Emotion Recognition through Computer Vision and Deep Learning

Tumelo Mashabane (0861376)
Jainil Shah (0885959)
Catherine Rachel Samuel Rajkumar (0888987)
Sai Swaroop Madugula (0898140)

Introduction

Computer Vision can be described as the process of using machines to understand and analyze images as well as videos. It enables a computer to see, identify and process images in the same way that a human would do, and then provide appropriate output. Computer Vision is closely linked with artificial intelligence, as the system must interpret and perform several analysis techniques on the image or video. Image processing is one of the parts of the computer vision. Image processing consists of several techniques which are used to modify various attributes of the images. This allows the system to analyse the image with ease and produce an accurate output.

Ever since computers were developed, scientists and engineers thought of artificially intelligent systems that that are mentally and/or physically equivalent to humans. In recent years, the increase of generally all computation resources has provided a strong aiding hand for developing fast learning machines, with the internet suppling an enormous amount of data for training. These two developments boosted the research on smart self-learning systems, with neural networks among the most promising techniques.

Throughout this semester a wide variety of skills and techniques were learnt about computer vision and Image Analysis from this course. As part of our final evaluation our group selected a project and worked on it to display our new-found knowledge.

The **objective** of our project was to create a system that can recognise human emotion. Facial expressions are one of the most natural and immediate way for a human to convey their feelings or emotions. Emotion Recognition can be defined as the process of identifying human emotions from their facial expressions as well as from verbal expressions.

Many will say that the key step in the humanization of robotics is the ability to classify the emotion of the human operator. In our project we present the design of an intelligent system capable of emotion recognition trough facial expressions. A sophisticated well-tuned neural network architecture is customized, trained, and subjected to classification tasks, after which the network

is further optimized. The applicability of the final model is portrayed in an application that has a GUI that offers the options of instantaneous emotion analysis be it live or saved image of the user.

1. Emotion Recognition

We began by doing a study into Emotion recognition. We had to learn and understand what emotion recognition is. Literature suggests *emotion recognition* is best defined as the process of identifying human emotion, most typically from facial expressions as well as from verbal expressions; through audio and visual. These are two things that humans do automatically, but computational methodologies have also been developed, these methodologies leverage techniques from various disciplines, such as Signal processing, machine learning and computer vision.

It is important to point out that human emotion be it through facial expressions and body language is a universal key feature in human interaction. A review in literature showed that in the nineteenth century, Charles Darwin published upon globally shared facial expressions that play an important role in non-verbal communication [3]. In 1971, Ekman & Friesen declared that facial behaviors are universally associated with particular emotions [5]. Humans, including also animals, develop similar muscular movements belonging to a certain mental state, despite their place of birth, race, education, etcetera. Hence, if properly modelled, this universality can be a very convenient feature in human- machine interaction: a well-trained system can understand emotions, independent of who the subject is.

One should keep in mind that facial expressions are not necessarily directly translatable into emotions, nor vice versa. Facial expression is additionally a function of e.g. mental state, while emotions are also expressed via body language and voice [6]. More elaborate emotion recognition systems should therefore also include these other two contributors. However, this is out of the scope of this research and will remain a recommendation for future work.

As a final point of attention, emotions should not be confused with mood, since mood is considered to be a long-term mental state. Accordingly, mood recognition often involves longstanding analysis of someone's behaviour and expressions and will therefore be omitted in this work.

1.2. Image classification techniques

The growth of available computational power on consumer computers in the beginning of the twenty-first century gave a boost to the development of algorithms used for interpreting pictures. In the field of image classification, two starting points can be distinguished. On the one hand pre-programmed feature extractors can be used to analytically break down several elements in the picture in order to categorize the object shown. Directly opposed to this approach, self-learning neural networks provide a form of 'black- box' identification technique. In the latter concept, the system itself develops rules for object classification by training upon labelled sample data.

An extensive overview of analytical feature extractors and neural network approaches for facial expression recognition is given by Fasel and Luettin [6]. It can be concluded that by the time of writing, at the beginning of the twenty-first century, both approaches work approximately equally well. However, given the current availability of training data and computational power it is the expectation that the performance of neural network-based models can be significantly improved by now. Some recent achievements will be listed below.

i. A breakthrough publication on automatic image classification in general is given by Krizhevsky and Hinton [9]. This work shows a deep neural network that resembles the functionality of the human visual cortex. Using a self-developed labelled collection of 60000 images over 10 classes, called the CIFAR-10 dataset, a model to categorize objects from pictures is obtained.

ii. In another work which adopts the CIFAR-10 dataset [2], a very wide and deep network architecture is developed, combined with GPU sup- port to decrease training time. On popular datasets, such as the MNIST handwritten dig- its, Chinese characters, and the CIFAR-10 images, near-human performance is achieved. The extremely low error rates

beat prior state-of-the- art results significantly. However, it has to be mentioned that the network used for the CIFAR- 10 dataset consists of 4 convolutional layers with 300 maps each, 3 max pooling layers, and 3 fully connected output layers. As a result, although a GPU was used, the training time was several days.

iii. In 2010, the introduction of the yearly Imagenet challenge [4] boosted the research on image classification and the belonging huge set of labelled data is often used in publications ever since. In a later work of Krizhevsky et al. [10], a CNN with 5 convolutional, 3 max pooling, and 3 fully connected layers is trained with 1.2 million high resolution images from the ImageNet LSVRC-2010 contest. After implementing techniques to reduce overfitting, the results are promising compared to previous state-of-the- are models. Furthermore, experiments with lowering the network size were conducted, stating that the number of layers can be significantly reduced while the performance drops only a little.

iv. One of the most recent studies on emotion recognition describes a neural network able to recognize race, age, gender, and emotion from pictures of faces [7]. The dataset used for the latter category is originating from the Facial Expression Recognition Challenge (FERC-2013). A clearly organized deep network consisting of 3 convolutional layers, 1 fully connected layer, and some small layers in between obtained an average ac- curacy of 67% on emotion classification, which is equal to previous state-of-the-art publications on the same dataset. Furthermore, this thesis lays down a valuable analysis of the effect of adjusting the network size, pooling, and dropout.

Underlined by some other literature, the most promising concept for facial expression analysis is the use of deep convolutional neural networks. How- ever, the network from [2] (ii) is considered to be too heavy for our limited amount of available processing resources. The original network from [10] (iii) is large as well, but smaller versions are claimed to be equally suitable.

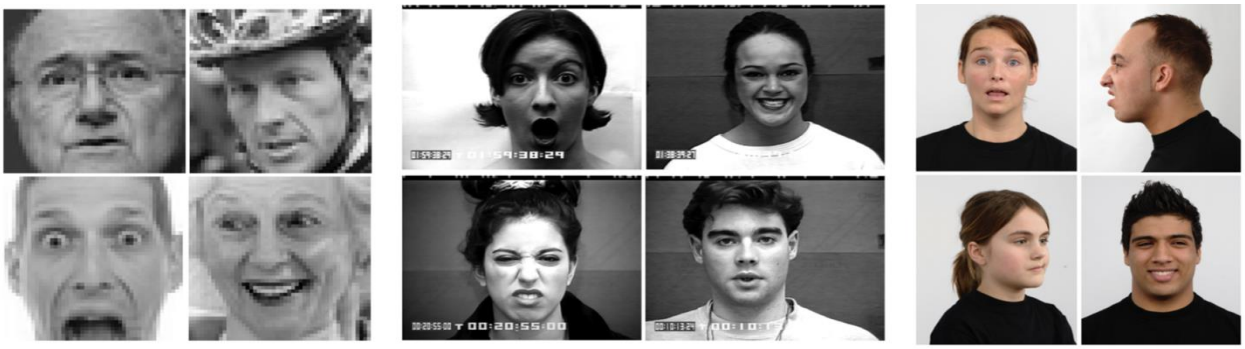

Figure 1: Samples images from different datasets.

2. Deep learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

2.2. Why Deep Learning?

In a word, accuracy. Deep learning achieves recognition accuracy at higher levels than ever before. This helps consumer electronics meet user expectations, and it is crucial for safety-critical applications like driverless cars. Recent advances in deep learning have improved to the point where deep learning outperforms humans in some tasks like classifying objects in images.
While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

1. Deep learning requires large amounts of **labeled data**. For example, driverless car development requires millions of images and thousands of hours of video.
2. Deep learning requires substantial **computing power**. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

2.3. How it works

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks.

The term “deep” usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.

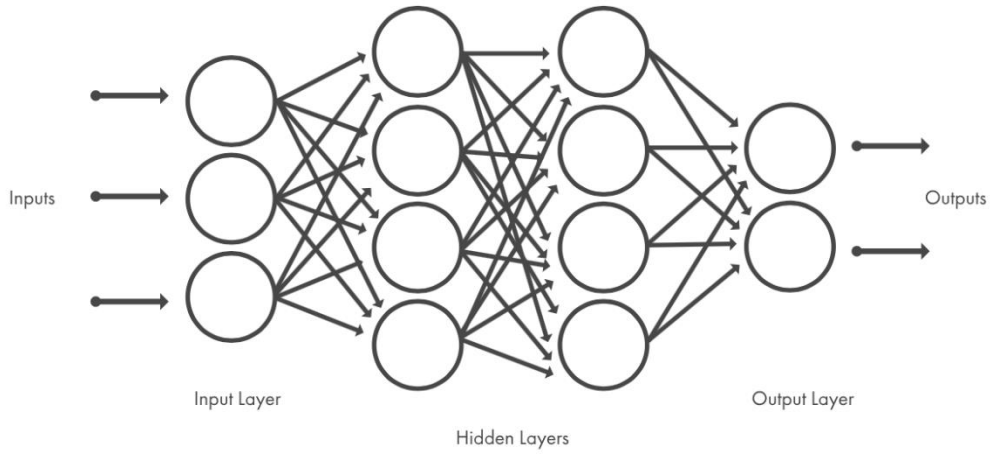

Figure 2: A convolutional neural network.

Figure 2: Neural networks, which are organized in layers consisting of a set of interconnected nodes. Networks can have tens or hundreds of hidden layers.

One of the most popular types of deep neural networks is known as convolutional neural networks (CNN or ConvNet). A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images this will be further discussed in the design chapter.

CNNs eliminate the need for manual feature extraction, so you do not need to identify features used to classify images. The CNN works by extracting features directly from images. The relevant features are not pretrained; they are learned while the network trains on a collection of images. This automated feature extraction makes deep learning models highly accurate for computer vision tasks such as object classification.

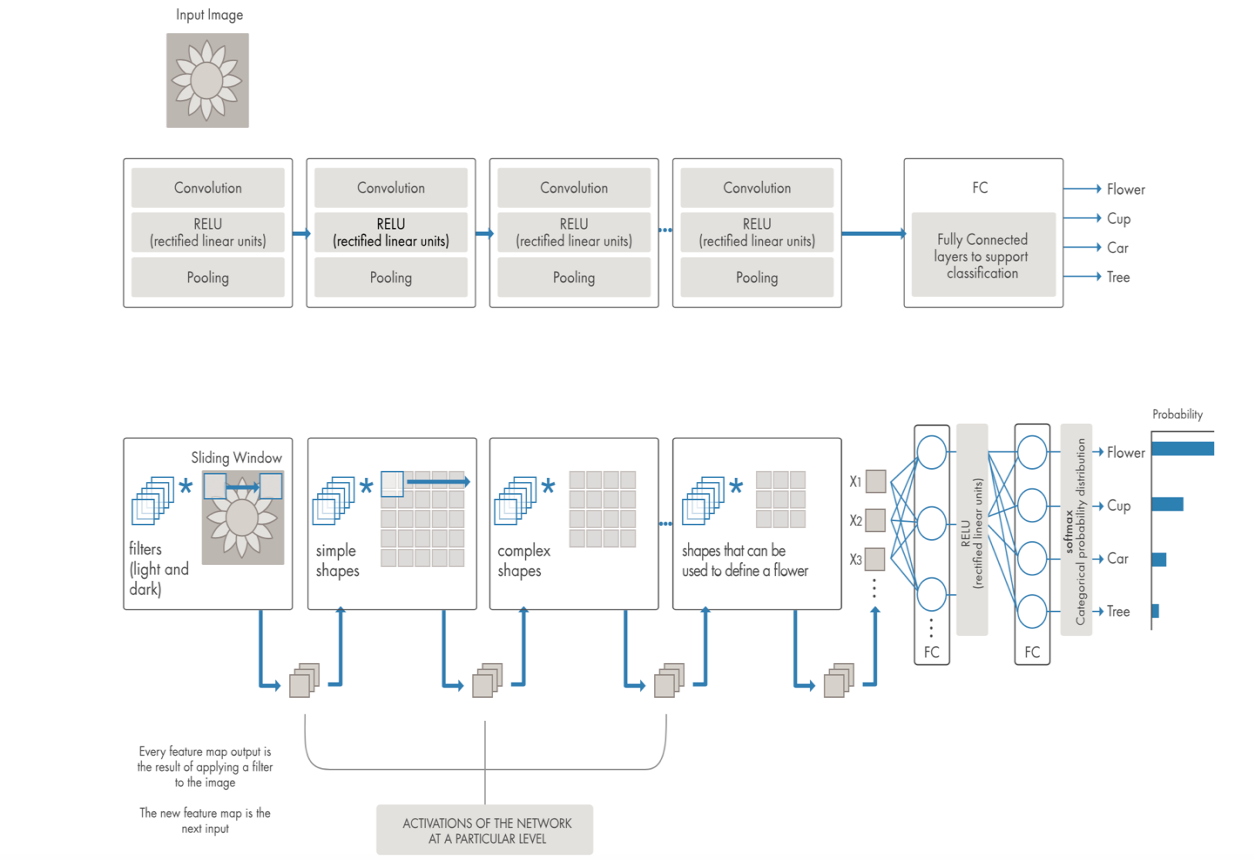

Figure 3: Example of a network with many convolutional layers.

Filters are applied to each training image at different resolutions, and the output of each convolved image serves as the input to the next layer.

CNNs learn to detect different features of an image using tens or hundreds of hidden layers. Every hidden layer increases the complexity of the learned image features. For example, the first hidden layer could learn how to detect edges, and the last learns how to detect more complex shapes specifically catered to the shape of the object we are trying to recognize.

3. Design

The Design phase of the project involves gathering data, creating the convolution network and making sure it's working in the desired way. Firstly, we shall commence with the acquisition of the data.

3.2. Dataset

Neural networks, or deep networks in particular, are known for their need for large amounts of training data. Moreover, the choice of images used for training are responsible for a big part of the performance of the eventual model. This implies the need for a both high qualitative and quantitative dataset. For emotion recognition, several datasets are available for research, varying from a few hundred high resolution photos to tens of thousands smaller images. The one we will work with is the Facial Expression Recognition Challenge (FERC-2013) [8].

The FERC-2013 dataset has 27,980 low resolution images. It can be noticed that the FERC-2013 set shows emotions *in the wild*. This makes the pictures a bit harder to interpret, but given the large size of the dataset, the diversity can be beneficial for the robustness of a model.

We reason that, once trained, images from 'clean' datasets can easily be classified, but not vice versa.

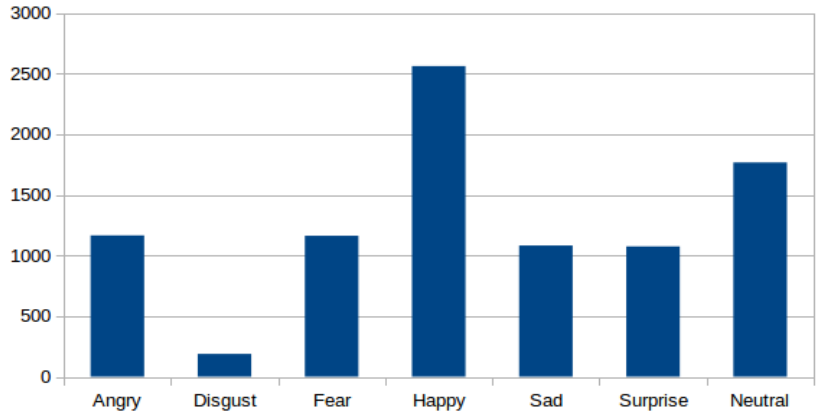

Figure 4: Number of images per emotion in the training set.

Please note that non-frontal faces and pictures with the label contemptuous are taken out of the RaFD data, since these are not represented in the FERC-2013 training set. Furthermore, with use of the Haar Feature-Based Cascaded Classifier inside the OpenCV framework [15], all data is preprocessed. For every image, only the square part containing the face is taken, rescaled, and converted to an array with 48x48 grey-scale values.

3.3. Preparing the Environment

The most important tool that was used was the Google API tensorflow. Tensorflow was used as a backend for the training model Keras. Other tools include

- numpy
- matplotlib
- opencv

To work more comfortably an isolated environment, we created an environment in anaconda where the tools were installed.

```
import tensorflow as tf

import keras
from keras import models
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.models import load_model

from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator

import numpy as np
import matplotlib.pyplot as plt

#------------------------------
#cpu - gpu configuration
config = tf.ConfigProto( device_count = {'GPU': 0 , 'CPU': 56} ) #max: 1 gpu, 56 cpu
sess = tf.Session(config=config)
keras.backend.set_session(sess)
#------------------------------
#variables
num_classes = 7 #angry, disgust, fear, happy, sad, surprise, neutral
batch_size = 256
epochs = 5
```

Figure 5: Importing necessary Packages.

3.4. Convolutional Neural Network

In this part of the project we create the CNN that will learn how to classify the images. The 3 primary layer types are discussed below.

Convolutional layer, which apply a specified number of convolution filters to the image. For each sub region, the layer performs a set of mathematical operations to produce a single value in the output feature map. Convolutional layers then typically apply a ReLU activation function to the output to introduce nonlinearities into the model.

Dense layer, which perform classification on the features extracted by the convolutional layers and down sampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

Pooling layer, which down sample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. A commonly used

pooling algorithm is max pooling, which extracts sub regions of the feature map (e.g., 2x2-pixel tiles), keeps their maximum value, and discards all other values.

Since this is a classification problem, we are ending the network with a Dense layer of size 7 as that is the number of classes that we have. Furthermore, a SoftMax activation function is used when connecting the layers SoftMax was used for its ability to give probabilities in logistic regression problem. This unit will encode the probability that the network is looking at one class or the others.

Note that CNN has 3 layers of which at the output of each layer we are going to do a pooling. This is done for feature extraction and thus the network learns. To combat the problem of overfitting dropout layers was used, dropping 20%. This was effective way to avoid overfitting. The first and second convolution layer have 64 channels and the third has 128.

```
model = models.Sequential()

#1st convolution layer
model.add(Conv2D(64, (5, 5), activation='relu', input_shape=(48,48,1)))
model.add(MaxPooling2D(pool_size=(5,5), strides=(2, 2)))

#2nd convolution layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(AveragePooling2D(pool_size=(3,3), strides=(2, 2)))

#3rd convolution layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(AveragePooling2D(pool_size=(3,3), strides=(2, 2)))

model.add(Flatten())

#fully connected neural networks
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(7, activation='softmax'))
```

Figure 6: Addition of convolutional layers.

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 44, 44, 64)        1664
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 18, 18, 64)        36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
average_pooling2d_1 (Average (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 128)         147584
_________________________________________________________________
average_pooling2d_2 (Average (None, 1, 1, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              132096
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 7175
=================================================================
Total params: 1,485,831
Trainable params: 1,485,831
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Summary of the classifier.

3.5. Training

As can be seen the final feature map has a shape size (1, 1, 128) outputs were flattened into vectors of shape 128, before going through three Dense layers and two Dropout layers. Observe that we have `1,485,831` trainable parameters, these are the parameters that the CNN is going to learn to classify in the images.

At this point now, we created the image data generator, of which is feed by the variable that we had created `x_train` and `y_train`. There after we create a variable `trainer`. The `trainer` is the what shall initiate the training. not that right from the beginning we had defined the number of epochs and the batch size. This is a neater way to organise the work.

Note that `fit` is used to fit our model into the generator, we do it using the `fit_generator` method. It expects as first argument a Python generator that will yield batches of inputs and targets indefinitely. Because the data is being generated endlessly, the generator needs to know example how many samples to draw from the generator before declaring an `epoch` over. This is the role of the `steps_per_epoch` argument: after having drawn `steps_per_epoch` batches from the generator, i.e. after having run for `steps_per_epoch` gradient descent steps, the fitting process will go to the next epoch. In our case, batches are 256-sample large, so it will take 256 batches until we see our target of 28709 samples.

```
gen = ImageDataGenerator()
train_generator = gen.flow(x_train, y_train, batch_size=batch_size)

model.compile(loss='categorical_crossentropy'
    , optimizer=keras.optimizers.Adam()
    , metrics=['accuracy'])

fit = True

if fit == True:

    trainer = model.fit_generator(
            train_generator, steps_per_epoch=batch_size, epochs=epochs)
else:
    model.load_weights('C:/Users/4217/Desktop/tumelo/VISION/weights.h5') #load weights
```

```
Epoch 1/5
256/256 [==============================] - 248s 971ms/step - loss: 1.8046 - acc: 0.2517
Epoch 2/5
256/256 [==============================] - 247s 967ms/step - loss: 1.6001 - acc: 0.3603
Epoch 3/5
256/256 [==============================] - 240s 938ms/step - loss: 1.3979 - acc: 0.4616
Epoch 4/5
256/256 [==============================] - 235s 917ms/step - loss: 1.2871 - acc: 0.5047
Epoch 5/5
256/256 [==============================] - 236s 921ms/step - loss: 1.1964 - acc: 0.5430
```

Figure 8: Training Progress and Accuracy.

As a good practice the training data will be store in a location on our drive so that its not lost. There after we evaluate the accuracy of the model that our CNN has made.

```
model.save('C:/Users/4217/Desktop/tumelo/VISION/weights.h5')
```

Figure 9: Trained Model stored in a .h5 file.

3.6. Testing

In the previous section the detector was trained and all the knowledge that was learn was saved in the weights.h5 file. Now comes the time to test the emotion detector, we selected a random image

from the internet named *test_image1*. There after we evaluate the image to detect the emotion the person in the image is expressing.

```
model.load_weights('C:/Users/4217/Desktop/tumelo/VISION/weights.h5')
```

```
img_path = 'C:/Users/4217/Desktop/tumelo/VISION/test_image1.jpg'
imgd = image.load_img(img_path)
img_tensor = image.img_to_array(imgd)
img_tensor /= 255

plt.imshow(img_tensor)
```

```
<matplotlib.image.AxesImage at 0x272800983c8>
```

Figure 10: Loading the trained classifier and the test image.

In the first part of the code snippet it can be seen that the model is loaded. The second part shows how the image is then converted into and array.

Note that before the image can be evaluated, we have to pre-process the image. This is so that we are able to have a tensor, which can be generalised as a matrix that the model can interpret. On top of that we even convert the image into a grayscale image. This is to further increase compatibility with our trained model. We preprocess the image into a 4D tensor.

```
img = image.load_img(img_path, grayscale=True, target_size=(48, 48))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
# Remember that the model was trained on inputs
# that were preprocessed in the following way:
img_tensor /= 255.
```

Figure 11: Conversion of test image into array (Pre-Processing).

The image is ready to be evaluated. For a much-structured evaluation we shall create a bar chart is created so that we can understand and see the results and see how the image faired against all the possible emotions that could have be observed.

```
#Create bar chart
def emotion_analysis(emotions):
    objects = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
    y_pos = np.arange(len(objects))

    plt.bar(y_pos, emotions, align='center', alpha=0.5)
    plt.xticks(y_pos, objects)
    plt.ylabel('percentage')
    plt.title('emotion')

    plt.show()

custom = model.predict(img_tensor)
emotion_analysis(custom[0])

img_tensor_prime = img_tensor
img_tensor_prime = np.array(img_tensor, 'float32')
img_tensor_prime = img_tensor.reshape([48, 48]);

plt.gray()
plt.show()
```

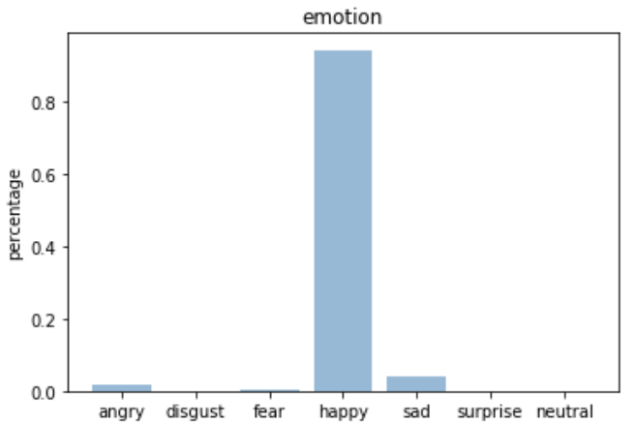

Figure 12: Image Evaluation and Emotion Recognition.

We can see that the emotion the boy in the image is showing is happiness. This is at a 94.29% confidence value as shown people.

```
print (max(custom[0]))
```

```
0.9428979
```

Figure 13: Printing the confidence level.

With is we can conclude that the image detector works and can be used to classify human emotion. But then comes as very important question which is "How does the emotion detector learn the facial characteristics of people to classify the emotion?". This is discussed in the next section.

3.7. Visualizing Trained data

It is often said that deep learning models are "black boxes", learning representations that are difficult to extract and present in a human-readable form. While this is partially true for certain types of deep learning models, it is definitely not true for convolutional networks.

There are various points that the learning process can be observed, but the point of interest for us is at the intermediate activations as this is where the feature extraction takes place.

In order to extract the feature maps, we want to look at, we created a Keras model that takes batches of images as input, and outputs the activations of all convolution and pooling layers. To do this, we used the Keras class `model`. `model` is instantiated using two arguments: an input tensor (or list of input tensors), and an output tensor (or list of output tensors). The resulting class is a Keras model.

```
# Extracts the outputs of the top 8 layers:
layer_outputs = [layer.output for layer in model.layers[:8]]
# Creates a model that will return these outputs, given the model input:
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

Figure 14: Using Keras model to extract feature maps.

When fed an image input, this model returns the values of the layer activations in the original model. For instance, this is the activation of the first convolution layer for our boy image input.

As mentioned in section 2, the first layer has 64 channels, in the following evaluation we visualize what the convolutional neural network sees in the first channel.

```
activations = activation_model.predict(img_tensor)

first_layer_activation = activations[0]
plt.matshow(first_layer_activation[0, :, :, 0], cmap='viridis')
```

```
<matplotlib.image.AxesImage at 0x272801f3470>
```

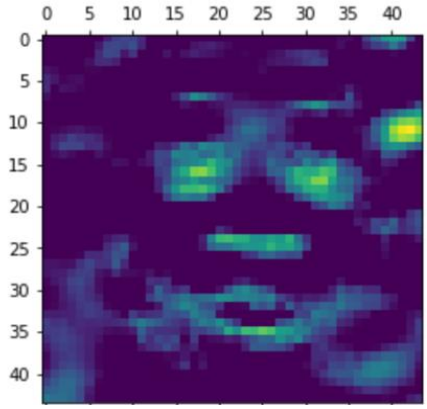

Figure 15: Visualizing how the neural network perceives the image during first layer.

This channel is able to encode some edge detection. As we look at other channels, we shall see how this varies between channels and layers.

At this point, we can plot a complete visualization of all the activations in the network. We'll extract and plot every channel in each of our 8 activation maps, and we will stack the results in one big image tensor, with channels stacked side by side.

```
# These are the names of the layers, so can have them as part of our plot
layer_names = []
for layer in model.layers[:7]:
    layer_names.append(layer.name)

images_per_row = 16

# Now let's display our feature maps
for layer_name, layer_activation in zip(layer_names, activations):
    # This is the number of features in the feature map
    n_features = layer_activation.shape[-1]

    # The feature map has shape (1, size, size, n_features)
    size = layer_activation.shape[1]

    # We will tile the activation channels in this matrix
    n_cols = n_features // images_per_row
    display_grid = np.zeros((size * n_cols, images_per_row * size))

    # We'll tile each filter into this big horizontal grid
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             col * images_per_row + row]
            # Post-process the feature to make it visually palatable
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] = channel_image
    # Display the grid
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
plt.show()
```

Figure 16: Extracting the features and storing them in an image tensor.

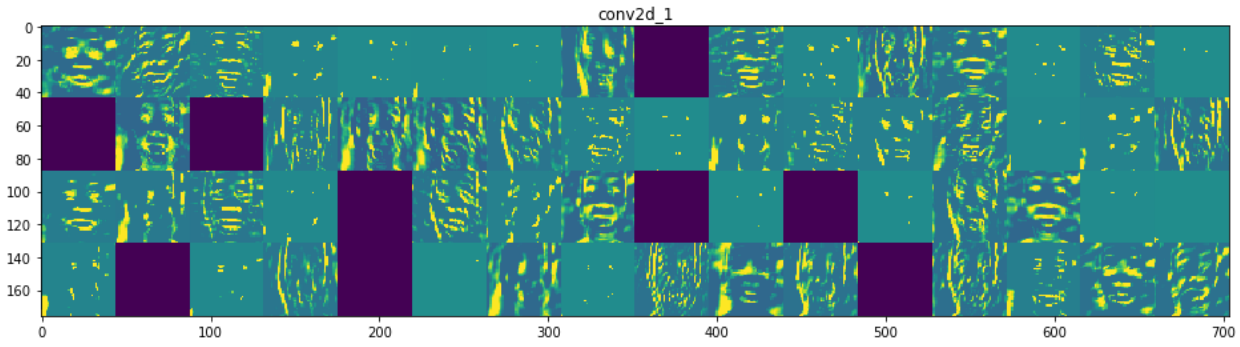

Figure 17:

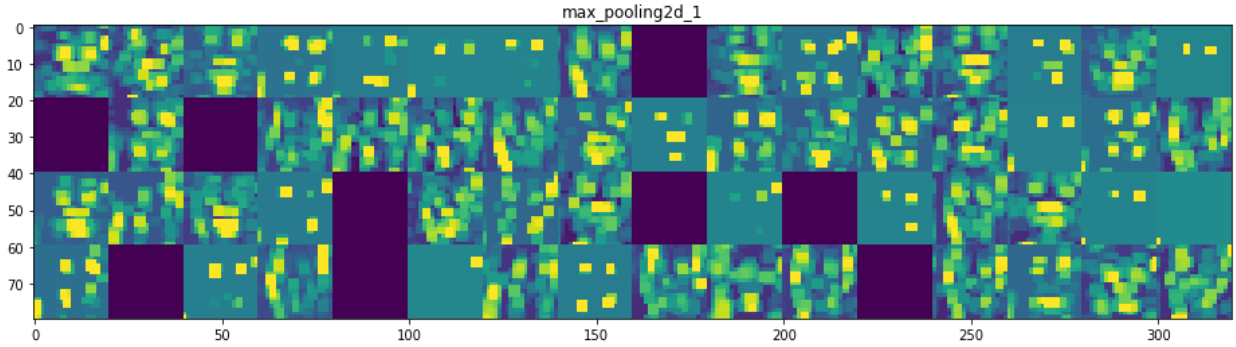

Figure 18:

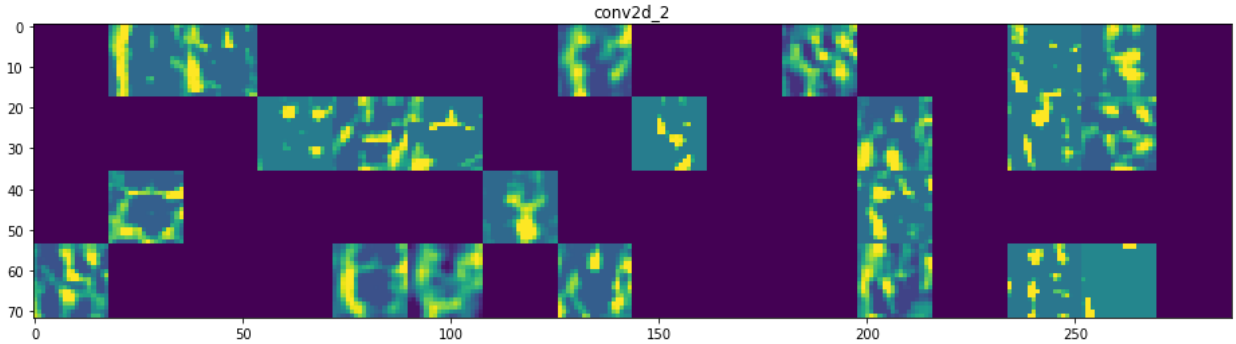

Figure 19:

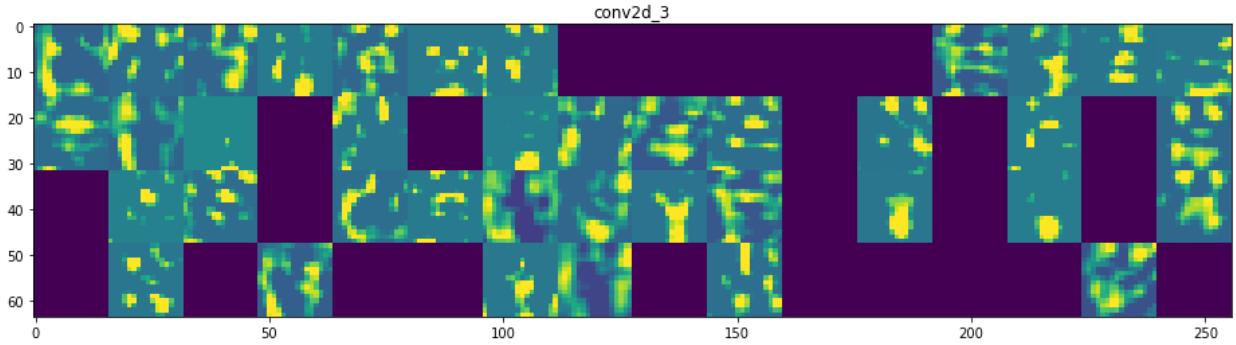

Figure 20:

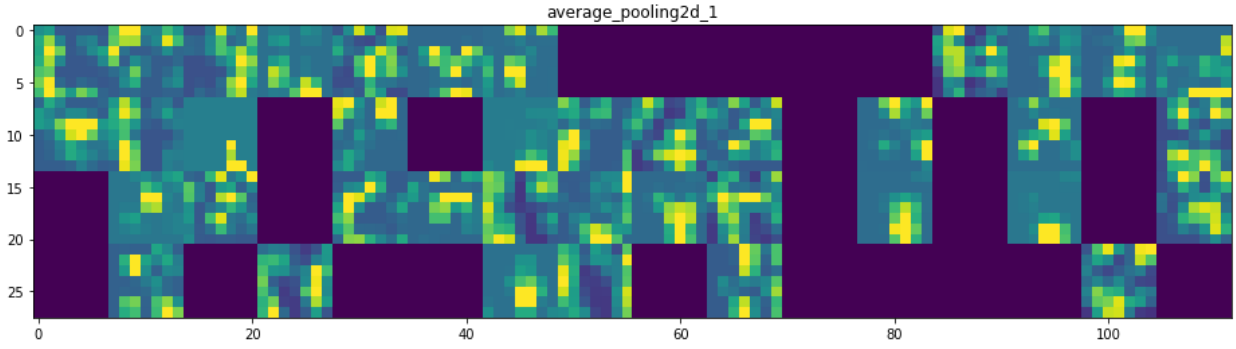

Figure 21:

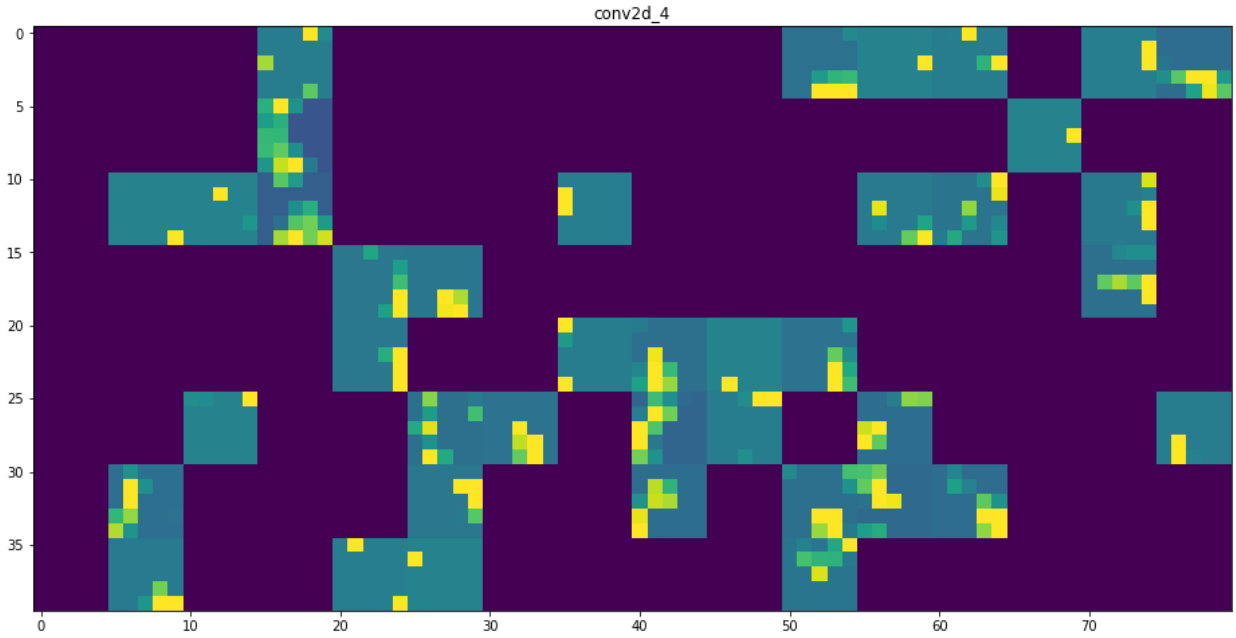

Figure 22:

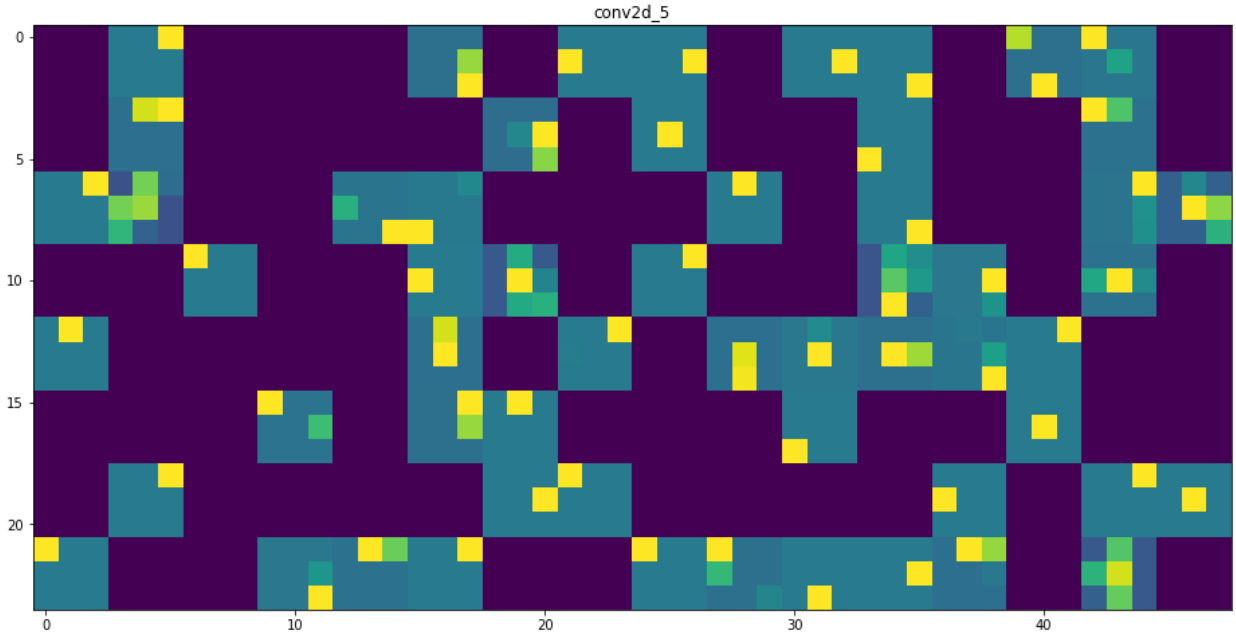

Figure 23:

Some interesting things to notes:

- The first layer is a collection of various edge detectors. At that stage, the activations are still retaining almost all of the information present in the initial picture.

- As we procced, the activations become increasingly abstract and less visually interpretable. They start encoding higher-level concepts such as "the boy's ears" or "the boy's eyes". Higher-up presentations carry increasingly less information about the visual contents of the image, and increasingly more information related to the class of the image.
- The sparsity of the activations is increasing with the depth of the layer: in the first layer, all filters are activated by the input image, but in the following layers more and more filters are blank. This means that the pattern encoded by the filter isn't found in the input image.

We have just proven a very important universal characteristic of the representations learned by deep neural networks: the features extracted by a layer get increasingly abstract with the depth of the layer. The activations of layers higher-up carry less and less information about the specific input being seen, and more and more information about the target (in our case, the emotion class in the image: Happy, Angry, Fear, Disgust, Surprise, Sad or Neutral).

A deep neural network effectively acts as an information distillation pipeline, with raw data going in (in our case, RBG pictures), and getting repeatedly transformed so that irrelevant information gets filtered out (e.g. the specific visual appearance of the image) while useful information gets magnified and refined (e.g. the class of the image).

Now the classifier is ready to be incorporated into the UI where we are going to evaluate images live from a webcam.

4. GUI AND RESULTS

In order to create a Graphical User Interface, we have used the PYQT5 package in python programming environment. The PyQt is a python framework used for developing desktop application. It is a crosse-platform application and widget toolkit for creating classic and embedded graphical user interface. The Pyqt API consists of numerous modules which large number of classes and functions. There are two main components in PYQT to handle different types of scenarios. The QtCore module handles the Non-GUI functionality which relates to tasks like working with directories or files. Other than this, there is another module known as QtGui which consists of all the graphical controls.

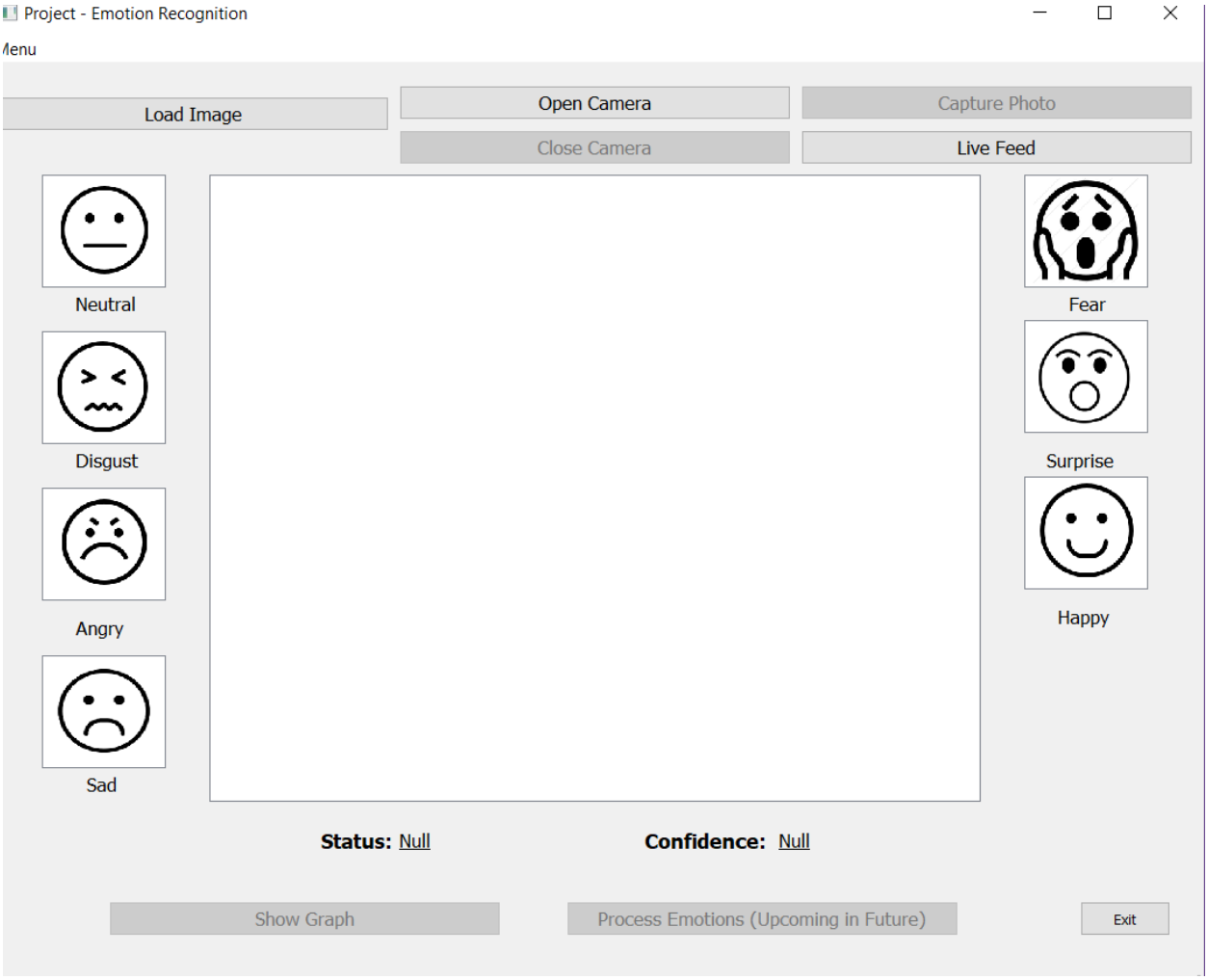

Figure 24: Graphical User Interface.

As we can see in Figure 24, we have created a simple GUI with the help of PyQt. For our objective, we have implemented an interface using several components each playing a specific role:

4.1. Buttons

In our GUI, we have placed a total of seven buttons (excluding the button for future work). Each button defines a specific purpose of the UI.

```
self.loadImage.clicked.connect(self.loadImageClicked)
self.showGraph.clicked.connect(self.showGraphClicked)
self.openCamera.clicked.connect(self.openCameraClicked)
self.closeCamera.clicked.connect(self.closeCameraClicked)
self.capturePhoto.clicked.connect(self.capturePhotoClicked)
self.exit.clicked.connect(self.exitClicked)
self.liveFeed.clicked.connect(self.liveFeedClicked)
```

Figure 25: Execution of methods upon Button Clicked.

In the Figure, we have designed our interface in such a way that whenever the user clicks on a specific button, the program executes the method (or function) associated to that button. For example, if the user clicks on the load Image button, the program executes the loadImageClicked() function.

Each button has a specific functionality:

4.1.1. Load Image

This button is used to load the image from the system directories. Whenever the user presses this button. A file dialog window opens, and the user is given the option to choose the file that they want to begin processing on. This lets the user try out different types of images without having to change the path for every image manually. After the user selects the image, our system automatically begins its detection and the relevant emotion is recognised and displayed (in the image holder by changing the black and white emoticon to color) as the output along with the confidence level, status and the graph which will be explained in the later section.

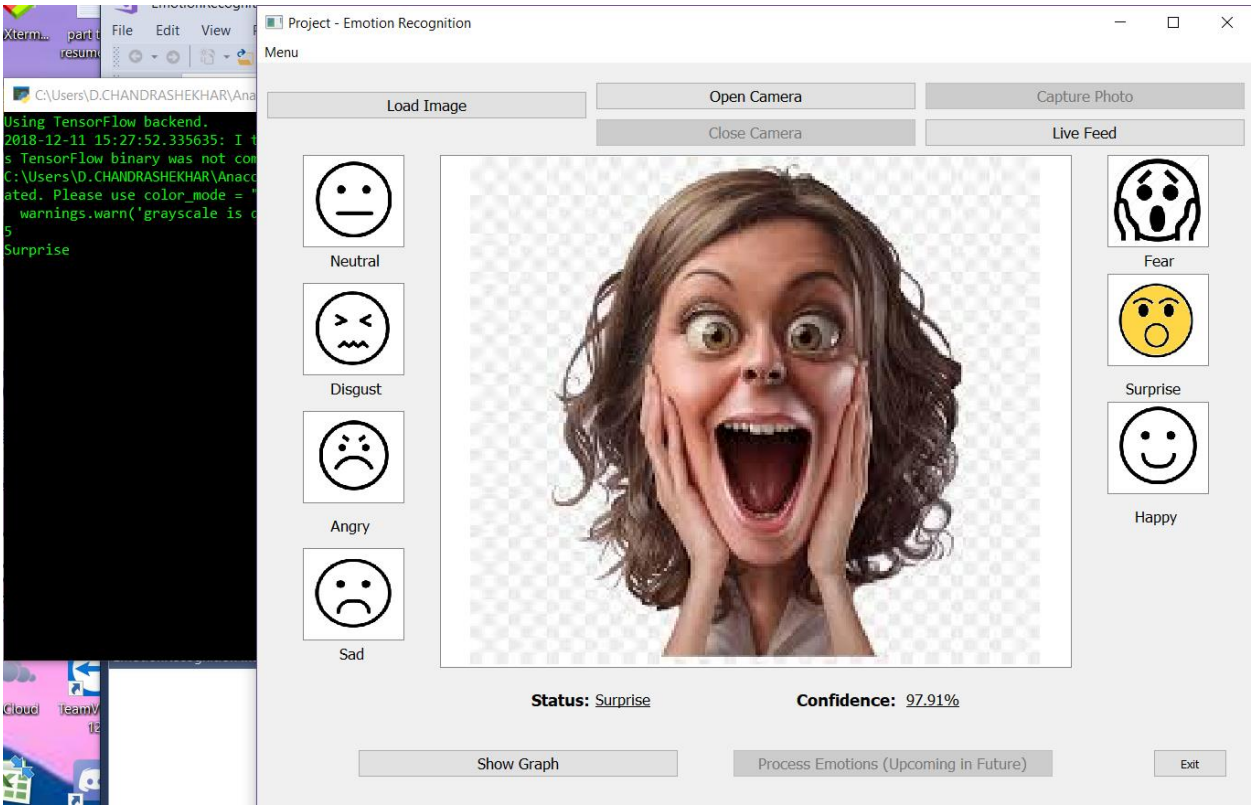

Figure 26: Load image and working of system.

Here in figure 26, the user clicks on the load image button and selects an image to process. After selecting an image, the image is preprocessed and classified by the trained model and the output is displayed. Here we can see that the emotion in the image figure is surprise. Our model successfully detects the emotion and converts the emoticon pertaining to the surprise field and highlights it into a colored image depicting that the emotion is surprise. In addition to this, the status is displayed as `surprise` which the recognised emotion and also, the confidence level is displayed which is 97.91%

.

4.1.2. Open Camera and Capture Photo and Close Camera

We have designed three buttons which provide the user to open the camera, capture an image and close the camera after they are done. When the user selects this option, the interface opens the webcam of the system (if it is present). After this, the user can select the capture photo button to capture a real time picture and store it in the parent directory of the source code file. After the image is successfully captured, we can click on the close camera button to switch

off the camera. After this, we can select the load image button again, and select the image which was previously captured to recognize the emotion of that image. This functionality provides the user to test out system using a real time image.

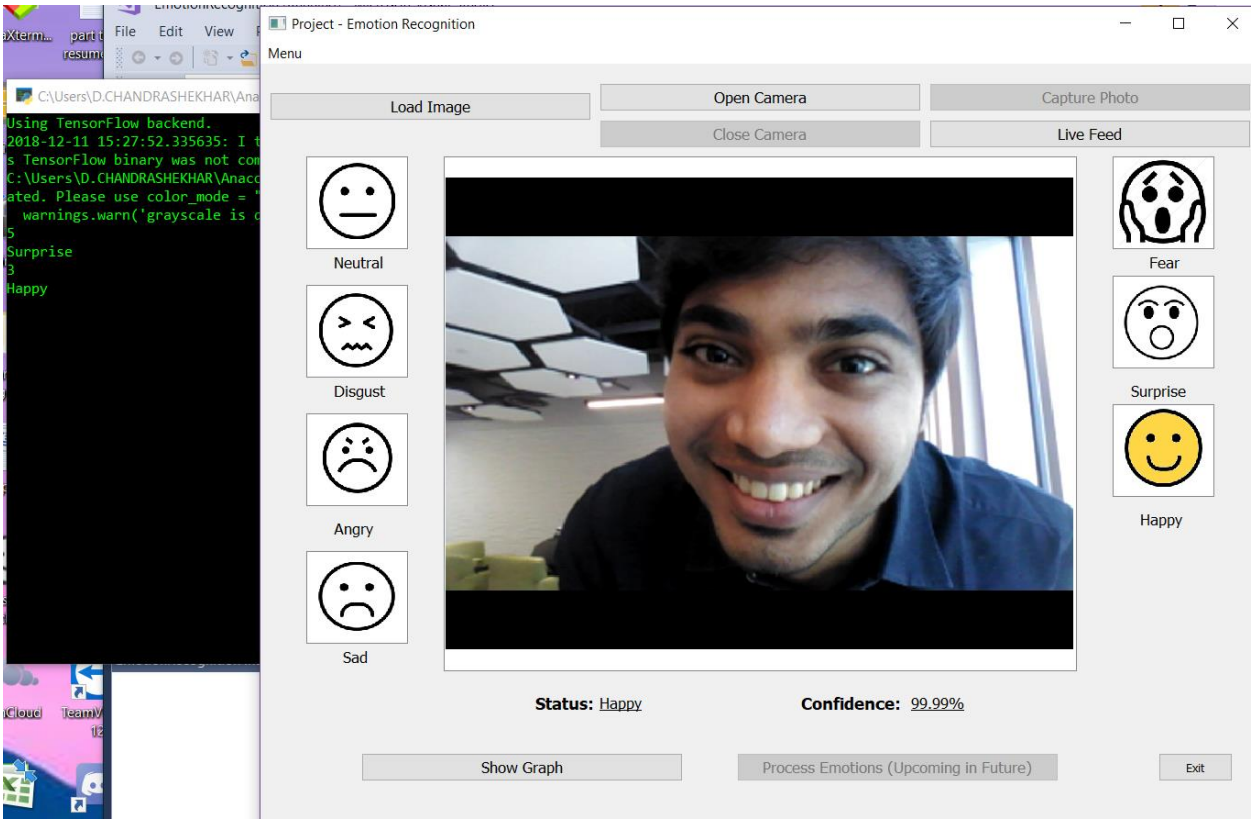

Figure 27: Capturing real time photo and emotion detection.

In figure 27, We are performing a real time image capture and detection. The user presses the open camera button and then, captures an appropriate photo. Then, they can click on close camera button to stop the capture. After that, the user clicks on the load image button and selects the image which is saved to the disk. Here we can see that, the model accurately recognises the emotion with a confidence level of 99.99%. However, this is not always accurate. It must be noted that factors such as lighting, orientation, background and color intensity will heavily influence the detection process.

4.1.3. Live Feed

This button allows the user to test the emotion recognition system in real time video capture. When the user clicks this button, the camera is opened, and the live feed is

displayed to the user. Now the user can try to enact different emotions to test the system. If the live feed is running, the system continuously detects the face and tries to recognize the emotions. This is done by taking each frame in the video and performing processing on each frame to gain the output. The user can see how the detector works in real time with this option.

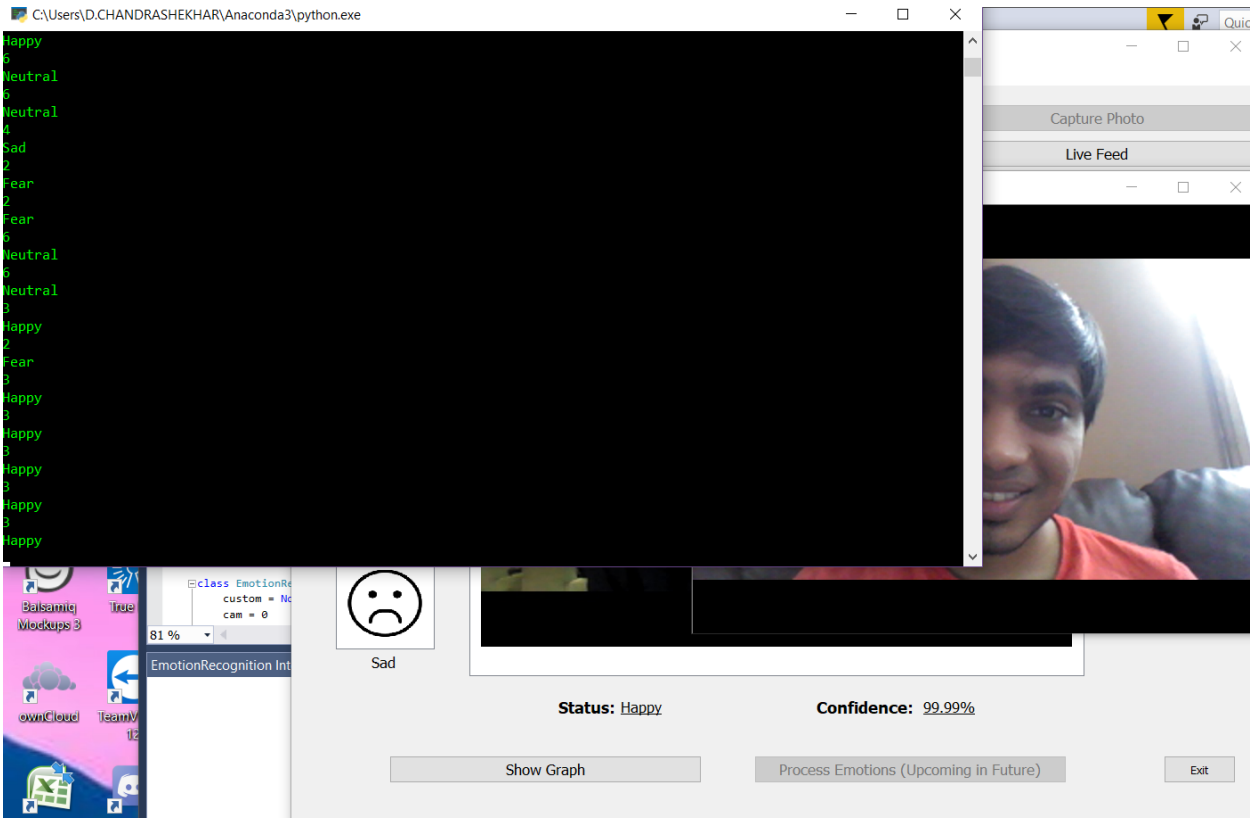

Figure 28: Live feed emotion Recognition.

In the above figure, we can observe that the recognition process is running continuously for the live feed. The user clicks on the live feed button and opens the webcam and the live video is streamed. Here the model detects the emotions as the person emotes in different ways. The output of this step is shown in the terminal. The timeout for detection is 1 sec. That means the model continuously detects the live feed with an interval of 1 second.

As mentioned earlier, the live feed process works by taking each frame of the video and processing it with the model. This is done with the help of the OpenCV. OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision [8(wiki)]. It is mainly used for image and video processing.

```
def liveFeedClicked(self):
    global cam, img, flagCam
    flagCam = 0
    cam = cv2.VideoCapture(0)
    self.openCamera.setDisabled(True)
    if(cam.isOpened() == True):
        self.closeCamera.setDisabled(False)
        self.capturePhoto.setDisabled(True)
```

Figure 29: Initialising the VideoCapture using openCV.

In figure 29, we are initialising the VideoCapture module of cv2. This variable is further used to open camera as well as reading the images in the feed.

```
while flagCam==0:
    ret, img = cam.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.resize(gray, (48, 48))

    #img = cv2.resize(img,(48,48))

    img_tensor = image.img_to_array(gray)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    #img_tensor /= 255.
    custom = model.predict(img_tensor)

    time.sleep(1)

    maxindex = custom.argmax()
    print(maxindex)

    emotion_array = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neut
    print(emotion_array[maxindex])


    cv2.imshow('Camera Feed',img)
    k = cv2.waitKey(30)
    if k == 27:
        break
cam.release()
cv2.destroyAllWindows()
```

Figure 31: Live feed emotion Recognition.

We can see in figure 31, the camera is opened, and the frames are being read with the help of `cam` variable which was previously initialised. Each frame is converted into a color frame and then resized to 48x48. This is because the trained model requires a lot of

processing power in case of images of large resolution. After resizing the frame, the image is processed into a tensor and sent to the prediction model.

4.1.4 Show Graph

This option allows us to observe how the detector has perceived the emotion. If an image is given to the system for detection, the classifier displays the emotion which has the highest confidence level. In this show graph option, we can see how the detector assigns confidence levels to each emotion and displays the result. From this graph, we can see how accurate the classifier is for a given image is. In a few cases, albeit classifier recognises the correct emotion, we can observe the graph the classifier assigning confidence level values close to some emotions. For example, a surprised face can be detected accurately but the confidence levels for the `fear` and `surprise` emotions might be very close. So, in order to handle these cases, we are taking the emotion which has the highest confidence of all seven emotions.

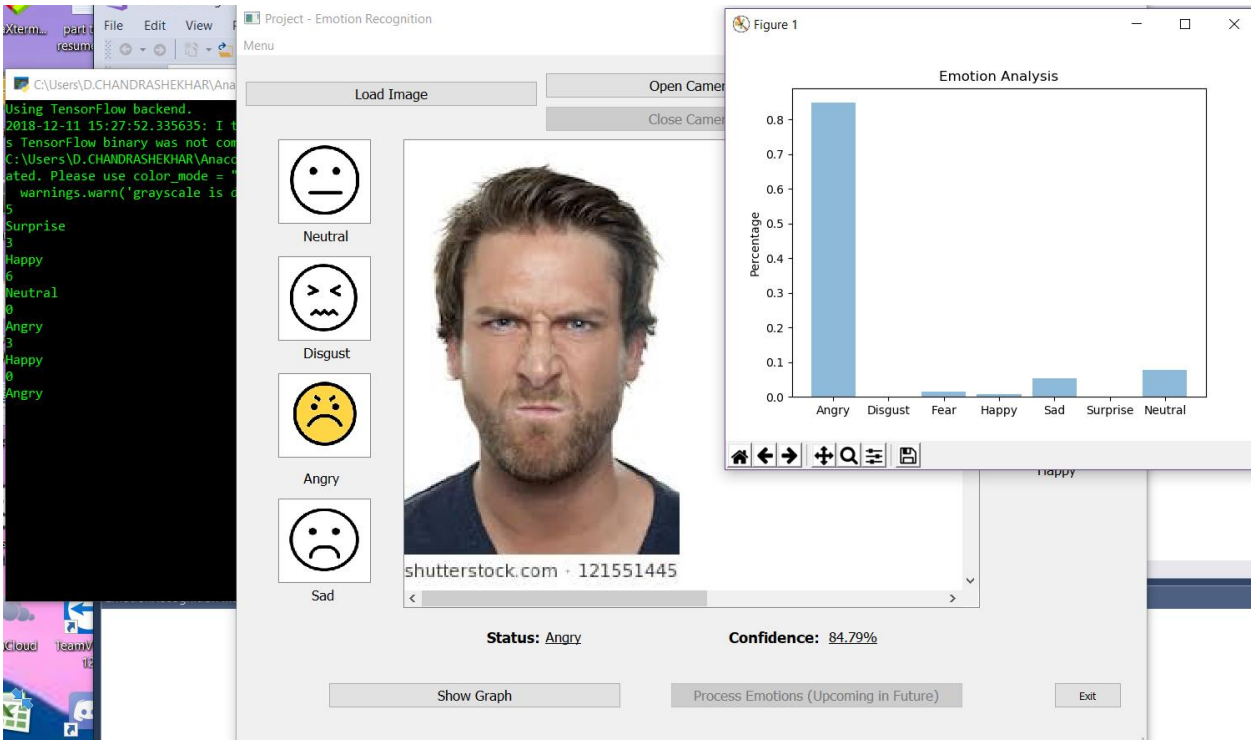

Figure 29: Graph consisting of various confidence levels of different emotions for a given image.

As you can observe in figure 29, the user clicks on load image and selects an image for processing. The model detects the emotion and displays it by changing the color of

the relevant emoticon. After that, if the user clicks on the show graph button, a new window opens displaying a bar graph which consists of the model's confidence levels computations. Here we can see that the model calculates various emotion's confidence level of the above image. Although, the emotion 'angry' has a high confidence level, we can observe that there exists a value for the emotions: fear, happy, sad and neutral (each having the confidence levels around 2%, 1.3%, 5% and 8% respectively). Angry, being the emotion with the maximum confidence level, it is selected as the recognised emotion.

4.1.5 Exit

As the name suggests, this option is used to exit the user interface and shutdown the program.

4.2. Image Holders

As you can see in the interface, there are icons pertaining to each emotion. To accomplish this, we used the QGraphicScene class present in the pyqt5 module. The QgraphicScene provides a surface for managing graphical items. Each item is considered as a container.

```
# Members.
self.imageHolderScene = QGraphicsScene()
self.neutralScene = QGraphicsScene()
self.disgustScene = QGraphicsScene()
self.angryScene = QGraphicsScene()
self.sadScene = QGraphicsScene()
self.fearScene = QGraphicsScene()
self.surpriseScene = QGraphicsScene()
self.happyScene = QGraphicsScene()
```

Figure 30: Image Holders.

As you can see in the figure, the QGraphicsScene() is used to assign image holders for each emotion. After specifying targets for all emotions, we load an image into that specific holder to display the image. In addition to this, each time the detector recognizes the emotion, it changes the image in the holder. It displays the colored image which is relevant to the output emotion.

4.3. Integration of Classifier and Process of Detection

```
self.showGraph.setDisabled(False)
# Initialize scene.
self.image = imageio.imread(filePath)
self.image = cv2.resize(self.image,(48,48))

self.imageHolderScene.clear()
self.imageHolderScene.addPixmap(QPixmap(filePath))
self.imageHolder.fitInView(self.imageHolderScene.itemsBoundingRect(), QtCore.Qt.KeepAspectRatio)

model = models.Sequential()

#1st convolution layer
model.add(Conv2D(64, (5, 5), activation='relu', input_shape=(48,48,1)))
model.add(MaxPooling2D(pool_size=(5,5), strides=(2, 2)))

#2nd convolution layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(AveragePooling2D(pool_size=(3,3), strides=(2, 2)))

#3rd convolution layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(AveragePooling2D(pool_size=(3,3), strides=(2, 2)))

model.add(Flatten())

#fully connected neural networks
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(7, activation='softmax'))

model.load_weights('facial_expression_model_weights.h5')

img = image.load_img(filePath, grayscale=True, target_size=(48, 48))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.
custom = model.predict(img tensor)
```

Figure 31: Integration of GUI and trained model.

5. CONCLUSION AND FUTURE WORK

By using Convolutional neural network, we have observed that our trained model is significantly accurate when compared with other models. The usage of CNNs are motivated by the fact that they can capture / are able to learn relevant features from an image/video at different levels like a human brain. This is also known as feature leaning.

6. REFERENCES

[1]. Minchul Shin, Munsang Kim, Dong-Soo Kwon “Baseline CNN structure analysis for facial expression recognition” IEEE Intl. RO-MAN, Nov. 2016.

[2]. Vibha Salunke, C.G. Patil “A New Approach for Automatic Face Emotion Recognition and Classification Based on Deep Networks” IEEE Intl. ICCUBEA, Sep. 2018.

[3]. Ariel Ruiz-Garcia, Mark Elshaw, Abdulrahman Altahhan, Vasile Palade “Stacked deep convolutional auto-encoders for emotion recognition from facial expressions” IEEE Intl. IJCNN, July 2017.

[4]. G A Rajesh Jumar, Ravi Kant Kumar, Goutam Sanyal “Facial emotion analysis using deep convolution neural network” IEEE Intl. ICSPC, July 2017.

[5]. https://en.wikipedia.org/wiki/OpenCV

[6]. http://pyqt.sourceforge.net/Docs/PyQt5/

[7]. https://keras.io/

[8]. https://jupyter-notebook.readthedocs.io/en/stable/