ABSTRACT

In today's world, which is influenced by social media, image sharing has become an integral part to mingle with different people across the world. People use social media for many purposes like, chatting, sharing of pictures, etc. There are numerous applications available, which are used extensively for image sharing like Instagram, Snapchat, etc. People are trying to find new innovative ways to represent themselves to rest of the world. Nevertheless, many people still feel insecure to share their original images, because we can see many cases, where images of individuals have been misused for faulty purposes.

So, we have come up with an innovative solution, where one can turn their own picture into an animated image, using Artificial Intelligence. The process of converting real-life high-quality pictures into practical cartoon scenes is known as **cartoonization**. Here, when image is given as an input to our software, by means of image processing, it is processed in the system. And then, an animated image is produced as an output to the given image. The white-box model is used in this, which gives more emphasis on human painting behaviours and cartoon images of different styles. The model decomposes images into three different cartoon representations, namely Surface Representation, Structure Representation and Textured Representation, which further counsel the network optimization to generate cartoonified images.

People can share these animated images, which removes privacy concerns. Also, people can represent themselves differently in a funnier way. Even during communication, one can use these images, instead of emoticons. Also, detecting the emotion of the person in the image would prove to be useful as the person need not update the status manually. The emotion detection algorithm would automatically perform facial recognition and detects the expression on the person's face and conveys it to other people accordingly.

Keywords – Cartoonification, Emotion Detection

G. Pavan Harsha Vardhan (2451-17-733-009)

Prashansa Evangeline Bonapalle (2451-17-735-055)

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Cartoonification is a recent colloquialism for the process of making something that's real look cartoonish i.e., drawing it in a (usually ridiculously) oversimplified, child-friendly or 'delightful' manner. The drastic increase in the usage of social media in today's world has made social media platforms come up with innovative ideas for people to communicate. One such feature that can be implemented is the usage of cartoonified images. Image processing is a very effective tool to modify and work on images. We can implement image processing solutions in an effective way, that can help people to share their images.

Generally human beings convey their intentions and emotions through nonverbal ways such as gestures, facial expressions and involuntary languages. This system can be significantly useful, nonverbal way for people to communicate with each other. This can be treated as a major drawback of texting. But, this texting can be improvised with cartoonified images which bring life to online communication. Cartoons are powerful because they provide a shortcut to a smile. Cartoons are more effective at clarifying complex issues, revealing simple truths we all share, and adding much needed visual interest and levity.

These cartoonified images would prove to be even more useful if the system or the platform itself can detect the emotion of the person in the image. We see many social media platforms, which ask us to select the status or the feeling that is to be displayed while sharing an image. This emotion detection algorithm will automatically detect the emotion of the person and update it accordingly.

The system classifies facial expression of the same person into the basic emotions namely anger, disgust, fear, happiness, sadness and surprise. The main purpose of this system is efficient interaction between human beings and machines using eye gaze, facial expressions, cognitive modelling etc. Here, detection and classification of facial expressions can be used as a natural way for the interaction between man and machine. And the system intensity vary from person to person and also varies along with age, gender, size and shape of face, and further, even the expressions of the same person do not remain constant with time.

However, the inherent variability of facial images caused by different factors like variations in illumination, pose, alignment, occlusions makes expression recognition a challenging task. Some surveys on facial feature representations for face recognition and expression analysis addressed these challenges and possible solutions in detail.

1.1 PROBLEM STATEMENT

The use of Social Media has increased extensively in the past few years. The social media platforms have grown drastically and help millions of people around the world to communicate effectively. These social media platforms have made calling, texting and reaching out to people around the world easier. In a world that runs by means of communication, it is important for anyone to visually express their feeling, even virtually. Emoticons and stickers can be used to communicate, as seen in many mediums of communication. Stickers have also come into a major use for communicating with other people. Emoticons and stickers do serve as a source if entertainment, while communicating through texting, but can they be used to express feelings virtually?

Another such problem is, sharing pictures through social media. Teenagers and youngsters today, are keen about how their social media profiles look, rather than their academic profiles. This raises security concerns among the people who, on a regular basis post their pictures in their social media handles. Does this affect them? To a larger extent, yes they do.

Few major problems associated with photo sharing:

Expressing Feelings: This has been one of the major issues. People keep texting with emoticons and stickers, but are they effective enough to express virtually? Do they help? Is it same as if we talk in reality?

Photo Morphing: Pictures are often morphed and used inappropriately. They tend to damage reputation of people in the society.

Security: People in our society have become very conscious and are trying to supress themselves, without sharing images on their handles. One shouldn't restrict themselves or others, because of the security concerns. This issue is the one that must be fixed.

Figure 1.1

Innumerable cases are seen where pictures picked up from people's social handles and are morphed in an inappropriate manner. Though it is not the real person in the picture, it affects him/her mentally.

Figure 1.2

In this situation, a growing need for an application, that can eliminate the insecurities and help people convey their expressions arises. The emoticons and stickers do not convey the exact emotion of a person. The emotion detection algorithm can automatically detect the emotion and share it as well.

1.2 OBJECTIVES

- Developing a web application that cartoonifies the image uploaded to it.
- Cartoonification of the image using Image Processing.
- To experiment Machine Learning Algorithm in Computer Vision Fields.

- To perform Facial Recognition and detecting the emotion of the person in the image, thus facilitating Human-Computer Interaction.
- To overcome the disadvantages of sharing real images online.
- To update the status or feelings of a person easier by detecting the emotion automatically.

1.3 SCOPE

This cartoonifying an image resolves the security concerns among the people, who tend to express themselves on a regular basis on the social media. It is a source by which a new and fancied form of our image can be displayed. This can be used to provide a comic view of oneself. Some camera also provides few filters, that can express the pictures in different ways. But, they do not quite serve the purpose as cartoonifying an image does. Even the emoticons aren't quite suitable to express the actual feelings. This software can also be used to reduce the risk of cybercrimes. It doesn't allow the unauthorized usage of the original images.

The emotion detection can also be used to update the status automatically, instead of choosing it manually every time. It can be used in social media platforms to share our emotions with various people, like friends and relatives across the world. This makes reaching out to the world easier, better and faster.

1.4 PROPOSED SYSTEM

The project is divided into 2 modules, the first one cartoonifies the uploaded image and the second module detects the emotion from it. The first module aims to take an image as input and performs image processing on it, thus cartoonifying it using OpenCV and produces the cartoonified image as an output. The second module aims to develop a convolution neural network for classifying human emotions. The model is first trained using a dataset consisting of facial images and their expressions. Then this model takes the cartoonified image as the input and outputs the expression or emotion of the person in the image.

Figure 1.3

Figure 1.4

CHAPTER 2
LITERATURE SURVEY

The core technology that supports the cartoonifying application, is image processing and the for emotion detection, the algorithms, through which the model is trained to detect the emotion of the person in the image, that is given as input. The usage of this application begins with capturing the image, which needs to be transformed. The image must be scanned, so that the image obtained can be transformed right from the pixel level. Using pixelation and different algorithms of image processing, the appropriate output image can be produced, after the image is passed through the networks. This is done for a single filter. The idea of developing this application comes from using image processing through Python. Developing a code in python to scan the image and applying AI algorithms and networks through white box model is the important step involved in this application. Then, the image undergoes facial recognition and the trained model detects the expression of the person in the image.

The model decomposes images into three different cartoon representations, which further counsel the network optimization to generate cartoonified images.

1. **Surface Representation:** It helps to extract smooth surfaces of the image that contains a weighted low-frequency component where the colour composition and surface texture are retained along with edges, textures, and details.
2. **Structure Representation**: It helps to derive global structural information and sparse colour blocks, once done we implement adaptive colouring algorithms like the Felzenswalb algorithm to develop structural representation that can help us to generate sparse visual effects for celluloid styled cartoon process.
3. **Textured Representation**: It helps us to retain painted details and edges. The three-dimensional image is converted to single-channel intensity map that helps to retain pixel intensity compromising colour and luminance, it follows the approach of manual artist that first draw a line sketch with contours and then apply colours to it.

The extracted outputs are fed to a Generative Neural Networks (GAN) framework, which helps to optimize our problem making the solution more flexible and diversified.

2.1 IMAGE PROCESSING

Image processing is performed in a very basic manner using Python.

Step 1: Import the required library. Skimage package enables us to do image processing using Python.

Step 2 : Import the image. Once we have all the libraries in place, we need to import our image file to python.

Step 3 : Analysing the model using the pixels obtained through image.

Step 4 : Validated whether all pixels are transformed.

Utilization of libraries is very important to use different applications in python. Different libraries like numpy, seaborn, etc. can be used for calculating operations in this application.

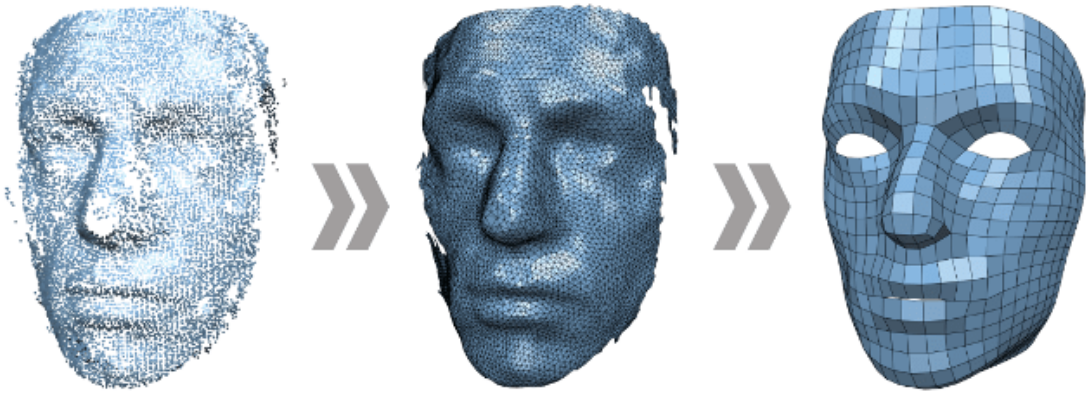

Figure 2.1

In particular, digital image processing is a concrete application of, and a practical technology based on:

- Classification

- Feature extraction
- Multi-scale signal analysis
- Pattern recognition
- Projection

Some techniques which are used in digital image processing include:

- Anisotropic diffusion
- Hidden Markov models
- Image editing
- Image restoration
- Independent component analysis
- Linear filtering
- Neural networks
- Partial differential equations
- Pixelation
- Point feature matching
- Principal components analysis
- Self-organizing maps
- Wavelets

Pixelation and neural networks are few important techniques, on which we have our eyes set on, to obtain the cartoonification feature.

2.2 FILTERING

Digital filters are used to blur and sharpen digital images. Filtering can be performed by:

- convolution with specifically designed kernels (filter array) in the spatial domain
- masking specific frequency regions in the frequency (Fourier) domain

2.3 PIXELATION

In computer graphics, pixelation is caused by displaying a bitmap or a section of a bitmap at such a large size that individual pixels, small single-coloured square display elements that comprise the bitmap, are visible. Such an image is said to be pixelated.

Early graphical applications such as video games ran at very low resolutions with a small number of colours, resulting in easily visible pixels. The resulting sharp edges gave curved objects and diagonal lines an unnatural appearance. However, when the number of available colours increased to 256, it was possible to gainfully employ anti-aliasing to smooth the appearance of low-resolution objects, not eliminating pixelation but making it less jarring to the eye. Higher resolutions would soon make this type of pixelation all but invisible on the screen, but pixelation is still visible if a low-resolution image is printed on paper.

In the realm of real-time 3D computer graphics, pixelation can be a problem. Here, bitmaps are applied to polygons as textures. As a camera approaches a textured polygon, simplistic nearest neighbour texture filtering would simply zoom in on the bitmap, creating drastic pixelation. The most common solution is a technique called *pixel interpolation* that smoothly blends or interpolates the colour of one pixel into the colour of the next adjacent pixel at high levels of zoom. This creates a more organic, but also much blurrier image. There are a number of ways of doing this; see *texture filtering* for details.

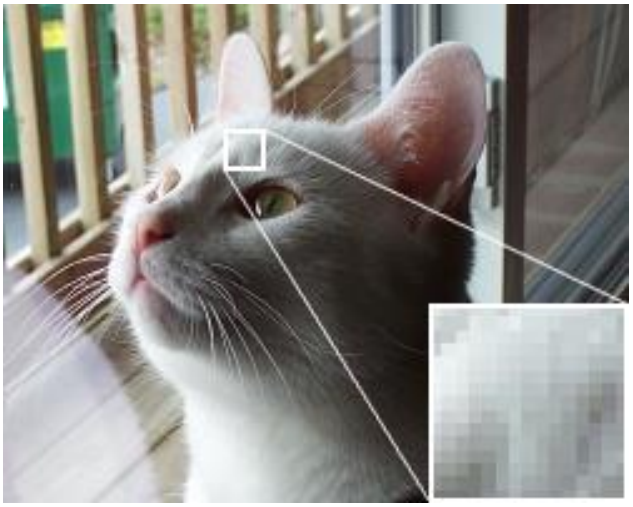

Figure 2.2

Pixelation is a problem unique to bitmaps. Alternatives such as vector graphics or purely geometric polygon models can scale to any level of detail. This is one reason vector graphics are popular for printing – most modern computer monitors have a resolution of about 100 dots per inch, and at 300 dots per inch printed documents have about nine times as many pixels per unit of area as a screen. Another solution sometimes used is procedural textures, textures such as fractals that can be generated on-the-fly at arbitrary levels of detail.

2.4 NEURAL NETWORKS

A neural network is a system of interconnected artificial “neurons” that exchange messages between each other. The connections have numeric weights that are tuned during the training process, so that a properly trained network will respond correctly when presented with an image or pattern to recognize. The network consists of multiple layers of feature-detecting “neurons”. Each layer has many neurons that respond to different combinations of inputs from the previous layers. The layers are built up so that the first layer detects a set of primitive patterns in the input, the second layer detects patterns of patterns, the third layer detects patterns of those patterns, and so on. Typical CNNs use 5 to 25 distinct layers of pattern recognition. Neural networks are inspired by biological neural systems. The basic computational unit of the brain is a neuron and they are connected with synapses.

2.4.1 CONVOLUTION NEURAL NETWORK

A Convolutional neural network is a neural network comprised of convolution layers which does computational heavy lifting by performing convolution. Convolution is a mathematical operation on two functions to produce a third function. It is to be noted that the image is not represented as pixels, but as numbers representing the pixel value. In terms of what the computer sees, there will simply just be a matrix of numbers. The convolution operation takes place on these numbers. We utilize both fully-connected layers as well as convolutional layers. In a fully-connected layer, every node is connected to every other neuron. They are the layers used in standard feed-forward neural networks. Unlike the fully connected layers, convolutional layers are not connected to every neuron. Connections are made across localized regions. A sliding "window" is moved across the image. The size of this window is known as the kernel or the filter. They help recognize patterns in the data.

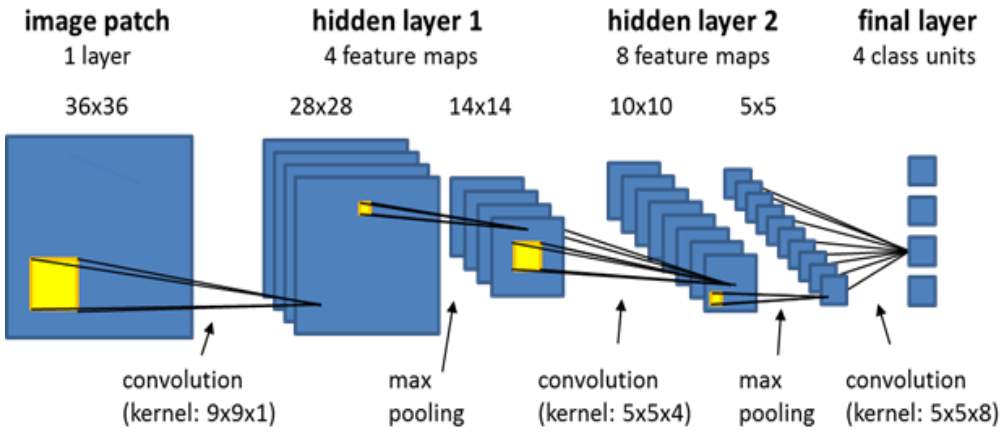

Figure 2.3

2.5 DATASET DESCRIPTION

The dataset contains 35,685 examples of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression in to one of

seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The dataset contains two columns, "emotion" and "pixels". The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. The "pixels" column contains a string surrounded in quotes for each image. The contents of this string are a space-separated pixel values in row major order. This dataset was prepared by Pierre-Luc Carrier and Aaron Courville.

2.6 RELATED WORK

There are lots of methods to help to generate cartoon image data. Ha et al. [Ha and Eck 2017] develop a method to automatically generate sketch drawings based on a neural representation. They use sketch-RNN to achieve both conditional and unconditional sketch generation, which can draw simple cartoon sketch drawings like cartoon cats, buses, penguins and so on. This method can provide lots of sketch drawings in a reasonable amount of time, but the quality is not good enough because the dataset they use contains just doodles drawn by human. Liu et al. [Liu et al. 2017] use GAN to paint black-white sketches automatically, and achieve satisfactory results on Japanimation. However, it cannot be used to auto-paint simple cartoon images. The main reason is that Japanimation usually contains lots of complex colors but the cartoon image we need just contains several simple colors. Gatys et al. [Gatys et al. 2016] use Convolutional Neural Network(CNN) to transfer the style of one image to another one, which can be used to transfer cartoon styles. Given a source cartoon image, it can change the target image to the cartoon style, but the result is not stable. Huang et al. [Huang and Belongie 2017] furtherly optimize the style transfer method to real time. Dong et al.[Dong et al. 2017] propose a method to synthesize realistic images directly with natural language description. Given an image and text descriptions, it can apply the style that the text described to the image, but it does not achieve satisfactory results on cartoon images as well. Besides, 3D models can also help to generate data because of their rich information. Mitchell et al. [Mitchell et al. 2007] propose a practical Non-Photorealistic Rendering method which can be used to cartoonify 3D models.

CHAPTER 3

SYSTEM DESIGN

3.1 PROJECT FLOW

This data flow diagram represents the whole flow, the project passes through. Initially, the model is being trained with the dataset fer2013.csv, that is seen in the other dataflow diagram. As the image and the filter is given as an input, the model divides the functionalities to cartoonify the image, as well as detecting the emotion, through prediction from the trained CNN model. As we get the cartoonified image, this image is again subjected to emotion detection, to draw a comparison among the original and the cartoonified images.

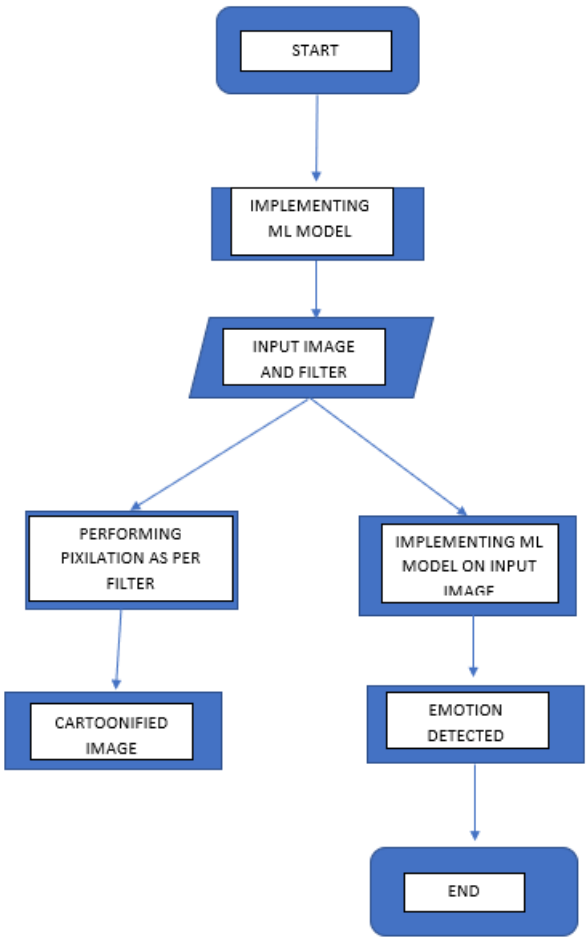

Figure 3.1

3.2 TRAINING THE MODEL

This flow chart represents training the ML model, to detect the emotion from the dataset fer2013.csv. Here, images are stored in the form of an array, which depicts the pixel location of n*n size. Using this array, the position of different facial attributes can be identified which classifies the images broadly under the categories of happy, sad, surprise, angry, neutral and disgust. The data is processed initially, to bring uniformity while training the model. Accuracy of the CNN model can be determined by testing the model against similar kind of data. At UI level, as the image is being inserted, it is subsequently being converted into an n*n pixel array, which when subjected with the ML model, can detect the emotion and printing the corresponding statement.

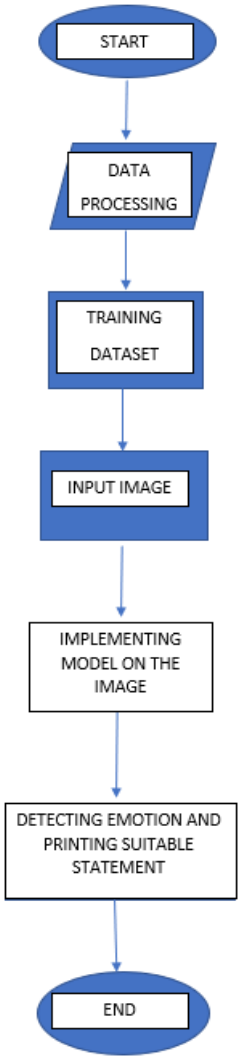

Figure 3.2

3.3 USE CASE

As the diagram depicts, 2 actors involved can be termed Admin and User. Admin maintains all system side functionalities. As the project involves training the model to detect the emotion, from a dataset, admin takes charge of carrying these tasks. An image and a filter is taken as an input from the user. Using this image, cartoonification algorithm is performed at the system side, and the emotion is detected from the image, which depends on the use case of the machine learning model developed from the dataset.

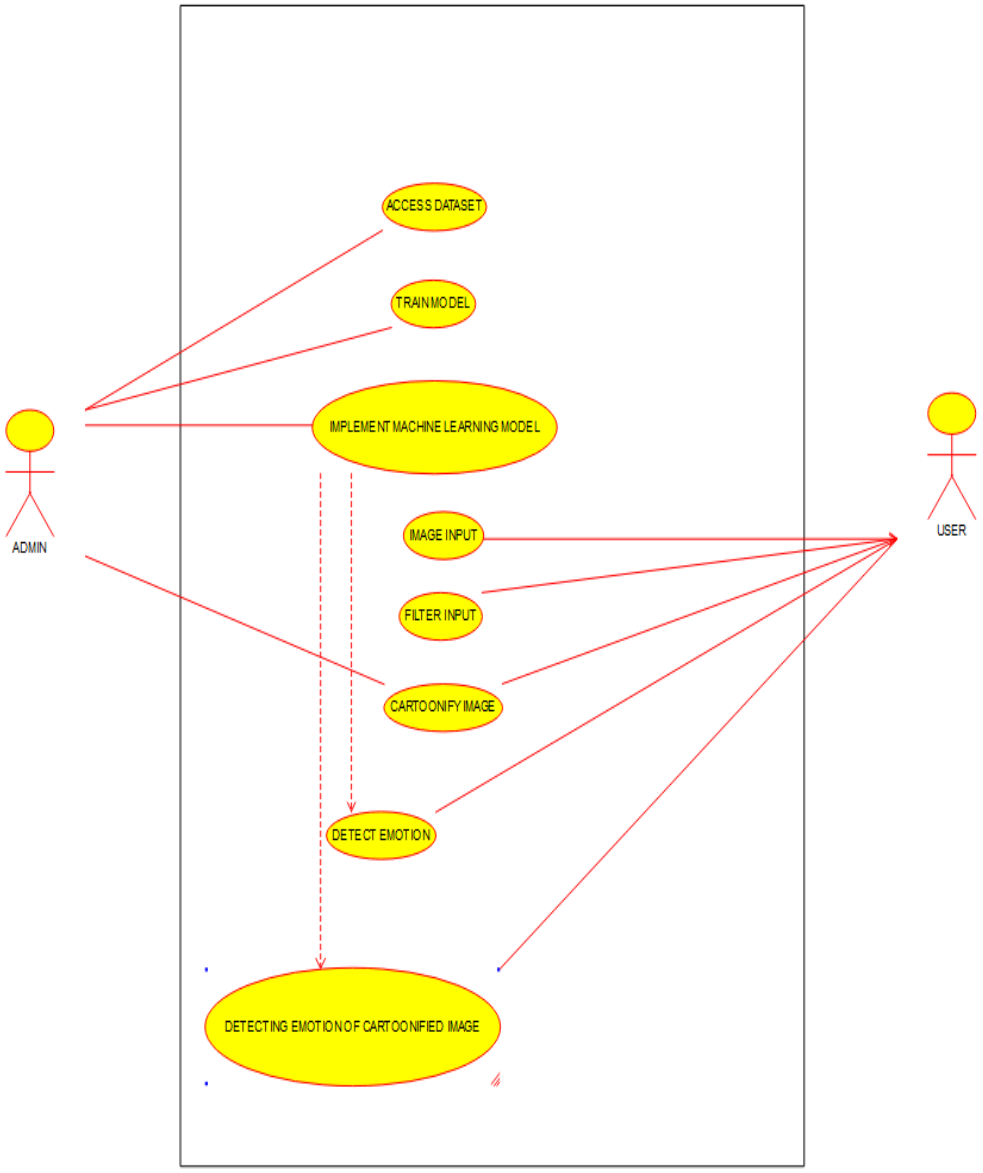

Figure 3.3

3.4 SEQUENCE

This diagram depicts the sequence of events that take place throughout the project. Initially, CNN model is trained using the dataset fer2013.csv. Model is developed with a particular accuracy, and this can be implemented to detect the emotion from the image, that is taken as an input from the user. Apart from obtaining the cartoonified image and detecting the emotion, additional function of detecting emotion from the cartoonified image also takes place.

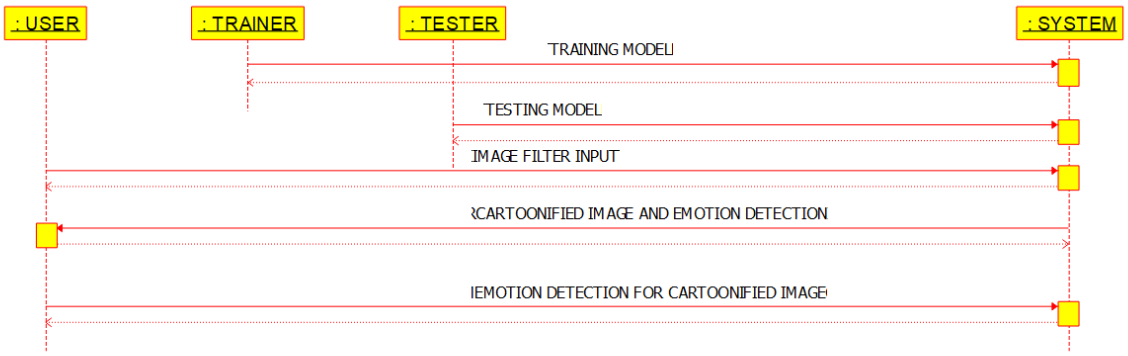

Figure 3.4

CHAPTER IV
REQUIREMENTS REVIEW

4.1 SOFTWARE REQUIREMENTS

4.1.1 PROGRAMMING LANGUAGE AND CODING TOOLS

4.1.1.1 GOOGLE COLAB

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs and TPUs. Unlike standalone Jupyter notebook, its preconfigured settings are targeted to focus on experimentation, and to lesser extent software development. This environment, based on Python Jupyter notebooks, gives the user free access to Tesla K80 GPUs. This opens up the ability of anybody to experiment with deep learning beyond simple datasets like MNIST. Being cloud based Colab allows you to write and execute Python in your browser with Zero configuration, free access to GPUs and Easy sharing. Google has also just recently opened up the free use of TPUs (Tensor Processing Units) within the environment.

Figure 4.1

4.1.1.2 PYTHON

Python is an interpreted, high-level, general-purpose programming language. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

4.1.2 SCRIPTING LANGUAGES

4.1.2.1 HTML

The HyperText Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document. HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> directly introduce content into the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

4.1.2.2. CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more

flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file which reduces complexity and repetition in the structural content as well as enabling the .css file to be cached to improve the page load speed between the pages that share the file and its formatting. Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

Figure 4.2

4.1.2.3 JAVASCRIPT

JavaScript often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. Over 97% of websites use it client-side for web page behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on the user's device. As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application

programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

4.1.2.4 jQuery

jQuery, at its core, is a Document Object Model (DOM) manipulation library. The DOM is a tree-structure representation of all the elements of a Web page. jQuery simplifies the syntax for finding, selecting, and manipulating these DOM elements. For example, jQuery can be used for finding an element in the document with a certain property (e.g. all elements with an h1 tag), changing one or more of its attributes (e.g. color, visibility), or making it respond to an event (e.g. a mouse click). jQuery also provides a paradigm for event handling that goes beyond basic DOM element selection and manipulation. The event assignment and the event callback function definition are done in a single step in a single location in the code. jQuery also aims to incorporate other highly used JavaScript functionality (e.g. fade ins and fade outs when hiding elements, animations by manipulating CSS properties). The jQuery library is typically distributed as a single JavaScript file that defines all its interfaces, including DOM, Events, and Ajax functions. It can be included within a Web page by linking to a local copy, or by linking to one of the many copies available from public servers. jQuery has a content delivery network (CDN) hosted by MaxCDN. Google in Google Hosted Libraries service and Microsoft host the library as well.

4.1.3 PACKAGES

4.1.3.1 KERAS

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the

coding necessary for writing deep neural network code. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

4.1.3.2 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.

Figure 4.3

4.1.3.3 TENSORFLOW

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications. Developers can build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

4.1.3.4 BASIC LIBRARIES

These are the basic libraries that transform Python from a general-purpose programming language into a powerful and robust tool for data analysis and visualization. They're the foundation that the more specialized tools are built on.

1. **NumPy** is the foundational library for scientific computing in Python, and many of the libraries on this list use NumPy arrays as their basic inputs and outputs. In short, NumPy introduces objects for multidimensional arrays and matrices, as well as routines that allow developers to perform advanced mathematical and statistical functions on those arrays with as little code as possible.
2. **SciKit-Learn** is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.
3. **Matplotlib** is a multi-platform data visualization library built on NumPy arrays .One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

4.2 HARDWARE REQUIREMENTS

- Processor : Any Updated Processer (Itanium series)
- Ram : Min 2 GB
- Hard Disk : Min 50 GB

CHAPTER V

IMPLEMENTATION

5.1 PREPROCESSING

- Importing the libraries and reading the CSV file.
- Dividing the data into X and Y labels, to create a dependency variable.
- Expanding features by using expand_dims() function.

Converting the image into a pixel array of size n*n that represents the positioning of facial attributes and the colour combinations of the image.

After data preprocessing:

- Number of Features: 2304(48x48)
- Number of Labels: 7
- Number of images in dataset: 35887

5.2 MODEL DEVELOPMENT

- Importing the required libraries for our CNN model.
- Initially, we declare the variables we will need for training our CNN model.
- We have 48x48-pixel resolution so we have width and height as 48. Then we have 7 emotions that we are predicting namely (0=Angry, 1=Disgust, 2=Fear, 3=Happy,4=Sad, 5=Surprise, 6=Neutral), so we have 7 labels. We will be processing our inputs with a batch size of 64.
- The dataset, fer2013.csv has been downloaded from Kaggle. Here data is represented in a favourable format, that do not require to use train_test_split() function to divide the data into train and test data. Along with X and Y labels, a new label is also used in this dataset, which is used only to distinguish the entry as training data or testing data. This validation is used to determine the accuracy of the model and to derive the efficiency and consistency with which the model can be used.

After we execute the code above, the output is-

Number of images in Training set: 32298

Number of images in Test set: 3589

5.3 DESIGNING THE CNN

This step is the most important part of the entire process as we design the CNN through which we will pass our features to train the model and eventually test it using the test features. The CNN model, that is developed is provided with ample features, which determine the prediction of the dependent variable, based upon the parameters that are provided to the model.

1. Sequential()-A sequential model is just a linear stack of layers which is putting layers on top of each other as we progress from the input layer to the output layer.
2. model.add(Conv2D())-This is a 2D Convolutional layer which performs the convolution operation as described at the beginning of this post.
3.model.add(BatchNormalization())-It performs the batch normalization operation on inputs to the next layer so that we have our inputs in a specified scale say 0 to 1 instead of being scattered all over the place.
4. model.add(MaxPooling2D())-This function performs the pooling operation on the data as explained in Literature Review. We are taking a pooling window of 2x2 with 2x2 strides in this model.
5. model.add(Dropout())-Dropout is a technique where randomly selected neurons are ignored during the training. This reduces overfitting.
6. model.add(Flatten())-This just flattens the input from ND to 1D and does not affect the batch size.
7. model.add(Dense())-According to Keras Documentation, Dense implements the operation: output = activation(dot(input, kernel)where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer. During testing, this layer is responsible for creating the final label for the image being processed. After the model.summary()function is executed, the summary representation of the model is printed.

5.4 FER DEMONSTRATION FILE

The fer2013.csv is the dataset, that is chosen for this project. Instead of the raw data, Kaggle provides the dataset for scaled quantities so that prediction of the emotion, can be determined by passing the pixel array as the parameter in the machine learning model. Here, as we have chosen a CNN model, a dense network of layers are operated together, where determining the emotion doesn't involve direct graphical prediction, but it takes the mapping of the nodes in the network into account. Considering raw data is obtained from the dataset, preprocessing must be performed, by a set of functions on the model.

Below are the functions that are used to process the data into a suitable format, as well as converting the image into a selected filter, by choosing gray-scale as the base line for all conversions.

- model_to_json()-This method is defined in keras.models. This method is responsible to load our saved CNNmodel.
- load_weights ()-This method is responsible to load all the weights from the saved CNN model.
- model.predict()-Given a trained model, predict the label of a new set of data. This method accepts one argument, and returns the learned label for each object in the array
- cv2.imshow()-Thismethod is used to display an image in a window.
- cv2.resize()-Resizing an image means by changing the dimensions of it, be it width alone, height alone or both can be performed using this method.
- destroyAllWindows()-This method is responsible to close the window and de-allocate any associated memory usage.
- cv2.adaptivethreshold()- This involves conversion of the image into smooth gray scale, and also deepens the image based upon the colour scaling depth that is used as the parameter.

cv2.bilateralFilter()-bilateralFilter can reduce unwanted noise very well while keeping edges fairly sharp. However, it is very slow compared to most filters.

Sigma values: For simplicity, you can set the 2 sigma values to be the same. If they are small, the filter will not have much effect, whereas if they are large, they will have a very strong effect, making the image look "cartoonish".

Filter size: Large filters () are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for offline applications that need heavy noise filtering.

This filter does not work inplace.

cv2.bitwise_and()-

Calculates the per-element bit-wise conjunction of two arrays or an array and a scalar.

The function cv::bitwise_and calculates the per-element bit-wise logical conjunction for:

Two arrays when src1 and src2 have the same size:

An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels():

A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels():

In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the second and third cases above, the scalar is first converted to the array type.

5.5 IMAGE PROCESSING

Image processing is the use of a digital computer to process digital images through an algorithm. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be modeled in the form of multidimensional systems. The generation and development of digital image processing are mainly affected by three factors: first, the development of computers; second, the development of mathematics (especially the creation and improvement of discrete mathematics theory); third, the demand for a wide range of applications in environment, agriculture, military, industry and medical science has increased.

In our project, we try to convert the image into a grayscale as a baseline approach. For every filter, that is to be converted, OpenCV provides a few set of functions that could smoothen, dampen or alter the intensity of the image by changing the parameters around a hexadecimal scale, which gives a multiple combination of colour scales to apply the filter on the image.

```
ReSized1 = cv2.resize(originalmage, (h,w))
grayScaleImage=cv2.cvtColor(originalmage, cv2.COLOR_BGR2GRAY)
ReSized2 = cv2.resize(grayScaleImage, (h,w))
smoothGrayScale = cv2.medianBlur(grayScaleImage, 5)
ReSized3 = cv2.resize(smoothGrayScale, (h,w))
getEdge=cv2.adaptiveThreshold(smoothGrayScale,255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 9, 9)
ReSized4 = cv2.resize(getEdge, (h,w))
colorImage = cv2.bilateralFilter(originalmage, 9, 300, 300)
ReSized5 = cv2.resize(colorImage, (h,w))
cartoonImage = cv2.bitwise_and(colorImage, colorImage, mask=getEdge)
```

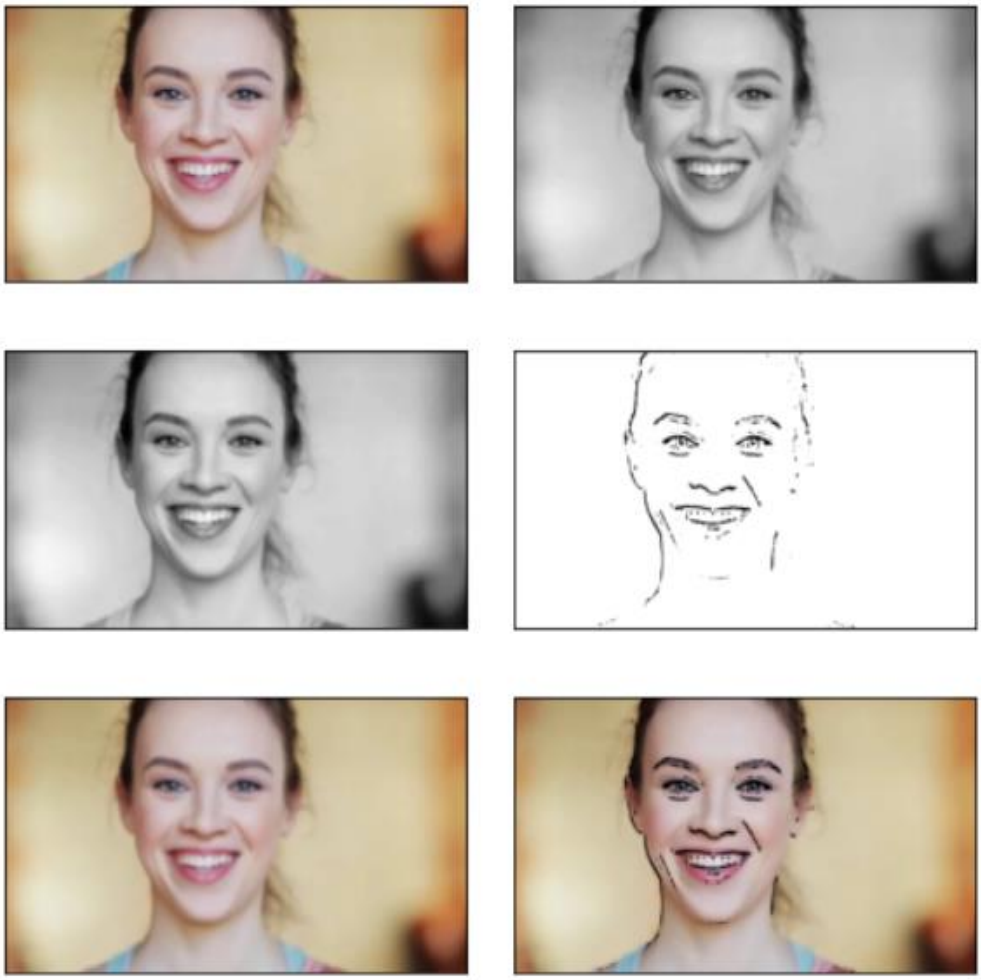

Figure 5.1

5.6 INTEGRATION AND DEPLOYMENT

The web application is supported by HTML where the processing of model and the cartoonification is performed in python. An Ajax request acts as a means of communication between the front-end and the back-end. The image that is uploaded in the HTML page is sent to the back-end through the Ajax request. After the required computations, i.e., cartoonification and emotion detection, a response is sent back to the HTML web page. In this way, the modules are integrated and work to produce the final output and display on the HTML page.

CHAPTER 6

RESULTS AND OBSERVATIONS

6.1 TRAINING MODEL

6.1.1 Summary of Model

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_11 (Conv2D)	(None, 46, 46, 64)	640
conv2d_12 (Conv2D)	(None, 46, 46, 64)	36928
batch_normalization_10 (Batc	(None, 46, 46, 64)	256
max_pooling2d_4 (MaxPooling2	(None, 23, 23, 64)	0
dropout_8 (Dropout)	(None, 23, 23, 64)	0
conv2d_13 (Conv2D)	(None, 23, 23, 128)	73856
batch_normalization_11 (Batc	(None, 23, 23, 128)	512
conv2d_14 (Conv2D)	(None, 23, 23, 128)	147584
batch_normalization_12 (Batc	(None, 23, 23, 128)	512
conv2d_15 (Conv2D)	(None, 23, 23, 128)	147584
batch_normalization_13 (Batc	(None, 23, 23, 128)	512
max_pooling2d_5 (MaxPooling2	(None, 11, 11, 128)	0
dropout_9 (Dropout)	(None, 11, 11, 128)	0
conv2d_16 (Conv2D)	(None, 11, 11, 256)	295168
batch_normalization_14 (Batc	(None, 11, 11, 256)	1024
conv2d_17 (Conv2D)	(None, 11, 11, 256)	590080
batch_normalization_15 (Batc	(None, 11, 11, 256)	1024
conv2d_18 (Conv2D)	(None, 11, 11, 256)	590080
batch_normalization_16 (Batc	(None, 11, 11, 256)	1024
max_pooling2d_6 (MaxPooling2	(None, 5, 5, 256)	0
dropout_10 (Dropout)	(None, 5, 5, 256)	0
conv2d_19 (Conv2D)	(None, 5, 5, 512)	1180160
batch_normalization_17 (Batc	(None, 5, 5, 512)	2048
conv2d_20 (Conv2D)	(None, 5, 5, 512)	2359808
batch_normalization_18 (Batc	(None, 5, 5, 512)	2048

```
conv2d_19 (Conv2D)           (None, 5, 5, 512)         1180160
_________________________________________________________________
batch_normalization_17 (Batc (None, 5, 5, 512)         2048
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 5, 5, 512)         2359808
_________________________________________________________________
batch_normalization_18 (Batc (None, 5, 5, 512)         2048
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 5, 5, 512)         2359808
_________________________________________________________________
batch_normalization_19 (Batc (None, 5, 5, 512)         2048
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
dropout_11 (Dropout)         (None, 2, 2, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_13 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_14 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_15 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_9 (Dense)              (None, 7)                 455
=================================================================
Total params: 9,014,727
Trainable params: 9,009,223
Non-trainable params: 5,504
```

6.1.2 Training Cycles (Epochs)

The following results contains epochs numbered from 1-16 and 91-100

```
None
Epoch 1/50
404/404 [==============================] - 62s 148ms/step - loss: 2.6772 - accuracy: 0.1759 - val_loss: 1.8773 - val_accuracy: 0.2489

Epoch 00001: val_loss improved from inf to 1.87734, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 2/50
404/404 [==============================] - 57s 141ms/step - loss: 1.8807 - accuracy: 0.2295 - val_loss: 1.8437 - val_accuracy: 0.2489

Epoch 00002: val_loss improved from 1.87734 to 1.84373, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 3/50
404/404 [==============================] - 57s 140ms/step - loss: 1.8487 - accuracy: 0.2477 - val_loss: 1.8190 - val_accuracy: 0.2489

Epoch 00003: val_loss improved from 1.84373 to 1.81899, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 4/50
404/404 [==============================] - 57s 140ms/step - loss: 1.8354 - accuracy: 0.2460 - val_loss: 1.8240 - val_accuracy: 0.2489

Epoch 00004: val_loss did not improve from 1.81899
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 5/50
404/404 [==============================] - 57s 140ms/step - loss: 1.8182 - accuracy: 0.2529 - val_loss: 1.8055 - val_accuracy: 0.2489

Epoch 00005: val_loss improved from 1.81899 to 1.80551, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 6/50
404/404 [==============================] - 57s 140ms/step - loss: 1.8164 - accuracy: 0.2515 - val_loss: 1.8198 - val_accuracy: 0.2489

Epoch 00006: val_loss did not improve from 1.80551
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 7/50
404/404 [==============================] - 57s 140ms/step - loss: 1.8170 - accuracy: 0.2491 - val_loss: 1.8329 - val_accuracy: 0.2489

Epoch 00007: val_loss did not improve from 1.80551
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 8/50
404/404 [==============================] - 57s 140ms/step - loss: 1.8061 - accuracy: 0.2534 - val_loss: 1.8311 - val_accuracy: 0.2495

Epoch 00008: val_loss did not improve from 1.80551
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 9/50
404/404 [==============================] - 57s 141ms/step - loss: 1.8000 - accuracy: 0.2551 - val_loss: 1.9086 - val_accuracy: 0.2489

Epoch 00009: val_loss did not improve from 1.80551
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 10/50
404/404 [==============================] - 57s 141ms/step - loss: 1.7690 - accuracy: 0.2727 - val_loss: 1.7461 - val_accuracy: 0.2816
```

```
Epoch 00010: val_loss improved from 1.80551 to 1.74605, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 11/50
404/404 [==============================] - 57s 140ms/step - loss: 1.7351 - accuracy: 0.2891 - val_loss: 1.6711 - val_accuracy: 0.3138

Epoch 00011: val_loss improved from 1.74605 to 1.67115, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 12/50
404/404 [==============================] - 57s 140ms/step - loss: 1.6821 - accuracy: 0.3159 - val_loss: 2.3156 - val_accuracy: 0.1512

Epoch 00012: val_loss did not improve from 1.67115
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 13/50
404/404 [==============================] - 57s 140ms/step - loss: 1.6259 - accuracy: 0.3387 - val_loss: 1.5944 - val_accuracy: 0.3540

Epoch 00013: val_loss improved from 1.67115 to 1.59438, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 14/50
404/404 [==============================] - 57s 140ms/step - loss: 1.5657 - accuracy: 0.3734 - val_loss: 1.6584 - val_accuracy: 0.3272

Epoch 00014: val_loss did not improve from 1.59438
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 15/50
404/404 [==============================] - 57s 140ms/step - loss: 1.5142 - accuracy: 0.4010 - val_loss: 1.4598 - val_accuracy: 0.4187

Epoch 00015: val_loss improved from 1.59438 to 1.45977, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 16/50
404/404 [==============================] - 57s 140ms/step - loss: 1.5016 - accuracy: 0.3996 - val_loss: 1.4063 - val_accuracy: 0.4198

Epoch 00016: val_loss improved from 1.45977 to 1.40627, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 17/50
404/404 [==============================] - 57s 140ms/step - loss: 1.4531 - accuracy: 0.4169 - val_loss: 1.5131 - val_accuracy: 0.3916

Epoch 00017: val_loss did not improve from 1.40627
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 18/50
404/404 [==============================] - 57s 140ms/step - loss: 1.4262 - accuracy: 0.4246 - val_loss: 1.5425 - val_accuracy: 0.3619

Epoch 00018: val_loss did not improve from 1.40627
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 19/50
404/404 [==============================] - 57s 140ms/step - loss: 1.4050 - accuracy: 0.4360 - val_loss: 1.3957 - val_accuracy: 0.4285
```

```
Epoch 00019: val_loss improved from 1.40627 to 1.39571, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 20/50
404/404 [==============================] - 57s 140ms/step - loss: 1.3804 - accuracy: 0.4432 - val_loss: 1.3413 - val_accuracy: 0.4514

Epoch 00020: val_loss improved from 1.39571 to 1.34130, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 21/50
404/404 [==============================] - 57s 140ms/step - loss: 1.3587 - accuracy: 0.4533 - val_loss: 1.4627 - val_accuracy: 0.4053

Epoch 00021: val_loss did not improve from 1.34130
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 22/50
404/404 [==============================] - 56s 140ms/step - loss: 1.3557 - accuracy: 0.4495 - val_loss: 1.3019 - val_accuracy: 0.4700

Epoch 00022: val_loss improved from 1.34130 to 1.30191, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 23/50
404/404 [==============================] - 56s 140ms/step - loss: 1.3414 - accuracy: 0.4635 - val_loss: 1.3475 - val_accuracy: 0.4542

Epoch 00023: val_loss did not improve from 1.30191
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 24/50
404/404 [==============================] - 56s 140ms/step - loss: 1.3263 - accuracy: 0.4777 - val_loss: 1.2956 - val_accuracy: 0.4915

Epoch 00024: val_loss improved from 1.30191 to 1.29558, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 25/50
404/404 [==============================] - 56s 140ms/step - loss: 1.2887 - accuracy: 0.5015 - val_loss: 1.2267 - val_accuracy: 0.5358

Epoch 00025: val_loss improved from 1.29558 to 1.22668, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 26/50
404/404 [==============================] - 56s 140ms/step - loss: 1.2669 - accuracy: 0.5189 - val_loss: 1.2851 - val_accuracy: 0.5156

Epoch 00026: val_loss did not improve from 1.22668
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 27/50
404/404 [==============================] - 56s 140ms/step - loss: 1.2480 - accuracy: 0.5223 - val_loss: 1.2688 - val_accuracy: 0.5090

Epoch 00027: val_loss did not improve from 1.22668
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 28/50
404/404 [==============================] - 56s 139ms/step - loss: 1.2329 - accuracy: 0.5282 - val_loss: 1.2008 - val_accuracy: 0.5421
```

```
Epoch 00028: val_loss improved from 1.22668 to 1.20083, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 29/50
404/404 [==============================] - 56s 140ms/step - loss: 1.2059 - accuracy: 0.5418 - val_loss: 1.2170 - val_accuracy: 0.5404

Epoch 00029: val_loss did not improve from 1.20083
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 30/50
404/404 [==============================] - 56s 140ms/step - loss: 1.1828 - accuracy: 0.5477 - val_loss: 1.1753 - val_accuracy: 0.5553

Epoch 00030: val_loss improved from 1.20083 to 1.17533, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 31/50
404/404 [==============================] - 56s 140ms/step - loss: 1.1399 - accuracy: 0.5635 - val_loss: 1.2250 - val_accuracy: 0.5376

Epoch 00031: val_loss did not improve from 1.17533
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 32/50
404/404 [==============================] - 57s 140ms/step - loss: 1.1483 - accuracy: 0.5723 - val_loss: 1.1904 - val_accuracy: 0.5584

Epoch 00032: val_loss did not improve from 1.17533
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 33/50
404/404 [==============================] - 57s 140ms/step - loss: 1.1173 - accuracy: 0.5792 - val_loss: 1.1625 - val_accuracy: 0.5683

Epoch 00033: val_loss improved from 1.17533 to 1.16253, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 34/50
404/404 [==============================] - 57s 140ms/step - loss: 1.0956 - accuracy: 0.5865 - val_loss: 1.2562 - val_accuracy: 0.5354

Epoch 00034: val_loss did not improve from 1.16253
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 35/50
404/404 [==============================] - 57s 141ms/step - loss: 1.0824 - accuracy: 0.5950 - val_loss: 1.1646 - val_accuracy: 0.5703

Epoch 00035: val_loss did not improve from 1.16253
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 36/50
404/404 [==============================] - 57s 140ms/step - loss: 1.0585 - accuracy: 0.5990 - val_loss: 1.2220 - val_accuracy: 0.5636

Epoch 00036: val_loss did not improve from 1.16253
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 37/50
404/404 [==============================] - 57s 140ms/step - loss: 1.0510 - accuracy: 0.6066 - val_loss: 1.2266 - val_accuracy: 0.5783
```

```
Epoch 00037: val_loss did not improve from 1.16253
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 38/50
404/404 [==============================] - 57s 140ms/step - loss: 1.0218 - accuracy: 0.6184 - val_loss: 1.1573 - val_accuracy: 0.5824

Epoch 00038: val_loss improved from 1.16253 to 1.15734, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 39/50
404/404 [==============================] - 57s 140ms/step - loss: 1.0133 - accuracy: 0.6284 - val_loss: 1.1513 - val_accuracy: 0.5887

Epoch 00039: val_loss improved from 1.15734 to 1.15126, saving model to /content/weights.hd5
INFO:tensorflow:Assets written to: /content/weights.hd5/assets
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 40/50
404/404 [==============================] - 57s 140ms/step - loss: 0.9897 - accuracy: 0.6379 - val_loss: 1.2308 - val_accuracy: 0.5537

Epoch 00040: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 41/50
404/404 [==============================] - 57s 140ms/step - loss: 0.9892 - accuracy: 0.6356 - val_loss: 1.2040 - val_accuracy: 0.5901

Epoch 00041: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 42/50
404/404 [==============================] - 57s 140ms/step - loss: 0.9447 - accuracy: 0.6575 - val_loss: 1.1875 - val_accuracy: 0.5803

Epoch 00042: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 43/50
404/404 [==============================] - 57s 140ms/step - loss: 0.9511 - accuracy: 0.6491 - val_loss: 1.1734 - val_accuracy: 0.5954

Epoch 00043: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 44/50
404/404 [==============================] - 57s 140ms/step - loss: 0.9054 - accuracy: 0.6759 - val_loss: 1.1842 - val_accuracy: 0.5972

Epoch 00044: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 45/50
404/404 [==============================] - 57s 140ms/step - loss: 0.8995 - accuracy: 0.6756 - val_loss: 1.1887 - val_accuracy: 0.5994

Epoch 00045: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 46/50
404/404 [==============================] - 57s 140ms/step - loss: 0.8710 - accuracy: 0.6852 - val_loss: 1.1964 - val_accuracy: 0.5960

Epoch 00046: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 47/50
404/404 [==============================] - 57s 140ms/step - loss: 0.8423 - accuracy: 0.7012 - val_loss: 1.2101 - val_accuracy: 0.5926
```

```
Epoch 00047: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 48/50
404/404 [==============================] - 57s 140ms/step - loss: 0.8378 - accuracy: 0.6966 - val_loss: 1.2321 - val_accuracy: 0.5902

Epoch 00048: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 49/50
404/404 [==============================] - 57s 140ms/step - loss: 0.8098 - accuracy: 0.7097 - val_loss: 1.2323 - val_accuracy: 0.6077

Epoch 00049: val_loss did not improve from 1.15126
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 50/50
404/404 [==============================] - 56s 140ms/step - loss: 0.8067 - accuracy: 0.7107 - val_loss: 1.2757 - val_accuracy: 0.6057
```

For training, dataset comprised of 32298 images is provided for model, from which 20% is again split for validation. So 25838 images are used for training the model and 6460 images are used for cross validation. The above outputs describe training accuracy of 92.86% and validation accuracy of 60.57%. Complete output can be viewed in appendix.

6.2 CARTOONIFYING

```
ReSized1 = cv2.resize(originalmage, (h,w))

ReSized2 = cv2.resize(grayScaleImage, (h,w))

ReSized3 = cv2.resize(smoothGrayScale, (h,w))

ReSized4 = cv2.resize(getEdge, (h,w))

ReSized5 = cv2.resize(colorImage, (h,w))
```

6.3 TESTCASES

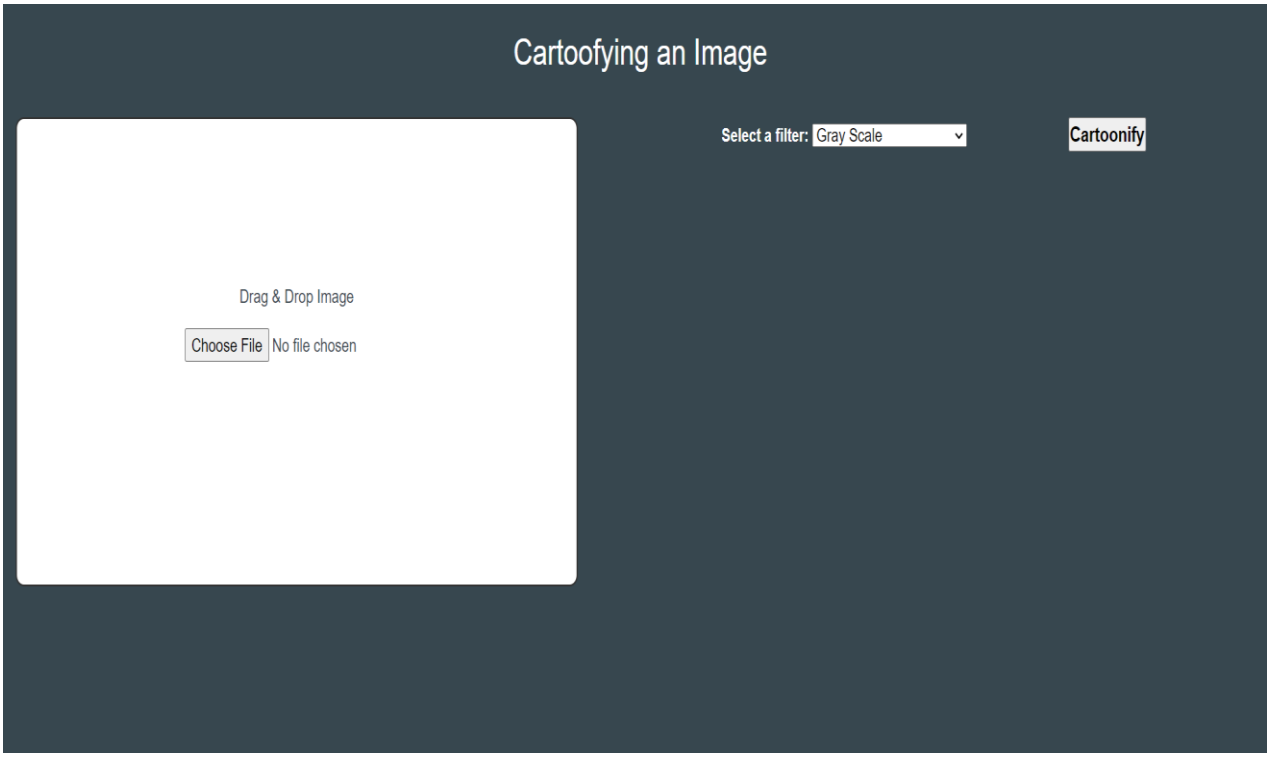

Figure 6.1

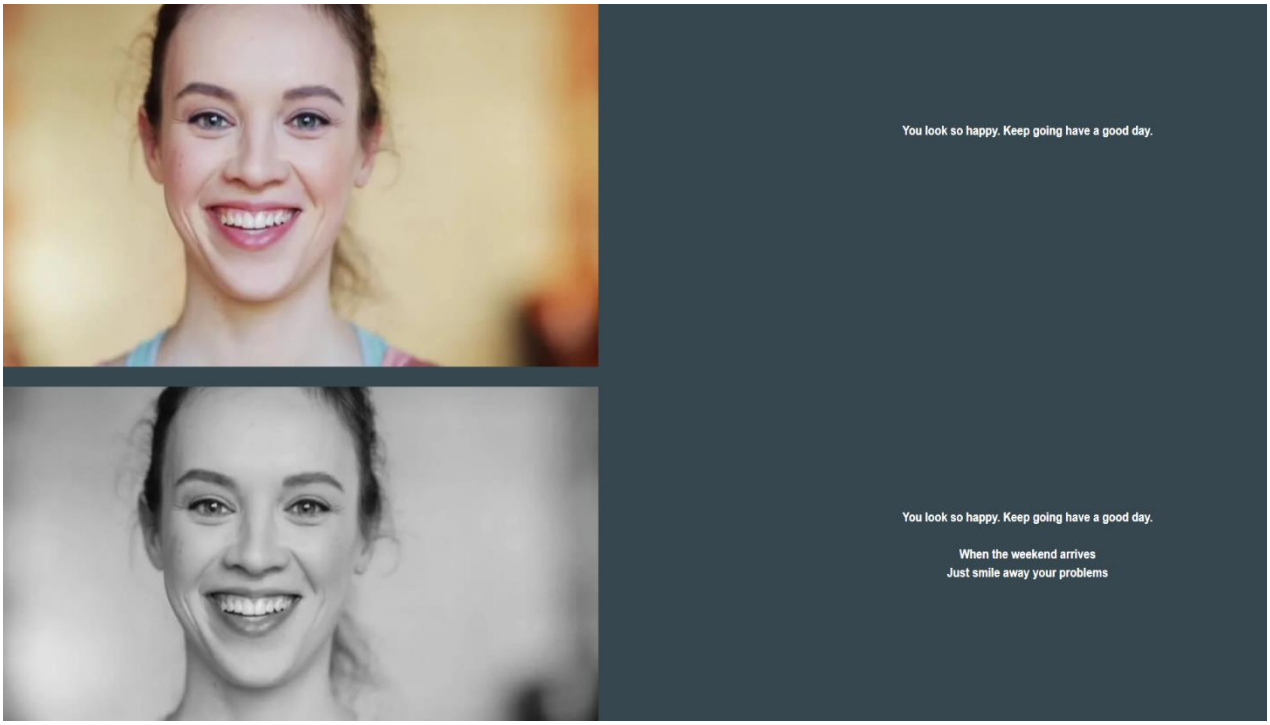

Figure 6.2

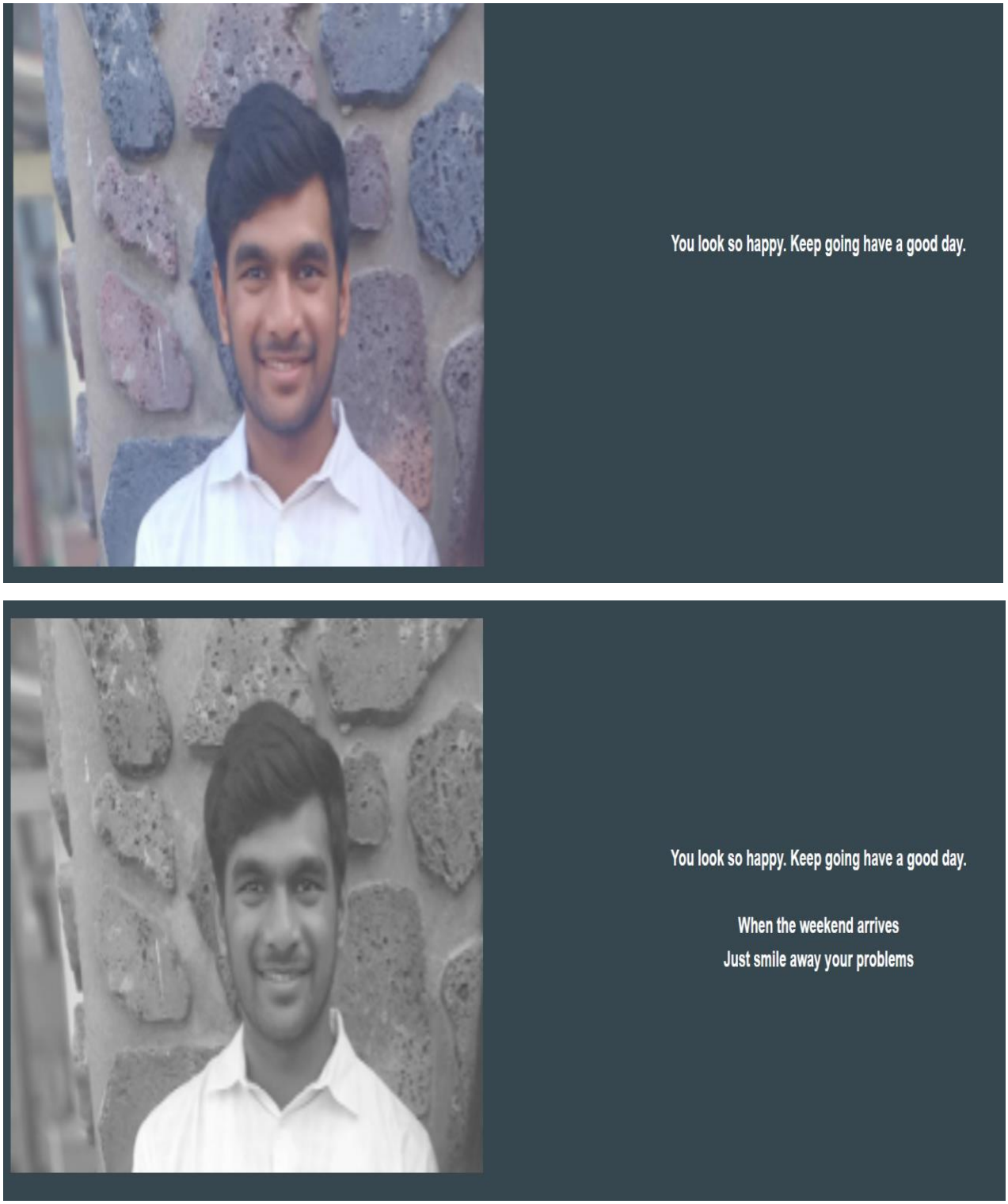
You look so happy. Keep going have a good day.
You look so happy. Keep going have a good day.
When the weekend arrives
Just smile away your problems

Figure 6.3


You look angry and frustrated. Please calm down
You look angry and frustrated. Please calm down
Quite an angry look. Peace out
The HULK look

Figure 6.4

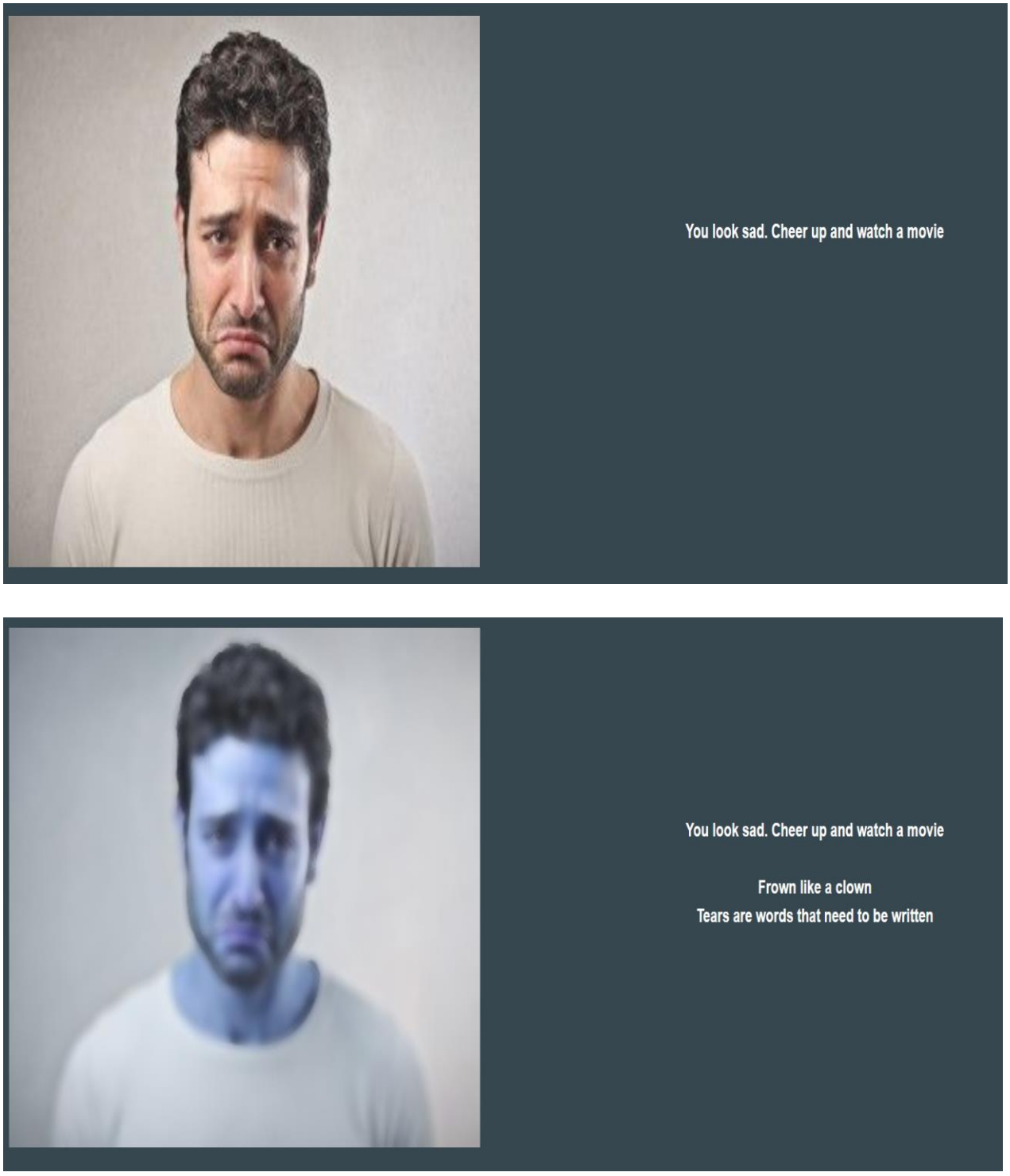
You look sad. Cheer up and watch a movie
You look sad. Cheer up and watch a movie
Frown like a clown
Tears are words that need to be written

Figure 6.5

CHAPTER 7

CONCLUSION

The idea to build this application arose from the privacy concerns of people in the recent times. We have built an application which performs the dual functioning of cartoonifying an image, as well as detecting the emotion in the image. Both the functions give accurate result and this application proves to be helpful among various domains, i.e., to post the images on social media, or share images online without security and privacy concerns, expression detection among the individuals. The cartoonifying of an image also involves edge detection and image filtering to achieve multiple filters. Moreover, the colour quantization can also be used for this purpose. Nowadays, stealing the data of the users and selling them to various advertising organizations or misusing the data that has been shared over the internet has become a major issue in the world of internet. Photos can be misused for identity theft or unlocking stolen smartphones where the face unlock mechanism was poorly implemented. This application can therefore help the users by reducing the risk of their pictures being misused. The animated images can also be used in chats, to express one's feelings in pictures without having to share the actual image. This objective was to develop a facial expression recognition system implementing the computer visions and enhancing the advanced feature extraction and classification in face expression recognition. Experiment results on the FER dataset show that our proposed method gives a good accuracy. Facial expression recognition is a very challenging problem. More efforts should be made to improve the classification performance for important applications.

FUTURE ENHANCEMENTS

Our future work will focus on improving the performance of the system and deriving more appropriate classifications which may be useful in many real world applications. The project can also be expanded using various other colour combinations for more filters and the performance of the emotion detection system can be improved with a much larger dataset.

REFERENCES

1. ToonNet: A cartoon image dataset and a DNN-based semantic classification system by Yanqing Zhou, Yongxu Jin,
2. M. Mohammadpour, H. Khaliliardali, S. M. R. Hashemi and M. M. AlyanNezhadi, "Facial emotion recognition using deep convolutional networks," 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, 2017, pp. 0017-0021, doi: 10.1109/KBEI.2017.8324974.
3. MalyalaDivya, R Obula Konda Reddy, C Raghavendra “Effective Facial Emotion Recognition using Convolutional Neural Network Algorithm,” International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-4, November 2019
4. Ruiz-Garcia A., Elshaw M., Altahhan A., Palade V. (2016) Deep Learning for Emotion Recognition in Faces. In: Villa A., Masulli P., Pons Rivero A. (eds) Artificial Neural Networks and Machine Learning – ICANN 2016. ICANN 2016. Lecture Notes in Computer Science, vol 9887. Springer, Cham.
5. Yijun Gan. 2018. Facial Expression Recognition Using Convolutional Neural Network. In Proceedings of the 2nd International Conference on Vision, Image and Signal Processing (ICVISP 2018). Association for Computing Machinery, New York, USA, Article 29, 1–5. DOI:https://doi.org/10.1145/3271553.3271584
6. D. V. Sang, N. Van Dat and D. P. Thuan, "Facial expression recognition using deep convolutional neural networks," 2017 9th International Conference on Knowledge and Systems Engineering (KSE), Hue, 2017, pp. 130-135, doi: 10.1109/KSE.2017.8119447.
7. Rajesh Kumar G A, Ravi Kant Kumar, Goutam Sanyal, “Facial Emotion Analysis using Deep Convolution Neural Network”, International Conference on Signal Processing and Communication (ICSPC’17) – 28th & 29th July 2017.
8. A. Mollahosseini, D. Chan, and M. H. Mahoor, “Going deeper in facial expression recognition using deep neural networks,” in Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on. IEEE, 2016, pp. 1– 10.

9. Sang, Dinh & Dat, Nguyen & Thuan, Do. (2017). Facial expression recognition using deep convolutional neural networks. 130-135. 10.1109/KSE.2017.8119447.

10. Learning to Cartoonize Using White-Box Cartoon Representations CVPR 2020 · Xinrui Wang, Jinze Yu
11. Cartoonifying of an Image by G. Jahnavi, P. Anusha, Ch. Sravani, T. Venkateshwarlu
12. Cartoonifying an Image using OpenCV and Python by Vaishali Sudarshan and Amritesh Singh, Jain University, Karnataka.
13. Arushi Raghuvanshi, Vivek Choksi, Facial Expression Recognition with Convolutional Neural Networks, Stanford University, 2016.
14. Abir Fathallah, Lotfi Abdi, and Ali Douik, Facial Expression Recognition via Deep Learning, IEEE/ACS, 2017.

APPENDIX

1. SOURCE CODE

1.1 Training the model

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.models import Sequential #Initialise neural network model as a sequential network
from keras.layers import Conv2D #Convolution operation
from keras.layers.normalization import BatchNormalization
from keras.regularizers import l2
from keras.layers import Activation #Applies activation function
from keras.layers import Dropout#Prevents overfitting by randomly converting few outputs to zero
from keras.layers import MaxPooling2D # Maxpooling function
from keras.layers import Flatten # Converting 2D arrays into a 1D linear vector
from keras.layers import Dense # Regular fully connected neural network
from keras import optimizers
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, TensorBoard, ModelCheckpoint
from sklearn.metrics import accuracy_score

def load_data(dataset_path):
 data = []
 test_data = []
 test_labels = []
 labels =[]
 with open(dataset_path, 'r') as file:
  for line_no, line in enumerate(file.readlines()):
    if 0 < line_no <= 35887:
      curr_class, line, set_type = line.split(',')
      image_data = np.asarray([int(x) for x in line.split()]).reshape(48, 48)
      image_data =image_data.astype(np.uint8)/255.0
      if (set_type.strip() == 'PrivateTest'):
        test_data.append(image_data)
        test_labels.append(curr_class)
      else:
        data.append(image_data)
        labels.append(curr_class)

  test_data = np.expand_dims(test_data, -1)
  test_labels = to_categorical(test_labels, num_classes = 7)
  data = np.expand_dims(data, -1)
  labels = to_categorical(labels, num_classes = 7)

  return np.array(data), np.array(labels), np.array(test_data), np.array(test_labels)
dataset_path = "/content/fer2013.csv"
train_data, train_labels, test_data, test_labels = load_data(dataset_path)
print("Number of images in Training set:", len(train_data))
print("Number of images in Test set:", len(test_data))
```

```
epochs = 50
batch_size = 64
learning_rate = 0.001
model = Sequential()
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(48, 48,1), kernel_regularizer=l2(0.01)))
model.add(Conv2D(64, (3, 3), padding='same',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
adam = optimizers.Adam(lr = learning_rate)
model.compile(optimizer = adam, loss = 'categorical_crossentropy', metrics = ['accuracy'])
print(model.summary())
lr_reducer = ReduceLROnPlateau(monitor='val_loss', factor=0.95, patience=3)
early_stopper = EarlyStopping(monitor='val_acc', min_delta=0, patience=6, mode='auto')
checkpointer = ModelCheckpoint('/content/weights.hd5', monitor='val_loss', verbose=1, save_best_only=True)
model.fit(train_data,train_labels,epochs=epochs,batch_size=batch_size,validation_split=0.2,shuffle=True,
callbacks=[lr_reducer,checkpointer,early_stopper])

from keras.models import model_from_json
model_json = model.to_json()
with open("/content/model.json","w") as json_file:
 json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("/content/model.h5")
print("Saved model to disk")
```

1.2 WEB APPLICATION

1.2.1 FRONT-END

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script type="text/javascript" src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.5.1.js"></script>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>
    <link href="https://fonts.googleapis.com/css2?family=Libre+Baskerville&
    family=Sora:wght@700&family=Ubuntu:ital,wght@1,500&display=swap" rel="stylesheet">
    <title>Home</title>
</head>
<body>
    <h2 align="center">Cartoofying an Image</h2><br>
        <form method="POST" action="http://127.0.0.1:5000/" enctype = "multipart/form-data">
        <div class="cont">
    <div id="userActions">
                <br><br><br><br><br><br>
        <p>Drag &amp;amp; Drop Image</p>
        <input type="file" id="fileUpload" name="fileupload"/>
        </div>
        <div style="margin-left: 13%;">

        <label for="filters" style="color : white"; > <b>Select a filter:</b></label>
        <select id="filter" name="filter">
                <option value="1">Gray Scale</option>
                <option value="2">Sharpened Gray Scale</option>
                <!-- <option value="3">Sketchy</option> -->
                <option value="4">UV Filter</option>
                <option value="5">Water Colored</option>
        </select>
        <input type="submit" value="Cartoonify" style="margin-left: 18%; font-weight:bolder; font-size: large; ">
    </div>
<div></div>
        <br>
```

```
        <div id="cimage">
        </div>
    <div id="em1" name="em1"></div>
        <!-- <div></div> -->

</div>
        </form>

</body>
<style>
    *{ margin: 0; padding: 0; }
body{
        background-color: #37474F;
        font-family: Arial, sans-serif;
        padding: 15px;
}
h2{
    margin: 1% 8%;
    color: #FFFFFF;
}
.error{ color: #B71C1C; }
#userActions{
        display: table-cell;
        height: 55vh;
        width: 90%;
        vertical-align: middle;
        text-align: center;
        color: #37474F;
        background-color: #FFFFFF;
        border: solid 2px #333333;
        border-radius: 10px;
}
/*.formclass{
        position: relative;

}*/
/* #cimage{
        margin-top: 350px;
} */
.cont{
         display: grid;
         grid-template-columns: 50% 50%;
}
#userActions input{
         width: 80%;
         margin-left:30%;
}
#emtn{
         font-weight: bolder;
         font-size: larger;
         margin-top: 190px;
         margin-left: 25%;
         color: white;
         text-align: center;
}
#imgPrime{
 width: 95%;
 height: 95%;
 /*position: relative;
 margin-bottom: 85%;
 margin-left: 0%;
 display: none;*/

}
button{
     margin-top: 10px;
     margin-left: 44%;
     padding: 3px;
     border-radius: 5px;
     background-color: rgb(177, 247, 212);
     color: rgb(13, 13, 14);
     font-size: large;
     font-weight: 600;
}
.signupbtn:hover{
     font-size: larger;
         border-radius: 12%;
}
</style>
<script>
```

```
<script>
    'use strict';
        $(document).ready(function(){
        console.log("Hi");
    jQuery("form").on("submit", function(e){
                console.log("Hi");
    e.preventDefault();
    jQuery.ajax({
            type: 'POST',
            url: jQuery(this).attr('action'),
            enctype: 'multipart/form-data',
            data: new FormData(this),
            processData: false,
            contentType: false,
            success: function (data) {
                jQuery('#cimage').append('<img id="imgPrime" src="'+data[0]+'" alt="Image placeholder" />');
                                jQuery('#em').append('<p id="emtn">'+data[1]+'</p>');
                                jQuery('#em1').append('<p id="emtn">'+data[2]+'</p>');
                                jQuery('#oimage').append('<img id="imgPrime" src="'+data[3]+'" alt="Image placeholder" />');

            }
            }
    );
});
});
/**
// |||||||||||||||||||||||||||||| \\
//     Global Object $: Generic controls
// |||||||||||||||||||||||||||||| \\
**/
(function(){
        // http://stackoverflow.com/questions/4083351/what-does-jquery-fn-mean
        var $ = function( elem ){
                if (!(this instanceof $)){
      return new $(elem);
                }
                this.el = document.getElementById( elem );
        };
        window.$ = $;
        $.prototype = {
                onChange : function( callback ){
                        this.el.addEventListener('change', callback );
                        return this;
                }
        };
})();

/**
// |||||||||||||||||||||||||||||| \\
//      Drag and Drop code for Upload
// |||||||||||||||||||||||||||||| \\
**/
var dragdrop = {
        init : function( elem ){
                elem.setAttribute('ondrop', 'dragdrop.drop(event)');
                elem.setAttribute('ondragover', 'dragdrop.drag(event)' );
        },
        drop : function(e){
                e.preventDefault();
                var file = e.dataTransfer.files[0];
                runUpload( file );
        },
        drag : function(e){
                e.preventDefault();
        }
};
```

```
window.onload = function(){
	if( window.FileReader ){
		// Connect the DIV surrounding the file upload to HTML5 drag and drop calls
		dragdrop.init( $('userActions').el );
		//	Bind the input[type="file"] to the function runUpload()
		$('fileUpload').onChange(function(){ runUpload( this.files[0] ); });
	}else{
		// Report error message if FileReader is unavilable
		var p	= document.createElement( 'p' ),
				msg = document.createTextNode( 'Sorry, your browser does not support FileReader.' );
		p.className = 'error';
		p.appendChild( msg );
		$('userActions').el.innerHTML = '';
		$('userActions').el.appendChild( p );
	}
};

</script>
</html>
```

1.2.2 BACK-END

```
from flask import Flask, render_template,request,json
from werkzeug.utils import secure_filename
import requests
import cv2 #for image processing
import numpy as np #to store image
import sys
import matplotlib.pyplot as plt
import os
import random
import numpy as np
from keras.models import model_from_json
from keras.preprocessing import image

app = Flask(__name__)

#gpath=''
#em1=''
def save(imagex, ImagePath):
    newName="cartoonified_Image_1"
    path1 = os.path.dirname(ImagePath)
    extension=os.path.splitext(ImagePath)[1]
    path = os.path.join(path1, newName+extension)
    #gpath=path
    cv2.imwrite(path, imagex)
    em1=c_emotion(path)
    # print(em1)
    return path,em1
```

```
def cartoonify(ImagePath,filter):
    originalmage = cv2.imread(ImagePath)
    originalmage = cv2.cvtColor(originalmage, cv2.COLOR_BGR2RGB)
    w,h,o=originalmage.shape

    if originalmage is None:
        print("Can not find any image. Choose appropriate file")
        sys.exit()

    ReSized1 = cv2.resize(originalmage, (h,w))

    grayScaleImage= cv2.cvtColor(originalmage, cv2.COLOR_BGR2GRAY)
    ReSized2 = cv2.resize(grayScaleImage, (h,w))

    smoothGrayScale = cv2.medianBlur(grayScaleImage, 5)
    ReSized3 = cv2.resize(smoothGrayScale, (h,w))

    getEdge = cv2.adaptiveThreshold(smoothGrayScale, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
    cv2.THRESH_BINARY, 9, 9)

    ReSized4 = cv2.resize(getEdge, (h,w))

    colorImage = cv2.bilateralFilter(originalmage, 9, 300, 300)
    ReSized5 = cv2.resize(colorImage, (h,w))

    cartoonImage = cv2.bitwise_and(colorImage, colorImage, mask=getEdge)

    ReSized6 = cv2.resize(cartoonImage, (h,w))
    images=[ReSized1, ReSized2, ReSized3, ReSized4, ReSized5, ReSized6]
    fig, axes = plt.subplots(1,2, figsize=(8,8), subplot_kw={'xticks':[], 'yticks':[]},
    gridspec_kw=dict(hspace=0.1, wspace=0.1))
    for i, ax in enumerate(axes.flat):
        if(i==0):
          ax.imshow(images[i], cmap='gray')
        else:
          ax.imshow(images[filter], cmap='gray')
    name,em1=save(images[filter],ImagePath)
    return name,em1
```

```
def emotion(ImagePath):
    model = model_from_json(open('model.json',"r").read())
    #load weights
    model.load_weights('model.h5')
    face_haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    test_img=cv2.imread(ImagePath)
    gray_img= cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)
    faces_detected = face_haar_cascade.detectMultiScale(gray_img,1.32, 5)
    img_pixels=[0]
    for (x,y,w,h) in faces_detected:
        cv2.rectangle(test_img,(x,y),(x+w,y+h),(255,0,0),thickness=7)
        roi_gray=gray_img[y:y+w,x:x+h]
        roi_gray=cv2.resize(roi_gray,(48,48))
        img_pixels = image.img_to_array(roi_gray)
        img_pixels = np.expand_dims(img_pixels, axis = 0)
        img_pixels /= 255
    # print(img_pixels)
    if(img_pixels.any()==0):
        return "Not Found"
    a=random.randint(0,2)
    predictions = model.predict(img_pixels)
    max_index = np.argmax(predictions[0])
    angry=('You look angry and frustrated. Please calm down','Angry 2')
    disgust=('cap1','cap2')
    fear=('cap1','cap2')
    happy=('cap1','cap2')
    sad=('cap1','cap2')
    surprise=('cap1','cap2')
    neutral=('cap1','cap2')

    # emotions= (angry, disgust, fear, happy, sad, surprise, neutral)
    emotions = ('You look angry and frustrated. Please calm down',
    'A look of disgust. Seems like you need to go to your happy place.',
    'You look frightened. Everything is going to be okay',
    'You look so happy. Keep going have a good day.',
    'You look sad. Cheer up and watch a movie',
    'Looks like someone has got a big surprise',
    'You seem pretty calm without any emotions right now.')
    predicted_emotion = emotions[max_index]
    # predicted_emotion=predict[a]
    return predicted_emotion
```

```
def c_emotion(ImagePath):
    model = model_from_json(open('model.json',"r").read())
    #load weights
    model.load_weights('model.h5')
    face_haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    test_img=cv2.imread(ImagePath)
    gray_img= cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)
    faces_detected = face_haar_cascade.detectMultiScale(gray_img,1.32, 5)
    img_pixels=[0]
    for (x,y,w,h) in faces_detected:
        cv2.rectangle(test_img,(x,y),(x+w,y+h),(255,0,0),thickness=7)
        roi_gray=gray_img[y:y+w,x:x+h]
        roi_gray=cv2.resize(roi_gray,(48,48))
        img_pixels = image.img_to_array(roi_gray)
        img_pixels = np.expand_dims(img_pixels, axis = 0)
        img_pixels /= 255
    # print(img_pixels)
    if(img_pixels.any()==0):
        return "Not Found"
    a=random.randint(0,2)
    predictions = model.predict(img_pixels)
    max_index = np.argmax(predictions[0])

    # emotions= (angry, disgust, fear, happy, sad, surprise, neutral)
    emotions = ('<p id="emtn">You look angry and frustrated. Please calm down<br>
    <br>Quite an angry look. Peace out<br>The HULK look</p>',
    '<p id="emtn">A look of disgust. Seems like you need to go to your happy place.<br>
    <br>My face when you make bad jokes<br> Ewwwwwwwww! </p>',
    '<p id="emtn">You look frightened. Everything is going to be okay<br><br>The time I
    saw Conjuring alone<br>Fear of growing up</p>',
    '<p id="emtn">You look so happy. Keep going have a good day.<br><br>When the weekend
    arrives<br>Just smile away your problems</p>',
    '<p id="emtn">You look sad. Cheer up and watch a movie<br><br>Frown like a clown<br>
    Tears are words that need to be written</p>',
    '<p id="emtn">Looks like someone has got a big surprise<br><br>When I see an empty
    box on the street<br>The Michael Scott look</p>',
    '<p id="emtn">You seem pretty calm without any emotions right now.<br><br>Calm before
    the storm<br>Peace out like BUDDHA</p>')
    predicted_emotion = emotions[max_index]
    # predicted_emotion=predict[a]
    return predicted_emotion
```

```
@app.route("/", methods = ["GET", "POST"])
def new():
    if request.method == "POST":
        f = request.files['fileupload']
        f.save(secure_filename(f.filename))
        filter=int(request.values.get("filter"))
        ImagePath="./"+str(f.filename)
        nam,em1=cartoonify(ImagePath,filter)
        em=emotion(ImagePath)
        #em1=emotion(gpath)
        di=dict()
        di[0]=nam
        di[1]=em
        di[2]=em1
        di[3]=ImagePath
        print(di)
        return di
    return render_template("index.html")



@app.after_request
def after_request(response):
    header = response.headers
    header['Access-Control-Allow-Origin'] = '*'
    header['Access-Control-Allow-Headers'] = 'Content-Type, Authorization'
    header['Access-Control-Allow-Methods'] = 'OPTIONS, HEAD, GET, POST, DELETE, PUT'
    return response

if __name__ == "__main__":
    app.run(debug=True)
```