EECS 4070

Professor: Jeff Edmonds

Project:

Emotion detection using Convolutional Neural Networks

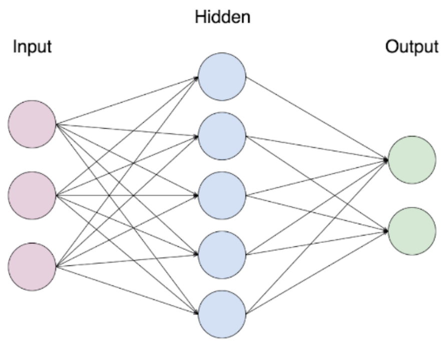
Name	Student number
Syed Omair Anwar	214729131
Bardia Goharanpour	214613996

Introduction

With this project, we set out to use our previous knowledge of Neural Networks to create a practical software. We extended on the Cat/Dog detection neural net to create one that can detect different facial features and emotions. So, it could potentially distinguish between someone being happy, sad and so on. For this project, we only focused on happy and sad emotions. To achieve this, we use a large database of pictures that we feed the neural net to train it, then we use the webcam on our laptops to obtain an image of our face, then use the trained model to determine the emotion we are displaying. We'll go into a lot more detail in each section.

Background

Neural Networks: A neural network is a software solution to replicate the brain's ability to learn and adapt. It does so by being trained on data related to the specific problem it's trying to solve. A neural net (as shown below) consists of an input layer, which takes in the data, an output layer, that makes the final prediction and classification, and in between, 0 or more hidden layers. Each layer is filled with neurons which have values and weights associated to them. Value comes either from the data (for the input layer) or comes from the previous layer. Weight determines how important this said data is to the final prediction. So, a value or attribute that has a high significance to the final result will have a high weight value whereas an insignificant neuron will have a low weight value. Each layer has an output that is the sum of each neurons weight*value. In the case of input or hidden layers, this output feeds into the next layer as input data. For the output layer on the other hand, this sum is only used to make the final prediction. Each layer's output comes from putting the mentioned sum through an "Activation function". There are many activation functions and depending on the situation, we use the appropriate one. Examples are "Rectifier", "Sigmoid", "Threshold" and so on. The main purpose of an activation function is to decide if the neurons in the next layer should be activated at all or not.



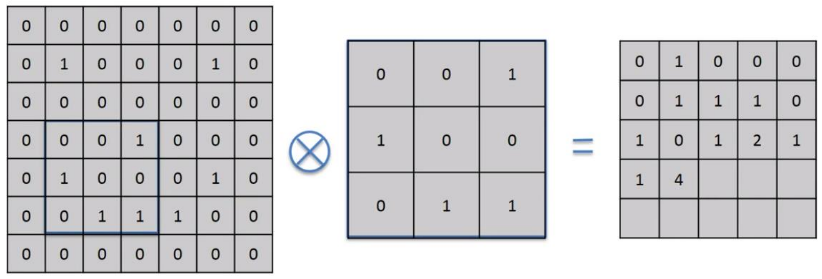
* There could be zero or more hidden layers

The data we choose to feed to the neural net is going to be one for which we know the result of the prediction for. In our example, we already know which pictures are of people being happy and which are people being sad. We then run these pictures through the net, get the predicted value, compare it to our expected result, and use the difference between the two values to adjust the weights of all the neurons in our neural net so that it could more closely replicate the result we want it to. This process is how the NN learns. This difference between the expected and obtained results is calculated using the "Loss function". This decides how drastically we react to wrong answer and by how much we correct our weights.

In this project, we are using a convolutional neural network (CNN) to make image-based predictions.

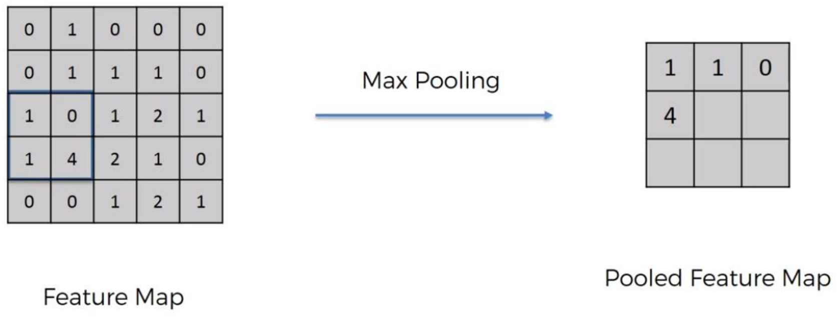
A CNN preprocesses images that are fed to an artificial neural network. To do so, images are sorted into folders based on their categories that the neural net will predict. 80% of the images are labeled as our training set, and the remaining 20% as our test set. Both the training set and the test sets are split into the different categories the neural network will try to predict.

Training a CNN is a multistep process where features in an image and their locations are recorded then fed to an artificial neural network. In the convolution stage, we use a feature detector, which covers a very small portion of the input image and tries to match pixel values. If the values of the pixels on the detector match those of the portion of the input image underneath it, then that is recorded into a feature map and the feature detector is then passed over the image to complete a feature map. This feature map contains the location and impact of the feature that the feature detector filter has. In the context of the image below, the input image on the left is turned into an array of ones and zeroes and the 3x3 feature detector has values of one or zero for each of its pixel. When this is passed over the input image, the number of ones that overlap are recorded in the feature map then the feature detector is moved over by a specified number of pixels and the process is repeated until the feature map is correctly populated. The convolution process is then repeated with multiple feature detectors, creating an equal number of feature maps with locations for different features for that input image.



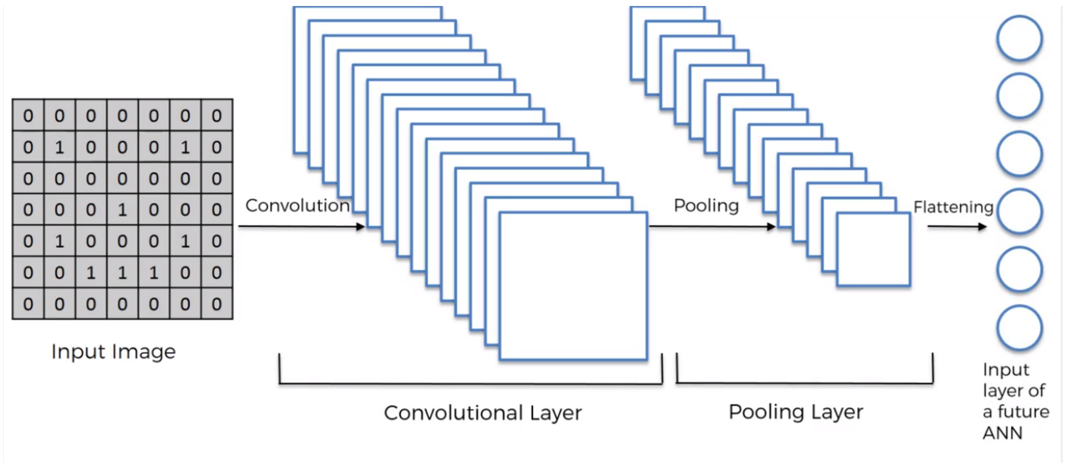
A feature could be thought of as a shape in the input image; for example, a line that can be identified by the feature detector. We then use an activation function with the output to increase nonlinearity of the values that are fed into the max pooling before they are fed into any additional convolutional layers.

Max pooling is done to highlight the importance of the most impactful features in a feature map and compress this information into a smaller, pooled feature map. We specify the number of pixels that we focus on at a given step.

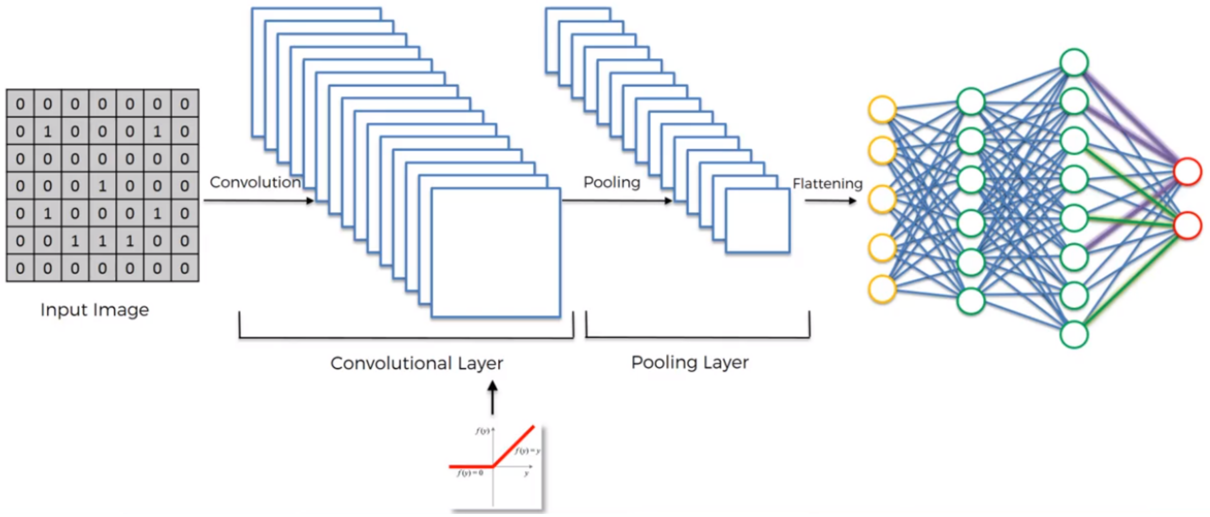


In the image above, we are using a 2x2 2D array. This array covers a portion of feature map and the highest value in that location on the feature map is recorded into a cell in the pooled feature map, and we move by 2 cells each time to fill the next cell in the pooled feature map. This step highlights the features that have the most importance and reduces the amount of additional data in the feature map that is not very impactful. We can then pass this to additional convolution and pooling layers to optimize our results.

The next step is to flatten these layers into a one-dimensional vector to feed it into an artificial neural network. The picture below shows all the previous steps before inputting the values to the ANN.



An ANN is used to classify what the image is. In the training step, the neural network makes a prediction of what the input image is and then based on the expected value, it back propagates to adjust its weights as specified by the loss function. This improves its accuracy over the course of its training. Below is a summary of the entire training process. The input image is read, processed, and has features identified and pooled before this information is flattened and fed to the neural network.



Preparation

For this part, we needed to collect all the data, in our case pictures, necessary to train the neural net. From recent experience working on the Cat/Dog recognizing CNN, we knew 5,000 pictures of each category is a great amount. While for the cat and dog situation we had a link from the course, we were going online to download the images which were already categorized, here we had no such thing. So, we tried a few ways and finally got a python script working that would take arguments of type, number of downloads and keywords to search. The script took these values, made a Google Images search based on them and downloaded however many pictures we asked for. We gave it the type of jpg, number of 1000 downloads each time, and we made around a dozen different search keywords for both sad and happy. Reason for that was usually after a certain number, the result began to not be what we really asked for; So, we had to find new keywords that would return the same general result we wanted. For example, for the "Sad" section, we searched things like "Sad adult man", "Sad woman", "Sad teenager" and things like that, just to keep it varied and get a ton of results.

Next, we had to go through all these search results by hand and delete the useless/irrelevant ones. A lot of times, a search of "Sad" would bring results of happy people, or emojis, or pictures with way too much background for the neural net to pick up on subtle facial features, or even sometimes the correct emotion but way too many people in the same photo. So, after going through these for hours and categorizing them correctly, we ended up with two folders, "Sad" and "Happy", each containing 5000 pictures. Then we had to split each of those into 2 folders, 4000 for the training set, and 1000 for the test set. The neural net would train on the 80% that exist in the training set, then it would test itself on the test set and return to us an accuracy percentage of how well our trained neural net is performing.

Libraries

Our program was developed with the help of many libraries for different purposes such as creating a neural net and processing the images we would feed to it. We used three libraries that assisted in creating a neural network:

1. Theano: A library for creating and optimizing data structures we used within the scope of the project.

2. TensorFlow: A library by google that is used as the backend for our neural network. We used the GPU version as we needed the added processing power to work with manipulating many images at a faster speed than working with a CPU.

3. Keras: Another important library for creating the neural network. It has features that are important to specifying the structure of our network.

We used multiple features of the Keras library: Sequential, Convolution2D, Maxpooling2D, Flatten, Dense, Dropout. Our neural network is an object of the type sequential. We used Convolution2D, MaxPooling2D and Flatten in the image preprocessing phase. We also used Keras' image preprocessing libraries to read the input images. Once our neural network was trained, we saved it so we could load it again later with the load_model feature of Keras.

In addition to these, we used additional image processing libraries like skimage, cv2 and PIL. Using Skimage, we read images; using cv2, we were able to crop them and using PIL, we obtained the dimensions of the images. The PyAutoGUI library allowed us to screenshot the image to feed to the neural net to predict. This was used with the camera application to predict emotion in real time. Using the subprocess library, we were able to run an executable that was a compiled C# code that allowed text to be displayed on screen.

In the C# code, we used system libraries such as IO to read a text file with the result, then drawing and InteropServices to display the resulting prediction on screen, over all other windows.

Implementation

In this section, we had to start by designing the neural net, that meant deciding how many layers we'll have, what kind of activation function to use, and so on. For our image processing part, we used 64, 3x3 feature detectors. Our input is a 256x256 image turned into 3 different layers, each representing one of the RGB colors. We first ran it using 128x128 images because the run-time to train was too long, but then we switched to 256x256 and it took much longer to train but our accuracy increased. Our activation function for the input layer was "Relu" (Rectified Linear Unit). Then we did all the different steps described in the previous sections such as flattening, max pooling, dense, dropout and so on. In the end, the image turns into a long one-dimensional array that gets fed to the neural net. Our neural net uses the optimizer function of "adam" and since our result is only one of two choices, our loss function is "binary cross entropy". If we had more than two emotions to recognize, we would've had to use "categorical cross entropy". Before we feed the images to our net, we try to transform them a little every time by rescaling, sheering, zooming, and horizontally or vertically flipping the image occasionally. The purpose of this is to make sure our model can also recognize images that are a little tilted or transformed. For example, if someone is looking at the camera from an angle, or their head is a little tilted while they're smiling, our neural net should still be able to pick up on that. Next up in the code, we set the directories for the training and the test set, we specify what size we want to use them at, how many per batch we grab from there, and we specify that there's only going to be two classes of images by setting "class_mode = 'binary'". Next comes the most time-consuming part of the whole thing, letting the neural net train! Side note, we were supposed to have 8000 pictures in our training set, but 7 of our images had format issues and we had to delete them. We figured training the neural net on 7993 pictures as opposed to 8000 should still be perfectly fine.

We let the training run and it took approximately 2 hours using TensorFlow-GPU. Next up, to avoid running the training every single time we close the software and having to wait another 2 hours, we figured out how to save our trained model to a file type of ".h5py" so that we could keep and load the model in a manner of seconds from now on! From this point on, we split our code into two separate files, one that was only used to train and save the neural net, and the second one loaded the already saved model and used it to run the live webcam demo.

After having saved the model, we can call upon it separately without having to spend the time doing so. The model's structure and weights are saved. Using load_model from the Keras library. We specify class labels, which are "Happy" and "Sad" in our case and we are now ready to test our model in real-time. in a loop, we first screenshot whatever is on screen. It is assumed that we will be running the webcam at this time. We save the screenshot and then crop it using cv2 and PIL and rewrite it to the same filename. We then predict the emotion that is in the screenshotted image and write it to a file. We then use the subprocesses library to call the exe generated by the compiled C# code.

In separate C# code, we used the streamreader to read the file with the result, we then used a graphics object to display the result the text on top of all the windows currently open. On every refresh, it draws a background to the text and then writes the text to remove any rendering errors from the previously displayed value.

While this value is getting displayed, our program continues the loop and checks the next screenshot to see if there were any changes since the last screenshot. The overall delay between this is about 2 seconds. This is enough time to run all the code in the loop before changes while keeping it short enough to detect and display changes in real time.

Results

We finally were able to run the application and play around and see results. Not only did it pickup on the basic sad and happy emotions, it managed to pick up on smaller nuances that we didn't necessarily anticipate. For example, if you cover your face with your hands, it considers that a sad emotion because there was a good number of pictures from our Google Images results that had people showing that gesture. Another it picked up was if your head is tilted down, it picks that up as a sad gesture too. It recognized different types of smiling, showing teeth, not showing teeth, and so on. Overall, I'd say it worked much better that we expected it to. Since we mostly fed it pictures with one person in them, it doesn't exactly work with both of us being in frame at the same time. Another problem is you must be relatively close to the camera so that your face takes up a good amount of the screen space.

Summary

In this project we combined multiple technologies to create a convolutional neural network to understand and learn human emotion. We first obtained a large dataset of images using a search engine image download script and specified keywords related to the emotion we were searching for. As the results returned between 600-800 results, we had to run multiple searches with creative keywords. This worked in our favor as we were able to show different expressions and features as part of the same emotion such as a frown or hands over the face demonstrating sad faces or smiles with and without teeth showing both being recognized as happy. We arranged these files into folders of 4000 each for the training set for each emotion and 1000 each for the test set.

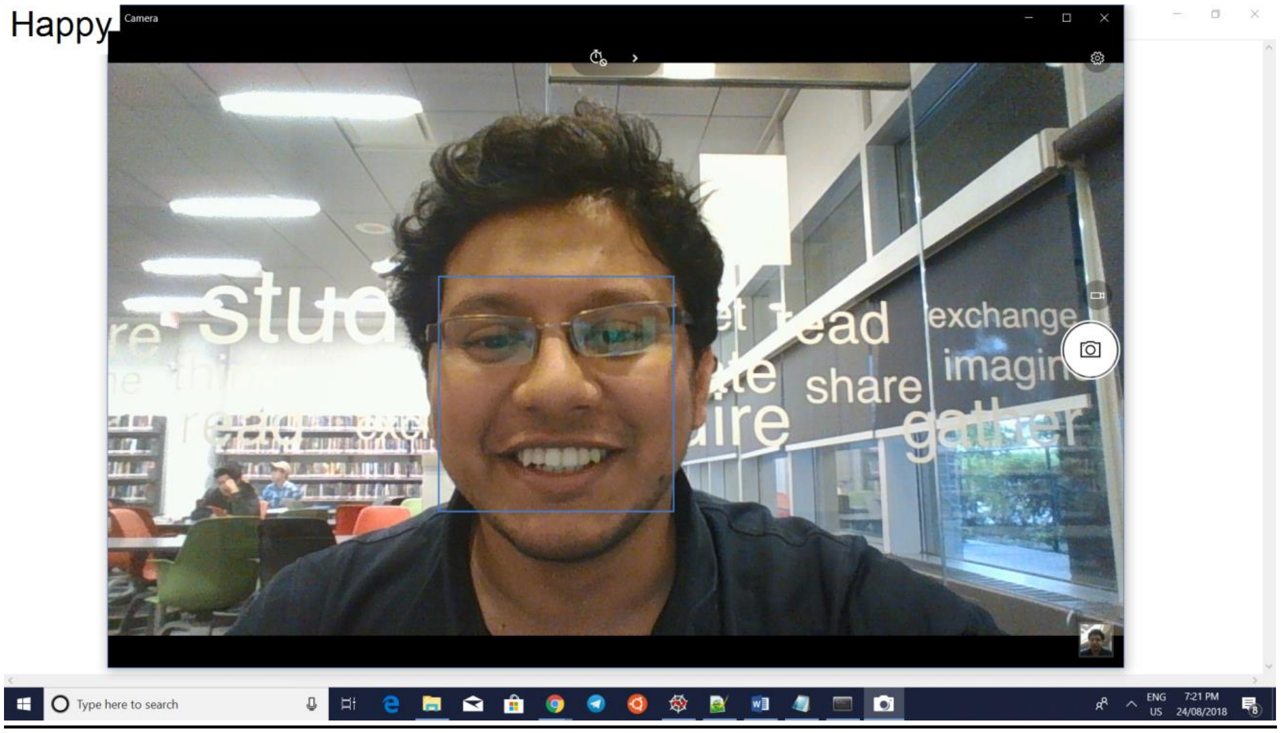
The next phase was writing the code. We created a convolutional neural network that first created feature maps and then pooled feature maps, flattened the data and fed it to the neural network in a 1D vector. The CNN we created trained the using backpropagation and the binary cross entropy loss function to adjust the weight and improve on it.

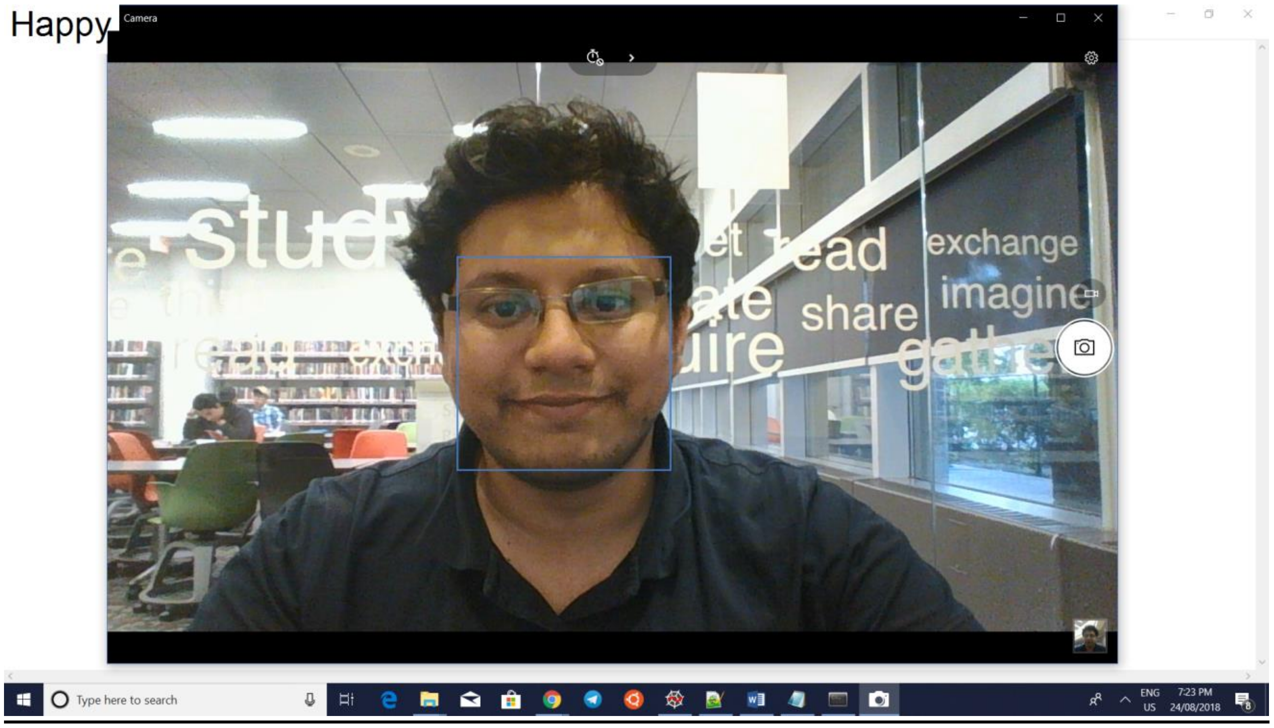
Once trained, we saved this neural network so that we had a model without having to spend the hours of training getting it prepared. We then figured out how to screenshot the display are of our computer and crop out unnecessary data. We ran this in a loop in which the screenshotted image was fed to the neural network to make a prediction, the result was saved in a file which we read from and displayed on screen. This was all done at regular short intervals to ensure that changes to emotion were picked up in real time.

With this project, we learned about data preprocessing for a neural network, working with many libraries, training and saving a neural network to use to make predictions, displaying on screen, image manipulation and screen capturing. We also learned to work with cross application communication while working with code written in different languages.

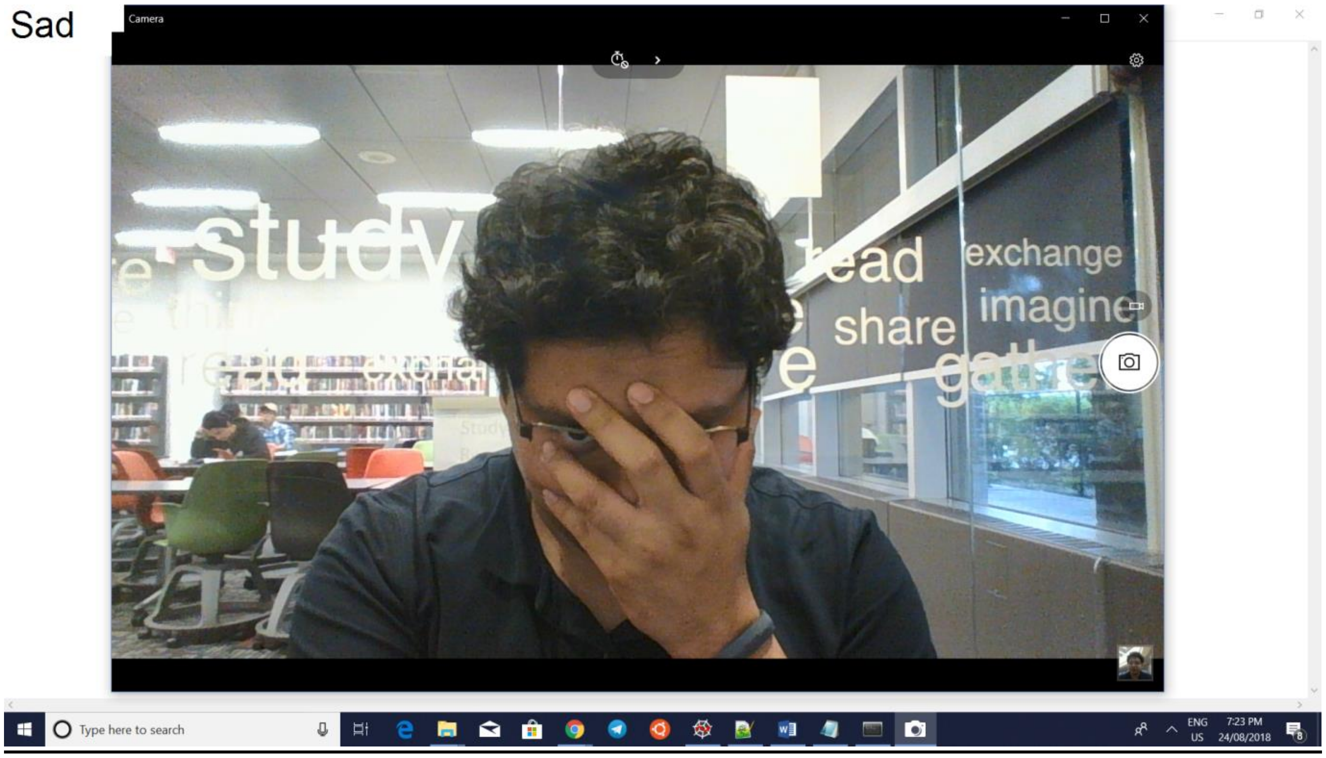
YouTube link of our test run:
https://www.youtube.com/watch?v=YPfWSsoYjp4

Test run screenshots

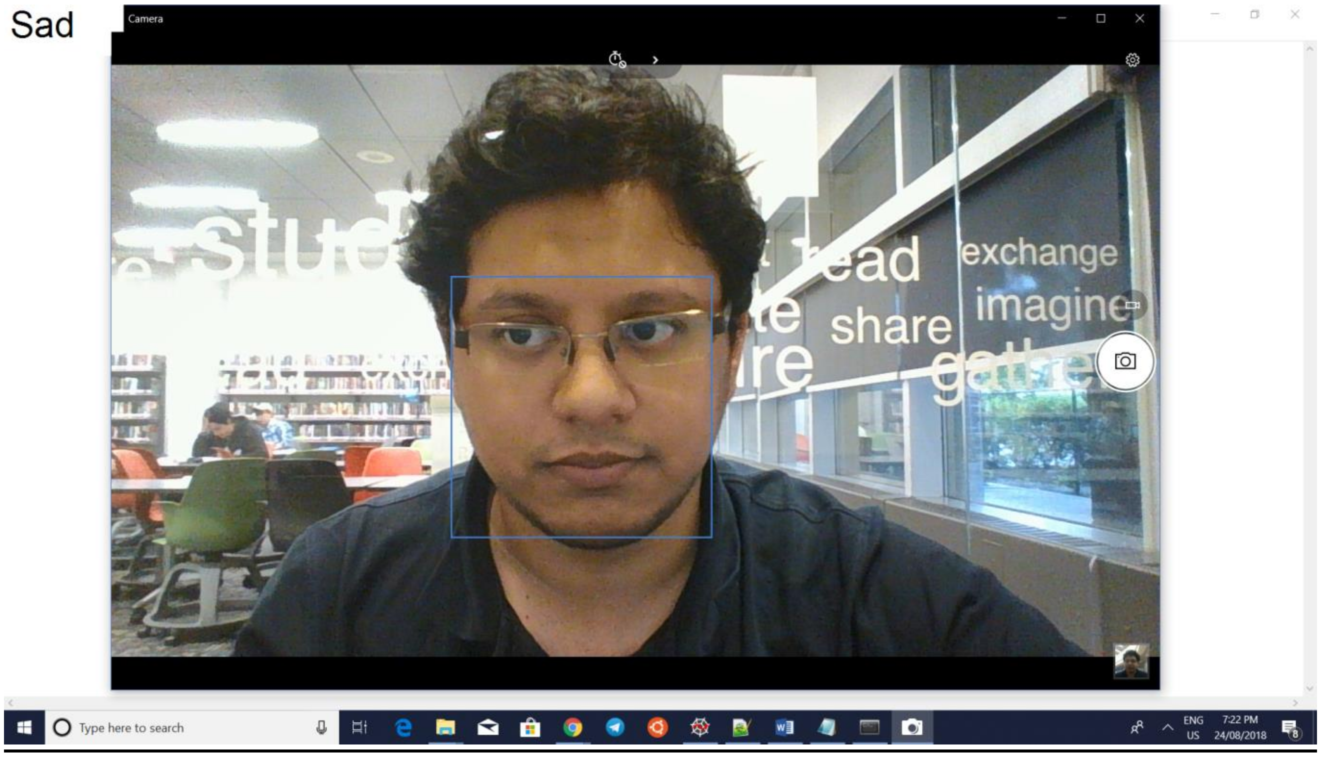




Sad


Camera
study
read
exchange
imagine
share
Type here to search
ENG
US
7:23 PM
24/08/2018

Sad


Camera
read
exchange
imagine
share
Type here to search
ENG
US
7:22 PM
24/08/2018