AI IMAGE COLOURISATION

18CSC305J ARTIFICIAL INTELLIGENCE

Course Project Report

Submitted by

AKSHAY NARISETTI (RA1911003010684)

PARTHIB RANJAN RAY (RA1911003010664)

PUSHKAR SINHA (RA1911003010676)

ROHAN GARODIA (RA1911003010680)

In partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTING TECHNOLOGIES

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

April 2022

PROJECT TITLE:- AI IMAGE COLOURIZER USING OPENCV AND DEEP LEARNING

(1)WHY AI IS AN IMPORTANT PART OF EVERYDAY LIFE?

- Artificial Intelligence (AI) and its multiple sub-domains are being increasingly employed in various industries and businesses to aid in repetitive processes. But there has been a burgeoning interest from established tech giants and startups in using AI to make everyday life a walk in the park.
- AI has been highly instrumental in optimizing the way we entertain ourselves, interact with our mobile devices, to even driving vehicles for us. We tend to encounter Machine Learning (ML) algorithms and Natural Language Processing (NLP) in several everyday tasks more than we know.

(2)SOME APPLICATIONS OF AI IN EVERYDAY LIFE

- AI and ML-powered software and devices are mimicking human thought patterns to facilitate the digital transformation of society. AI systems perceive their environment, deal with what they perceive, solve problems and act to help with tasks to make everyday life easier.
- Some applications are as follows:-
- **Voice Assistants**
- **Entertainment Streaming Apps**
- **Personalized Marketing**
- **Smart Input Keyboards**
- **Navigation and Travel and many more…………….**

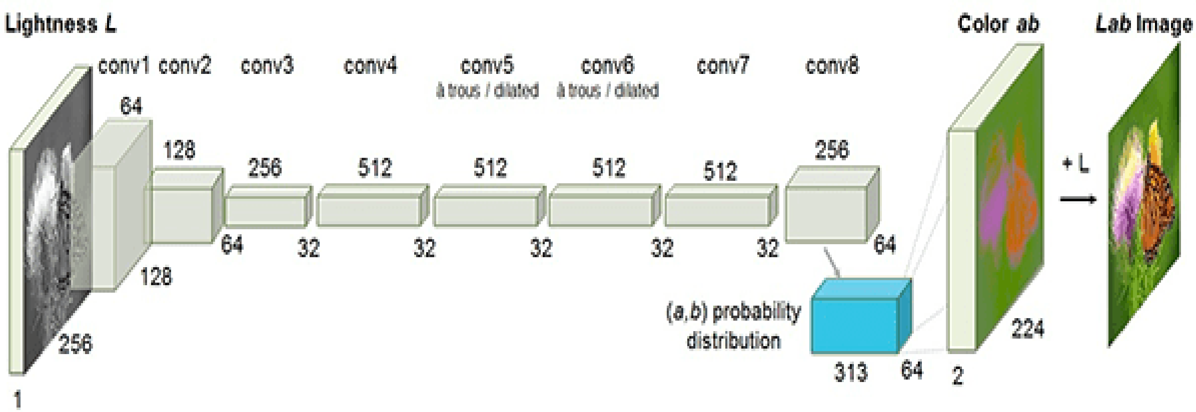
(3)AI IMAGE COLOURIZER AND ITS NEED IN TODAYS WORLD:

- Image colorization is the process of taking an **input grayscale (black and white) image** and then producing an **output colorized image** that represents the semantic colors and tones of the input (for example, an ocean on a clear sunny day must be plausibly "blue" — it can't be colored "hot pink" by the model).
- **Previous methods for image colorization either:**

1. **Relied on significant human interaction and annotation**
2. **Produced desaturated colorization**

The novel approach we are going to use here today instead relies on deep learning. We will utilize a Convolutional Neural Network capable of colorizing black and white images with results that can even "fool" humans!

This was done to revive many classic photographs, films and videos and make it more vivid and colourful for the human brain to remember better as visual learning is the best learning.



(4)How can we colorize black and white images with deep learning?

- The technique we'll be covering here today is from Zhang et al.'s 2016 ECCV paper, ***Colorful Image Colorization***.
- Zhang et al. decided to attack the problem of image colorization by using Convolutional Neural Networks to "hallucinate" what an input grayscale image would look like when colorized.
- To train the network Zhang et al. started with the **ImageNet dataset** and converted all images from the RGB color space to the **Lab color space.**
- Similar to the RGB color space, the Lab color space has *three channels*. But *unlike* the RGB color space, Lab encodes color information differently:
- The ***L* channel** encodes lightness intensity only
- The ***a* channel** encodes green-red.
- And the ***b* channel** encodes blue-yellow
- 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.**

(5) SUMMARY OF THE ABOVE PROCESS

1. Convert all training images from the RGB color space to the Lab color space.
2. Use the ***L* channel** as the input to the network and train the network to predict the ***ab* channels.**

3. Combine the input ***L* channel** with the predicted ***ab* channels.**
4. Convert the Lab image back to RGB.

- To produce more plausible black and white image colorizations the authors also utilize a few additional techniques including mean annealing and a specialized loss function for color rebalancing

(6) PROJECT STRUCTURE:

- Create the source code , models to be trained and the images to be colourized in a particular folder and open it using Jupyter notebook or Google Colab
- use the tree command to inspect the project structure.
- We have four sample black and white images in the images/ directory.
- Our Caffe model and prototxt are inside the model/ directory along with the cluster points NumPy file.
- We'll be reviewing two scripts today:
- bw2color_image.py
- bw2color_video.py
- The image script can process any black and white (also known as grayscale) image you pass in.
- Our video script will either use your webcam or accept an input video file and then perform colorization.

```
$ tree --dirsfirst
.
├── images
│   ├── adrian_and_janie.png
│   ├── albert_einstein.jpg
│   ├── mark_twain.jpg
│   └── robin_williams.jpg
├── model
│   ├── colorization_deploy_v2.prototxt
│   ├── colorization_release_v2.caffemodel
│   └── pts_in_hull.npy
├── bw2color_image.py
└── bw2color_video.py

2 directories, 9 files
```

<u>DIRECTORIES IN THE FILE CREATED</u>

<u>(7) Colorizing black and white images with OpenCV</u>

- Let’s go ahead and implement black and white image colorization script with OpenCV.
- Open up the bw2color_image.py file and insert the following code:

Black and white image colorization with OpenCV and Deep Learning

```python
# import the necessary packages
import numpy as np
import argparse
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", type=str, required=True,
	help="path to input black and white image")
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")
args = vars(ap.parse_args())
```

Our colorizer script only requires three imports: NumPy, OpenCV, and argparse

Let's go ahead and use argparse to parse command line arguments. 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.
- Let's go ahead and load our model and cluster centers into memory:
- Line 21 loads our Caffe model directly from the command line argument values. OpenCV can read Caffe models via the cv2.dnn.readNetFromCaffe function.
- Line 22 then loads the cluster center points directly from the command line argument path to the points file. This file is in NumPy format so we're using np.load.
- Line 25-29 Load centers for *ab* channel quantization used for rebalancing.

- Treat each of the points as *1×1* convolutions and add them to the model.

Black and white image colorization with OpenCV and Deep Learning

```python
# load our serialized black and white colorizer model and cluster
# center points from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
pts = np.load(args["points"])

# add the cluster centers as 1x1 convolutions to the model
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")]
```

- Now let’s load, scale, and convert our image:
- To load our input image from the file path, we use cv2.imread on Line 34.
- Preprocessing steps include:
- Scaling pixel intensities to the range *[0, 1]* (Line 35).
- Converting from BGR to Lab color space (Line 36).

Black and white image colorization with OpenCV and Deep Learning

```python
31. # load the input image from disk, scale the pixel intensities to the
32. # range [0, 1], and then convert the image from the BGR to Lab color
33. # space
34. image = cv2.imread(args["image"])
35. scaled = image.astype("float32") / 255.0
36. lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)
```

Black and white image colorization with OpenCV and Deep Learning

```python
38. # resize the Lab image to 224x224 (the dimensions the colorization
39. # network accepts), split channels, extract the 'L' channel, and then
40. # perform mean centering
41. resized = cv2.resize(lab, (224, 224))
42. L = cv2.split(resized)[0]
43. L -= 50
```

- We’ll go ahead and resize the input image to 224×224 (**Line 41**), the required input dimensions for the network.
- Then we grab the L channel only (i.e., the input) and perform mean subtraction (**Lines 42 and 43**).
- Now we can pass the **input *L* channel** through the network to **predict the *ab* channels:**

```
Black and white image colorization with OpenCV and Deep Learning
45. # pass the L channel through the network which will *predict* the 'a'
46. # and 'b' channel values
47. 'print("[INFO] colorizing image...")'
48. net.setInput(cv2.dnn.blobFromImage(L))
49. ab = net.forward()[0, :, :, :].transpose((1, 2, 0))
50.
51. # resize the predicted 'ab' volume to the same dimensions as our
52. # input image
53. ab = cv2.resize(ab, (image.shape[1], image.shape[0]))
```

- A forward pass of the L channel through the network takes place on **Lines 48 and 49**
- Notice that after we called net.forward , on the same line, we went ahead and extracted the predicted ab volume

(8)POST PROCESSING OF IMAGE

- Grabbing the L channel from the ***original*** input image (**Line 58**) and concatenating the original L channel and ***predicted*** ab
- channels together forming colorized (**Line 59**).
- Converting the colorized image from the Lab color space to RGB (**Line 63**).
- Clipping any pixel intensities that fall outside the range *[0, 1]* (**Line 64**).
- Bringing the pixel intensities back into the range *[0, 255]* (**Line 69**). During the preprocessing steps (**Line 35**) we divided by 255 and now we are multiplying by 255.
- We've also found that this scaling and "uint8" conversion isn't a requirement but that it helps the code work between **OpenCV 3.4.x** and **4.x** versions.

(9) **Image Colourisation Results:-**

(10) **PURPOSE OF THE PROJECT**

- AI Image colourisation using Open CV and Deep learning will reduce the painstaking task of colouring images using human annotations and other traditional techniques.
- It will help us preserve many archival videos, photos etc in better quality which may have lost their quality due to not so good colouring techniques in the past.
- It will also help us bring us life a lot of black and white images for better understanding etc.
- Last but not least it will help us colourise personal photos of our forefathers for better display and preservation.

AI Image Colourizer Codes And Github Upload Screenshot

(1)Colourizer.py

```
import numpy as np
import cv2
import PySimpleGUI as sg
import os.path

version = '7 June 2020'

prototxt = r'model/colorization_deploy_v2.prototxt'
model = r'model/colorization_release_v2.caffemodel'
points = r'model/pts_in_hull.npy'
points = os.path.join(os.path.dirname(__file__), points)
prototxt = os.path.join(os.path.dirname(__file__), prototxt)
model = os.path.join(os.path.dirname(__file__), model)
if not os.path.isfile(model):
   sg.popup_scrolled('Missing model file', 'You are missing the file
"colorization_release_v2.caffemodel"',
                  'Download it and place into your "model" folder', 'You can
download this file from this location:\n',
r'https://www.dropbox.com/s/dx0qvhhp5hbcx7z/colorization_release_v
2.caffemodel?dl=1')
   exit()
net = cv2.dnn.readNetFromCaffe(prototxt, model)    # load model from
disk
pts = np.load(points)

# add the cluster centers as 1x1 convolutions to the model
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")]
```

```
def colorize_image(image_filename=None, cv2_frame=None):
    """
    Where all the magic happens.  Colorizes the image provided. Can
colorize either
    a filename OR a cv2 frame (read from a web cam most likely)
    :param image_filename: (str) full filename to colorize
    :param cv2_frame: (cv2 frame)
    :return: Tuple[cv2 frame, cv2 frame] both non-colorized and colorized
images in cv2 format as a tuple
    """
    # load the input image from disk, scale the pixel intensities to the
range [0, 1], and then convert the image from the BGR to Lab color
space
    image = cv2.imread(image_filename) if image_filename else
cv2_frame
    scaled = image.astype("float32") / 255.0
    lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)

    # resize the Lab image to 224x224 (the dimensions the colorization
network accepts), split channels, extract the 'L' channel, and then
perform mean centering
    resized = cv2.resize(lab, (224, 224))
    L = cv2.split(resized)[0]
    L -= 50

    # pass the L channel through the network which will *predict* the 'a'
and 'b' channel values
    'print("[INFO] colorizing image...")'
    net.setInput(cv2.dnn.blobFromImage(L))
    ab = net.forward()[0, :, :, :].transpose((1, 2, 0))

    # resize the predicted 'ab' volume to the same dimensions as our
input image
    ab = cv2.resize(ab, (image.shape[1], image.shape[0]))
```

```
    # grab the 'L' channel from the *original* input image (not the resized
one) and concatenate the original 'L' channel with the predicted 'ab'
channels
    L = cv2.split(lab)[0]
    colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)

    # convert the output image from the Lab color space to RGB, then clip
any values that fall outside the range [0, 1]
    colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2BGR)
    colorized = np.clip(colorized, 0, 1)

    # the current colorized image is represented as a floating point data
type in the range [0, 1] -- let's convert to an unsigned 8-bit integer
representation in the range [0, 255]
    colorized = (255 * colorized).astype("uint8")
    return image, colorized


def convert_to_grayscale(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convert webcam
frame to grayscale
    gray_3_channels = np.zeros_like(frame)  # Convert grayscale frame
(single channel) to 3 channels
    gray_3_channels[:, :, 0] = gray
    gray_3_channels[:, :, 1] = gray
    gray_3_channels[:, :, 2] = gray
    return gray_3_channels


# --------------------------------- The GUI ---------------------------------

# First the window layout...2 columns

left_col = [[sg.Text('Folder'), sg.In(size=(25,1), enable_events=True
,key='-FOLDER-'), sg.FolderBrowse()],
            [sg.Listbox(values=[], enable_events=True, size=(40,20),key='-FILE
LIST-')],
```

```
          [sg.CBox('Convert to gray first',key='-MAKEGRAY-')],
          [sg.Text('Version ' + version, font='Courier 8')]]

images_col = [[sg.Text('Input file:'), sg.In(enable_events=True, key='-IN
FILE-'), sg.FileBrowse()],
            [sg.Button('Colorize Photo', key='-PHOTO-'), sg.Button('Start
Webcam', key='-WEBCAM-'), sg.Button('Save File', key='-SAVE-'),
sg.Button('Exit')],
            [sg.Image(filename='', key='-IN-'), sg.Image(filename='', key='-
OUT-')],]
# ----- Full layout -----
layout = [[sg.Column(left_col), sg.VSeperator(), sg.Column(images_col)]]

# ----- Make the window -----
window = sg.Window('Photo Colorizer', layout, grab_anywhere=True)

# ----- Run the Event Loop -----
prev_filename = colorized = cap = None
while True:
    event, values = window.read()
    if event in (None, 'Exit'):
        break
    if event == '-FOLDER-':         # Folder name was filled in, make a list of
files in the folder
        folder = values['-FOLDER-']
        img_types = (".png", ".jpg", "jpeg", ".tiff", ".bmp")
        # get list of files in folder
        try:
            flist0 = os.listdir(folder)
        except:
            continue
        fnames = [f for f in flist0 if os.path.isfile(
            os.path.join(folder, f)) and f.lower().endswith(img_types)]
        window['-FILE LIST-'].update(fnames)
    elif event == '-FILE LIST-':    # A file was chosen from the listbox
        try:
            filename = os.path.join(values['-FOLDER-'], values['-FILE LIST-'][0])
```

```
            image = cv2.imread(filename)
            window['-IN-'].update(data=cv2.imencode('.png',
image)[1].tobytes())
            window['-OUT-'].update(data='')
            window['-IN FILE-'].update('')

            if values['-MAKEGRAY-']:
                gray_3_channels = convert_to_grayscale(image)
                window['-IN-'].update(data=cv2.imencode('.png',
gray_3_channels)[1].tobytes())
                image, colorized =
colorize_image(cv2_frame=gray_3_channels)
            else:
                image, colorized = colorize_image(filename)

            window['-OUT-'].update(data=cv2.imencode('.png',
colorized)[1].tobytes())
        except:
            continue
    elif event == '-PHOTO-':        # Colorize photo button clicked
        try:
            if values['-IN FILE-']:
                filename = values['-IN FILE-']
            elif values['-FILE LIST-']:
                filename = os.path.join(values['-FOLDER-'], values['-FILE LIST-
'][0])
            else:
                continue
            if values['-MAKEGRAY-']:
                gray_3_channels = convert_to_grayscale(cv2.imread(filename))
                window['-IN-'].update(data=cv2.imencode('.png',
gray_3_channels)[1].tobytes())
                image, colorized =
colorize_image(cv2_frame=gray_3_channels)
            else:
                image, colorized = colorize_image(filename)
```

```
            window['-IN-'].update(data=cv2.imencode('.png',
image)[1].tobytes())
         window['-OUT-'].update(data=cv2.imencode('.png',
colorized)[1].tobytes())
      except:
         continue
   elif event == '-IN FILE-':      # A single filename was chosen
      filename = values['-IN FILE-']
      if filename != prev_filename:
         prev_filename = filename
         try:
            image = cv2.imread(filename)
            window['-IN-'].update(data=cv2.imencode('.png',
image)[1].tobytes())
         except:
            continue
   elif event == '-WEBCAM-':       # Webcam button clicked
      sg.popup_quick_message('Starting up your Webcam... this takes a
moment....', auto_close_duration=1,  background_color='red',
text_color='white', font='Any 16')
      window['-WEBCAM-'].update('Stop Webcam',
button_color=('white','red'))
      cap = cv2.VideoCapture(1) if not cap else cap
      while True:                # Loop that reads and shows webcam until stop
button
         ret, frame = cap.read()     # Read a webcam frame
         gray_3_channels = convert_to_grayscale(frame)
         image, colorized = colorize_image(cv2_frame=gray_3_channels)
# Colorize the 3-channel grayscale frame
         window['-IN-'].update(data=cv2.imencode('.png',
gray_3_channels)[1].tobytes())
         window['-OUT-'].update(data=cv2.imencode('.png',
colorized)[1].tobytes())
         event, values = window.read(timeout=0)  # Update the window
outputs and check for new events
         if event in (None, '-WEBCAM-', 'Exit'): # Clicked the Stop Webcam
button or closed window entirely
```

```
            window['-WEBCAM-'].update('Start Webcam',
button_color=sg.theme_button_color())
            window['-IN-'].update('')
            window['-OUT-'].update('')
            break
   elif event == '-SAVE-' and colorized is not None:   # Clicked the Save
File button
        filename = sg.popup_get_file('Save colorized image.\nColorized
image be saved in format matching the extension you enter.',
save_as=True)
        try:
            if filename:
                cv2.imwrite(filename, colorized)
                sg.popup_quick_message('Image save complete',
background_color='red', text_color='white', font='Any 16')
        except:
            sg.popup_quick_message('ERROR - Image NOT saved!',
background_color='red', text_color='white', font='Any 16')
# ----- Exit program -----
window.close()
```

(2)Colourizer-Webcam.py

```
import numpy as np
import cv2
import PySimpleGUI as sg
import os.path

prototxt = r'model/colorization_deploy_v2.prototxt'
model = r'model/colorization_release_v2.caffemodel'
points = r'model/pts_in_hull.npy'
points = os.path.join(os.path.dirname(__file__), points)
prototxt = os.path.join(os.path.dirname(__file__), prototxt)
model = os.path.join(os.path.dirname(__file__), model)
if not os.path.isfile(model):
```

```
    sg.popup_scrolled('Missing model file', 'You are missing the
file "colorization_release_v2.caffemodel"',
                    'Download it and place into your "model" folder',
'You can download this file from this location:\n',
r'https://www.dropbox.com/s/dx0qvhhp5hbcx7z/colorization_
release_v2.caffemodel?dl=1')
    exit()
net = cv2.dnn.readNetFromCaffe(prototxt, model)     # load
model from disk
pts = np.load(points)

# add the cluster centers as 1x1 convolutions to the model
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")]

def colorize_image(image_filename=None, cv2_frame=None):
    """
    Where all the magic happens.  Colorizes the image provided.
Can colorize either
    a filename OR a cv2 frame (read from a web cam most likely)
    :param image_filename: (str) full filename to colorize
    :param cv2_frame: (cv2 frame)
    :return: cv2 frame colorized image in cv2 format
    """
    # load the input image from disk, scale the pixel intensities to
the range [0, 1], and then convert the image from the BGR to
Lab color space
    image = cv2.imread(image_filename) if image_filename else
cv2_frame
```

```
    scaled = image.astype("float32") / 255.0
    lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)

    # resize the Lab image to 224x224 (the dimensions the
colorization network accepts), split channels, extract the 'L'
channel, and then perform mean centering
    resized = cv2.resize(lab, (224, 224))
    L = cv2.split(resized)[0]
    L -= 50

    # pass the L channel through the network which will
*predict* the 'a' and 'b' channel values
    'print("[INFO] colorizing image...")'
    net.setInput(cv2.dnn.blobFromImage(L))
    ab = net.forward()[0, :, :, :].transpose((1, 2, 0))

    # resize the predicted 'ab' volume to the same dimensions as
our input image
    ab = cv2.resize(ab, (image.shape[1], image.shape[0]))

    # grab the 'L' channel from the *original* input image (not
the resized one) and concatenate the original 'L' channel with
the predicted 'ab' channels
    L = cv2.split(lab)[0]
    colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)

    # convert the output image from the Lab color space to RGB,
then clip any values that fall outside the range [0, 1]
    colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2BGR)
    colorized = np.clip(colorized, 0, 1)
```

```
    # the current colorized image is represented as a floating
point data type in the range [0, 1] -- let's convert to an
unsigned 8-bit integer representation in the range [0, 255]
    colorized = (255 * colorized).astype("uint8")
    return  colorized


def convert_to_grayscale(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  #
Convert webcam frame to grayscale
    gray_3_channels = np.zeros_like(frame)  # Convert grayscale
frame (single channel) to 3 channels
    gray_3_channels[:, :, 0] = gray
    gray_3_channels[:, :, 1] = gray
    gray_3_channels[:, :, 2] = gray
    return gray_3_channels


def make_video_window(title, location):
    return sg.Window(title, [[sg.Image(key='-IMAGE-')]],
finalize=True, margins=(0,0), element_padding=(0,0),
location=location)

def convert_cvt_to_data(cv2_frame):
    return cv2.imencode('.png', cv2_frame)[1].tobytes()


def main():
    # --------------------------------- The GUI ---------------------------------

    layout = [  [sg.Text('Colorized Webcam Demo', font='Any
18')],
```

```
            [sg.Button('Start Webcam', key='-WEBCAM-'),
sg.Button('Exit')]]

    # ----- Make the starting window -----
    window_start = sg.Window('Webcam Colorizer', layout,
grab_anywhere=True, finalize=True)

    # ----- Run the Event Loop -----
    cap, playback_active = None, False
    while True:
        window, event, values = sg.read_all_windows(timeout=10)
        if event == 'Exit' or (window == window_start and event is
None):
            break
        elif event == '-WEBCAM-':      # Webcam button clicked
            if not playback_active:
                sg.popup_quick_message('Starting up your Webcam...
this takes a moment....', auto_close_duration=1,
background_color='red', text_color='white', font='Any 16')
                window_start['-WEBCAM-'].update('Stop Webcam',
button_color=('white','red'))
                cap = cv2.VideoCapture(0) if not cap else cap
                window_raw_camera = make_video_window('Your
Webcam Raw Video', (300,200))
                window_gray_camera = make_video_window('Video
as Grayscale', (1000,200))
                window_colorized_camera =
make_video_window('Your Colorized Video', (1700,200))
                playback_active = True
            else:
                playback_active = False
                window['-WEBCAM-'].update('Start Webcam',
button_color=sg.theme_button_color())
```

```
            window_raw_camera.close()
            window_gray_camera.close()
            window_colorized_camera.close()
        elif event == sg.TIMEOUT_EVENT and playback_active:
          ret, frame = cap.read()  # Read a webcam frame

          # display raw image
          if window_raw_camera:
            window_raw_camera['-IMAGE-
'].update(data=convert_cvt_to_data(frame))
          # display gray image
          gray_3_channels = convert_to_grayscale(frame)
          if window_gray_camera:
            window_gray_camera['-IMAGE-
'].update(data=convert_cvt_to_data(gray_3_channels))
          # display colorized image
          if window_colorized_camera:
            window_colorized_camera['-IMAGE-
'].update(data=convert_cvt_to_data(colorize_image(cv2_frame
=gray_3_channels)))

        # if a window closed
        if event is None:
          if window == window_raw_camera:
            window_raw_camera.close()
            window_raw_camera = None
          elif window == window_gray_camera:
            window_gray_camera.close()
            window_gray_camera = None
          elif window == window_colorized_camera:
            window_colorized_camera.close()
            window_colorized_camera = None
```

```
        # If playback is active, but all camera windows closed,
indicate not longer playing and change button color
        if playback_active and window_colorized_camera is None
and window_gray_camera is None and window_raw_camera is
None:
            playback_active = False
            window_start['-WEBCAM-'].update('Start Webcam',
button_color=sg.theme_button_color())

    # ----- Exit program -----
    window.close()

if __name__ == '__main__':
    main()
```

(3)Colourization-deploy-v2.prototxt

```
name: "LtoAB"

layer {
  name: "data_l"
  type: "Input"
  top: "data_l"
  input_param {
    shape { dim: 1 dim: 1 dim: 224 dim: 224 }
  }
}

# *****************
# ***** conv1 *****
# *****************
layer {
  name: "bw_conv1_1"
```

```
  type: "Convolution"
  bottom: "data_l"
  top: "conv1_1"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 64
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu1_1"
  type: "ReLU"
  bottom: "conv1_1"
  top: "conv1_1"
}
layer {
  name: "conv1_2"
  type: "Convolution"
  bottom: "conv1_1"
  top: "conv1_2"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 64
    pad: 1
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1_2"
```

```
  type: "ReLU"
  bottom: "conv1_2"
  top: "conv1_2"
}
layer {
  name: "conv1_2norm"
  type: "BatchNorm"
  bottom: "conv1_2"
  top: "conv1_2norm"
  batch_norm_param{ }
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
}
# ****************
# ***** conv2 *****
# ****************
layer {
  name: "conv2_1"
  type: "Convolution"
  # bottom: "conv1_2"
  bottom: "conv1_2norm"
  # bottom: "pool1"
  top: "conv2_1"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 128
    pad: 1
    kernel_size: 3
  }
}
layer {
```

```
  name: "relu2_1"
  type: "ReLU"
  bottom: "conv2_1"
  top: "conv2_1"
}
layer {
  name: "conv2_2"
  type: "Convolution"
  bottom: "conv2_1"
  top: "conv2_2"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 128
    pad: 1
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu2_2"
  type: "ReLU"
  bottom: "conv2_2"
  top: "conv2_2"
}
layer {
  name: "conv2_2norm"
  type: "BatchNorm"
  bottom: "conv2_2"
  top: "conv2_2norm"
  batch_norm_param{ }
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
```

```
  param {lr_mult: 0 decay_mult: 0}
}
# ****************
# ***** conv3 *****
# ****************
layer {
  name: "conv3_1"
  type: "Convolution"
  # bottom: "conv2_2"
  bottom: "conv2_2norm"
  # bottom: "pool2"
  top: "conv3_1"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu3_1"
  type: "ReLU"
  bottom: "conv3_1"
  top: "conv3_1"
}
layer {
  name: "conv3_2"
  type: "Convolution"
  bottom: "conv3_1"
  top: "conv3_2"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
```

```
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu3_2"
  type: "ReLU"
  bottom: "conv3_2"
  top: "conv3_2"
}
layer {
  name: "conv3_3"
  type: "Convolution"
  bottom: "conv3_2"
  top: "conv3_3"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu3_3"
  type: "ReLU"
  bottom: "conv3_3"
  top: "conv3_3"
}
layer {
```

```
  name: "conv3_3norm"
  type: "BatchNorm"
  bottom: "conv3_3"
  top: "conv3_3norm"
  batch_norm_param{ }
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
}
# ****************
# ***** conv4 *****
# ****************
layer {
  name: "conv4_1"
  type: "Convolution"
  # bottom: "conv3_3"
  bottom: "conv3_3norm"
  # bottom: "pool3"
  top: "conv4_1"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 512
    kernel_size: 3
    stride: 1
    pad: 1
    dilation: 1
  }
}
layer {
  name: "relu4_1"
  type: "ReLU"
  bottom: "conv4_1"
```

```
  top: "conv4_1"
}
layer {
  name: "conv4_2"
  type: "Convolution"
  bottom: "conv4_1"
  top: "conv4_2"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 512
    kernel_size: 3
    stride: 1
    pad: 1
    dilation: 1
  }
}
layer {
  name: "relu4_2"
  type: "ReLU"
  bottom: "conv4_2"
  top: "conv4_2"
}
layer {
  name: "conv4_3"
  type: "Convolution"
  bottom: "conv4_2"
  top: "conv4_3"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 512
    kernel_size: 3
```

```
    stride: 1
    pad: 1
    dilation: 1
  }
}
layer {
  name: "relu4_3"
  type: "ReLU"
  bottom: "conv4_3"
  top: "conv4_3"
}
layer {
  name: "conv4_3norm"
  type: "BatchNorm"
  bottom: "conv4_3"
  top: "conv4_3norm"
  batch_norm_param{ }
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
}
# ****************
# ***** conv5 *****
# ****************
layer {
  name: "conv5_1"
  type: "Convolution"
  # bottom: "conv4_3"
  bottom: "conv4_3norm"
  # bottom: "pool4"
  top: "conv5_1"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
```

```
  convolution_param {
    num_output: 512
    kernel_size: 3
    stride: 1
    pad: 2
    dilation: 2
  }
}
layer {
  name: "relu5_1"
  type: "ReLU"
  bottom: "conv5_1"
  top: "conv5_1"
}
layer {
  name: "conv5_2"
  type: "Convolution"
  bottom: "conv5_1"
  top: "conv5_2"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 512
    kernel_size: 3
    stride: 1
    pad: 2
    dilation: 2
  }
}
layer {
  name: "relu5_2"
  type: "ReLU"
  bottom: "conv5_2"
```

```
  top: "conv5_2"
}
layer {
  name: "conv5_3"
  type: "Convolution"
  bottom: "conv5_2"
  top: "conv5_3"
  # param {lr_mult: 0 decay_mult: 0}
  # param {lr_mult: 0 decay_mult: 0}
  convolution_param {
    num_output: 512
    kernel_size: 3
    stride: 1
    pad: 2
    dilation: 2
  }
}
layer {
  name: "relu5_3"
  type: "ReLU"
  bottom: "conv5_3"
  top: "conv5_3"
}
layer {
  name: "conv5_3norm"
  type: "BatchNorm"
  bottom: "conv5_3"
  top: "conv5_3norm"
  batch_norm_param{ }
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
}
```

```
# ****************
# ***** conv6 *****
# ****************
layer {
  name: "conv6_1"
  type: "Convolution"
  bottom: "conv5_3norm"
  top: "conv6_1"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 2
    dilation: 2
  }
}
layer {
  name: "relu6_1"
  type: "ReLU"
  bottom: "conv6_1"
  top: "conv6_1"
}
layer {
  name: "conv6_2"
  type: "Convolution"
  bottom: "conv6_1"
  top: "conv6_2"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 2
    dilation: 2
  }
}
```

```
layer {
  name: "relu6_2"
  type: "ReLU"
  bottom: "conv6_2"
  top: "conv6_2"
}
layer {
  name: "conv6_3"
  type: "Convolution"
  bottom: "conv6_2"
  top: "conv6_3"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 2
    dilation: 2
  }
}
layer {
  name: "relu6_3"
  type: "ReLU"
  bottom: "conv6_3"
  top: "conv6_3"
}
layer {
  name: "conv6_3norm"
  type: "BatchNorm"
  bottom: "conv6_3"
  top: "conv6_3norm"
  batch_norm_param{ }
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
```

```
}
# ****************
# ***** conv7 *****
# ****************
layer {
  name: "conv7_1"
  type: "Convolution"
  bottom: "conv6_3norm"
  top: "conv7_1"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    dilation: 1
  }
}
layer {
  name: "relu7_1"
  type: "ReLU"
  bottom: "conv7_1"
  top: "conv7_1"
}
layer {
  name: "conv7_2"
  type: "Convolution"
  bottom: "conv7_1"
  top: "conv7_2"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    dilation: 1
  }
```

```
}
layer {
  name: "relu7_2"
  type: "ReLU"
  bottom: "conv7_2"
  top: "conv7_2"
}
layer {
  name: "conv7_3"
  type: "Convolution"
  bottom: "conv7_2"
  top: "conv7_3"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 1
    dilation: 1
  }
}
layer {
  name: "relu7_3"
  type: "ReLU"
  bottom: "conv7_3"
  top: "conv7_3"
}
layer {
  name: "conv7_3norm"
  type: "BatchNorm"
  bottom: "conv7_3"
  top: "conv7_3norm"
  batch_norm_param{ }
  param {lr_mult: 0 decay_mult: 0}
  param {lr_mult: 0 decay_mult: 0}
```

```
  param {lr_mult: 0 decay_mult: 0}
}
# ****************
# ***** conv8 *****
# ****************
layer {
  name: "conv8_1"
  type: "Deconvolution"
  bottom: "conv7_3norm"
  top: "conv8_1"
  convolution_param {
    num_output: 256
    kernel_size: 4
    pad: 1
    dilation: 1
    stride: 2
  }
}
layer {
  name: "relu8_1"
  type: "ReLU"
  bottom: "conv8_1"
  top: "conv8_1"
}
layer {
  name: "conv8_2"
  type: "Convolution"
  bottom: "conv8_1"
  top: "conv8_2"
  convolution_param {
    num_output: 256
    kernel_size: 3
    pad: 1
```

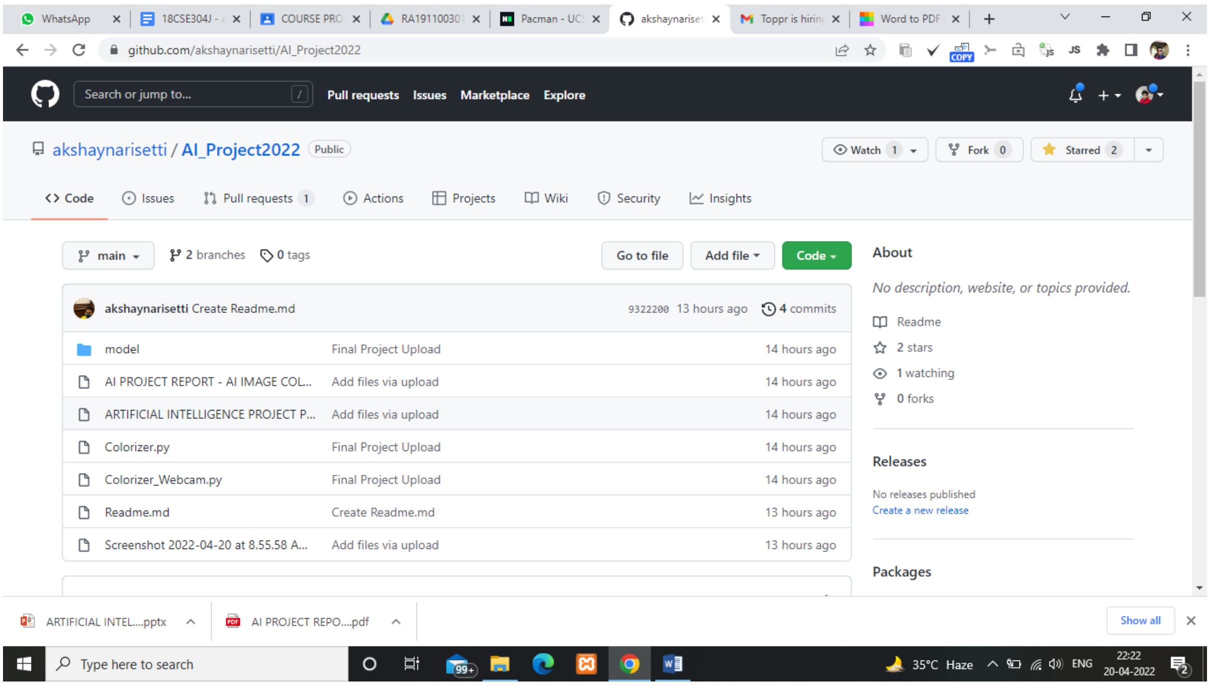
```
    dilation: 1
  }
}
layer {
  name: "relu8_2"
  type: "ReLU"
  bottom: "conv8_2"
  top: "conv8_2"
}
layer {
  name: "conv8_3"
  type: "Convolution"
  bottom: "conv8_2"
  top: "conv8_3"
  convolution_param {
    num_output: 256
    kernel_size: 3
    pad: 1
    dilation: 1
  }
}
layer {
  name: "relu8_3"
  type: "ReLU"
  bottom: "conv8_3"
  top: "conv8_3"
}
# ******************
# ***** Softmax *****
# ******************
layer {
  name: "conv8_313"
  type: "Convolution"
```

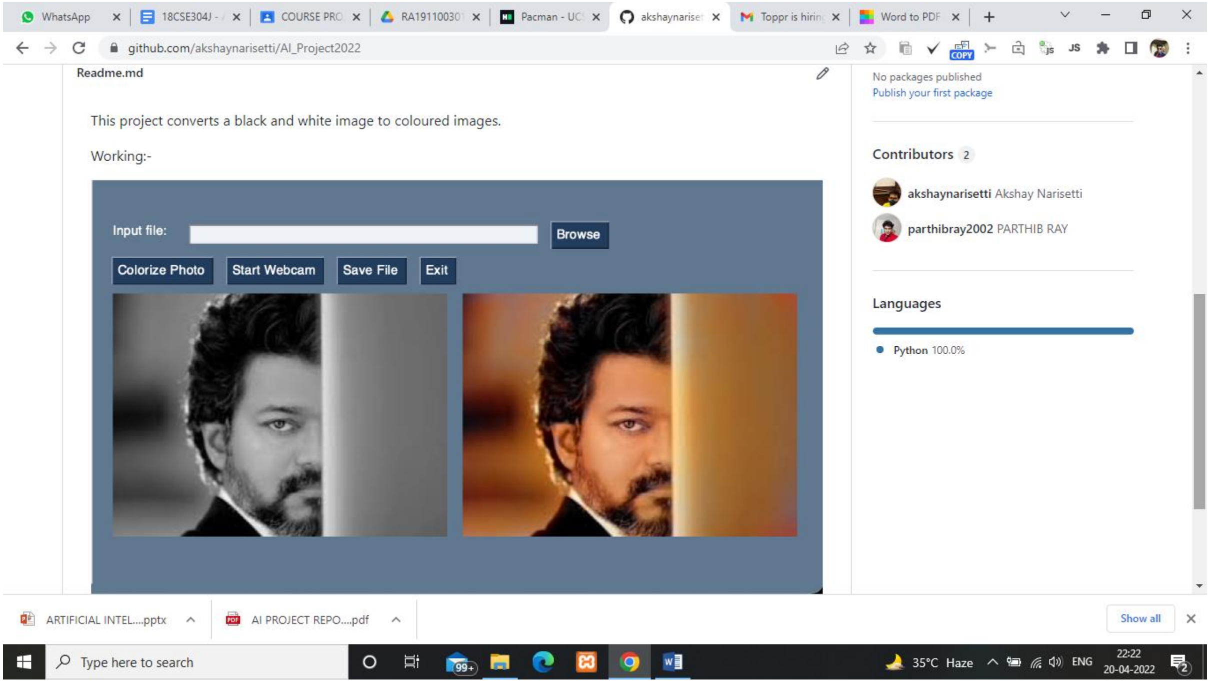
```
  bottom: "conv8_3"
  top: "conv8_313"
  convolution_param {
    num_output: 313
    kernel_size: 1
    stride: 1
    dilation: 1
  }
}
layer {
  name: "conv8_313_rh"
  type: "Scale"
  bottom: "conv8_313"
  top: "conv8_313_rh"
  scale_param {
    bias_term: false
    filler {     type: 'constant'     value: 2.606    }
  }
}
layer {
  name: "class8_313_rh"
  type: "Softmax"
  bottom: "conv8_313_rh"
  top: "class8_313_rh"
}
# *******************
# ***** Decoding *****
# *******************
layer {
  name: "class8_ab"
  type: "Convolution"
  bottom: "class8_313_rh"
  top: "class8_ab"
```

```
  convolution_param {
    num_output: 2
    kernel_size: 1
    stride: 1
    dilation: 1
  }
}
layer {
  name: "Silence"
  type: "Silence"
  bottom: "class8_ab"
}
```

GITHUB UPLOAD SCREENSHOTS

Link:- https://github.com/akshaynarisetti/AI_Project2022




github.com/akshaynarisetti/AI_Project2022
Readme.md
This project converts a black and white image to coloured images.
Working:-
Input file:
Browse
Colorize Photo
Start Webcam
Save File
Exit
No packages published
Publish your first package
Contributors 2
akshaynarisetti Akshay Narisetti
parthibray2002 PARTHIB RAY
Languages
Python 100.0%
ARTIFICIAL INTEL....pptx
AI PROJECT REPO....pdf
Show all
Type here to search
35°C Haze
ENG
22:22
20-04-2022

ARTIFICIAL INTELLIGENCE

PROJECT :- AI IMAGE COLOURIZER USING OPENCV AND DEEP LEARNING

WHY AI IS AN IMPORTANT PART OF EVERYDAY LIFE?

- Artificial Intelligence (AI) and its multiple sub-domains are being increasingly employed in various industries and businesses to aid in repetitive processes. But there has been a burgeoning interest from established tech giants and startups in using AI to make everyday life a walk in the park.
- AI has been highly instrumental in optimizing the way we entertain ourselves, interact with our mobile devices, to even driving vehicles for us. We tend to encounter Machine Learning (ML) algorithms and Natural Language Processing (NLP) in several everyday tasks more than we know.

SOME APPLICATIONS OF AI IN EVERYDAY LIFE

- AI and ML-powered software and devices are mimicking human thought patterns to facilitate the digital transformation of society. AI systems perceive their environment, deal with what they perceive, solve problems and act to help with tasks to make everyday life easier.
- Some applications are as follows:-
- **Voice Assistants**
- **Entertainment Streaming Apps**
- **Personalized Marketing**
- **Smart Input Keyboards**
- **Navigation and Travel and many more**…………. .

AI IMAGE COLOURIZER AND ITS NEED IN TODAYS WORLD

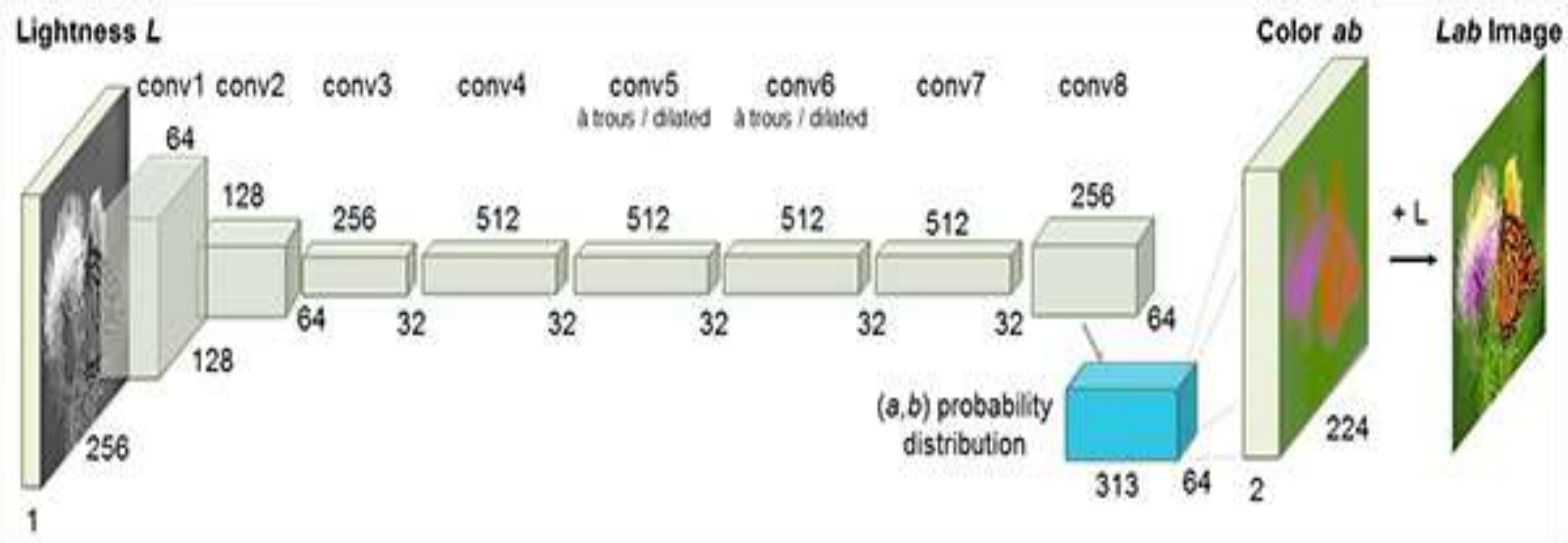
- Image colorization is the process of taking an **input grayscale (black and white) image** and then producing an **output colorized image** that represents the semantic colors and tones of the input (for example, an ocean on a clear sunny day must be plausibly “blue” — it can’t be colored “hot pink” by the model).
- **Previous methods for image colorization either:**

1. **Relied on significant human interaction and annotation**
2. **Produced desaturated colorization**

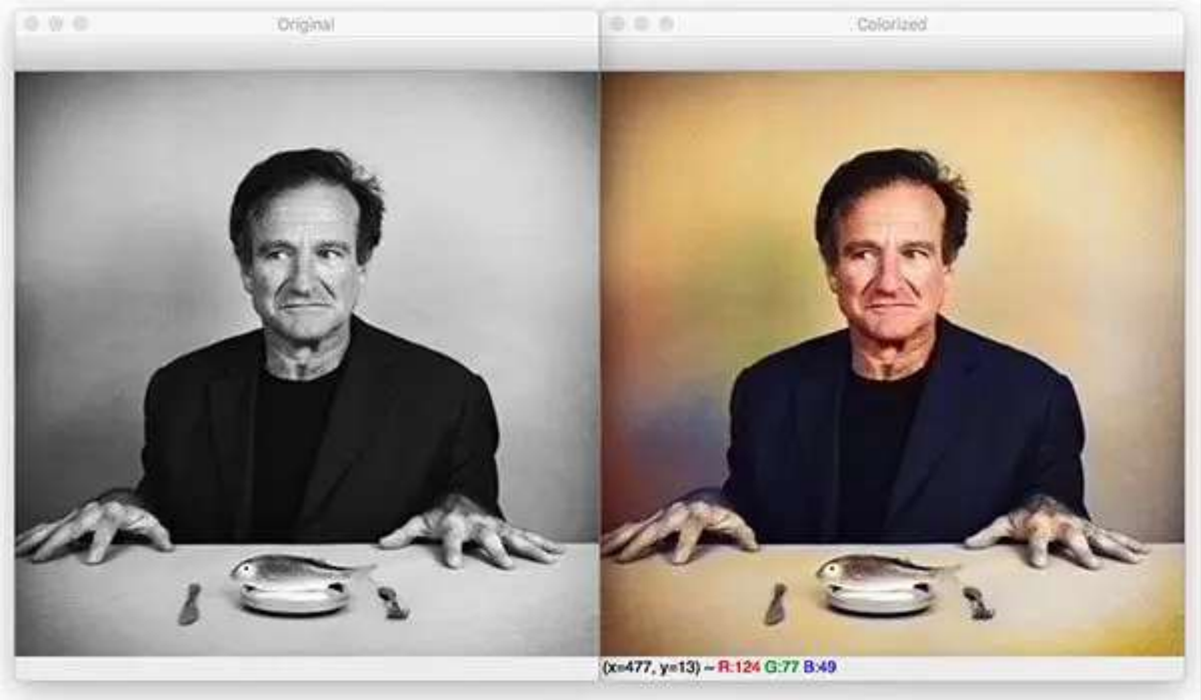
The novel approach we are going to use here today instead relies on deep learning. We will utilize a Convolutional Neural Network capable of colorizing black and white images with results that can even “fool” humans!

This was done to revive many classic photographs, films and videos and make it more vivid and colourful for the human brain to remember better as visual learning is the best learning.

TECHNIQUE GOING ON BEHIND FOR COLOURISING



SOME EXAMPLES OF THE PROCESS WE USED



HOW CAN WE COLORIZE BLACK AND WHITE IMAGES WITH DEEP LEARNING?

- The technique we'll be covering here today is from Zhang et al.'s 2016 ECCV paper, ***Colorful Image Colorization***.
- Zhang et al. decided to attack the problem of image colorization by using Convolutional Neural Networks to "hallucinate" what an input grayscale image would look like when colorized.
- To train the network Zhang et al. started with the **ImageNet dataset** and converted all images from the RGB color space to the **Lab color space.**
- Similar to the RGB color space, the Lab color space has *three channels*. But *unlike* the RGB color space, Lab encodes color information differently:
- The ***L* channel** encodes lightness intensity only
- The ***a* channel** encodes green-red.
- And the ***b* channel** encodes blue-yellow
- 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.**

SUMMARY OF THE ABOVE PROCESS

1. Convert all training images from the RGB color space to the Lab color space.
2. Use the ***L* channel** as the input to the network and train the network to predict the ***ab* channels.**
3. Combine the input ***L* channel** with the predicted ***ab* channels.**
4. Convert the Lab image back to RGB.

- To produce more plausible black and white image colorizations the authors also utilize a few additional techniques including mean annealing and a specialized loss function for color rebalancing

PROJECT STRUCTURE

- Create the source code , models to be trained and the images to be colourized in a particular folder and open it using Jupyter notebook or Google Colab
- use the **tree** command to inspect the project structure.
- We have four sample black and white images in the images/ directory.
- Our Caffe model and prototxt are inside the model/ directory along with the cluster points NumPy file.
- We'll be reviewing two scripts today:
- **bw2color_image.py**
- **bw2color_video.py**
- The **image** script can process any black and white (also known as grayscale) image you pass in.
- Our **video** script will either use your webcam or accept an input video file and then perform colorization.

PROJECT STRUCTURE (CONTD)

Black and white image colorization with OpenCV and Deep Learning

```
$ tree --dirsfirst
.
├── images
│   ├── adrian_and_janie.png
│   ├── albert_einstein.jpg
│   ├── mark_twain.jpg
│   └── robin_williams.jpg
├── model
│   ├── colorization_deploy_v2.prototxt
│   ├── colorization_release_v2.caffemodel
│   └── pts_in_hull.npy
├── bw2color_image.py
└── bw2color_video.py

2 directories, 9 files
```

COLORIZING BLACK AND WHITE IMAGES WITH OPENCV

- Let’s go ahead and implement black and white image colorization script with OpenCV.

Black and white image colorization with OpenCV and Deep Learning

```python
# import the necessary packages
import numpy as np
import argparse
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", type=str, required=True,
	help="path to input black and white image")
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")
args = vars(ap.parse_args())
```

Our colorizer script only requires three imports: NumPy, OpenCV, and argparse

.

Let’s go ahead and **use `argparse` to parse command line arguments**. 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.
- Let’s go ahead and load our model and cluster centers into memory:

Black and white image colorization with OpenCV and Deep Learning

```
18. # load our serialized black and white colorizer model and cluster
19. # center points from disk
20. print("[INFO] loading model...")
21. net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
22. pts = np.load(args["points"])
23.
24. # add the cluster centers as 1x1 convolutions to the model
25. class8 = net.getLayerId("class8_ab")
26. conv8 = net.getLayerId("conv8_313_rh")
27. pts = pts.transpose().reshape(2, 313, 1, 1)
28. net.getLayer(class8).blobs = [pts.astype("float32")]
29. net.getLayer(conv8).blobs = [np.full([1, 313], 2.606, dtype="float32")]
```

- **Line 21** loads our Caffe model directly from the command line argument values. OpenCV can read Caffe models via the cv2.dnn.readNetFromCaffe function.
- **Line 22** then loads the cluster center points directly from the command line argument path to the points file. This file is in NumPy format so we’re using np.load.
- **Line 25-29** Load centers for ***ab* channel** quantization used for rebalancing.
- Treat each of the points as *1×1* convolutions and add them to the model.

- Now let's load, scale, and convert our image:

Black and white image colorization with OpenCV and Deep Learning

```
31. # load the input image from disk, scale the pixel intensities to the
32. # range [0, 1], and then convert the image from the BGR to Lab color
33. # space
34. image = cv2.imread(args["image"])
35. scaled = image.astype("float32") / 255.0
36. lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)
```

- To load our input image from the file path, we use cv2.imread on **Line 34**.
- Preprocessing steps include:
- Scaling pixel intensities to the range *[0, 1]* (**Line 35**).
- Converting from BGR to Lab color space (**Line 36**).

Black and white image colorization with OpenCV and Deep Learning

```
# resize the Lab image to 224x224 (the dimensions the colorization
# network accepts), split channels, extract the 'L' channel, and then
# perform mean centering
resized = cv2.resize(lab, (224, 224))
L = cv2.split(resized)[0]
L -= 50
```

- We'll go ahead and resize the input image to 224×224 (**Line 41**), the required input dimensions for the network.
- Then we grab the L channel only (i.e., the input) and perform mean subtraction (**Lines 42 and 43**).
- Now we can pass the **input *L* channel** through the network to **predict**

Black and white image colorization with OpenCV and Deep Learning

```python
45. # pass the L channel through the network which will *predict* the 'a'
46. # and 'b' channel values
47. 'print("[INFO] colorizing image...")'
48. net.setInput(cv2.dnn.blobFromImage(L))
49. ab = net.forward()[0, :, :, :].transpose((1, 2, 0))
50.
51. # resize the predicted 'ab' volume to the same dimensions as our
52. # input image
53. ab = cv2.resize(ab, (image.shape[1], image.shape[0]))
```

- A forward pass of the L channel through the network takes place on **Lines 48 and 49**
- Notice that after we called net.forward , on the same line, we went ahead and extracted the predicted ab volume

POST PROCESSING OF IMAGE

- Grabbing the L channel from the ***original*** input image (**Line 58**) and concatenating the original L channel and ***predicted*** ab
- channels together forming colorized (**Line 59**).
- Converting the colorized image from the Lab color space to RGB (**Line 63**).
- Clipping any pixel intensities that fall outside the range *[0, 1]* (**Line 64**).
- Bringing the pixel intensities back into the range *[0, 255]* (**Line 69**). During the preprocessing steps (**Line 35**) we divided by 255 and now we are multiplying by 255.
- We've also found that this scaling and "uint8" conversion isn't a requirement but that it helps the code work between **OpenCV 3.4.x** and **4.x** versions.

PURPOSE OF THE PROJECT

- AI Image colourisation using Open CV and Deep learning will reduce the painstaking task of colouring images using human annotations and other traditional techniques.
- It will help us preserve many archival videos, photos etc in better quality which may have lost their quality due to not so good colouring techniques in the past.
- It will also help us bring us life a lot of black and white images for better understanding etc.
- Last but not least it will help us colourise personal photos of our forefathers for better display and preservation.

TEAM MEMBERS:-

- Akshay Narisetti (RA1911003010684)
- Parthib Ranjan Ray (RA1911003010664)
- Pushkar Sinha (RA1911003010676)
- Rohan Garodia (RA1911003010680)