# CS559 - Biometrics
# Final Project Report

*A Novel Biometric Application : Facial Expression Recognition using Deep Learning Model*

Submitted By-

Nandana Veerabhadraswamy (A20448877)
Sahana Shreedhar Kulkarni (A20442482)
Manjari Kulkarni (A20453176)

# INDEX

# ABSTRACT

Facial emotions are important factors in human communication that help us understand the intentions of others. This project aims to classify the emotion on a person's face using a face detection model. The model classifies into one of seven categories which are angry, disgusted, fearful, happy, neutral, sad and surprised. This is achieved with the help of using deep convolutional neural networks. The model is created, built and deployed on a cloud platform called AWS Sagemaker . It accesses the dataset uploaded in AWS S3 which stores the train and test data . The model is then deployed on the AWS Deeplens device.

# INTRODUCTION

The automatic recognition of emotions has been an active research topic from early eras. This is because usually people infer the emotional states of other people, such as joy, sadness, and anger, using facial expressions and vocal tone.

Emotions are reflected from speech, hand and gestures of the body and through facial expressions. Hence extracting and understanding emotion has a high importance of the interaction between human and machine communication. Therefore, it is natural that research of facial emotion has been gaining a lot of attention over the past decades with applications not only in the perceptual and cognitive sciences, but also in affective computing and computer animations.There are several advances made in this field.

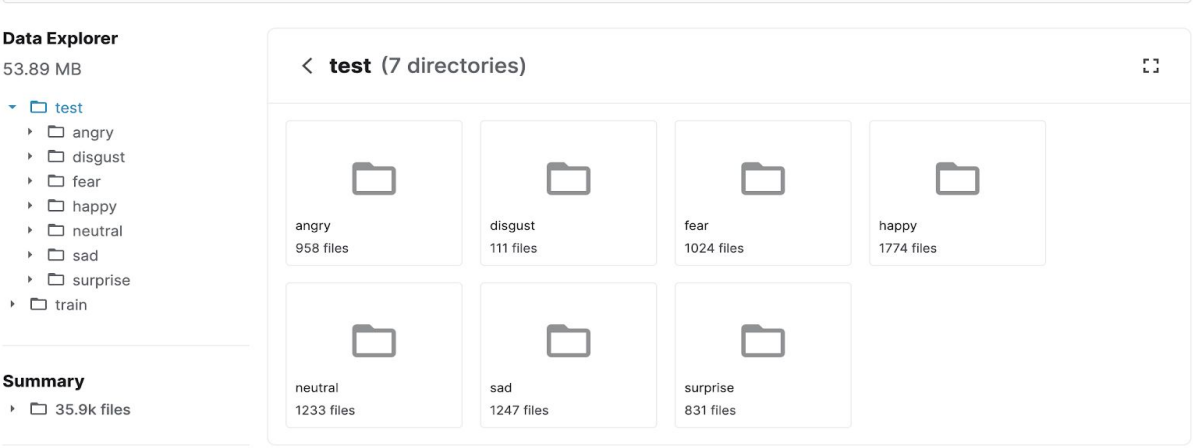
Face emotion detection has ample commercial applications as well . It is widely used by different companies to gauge consumer mood towards their product or brand in the digital world. However, in offline world users are also interacting with the brands and products in retail stores and showrooms and solutions to monitor user's reaction under such settings has remained a difficult task to track automatically. Besides, a skilled observation of the face is also relevant in the diagnosis and assessment of mental or physical diseases. The face appearance of a patient may indeed provide diagnostic clues to the illness, the severity of the disease and some vital patient’s values

In this project , we design and develop Deep Learning models for recognizing face emotion that can be applied in the domains listed above and may prove as an efficient solution in terms of cost and time.

# DESCRIPTION ABOUT DATASETS

The model is being trained on the FER-2013 dataset obtained from Kaggle.
( The link https://www.kaggle.com/msambare/fer2013)

The data consists of 48x48 pixel grayscale images of faces. The face images are of seven different categories based on the emotions like angry, disgust,fear,happy,neutral,sad and surprise as shown below.



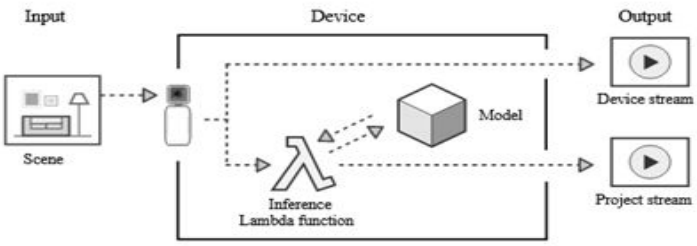

The entire dataset comprises 28,709 images, out of which we had taken 7178 images in the training dataset. These datasets are uploaded on AWS S3 bucket.

# PROPOSED SOLUTION

Our proposed solution is a face detection model that uses deep convolutional neural networks. The various critical technologies involved in this were as follows:

**AWS SageMaker** : It is a cloud machine-learning platform that enabled us to create , build and deploy our model on it.

**AWS DeepLens** : It is a deep learning enabled video camera . For our project we have used the DeepLens device with hardware model version 1.1.




**AWS S3**: We created an S3 bucket called bioproj-dataset and uploaded the datasets on it. (ARN's : s3://bioproj-dataset/train data/for train dataset and s3://bioproj-dataset/test data/ for test dataset)

**OpenCV** : We have used this python library for face detection purposes. It initially detected faces from the locally stored images using haar-cascade classifiers.

**TensorFlow** : We have used this open-source machine learning library for basic training and inference of deep neural networks.

# EXPERIMENTAL PROCEDURE

## ➢ Running the code locally

- The model was built and trained locally first and then executed on AWS environment. We have worked in a virtual environment to do so, because applications need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library's interface. [3] In order to create such an environment we have followed the following steps:

```
$ sudo easy_install pip
$ cd /usr/local/bin                    - Move to /bin
$ pwd                                  - Check that the path must now be /usr/local/bin
$ pip install virtualenvwrapper
$ WORKON_HOME=~/Envs
$ mkdir -p $WORKON_HOME
$ source /usr/bin/virtualenvwrapper.sh
$ mkvirtualenv base                    - Finally a virtual environment (env1) will be created
```

You will then see a (env1) prefix a part of the path in the terminal. In order to work on the environment that you have created, use the following command.

```
$ workon base
```

- The dependencies that have been used to develop the project are OpenCV, Tensorflow, NumPy, Matplotlib and other Python3 libraries. In order to install these libraries, we have created a dependencies.txt file where the dependencies to be installed are stored with its versions. Then these files can be run in the terminal using the command ‘ pip3 install -r dependencies.txt ’.

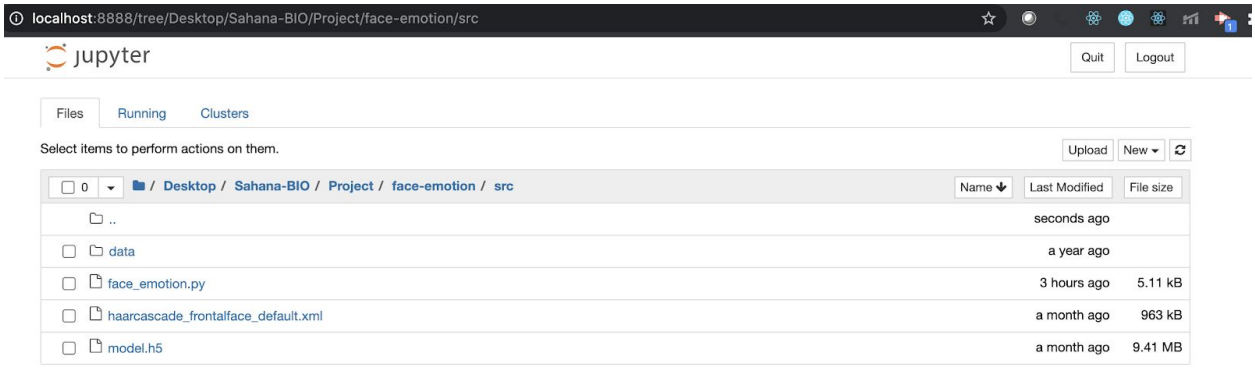
```
(base) sahanashreedharkulkarni@Sahanas-MBP Emotion-detection % pip3 install -r dependencies.txt
```

- Next we have worked on preparing the model. The code to train the model is given in ‘face_emotion.py’. There are 2 modes specified in this. There we can train the model by running ‘python3 face_emotion.py --arg train_model’. This will take nearly one and half hours with below mentioned parameter configuration on the local system.

- The model is trained by giving the parameters of batch_size=64, Epoch=50 for the model.fit_generator() function of Tensorflow. The batch_size and Epoch parameters can be changed to get a better accuracy which leads to obtaining a model with good fit. In the below screenshot we can see that the Epoch iterations value is 50. We obtained the best good fit model at the configuration defined above with an accuracy of 80%.

```
(base) sahanashreedharkulkarni@Sahanas-MBP src % python3 face_emotion.py --arg train_stream
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
WARNING:tensorflow:From face_emotion.py:92: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/50
448/448 [==============================] - 115s 256ms/step - loss: 1.7956 - accuracy: 0.2645 - val_loss: 1.6998 - val_accuracy: 0.3454
Epoch 2/50
 81/448 [====>.........................] - ETA: 1:32 - loss: 1.6834 - accuracy: 0.3247
```

model.h5 is the trained model that is created.



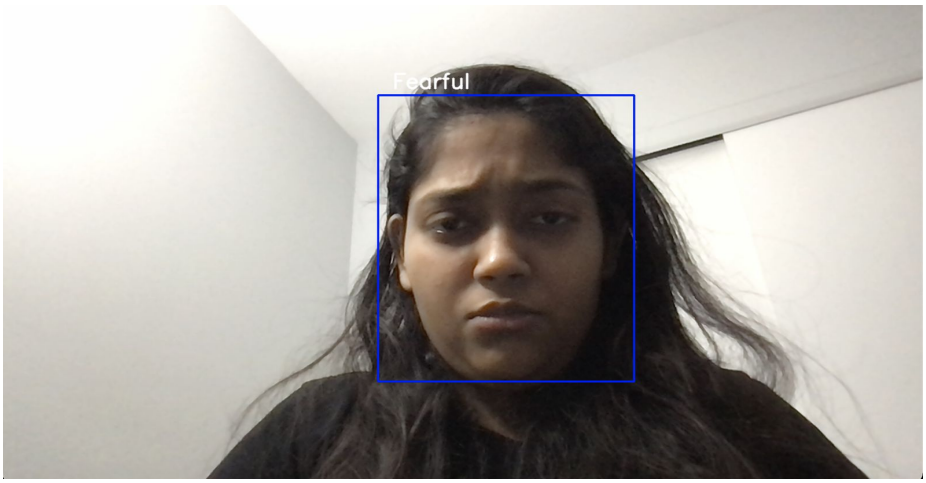

- The Tensorflow library function model.save_weights() to save the trained model and load the model using the library function model.load_weights().

- In order to read the face expressions we use the OpenCV library to process the video feed using the function cv2.VideoCapture(input).

- To identify the face and draw a boundary for the recognized face, OpenCV provides a haar cascade classifier function cv2.CascadeClassifier() for this.
https://github.com/opencv/opencv/tree/master/data/haarcascades.

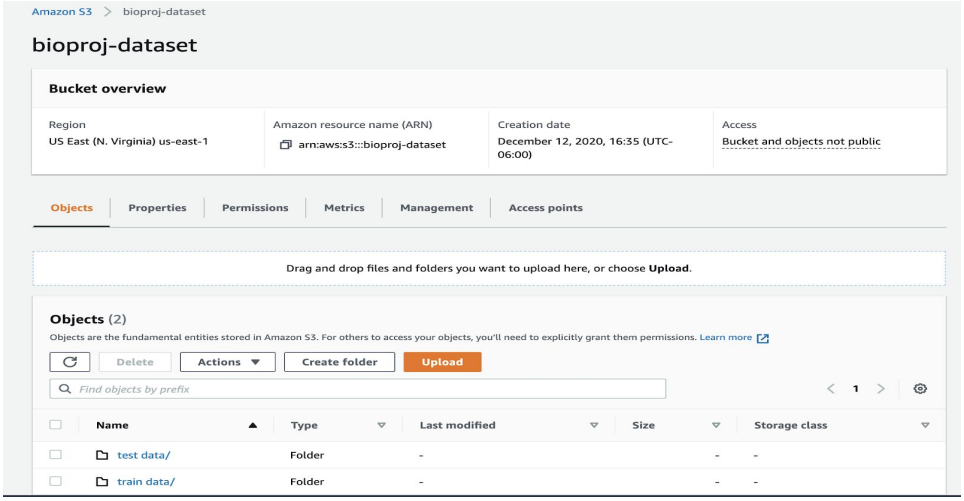
- Next in order to execute the application in real time execute the ‘face_emotion.py’ in stream mode. The command would be ‘python3 face_emotion.py --arg stream’.

- Finally you would see that the access to open the camera will be asked and once the permission is provided. The camera will switch on and you will be able to see the face emotion in real time.

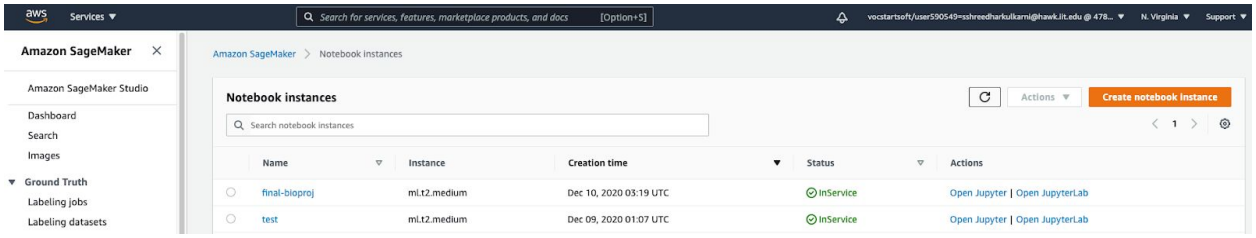



## ➢ Deployment to SageMaker

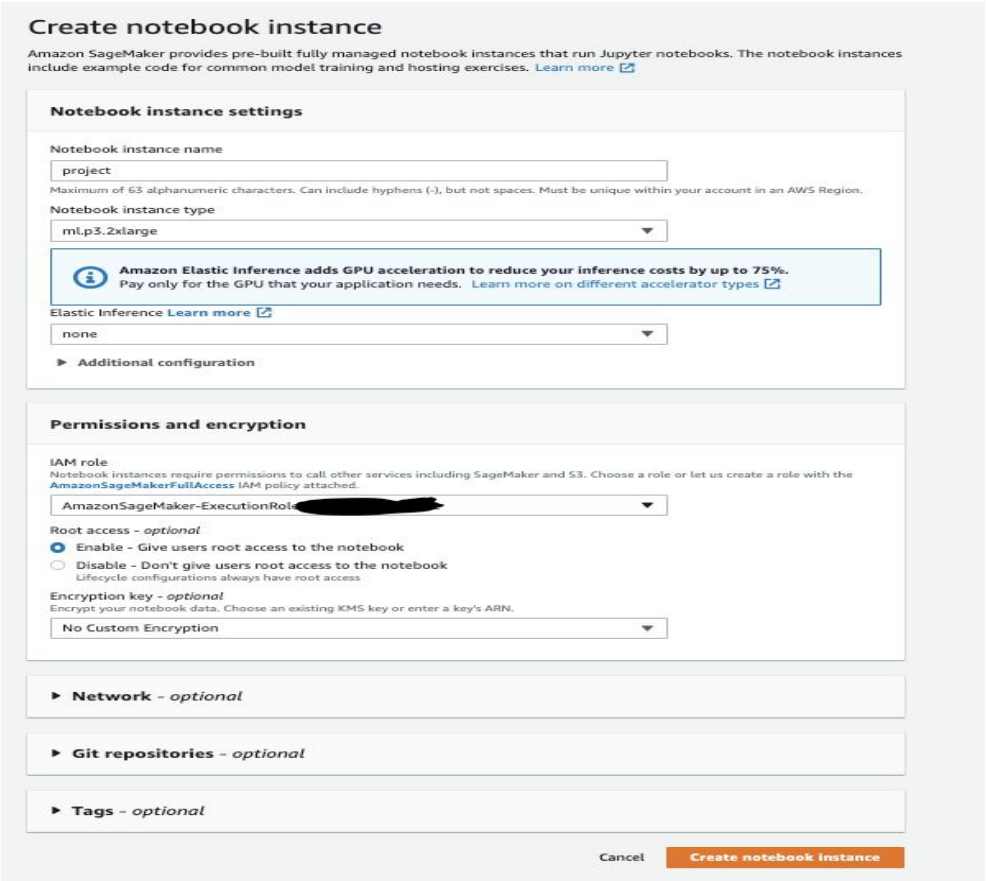
- Firstly we added the Data set to an S3 bucket. Using the **boto3** library, we read the train data from the S3 buckets to sagemaker and build the training model.



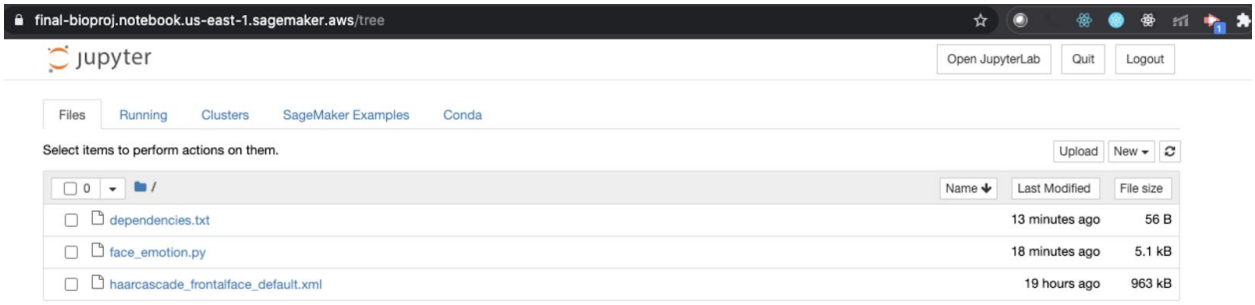
- Secondly, we will be creating an instance in Sagemaker. In order to do this we need to go to Sagemaker in the list of services in AWS. Navigate to the Notebook instance. Then click on ‘create notebook instance’.




- In order to create a notebook instance we will have to give a  name, instance type and IAM role, as shown below.

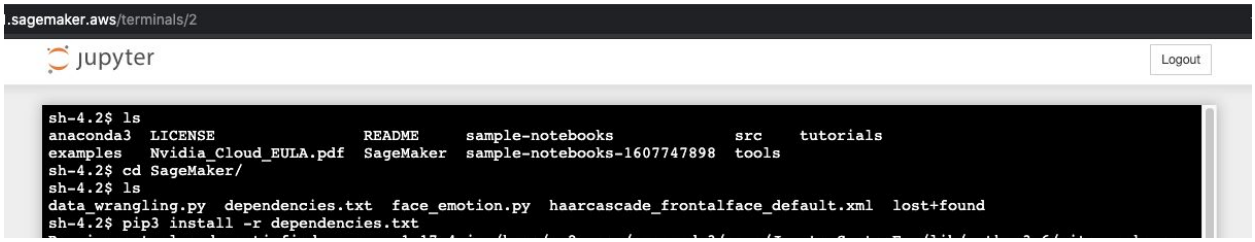


- So once the instance gets the status of 'InService', we can open the jupyter notebook by clicking on 'Open Jupyter'. Here we have to upload the files that we worked on locally before. This would look like:



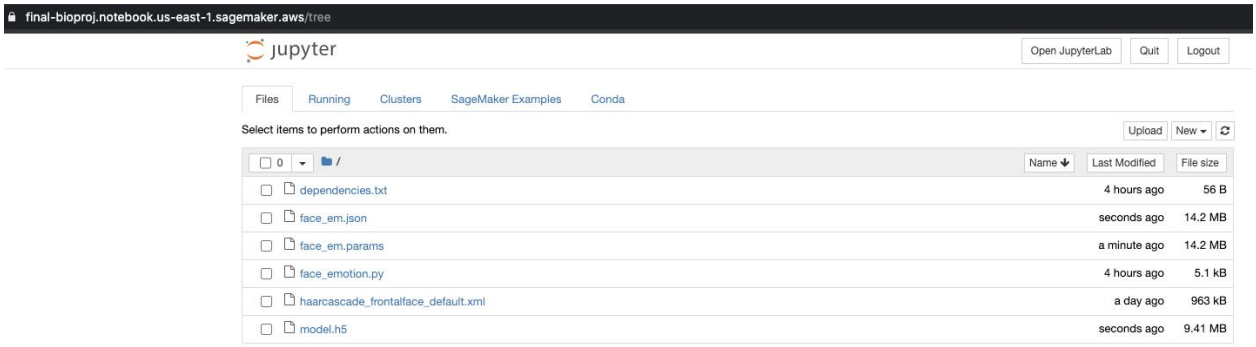

- Go to 'New' drop down shown in the left hand corner of the above screenshot. Here choose 'Terminal'. You will now get a terminal. You will have to Navigate to the folder. In order to do so execute the following commands in your new AWS jupyter terminal:

  $ ls - *You will find the list of folders*
  $ cd SageMaker/ - *Navigate to the SageMaker*
  $ ls - *When you list here, you will find the uploaded files*

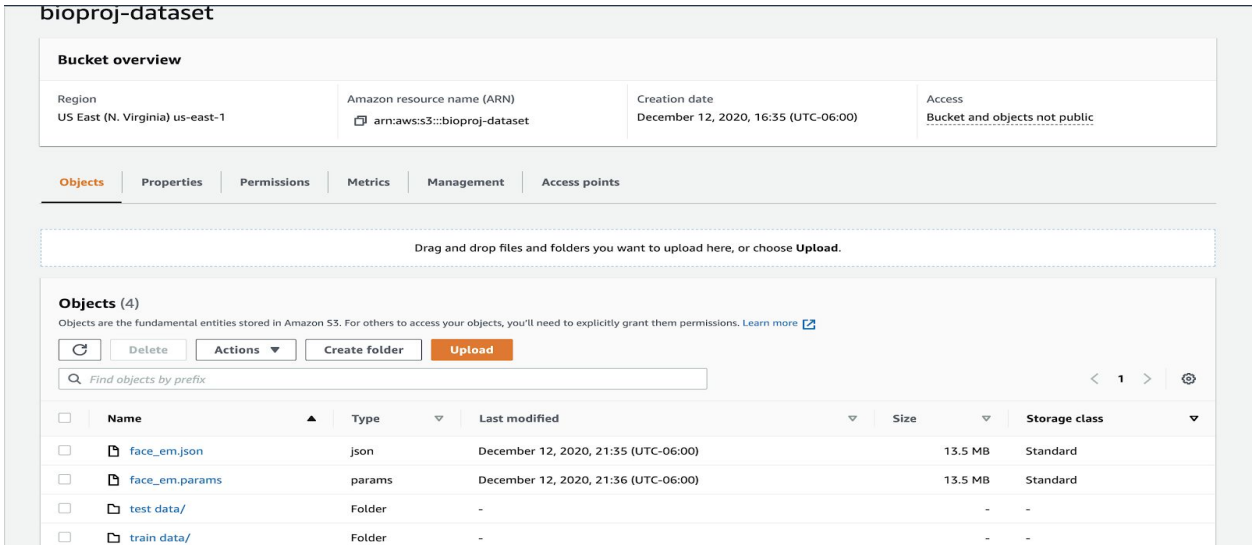



- You will now have to follow similar steps as that we did in the local. Install dependencies using 'pip3 install -r dependencies.txt'.

- We would then train the model 'python3 face_emotion.py --arg train_model'. The 'face_emotion.py' code is updated to create 'face_em.json' and face_em.params' which will be used as the input model for deeplens and will be stored on S3.

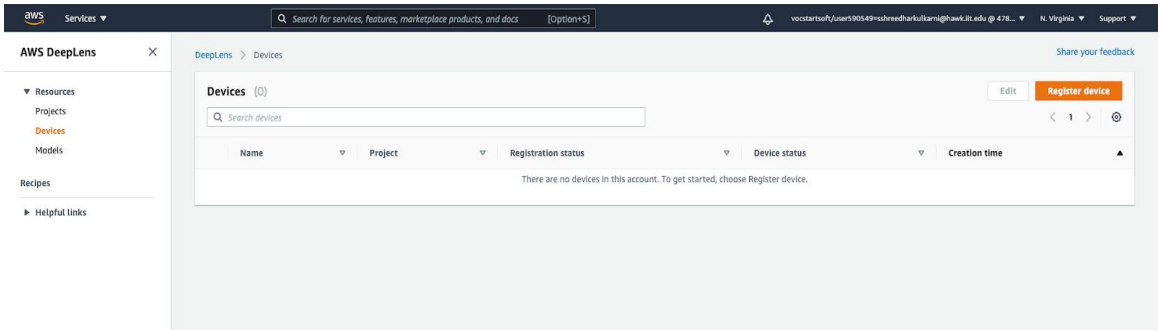



- The trained model is saved into S3 bucket and the deeplens device has to be set up as shown below.



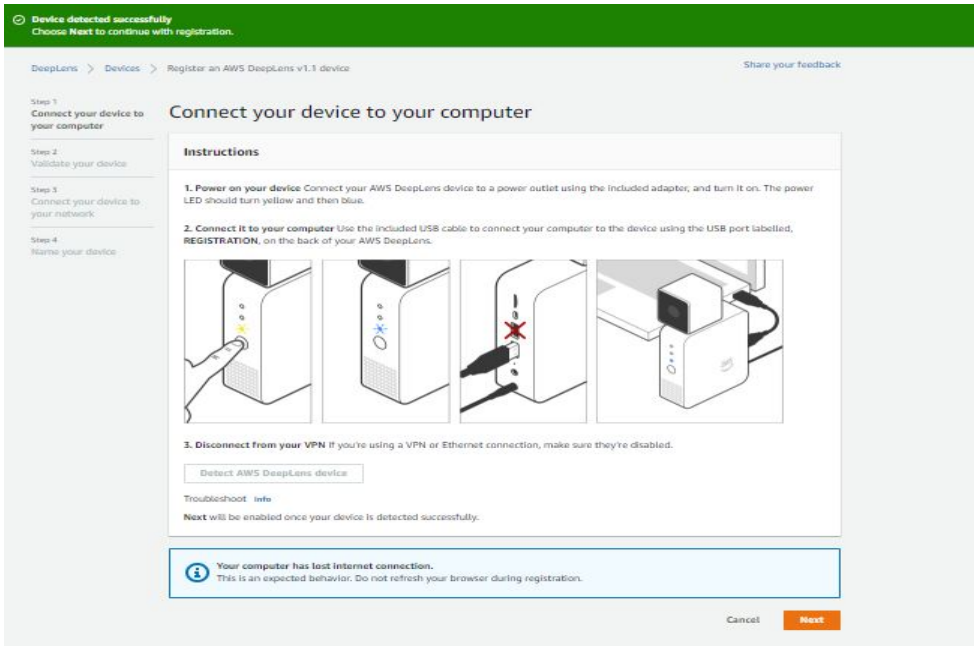

## ➢ Deployment in Deeplens

- Look for AWS DeepLens service in AWS. Here we would find resources like Project, Device and Model which we will be setting further.

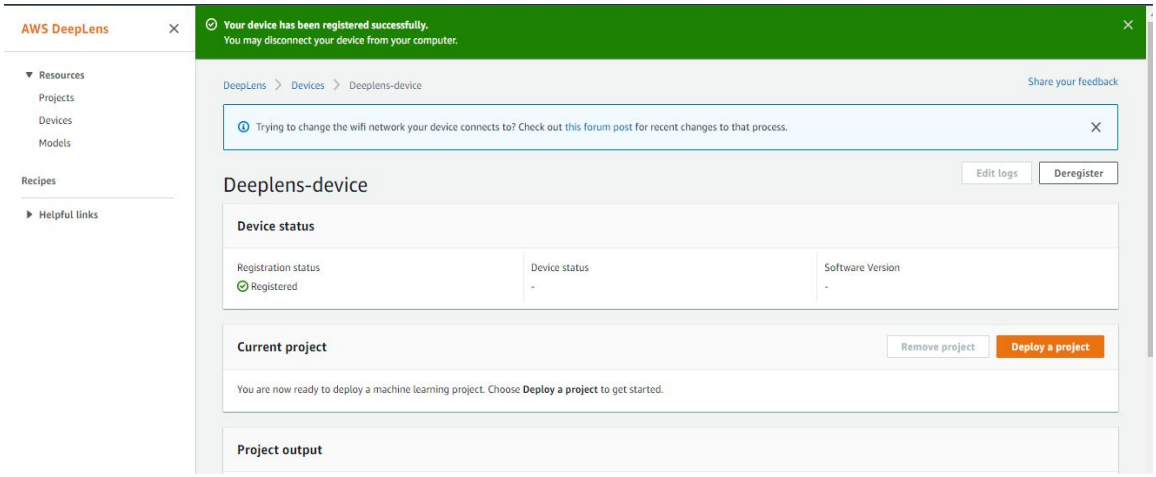



- Now in order to configure the deeplens, establish the connection to the power supply and also connect the USB to the computer. Next switch the deeplens ON. This would give yellow light initially and then turn to blue. We faced issues with the wifi not switching on in Deeplens, which we then rectified using the reset option in the deep lens. A picture of our deeplens is shown below:

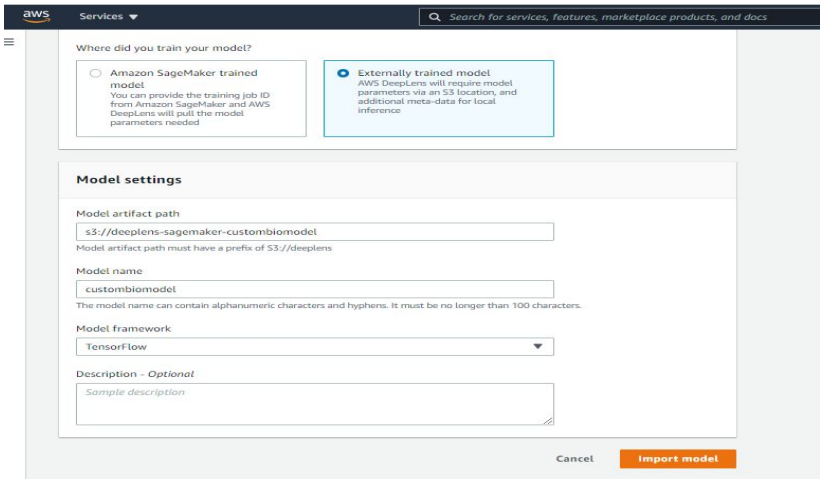

- Now go to the Device option in the Resources. Then ‘Register Device’. Choose the HWv1.1 option as that is the device version we have. Then click ‘Start’. Click on Detect device. Further, it will ask for a validation of the device, where you will have to provide the last four digits of device serial number. The successful detection of the device is shown below.



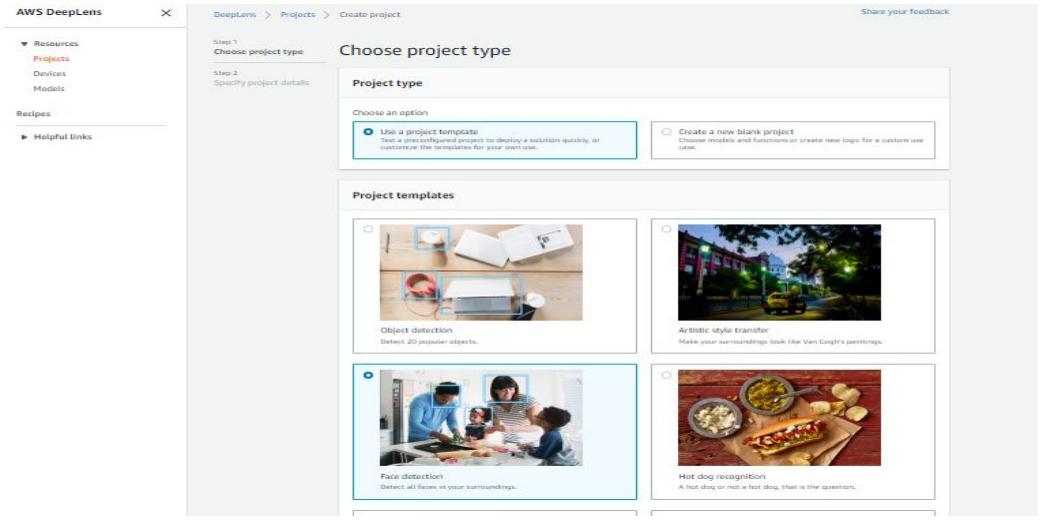

- The successful device registration is shown below:



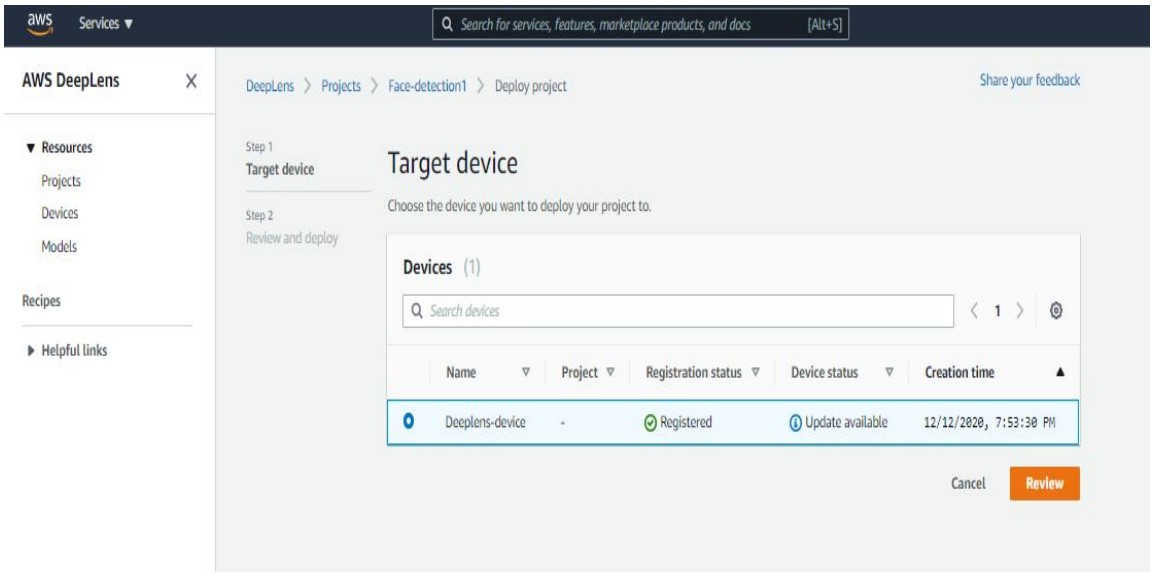
- Next we will be creating a model. Here we will be adding the S3 bucket link as shown below.



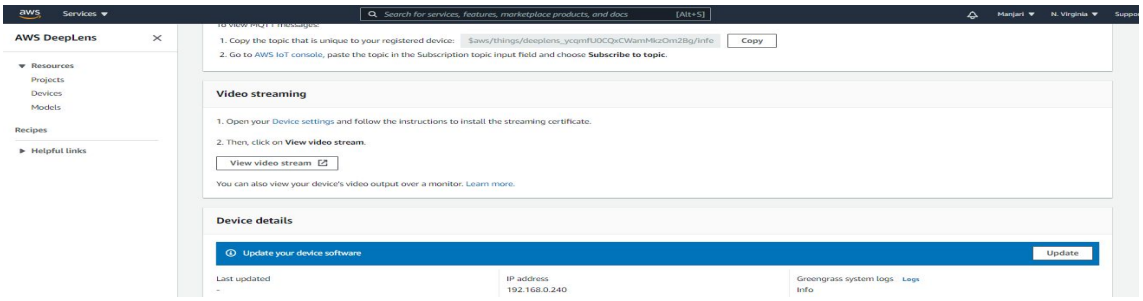

- Inorder to test the device, we first created a sample face detection template. This was done as shown below:



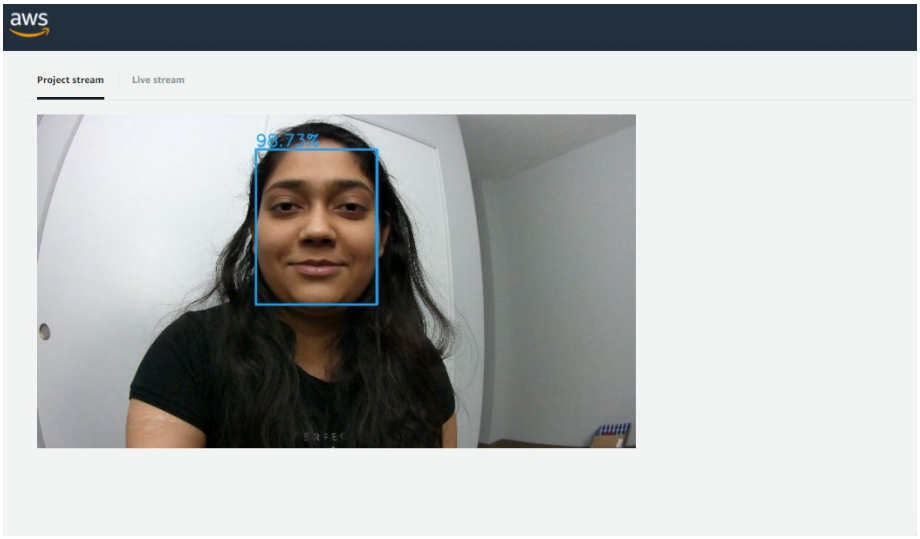
- Deploy the project to the deeplens. This can be done by choosing the ‘Deploy Project’ option given as a part of the project.



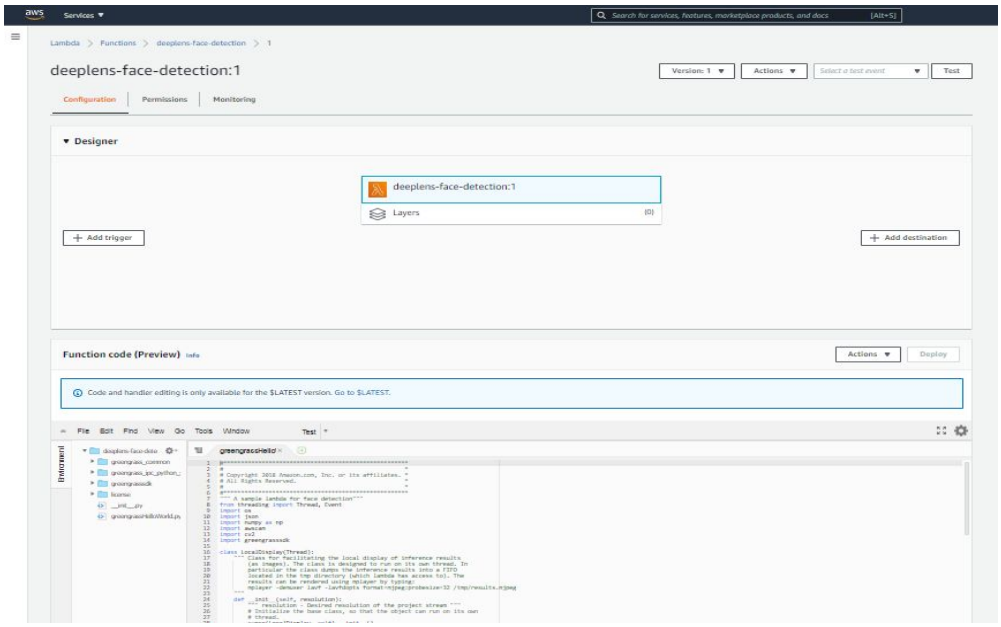

- Once the project is deployed to the deeplens device, we can check for the video stream. Now click ‘View Video Stream’.



- Now here we see the Project stream. The deep lens camera is able to detect the faces.




- Now in order to customize the functioning of the video stream for our project, we need to customise and build out lambda functions. A snip of the lambda function structure with customisation is shown below.

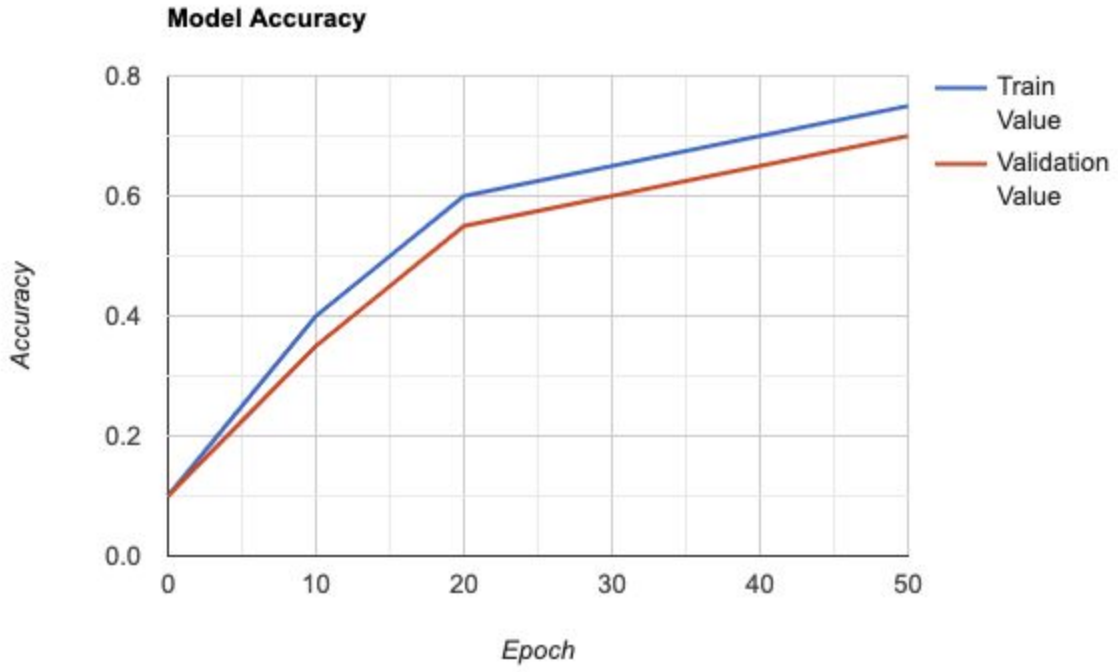


# MODEL EVALUATIONS

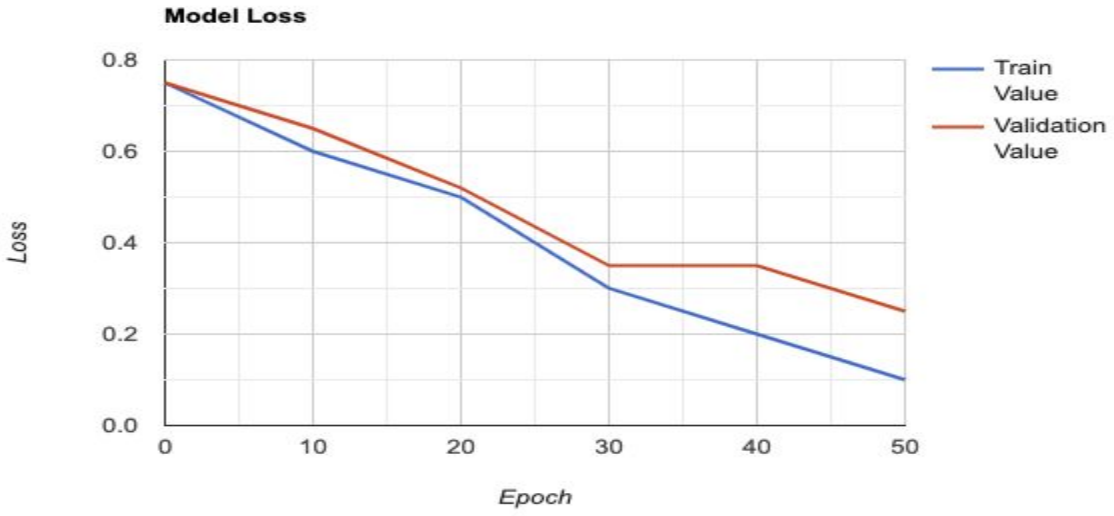
The model was trained by giving different parameters for batch_size, Epoch and dataset. Upon tuning parameters a final parameters of batch_size=64, Epoch=50 we obtained the **good fit** model with an accuracy of **80%**.

Below are the Model accuracy and Model loss graphs that were plotted with respect to the above tuned parameters.

**Model Accuracy:** Inferring from the graph below we can see that the accuracy of 'trained value' does not deviate more than 20 % from the 'validation value'.

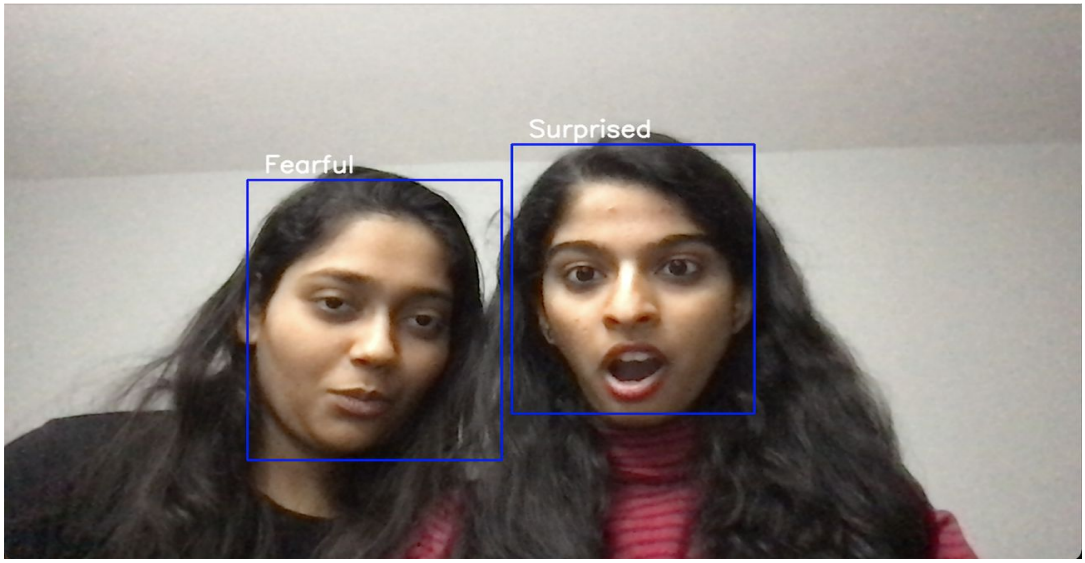



**Model Loss:** Inferring from the graph below we can see that the loss of 'trained value' does not deviate more than 20 % from the 'validation value'.

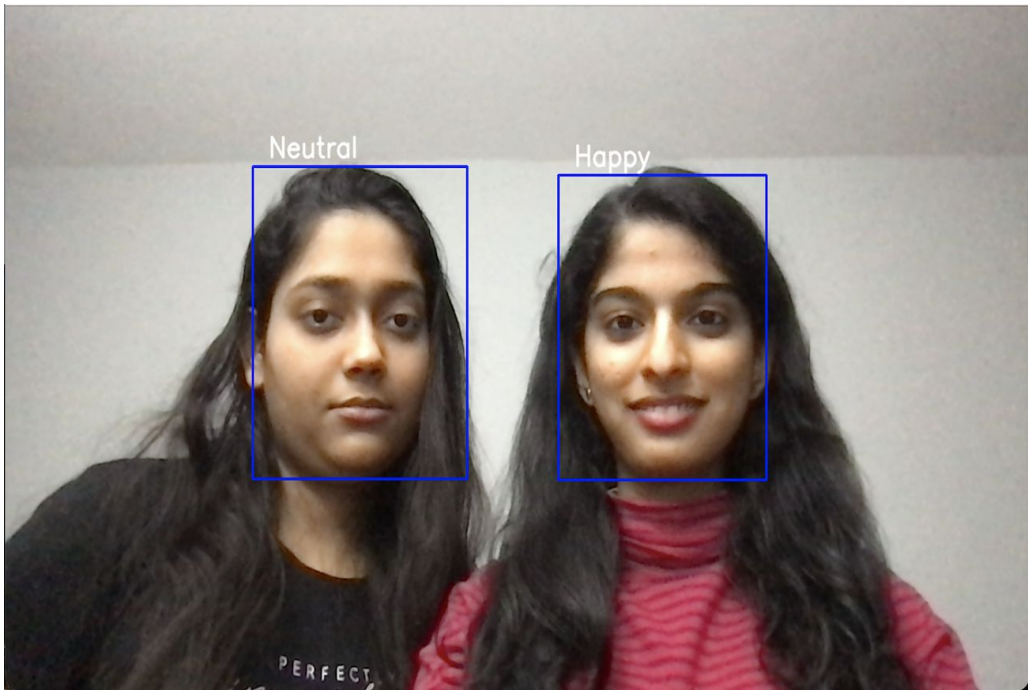



The model was able to classify the emotion of face detected to one of the seven categories as shown below:

Sample Output Screenshot 1 : Faces were detected in the blue  48x48 frame and the respective emotions were reflected such as Fearful and Surprised



Sample Output Screenshot 2 : Faces were detected in the blue  48x48 frame and the respective emotions were reflected such as Neutral and Happy




# CONCLUSION

The model classified the facial expressions into one of seven categories which are angry, disgusted, fearful, happy, neutral, sad and surprised. This was achieved with the help of using deep convolutional neural networks layer 4 classifiers. The model gave an accuracy of 80% with 50 epochs. The accuracy and loss of the model was also analysed.

The future work of the project would be to implement this facial emotion detection in lower end devices like mobile and other embedded devices , which we commonly interact in our daily life to tap maximum advantage of the technology.

# REFERENCES USED


[1] A. Jaiswal, A. Krishnama Raju and S. Deb, "Facial Emotion Detection Using Deep Learning," 2020 International Conference for Emerging Technology (INCET), Belgaum, India, 2020

[2] Deep Face Expression Recognition Survey - Shan Li and Weihong Deng* , Member, IEEE

[3] https://medium.com/@nickcancode/how-to-install-virtual-environment-wrapper-bfb656351b62

[4] Official AWS videos for building AWS DeepLens project using AWS SageMaker
https://www.youtube.com/watch?v=-JDLpSsO45A , https://www.youtube.com/watch?v=cTsCMTxEPHs

[5] AWS documentation on DeepLens
https://docs.aws.amazon.com/deeplens/latest/dg/deeplens-getting-started.html

[6] AWS documentation on SageMaker
https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html