1. INTRODUCTION

Image filters are primarily used to edit an image using computer software. An image filter generally changes the image at the pixel level, meaning each pixel individually is affected. It can be applied to 2-D and 3-D images. Typically, the image filter process includes options such as:

- Editing the colour scheme/theme/contrast of the image
- Adjusting image brightness
- Adding effects to the image
- Changing the texture

But in this system, there is no need of software for editing an image. It is automatic colorization ad hence converts very fast. Given a grayscale photograph as input, this attacks the problem of hallucinating a plausible color version of the photograph. This problem is clearly under constrained, so previous approaches have either relied on significant user interaction or resulted in desaturated colorizations. This propose a fully automatic approach that produces vibrant and realistic colorizations. We embrace the underlying uncertainty of the problem by posing it as a classification task and use class-rebalancing at training time to increase the diversity of colors in the result. The system is implemented as a feed-forward pass in a CNN (Convolution Neural Network) at test time and is trained on over a million color images. Moreover, this can be shown that colorization can be a powerful pretext task for self-supervised feature learning, acting as a cross-channel encoder. This approach results in state-of-the-art performance on several feature learning benchmarks. This approach consists 2 main systems namely existing system, proposed system. Existing system says about the drawbacks and proposed system is the newest system which we are undergoing currently.

1.1 Existing System:

Image manipulations are done by few applications like photoshop. In such applications, it requires user to assign colours to some regions and extends such information to the whole image. For every pixel of the photo, the user needs to change the colours in their own way. Many colours are assigned to it and with the filters of different shades that may be with or without high intensity. This may lead to time consumption and vary in the colour format.

Disadvantages:

- Time consumption.
- Colorizing each part of the image.
- Assigning a user to convert the grayscale image to coloured.

1.2 Proposed System:

Image colorization is the process of assigning colors to a grayscale image to make it more aesthetically appealing and perceptually meaningful. This is known to be a sophisticated task that often requires prior knowledge about the image content and manual adjustments in order to achieve artefact-free quality.

The approach for this is to learn the colour of each pixel from a colour image with similar content. We extract the information about colour from an image and transfer it to another image. Recently, deep learning has gained increasing attention among researchers in the field of computer vision and image processing. As a typical technique, convolutional neural networks (CNNs) have been well-studied and successfully applied to several tasks such as image recognition, image reconstruction, image generation, etc. A CNN consists of multiple layers of small computational units that only process portions of the input image in a feed-forward fashion. Each layer is the result of applying various image filters, each of which extracts a certain feature of the input image, to the previous layer. Thus, each

layer may contain useful information about the input image at different levels of abstraction.

Advantages:

- Easy conversion from grayscale to coloured.
- Adding filters without user involvement.
- Webcam and video manipulations.

1.3 Software and Hardware Requirements

Software Requirements:

This software requirements refers to the server-side and user-side required software specifications so that the module can be run with Image and Video Revising.

Software tools used:

- Python3
- OpenCV
- Deep Learning

Hardware Requirements:

This hardware Constraints refers to the server-side and user-side required hardware specifications so that the module can be run with System.

- Operating System: Windows 10 Home
- Processor: intel i5 7^{th} Gen

- RAM: Minimum 1GB
- Hard Disk: Minimum 32GB
- Sound card/Speakers
- Webcam

2. PHOTOSHOP

Photoshop can edit only photos with the interference of user. It cannot edit the images automatically. It requires user to assign colors to some regions and extends such information to the whole image. For every pixel of the photo, the user needs to change the colors in their own way. Many colors are assigned to it and with the filters of different shades that may be with or without high intensity. This may lead to time consumption and vary in the color format. Photoshop is also expensive. Progress monitoring facility is not present in fewer tools. Beginners may find the interface difficult to take in one's stride. Someone doing minor photo editing could likely find the functions he or she needs in a cheaper and more basic program. Photoshop cannot edit the videos and webcam. There are chances of harming your personal computer. Since video cannot be filtered and manipulated, to overcome this situation image and video revising is introduced. The following are the features in Adobe Photoshop.

- The latest photoshop is Adobe. Adobe Photoshop is not free and it has a secret source code. Source code is written by the programmer. Adobe Photoshop is relatively expensive, when compared with other photo-editing software.
- It has so many features and it is thus regarded as the industry standard in the graphic designing. You will spend much more for Adobe Photoshop. If you do not plan on editing the photos every day.
- Most beginners will not begin to touch the functionality of this program and they could get what they need from another program. Many people who are unfamiliar with editing the photos in Adobe Photoshop. The advanced functionality will be useless for them.
- There are many free and lower cost graphics and photo editors. They are available for the casual users and the enthusiasts who do not need all of the powerful features Photoshop offers.

- Adobe Photoshop requires a well-equipped computer. It is a heavyweight program and many of the tools are computationally intensive. It takes up a large space of room on your computer as well as RAM memory. Some of the tools in Adobe does not show progress bars. Some of your tools just don't have the progress bars that you may be used to seeing with similar programs. This creates a little more back and forth with your design sometimes to make sure that you've completed the steps necessary to get the job done.
- There are a lot of different tools that do very different things, so it takes time and patience with yourself in order to learn how to use each one correctly. Photoshop also takes up a large amount of room on your computer, by getting a memory for our computer.

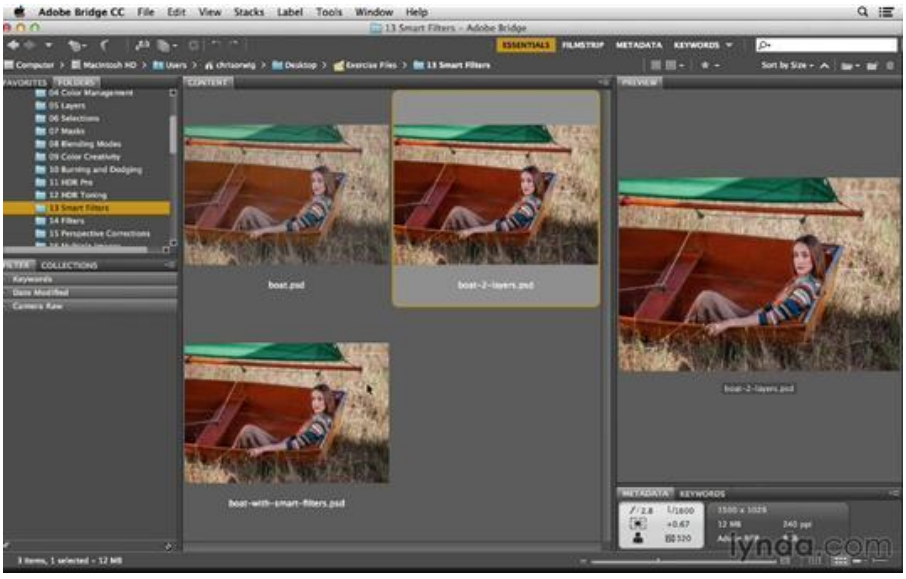

Figure 2.1 Adobe Photoshop

3. IMAGE AND VIDEO REVISING

Image and Video Revising is manipulating the images and video by colours and filters. To overcome the drawbacks of photoshop, this came into existence. It is an automatic colorization and filters which is written in code. It divides the pixel intensities from 0 to 1. Image colorization is the process of assigning colors to a grayscale image to make it more aesthetically appealing and perceptually meaningful. It is then converted from RGB (Red Green Blue) color to Lab color. Lab color encodes the lightness and the RGB colors. This makes the vision of photo and video with greater intensity. Image and Video colorizations makes the images, video, live webcam to lighten itself. In the live webcam and videos, if any dim light encounters, this procedure converts it into brighter by adding colors. This is known to be a sophisticated task that often requires prior knowledge about the image content and manual adjustments in order to achieve artefact-free quality.

Image and Video filters are used for adding filters to the pictures and live videos. This was before an application on mobile phone named Snapchat. But this is used on the computer which was not done before. It is a direct process of adding filters on a pic with the user choice.

The packages used for this colorization are:

1. **Numpy**: NumPy is a Python package which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc. NumPy array can also be used as an efficient multi-dimensional container for generic data. Now, let me tell you what exactly is a python numpy array.

2. **argparse**: Python argparse module is the preferred way to parse command line arguments. Parsing command-line arguments is a very common task, which Python scripts do and behave according to the passed values. Python argparse is the recommended

command-line argument parsing module in Python. It is very common to the getopt module but that is a little complicated and usually need more code for the same task.

3. **imutils**: A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and both Python 2.7 and Python 3. To install Python imutils, type "pip install imutils" in command prompt.

4. **time**: Python has a module named time to handle time-related tasks. To handle the time function type "import time" in the program. Time package mainly operates the webcam. It sets the webcam timing from 1 second to thousand seconds. Thus, it helps for the stable video operations to be dealt.

5. **OpenCV**: OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general-purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation. OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib. We import the package as "import cv2".

6. **Python 3:** Python is a general purpose and high-level programming language. You can use Python for developing desktop GUI applications, websites and web applications. Also, Python, as a high-level programming language, allows you to focus on core functionality of the application by taking care of common programming tasks. it can be used to build just about anything, which will be made easy with the right tools/libraries. Professionally, Python is great for backend web development, data analysis, artificial intelligence, and scientific computing. develop different applications like web applications, graphic user

interface-based applications, software development application, scientific and numeric applications, network programming, Games and 3D applications and other business applications.

7. **Command Prompt**: A command prompt is used in a text-based or "command-line" interface, such as a Unix terminal or a DOS shell. It is a symbol or series of characters at the beginning of a line that indicates the system is ready to receive input. It other words, it prompts the user for a command.

8. **Deep Learning**: Deep learning is a sub-field of machine learning dealing with algorithms inspired by the structure and function of the brain called artificial neural networks. In other words, it mirrors the functioning of our brains. Deep learning algorithms are similar to how nervous system structured where each neuron connected each other and passing information. Deep learning models work in layers and a typical model at least have three layers. Each layer accepts the information from previous and pass it on to the next one.

4. IMPLEMENTATION

We have taken a gray picture where in we have changed the grayscale to colorized image which is shown in the module-1 *figure 4.1*. The next module i.e., module-2 consists of converting the original video to grayscale and colorized such that it can change the shades of the original video. Module-3 consists of adding filters to image and video through webcam and also by taking the video clips.

4.1 Module 1- Black and White Image to Color:

To convert the black and white to color, we use OpenCV, python and deep learning for both images and video streams.

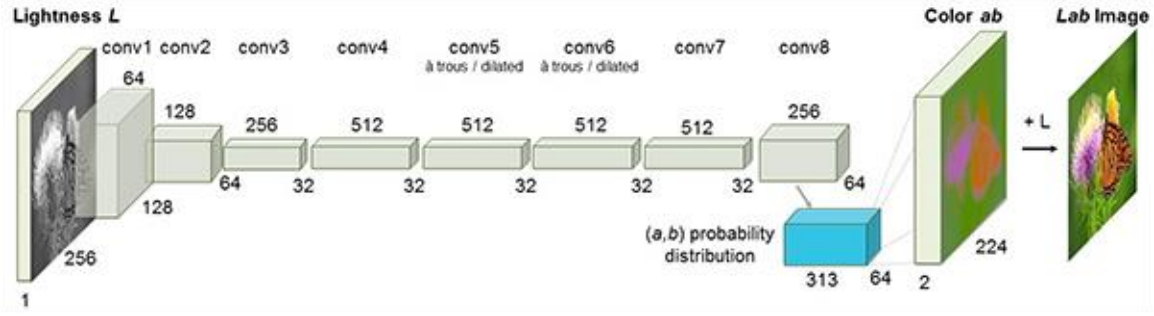

Figure 4.1 Conversion from grayscale to color

In this conversion, we convert grayscale to RGB to Lab color. Similar to the RGB color space, the Lab color space has three channels. But unlike the RGB color space, Lab encodes color information differently:

1. The ***L* channel** encodes lightness intensity only
2. The ***a* channel** encodes green-red.
3. And the ***b* channel** encodes blue-yellow.

Lab color does a better job representing how humans see color. Since the *L* channel encodes only the intensity, we can use the *L* channel as our grayscale input to the network. From there the network must learn to predict the *a* and *b* channels. Given the input *L* channel and the predicted *ab* channels we can then form our final output image.

Our colorizer script only requires three imports: NumPy, OpenCV, and argparse. This script requires that these four arguments be passed to the script directly from the terminal:

- Image: The path to our input black/white image.
- Prototxt: Our path to the Caffe prototxt file.
- Model: Our path to the Caffe pre-trained model.
- Points: The path to a NumPy cluster center points file.

With the above four flags and corresponding arguments, the script will be able to run with different inputs without changing any code. We load our Caffe model directly from the command line argument values. OpenCV can read Caffe models via the cv2.dnn.readNetFromCaffe function. We load ab channels quantization used for rebalancing by treating 1 1 convolutions and add them to this model.

By scaling the pixel intensities to between the range [0,1], any grayscale image can be converted to BGR color and then converting into Lab color. From there, we resize the ab color image with the same dimensions of input image. Grabbing the L channel from the original input image and concatenating the original L channel and predicted ab channels together forming colorized. Convert the colored image from lab color space to RGB. Bringing the pixel intensities back to [0,255]. During the preprocessing step, we divide by 255 and now we multiply with 255. The final output will result to colored image.

4.2 Module 2-Grayscale to Color from Original Video:

This script follows the same process as image colorizing except we will be processing the

frames of a video stream. Video script requires two imports:

- Video Stream: allows us to grab frames from a webcam or video file
- Time: will be used to pause to allow a webcam to warm up

Depending on whether we're working with a webcam or video file, we'll create our vs (i.e., "video stream") object here. From there, we'll load the colorizer deep learning model and cluster centres. We pre-process the video where we resize, scale, and convert to Lab. Then we grab the L channel, and perform mean subtraction. Then we'll post-process the result to from our colorized image. This is where we resize, grab our original L, and concatenate our predicted ab. From there, we convert from Lab to RGB, clip, and scale. Original webcam frame is shown along with our grayscale image and colorized result.

4.3 Module 3-Filter Module:

Structure of repository is as follows:

- video_detection.py is for video detection which can put filters according to your choice on live camera (Multiple face supported).
- hat&glass.py is for put hat & glasses on image which will you feed in your program.
- dogfilter.py is for put a dog face filter on a image which will you fee in program
- In images folder put images which you want to test on it.
- In Filters folder there are our Hat, Glass, and Dog filters pngs available.

1. **dog_filter.py:**

   In this we have imported cv2. In this first we will be setting up the path for the xml file incascadeclassifier which is used to recognize our frontal face. After this step we then set the path for our dog filter image. Once the path is set we then have to resize the image. Then the frontal face gets recognized and by using face width and face height the dog filters will be resized and gets added to the image.

2. **put_hat&glass.py:**

   In this we have imported cv2. In this first we will be setting up the path for the xml file in cascadeclassifier which is used to recognize our frontal face. After this step we then set the path for our dog filter image. Once the path is set we then have to resize the image. Then the frontal face gets recognized and by using face width and face height. The eye size and the length will be recognized the hat filters and the glass filter will be resized and gets added to the image.

3. **Video_detection.py:**

   In this we have imported cv2. In this first we will be setting up the path for the xml file in cascadeclassifier which is used to recognize our frontal face. After this step we then set the path for our dog filter image. Once the path is set we then have to resize the image. Then the frontal face gets recognized. We then use our webcam to run the video to put the filters. We will be setting time for the video so that it stops after the given time. We can make a choice of what filter to add. If we choose it as one, we can add the hat and glass filter. If we choose any other number, we get the dog filter.

4.4 Sample Code:

Image Colorization

Bw2color.py:

```
conv8 = net.getLayerId("conv8_313_rh")

pts = pts.transpose().reshape(2, 313, 1, 1)

net.getLayer(class8).blobs = [pts.astype("float32")]

net.getLayer(conv8).blobs = [np.full([1, 313], 2.606, dtype="float32")]

image = cv2.imread(args["image"])

scaled = image.astype("float32") / 255.0

lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)

resized = cv2.resize(lab, (224, 224))

L = cv2.split(resized)[0]

L -= 50

'print("[INFO] colorizing image...")'

net.setInput(cv2.dnn.blobFromImage(L))

ab = net.forward()[0, :, :, :].transpose((1, 2, 0))

ab = cv2.resize(ab, (image.shape[1], image.shape[0]))

L = cv2.split(lab)[0]

colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)
```

```
colorized = np.clip(colorized, 0, 1)

colorized = (255 * colorized).astype("uint8")

cv2.imshow("Original", image)

cv2.imshow("Colorized", colorized)

cv2.waitKey(0)
```

Video Colorization

Video_color.py:

```
from imutils.video import VideoStream

import numpy as np

import argparse

import imutils

import time

import cv2

ap = argparse.ArgumentParser()

ap.add_argument("-i", "--input", type=str,

        help="path to optional input video (webcam will be used otherwise)")

ap.add_argument("-p", "--prototxt", type=str, required=True,

        help="path to Caffe prototxt file")

ap.add_argument("-m", "--model", type=str, required=True,
```

```
        help="path to Caffe pre-trained model")
ap.add_argument("-c", "--points", type=str, required=True,
        help="path to cluster center points")
ap.add_argument("-w", "--width", type=int, default=500,
        help="input width dimension of frame")
args = vars(ap.parse_args())
webcam = not args.get("input", False)
if webcam:
        print("[INFO] starting video stream...")
        vs = VideoStream(src=0).start()
        time.sleep(2.0)
else:
        print("[INFO] opening video file...")
        vs = cv2.VideoCapture(args["input"])
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
pts = np.load(args["points"])
class8 = net.getLayerId("class8_ab")
conv8 = net.getLayerId("conv8_313_rh")
pts = pts.transpose().reshape(2, 313, 1, 1)
```

```
net.getLayer(class8).blobs = [pts.astype("float32")]
net.getLayer(conv8).blobs = [np.full([1, 313], 2.606, dtype="float32")]
while True:
        frame = vs.read()
        frame = frame if webcam else frame[1]
        if not webcam and frame is None:
                break
        frame = imutils.resize(frame, width=args["width"])
        scaled = frame.astype("float32") / 255.0
        lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)
        resized = cv2.resize(lab, (224, 224))
        L = cv2.split(resized)[0]
        L -= 50
        net.setInput(cv2.dnn.blobFromImage(L))
        ab = net.forward()[0, :, :, :].transpose((1, 2, 0))
        ab = cv2.resize(ab, (frame.shape[1], frame.shape[0]))
        L = cv2.split(lab)[0]
        colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)
        colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2BGR)
        colorized = np.clip(colorized, 0, 1)
```

```
        colorized = (255 * colorized).astype("uint8")
        cv2.imshow("Original", frame)
        cv2.imshow("Grayscale", cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY))
        cv2.imshow("Colorized", colorized)
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
                break
if webcam:
        vs.stop()
else:
        vs.release()
cv2.destroyAllWindows()
```

Image Filters

dogfilter.py:

```
import cv2
face = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
filename='C:/Users/admin/Project1/Filters_cv/images/gs.JPG'
img=cv2.imread(filename)
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
fl=face.detectMultiScale(gray,1.09,7)
dog=cv2.imread('C:/Users/admin/Project1/Filters_cv/Filters/dog.png')
def put_dog_filter(dog, fc, x, y, w, h):
    face_width = w
    face_height = h
    dog = cv2.resize(dog, (int(face_width * 1.5), int(face_height * 1.95)))
    for i in range(int(face_height * 1.75)):
        for j in range(int(face_width * 1.5)):
            for k in range(3):
                if dog[i][j][k] < 235:
                    fc[y + i - int(0.375 * h) - 1][x + j - int(0.35 * w)][k] = dog[i][j][k]
    return fc
for (x, y, w, h) in fl:
    frame = put_dog_filter(dog, img, x, y, w, h)
cv2.imshow('image',frame)
cv2.waitKey(20000)& 0xff
cv2.destroyAllWindows()
```

hat_glass.py:

```
import cv2
filename='C:/Users/admin/Project1/Filters_cv/images/rr.JPG'
```

```
img=cv2.imread(filename)

gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

fl=face.detectMultiScale(gray,1.09,7)

ey=face.detectMultiScale(gray,1.09,7)

hat=cv2.imread('C:/Users/admin/Project1/Filters_cv/Filters/hat.png')

glass=cv2.imread('C:/Users/admin/Project1/Filters_cv/Filters/glasses.png')

def put_hat(hat, fc, x, y, w, h):
    face_width = w
    face_height = h
    hat_width = face_width + 1
    hat_height = int(0.50 * face_height) + 1
    hat = cv2.resize(hat, (hat_width, hat_height))
    for i in range(hat_height):
        for j in range(hat_width):
            for k in range(3):
                if hat[i][j][k] < 235:
                    fc[y + i - int(0.40 * face_height)][x + j][k] = hat[i][j][k]
    return fc

def put_glass(glass, fc, x, y, w, h):
    face_width = w
```

```
    face_height = h
    hat_width = face_width + 1
    hat_height = int(0.50 * face_height) + 1
    glass = cv2.resize(glass, (hat_width, hat_height))
    for i in range(hat_height):
        for j in range(hat_width):
            for k in range(3):
                if glass[i][j][k] < 235:
                    fc[y + i - int(-0.20 * face_height)][x + j][k] = glass[i][j][k]
    return fc
for (x, y, w, h) in fl:
    frame = put_hat(hat, img, x, y, w, h)
for (x, y, w, h) in ey:
    frame=put_glass(glass,img, x, y, w, h)
cv2.imshow('image',frame)
cv2.waitKey(8000)& 0xff
cv2.destroyAllWindows()
```

video_detect.py:

```
import cv2
hat=cv2.imread('C:/Users/admin/Project1/Filters_cv/Filters/hat.png')
```

```
glass=cv2.imread('C:/Users/admin/Project1/Filters_cv/Filters/glasses.png')
dog=cv2.imread('C:/Users/admin/Project1/Filters_cv/Filters/dog.png')
def put_dog_filter(dog, fc, x, y, w, h):
    face_width = w
    face_height = h
    dog = cv2.resize(dog, (int(face_width * 1.5), int(face_height * 1.95)))
    for i in range(int(face_height * 1.75)):
        for j in range(int(face_width * 1.5)):
            for k in range(3):
                if dog[i][j][k] < 235:
                    fc[y + i - int(0.375 * h) - 1][x + j - int(0.35 * w)][k] = dog[i][j][k]
    return fc
def put_hat(hat, fc, x, y, w, h):
    face_width = w
    face_height = h
    hat_width = face_width + 1
    hat_height = int(0.50 * face_height) + 1
    hat = cv2.resize(hat, (hat_width, hat_height))
    for i in range(hat_height):
        for j in range(hat_width):
```

```
            for k in range(3):
                if hat[i][j][k] < 235:
                    fc[y + i - int(0.40 * face_height)][x + j][k] = hat[i][j][k]
    return fc
def put_glass(glass, fc, x, y, w, h):
    face_width = w
    face_height = h
    hat_width = face_width + 1
    hat_height = int(0.50 * face_height) + 1
    glass = cv2.resize(glass, (hat_width, hat_height))
    for i in range(hat_height):
        for j in range(hat_width):
            for k in range(3):
                if glass[i][j][k] < 235:
                    fc[y + i - int(-0.20 * face_height)][x + j][k] = glass[i][j][k]
    return fc
choice = 0
print('enter your choice filter to launch that: 1="put hat & glasses" ,any number="put fog filters" ')
choise= int(input('enter your choice:'))
```

```
webcam = cv2.VideoCapture(0)
while True:
    size=4
    (rval, im) = webcam.read()
    im = cv2.flip(im, 1, 0)
    gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    fl = face.detectMultiScale(gray,1.19,7)
    for (x, y, w, h) in fl:
        if choise ==1:
            im = put_hat(hat, im, x, y, w, h)
            im = put_glass(glass, im, x, y, w, h)
        else:
            im = put_dog_filter(dog, im, x, y, w, h)
    cv2.imshow('Hat & glasses',im)
    key = cv2.waitKey(30) & 0xff
    if key == 27:  # The Esc key
      break
```

5. RESULTS AND OBSERVATIONS

IMAGE REVISING

The *figure 5.1* is a clear picture of converting a grayscale image to a colorized image with using the pixel intensities and setting up their ranges.

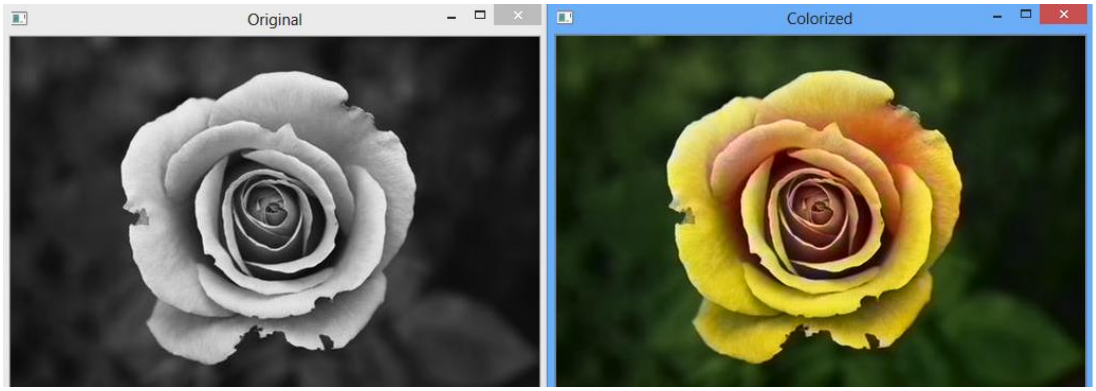

Figure 5.1 Image colorization

Adding different filters with the user's choice is shown in the *figure 5.2*. In this filtering, we resize the image of face and the filters such that, the filters will fit perfectly onto the face.

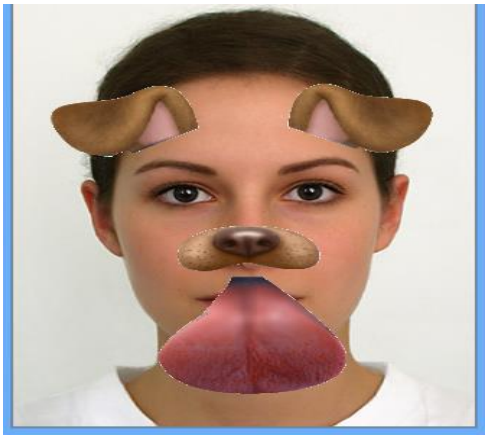

Figure 5.2 Image Filtering

VIDEO REVISING

This modification is done either through webcam or a video clip. It is a live colorizing the video to grayscale and colorized way which is shown in the *figure 5.3*. It is mainly used when there is a low brightness such that colorized can give an output with more lightness.

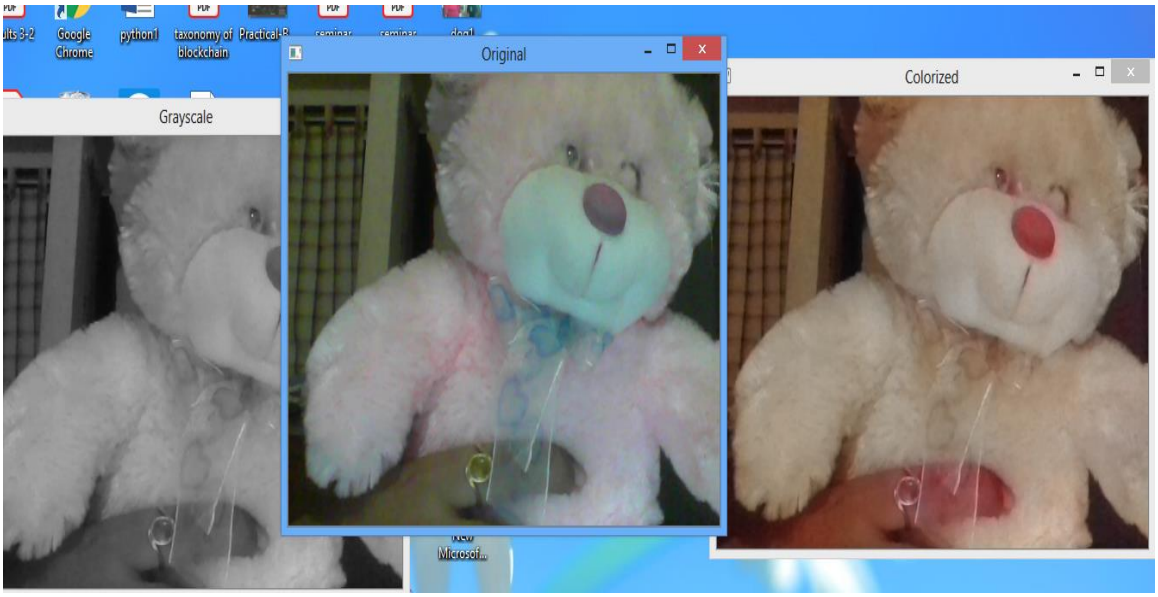

Figure 5.3 Video colorization

Video filters manipulates the images in our own personal computer. In this, only webcam is needed for the live recording and also can add some videos which is similar to video colorization. *Figure 5.4* describes the filters and its resizing depending upon the face structures.

Figure 5.4 Video filters

6. CONCLUSION AND FUTURE SCOPE

In these days, there is no picture without editing. By using image and audio revising, any live cam video can also be edited with the higher intensities and add different filters based upon the user. This is manually done without any human interference. Everyone is focused on adding different colors to their images. In an image, the focused part is only colorized and the remaining stays little darker. The background dims and the focused image lightens to make the users feel more attracted on it. User need to change their photo for every pixel if it is not automatic. In such case, this is very useful for them.

In future, there will be precorrection of images and adding more colors i.e., not only RGB (Red Green Blue) colors, but also different shades in it. This can be implemented by automatic coloring application so that it is freely used by the users without any huge amount of memory. It can also be improved by colorizing not only the grayscale image, but also an original image to the colorized by adding more filters in it.

REFERENCES

1. https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm
2. https://techtutorialsx.com/2018/06/02/python-opencv-converting-an-image-to-gray-scale/
3. https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Changing_ColorSpaces_RGB_HSV_HLS.php
4. https://towardsdatascience.com/facial-keypoints-detection-deep-learning-737547f73515
5. http://richzhang.io/colorization/