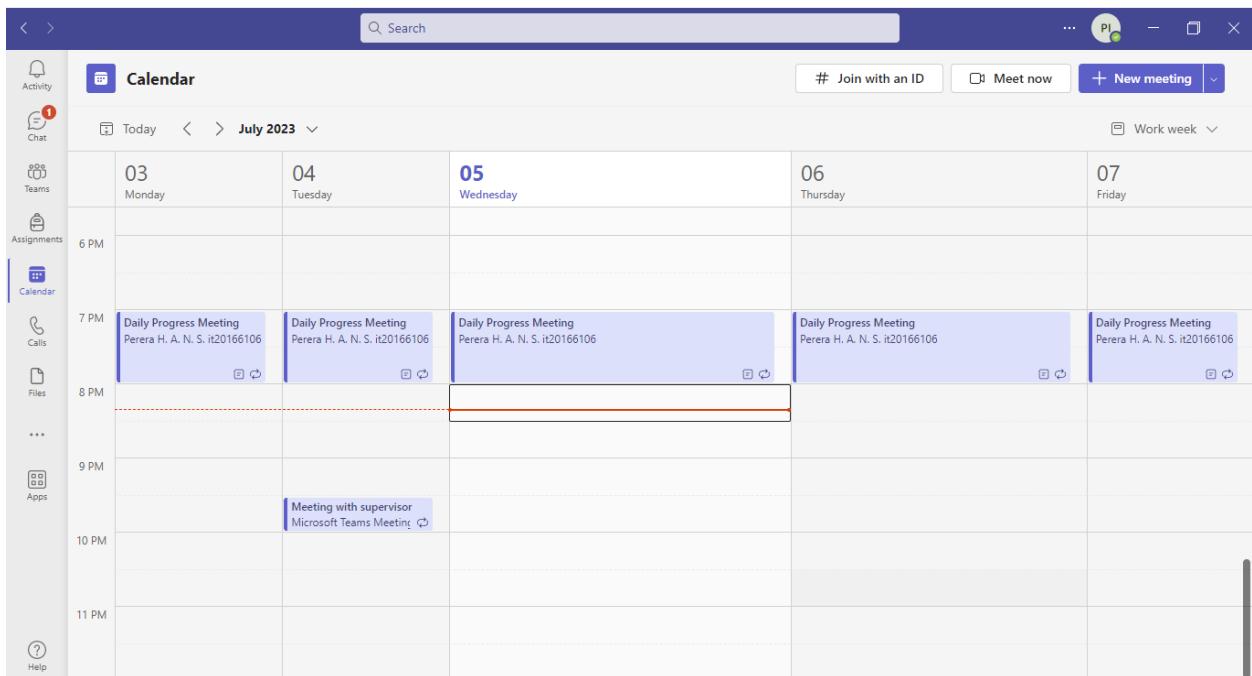


Figure 11: Screenshots of MS Teams Calendar Schedule

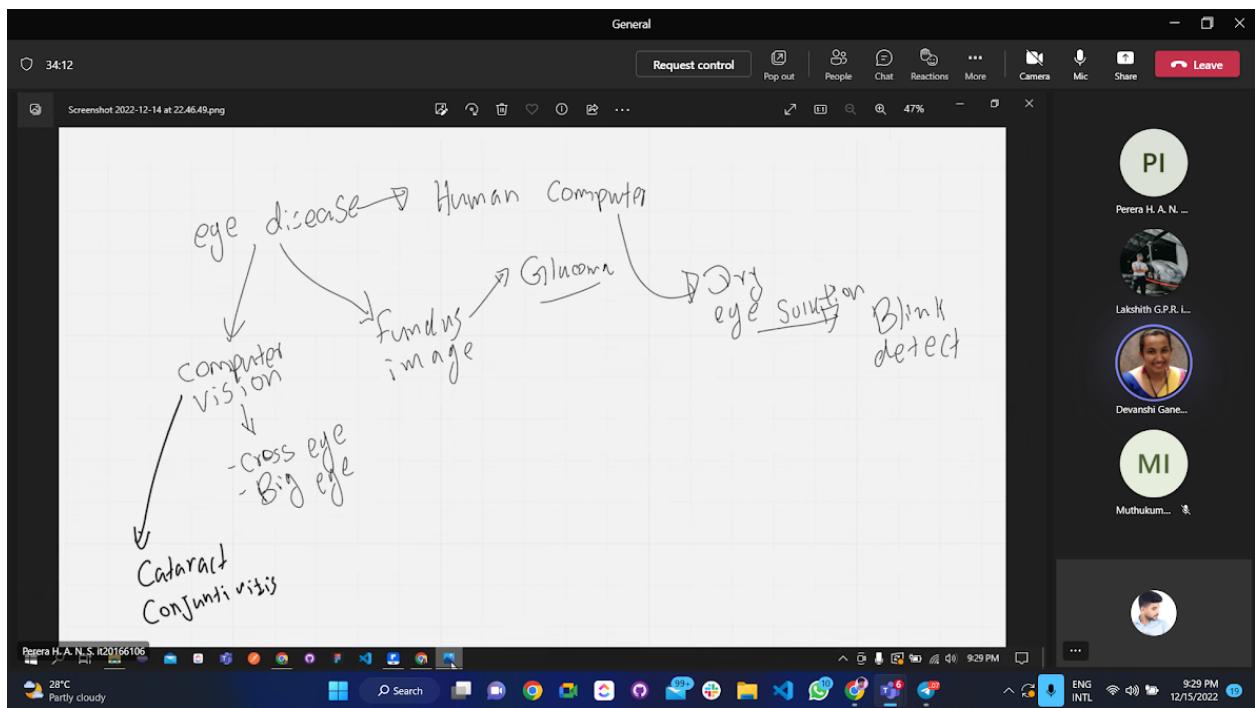


Figure 12: Screenshots of MS Teams Meetings