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
  }
}
```

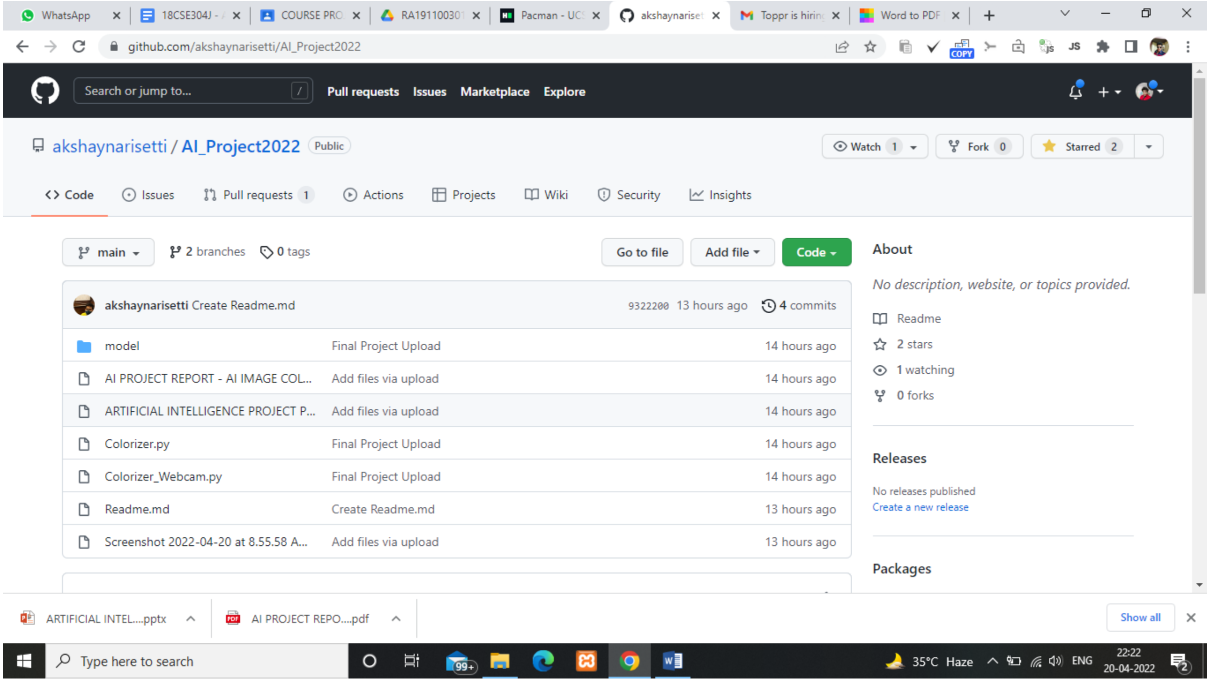
```
layer {
  name: "relu6_2"
  type: "ReLU"
  bottom: "conv6_2"
  top: "conv6_2"
}
layer {
  name: "conv6_3"
  type: "Convolution"
  bottom: "conv6_2"
  top: "conv6_3"
  convolution_param {
    num_output: 512
    kernel_size: 3
    pad: 2
    dilation: 2
  }
}
layer {
  name: "relu6_3"
  type: "ReLU"
  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



akshaynarisetti / AI_Project2022 Public

akshaynarisetti Create Readme.md 9322200 13 hours ago 4 commits

model	Final Project Upload	14 hours ago
AI PROJECT REPORT - AI IMAGE COL...	Add files via upload	14 hours ago
ARTIFICIAL INTELLIGENCE PROJECT P...	Add files via upload	14 hours ago
Colorizer.py	Final Project Upload	14 hours ago
Colorizer_Webcam.py	Final Project Upload	14 hours ago
Readme.md	Create Readme.md	13 hours ago
Screenshot 2022-04-20 at 8.55.58 A...	Add files via upload	13 hours ago

About

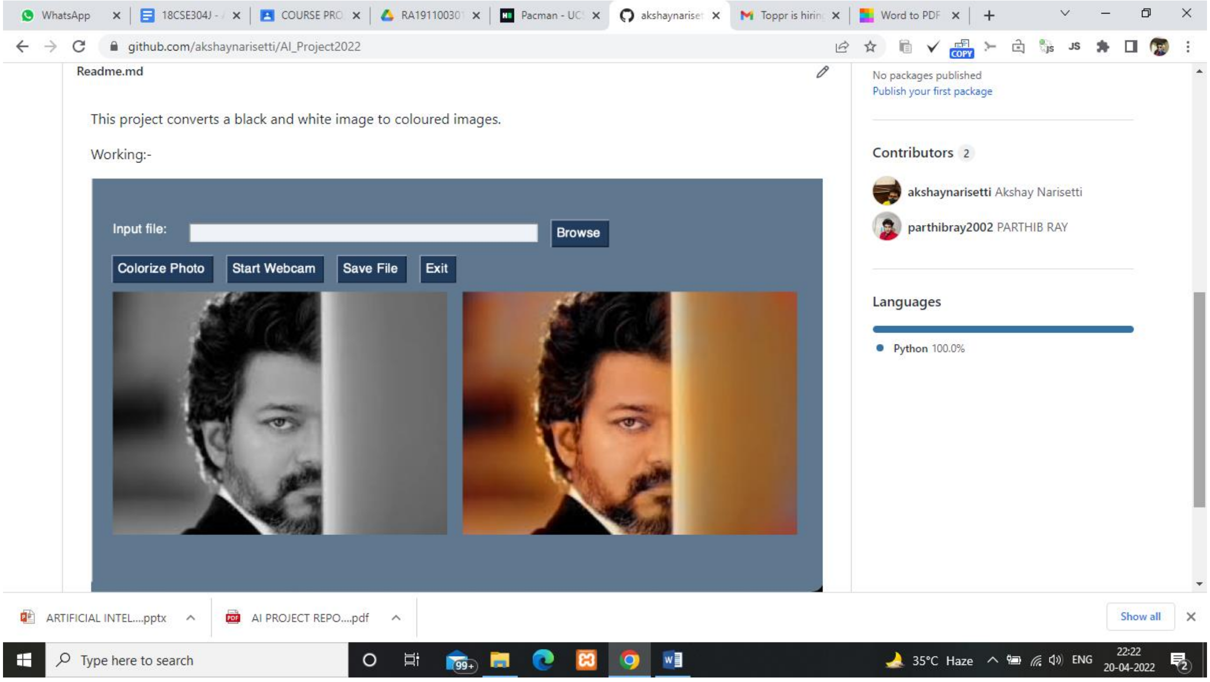
No description, website, or topics provided.

Readme
2 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages



Readme.md

This project converts a black and white image to coloured images.

Working:-

Input file:

No packages published

Publish your first package

Contributors 2

akshaynarisetti Akshay Narisetti

parthibray2002 PARTHIB RAY

Languages

● Python 100.0%