Image Enhancement with Super Resolution and B/W to Color Images using Python

A Special Assignment Report

Submitted in the
Open Elective Course

Pattern Recognition And Image Analysis

By

Nisarg Patel, Raj Parekh
18BCE136, 18BCE143
B.Tech Sem VII
Computer Science and Engineering

INSTRUMENTATION AND CONTROL ENGINEERING DEPARTMENT
INSTITUTE OF TECHNOLOGY, NIRMA UNIVERSITY
AHMEDABAD-382481
OCTOBER 2021

Abstract

Image enhancement is the process of adjusting digital images so that the results are more suitable for display or further image analysis. For example, you can remove noise, sharpen, or brighten an image, making it easier to identify key features. Nowadays, in the era of high-resolution images, enhancement of images in the context of resolution is required. If we consider images that are 4 decades older than that images are black and white in color. So, the colorization of images is also an important process under image enhancement. The existing algorithms give results but they are not quite pleasant to see. For this type of process, newly introduced algorithms called GAN can be useful. In this report, we will see its working and compare results with existing algorithms.

Keywords: Image Enhancement, Image Super Resolution, Image Colorization, Generative Adversial Network, Mean Squared Error, Regression Model, Machine Learning, Deep Learning, Image Processing.

1. Introduction

Image enhancement is the procedure of improving the quality and information content, surveyed original data before processing. Common practices include contrast enhancement, spatial filtering, density slicing, and FCC. Contrast enhancement or stretching is performed by linear transformation expanding the original range of gray level. Spatial filtering improves the naturally occurring linear features like fault, shear zones, and lineaments. Density slicing converts the continuous gray tone range into a series of density intervals marked by a separate color or symbol to represent different features [1]. In the era of high-resolution images line 1920*1080 or 4k images. So, the super-resolution of images should be considered in image enhancement. There are several existing mathematical-based algorithms, we can implement this easily in python through [2]. And the comparative study was done by the authors of [3].

The consideration of image colorization algorithms is also required for today's image enhancement purpose. Companies and research organizations can save a lot of data by storing the image in black and white format, and colorize them at the time of requirement. The traditional algorithm does not give pleasant images. So we thought of using a newly introduced deep learning-based algorithm call Generative Adversarial Network [4] for carrying out both of these tasks. In this report, we will discuss the implementation part and comparative studies.

1.1. Problem Statement

Our aim is to take low resolution, black and white images from the user and gave them high resolution colorized images, which will look realistic and pleasant in nature which is the main motive for image enhancement purpose. This kind of technology is much needed in the field of medical science, space research field, nanotechnology field, and many more.

1.2. Objective

Our objective is to give deep-learning based algorithm which will enhance the image using low computational power and visual appealing images. And the comparison of results with existing work for our test image.

2. Literature Surveyed

As part of our literature survey, for image super-resolution, we have considered work based on interpolation methods like the nearest neighbor, bilinear and bicubic. And compared the results with the output from GAN. For the work of model selection, we have explored few research papers to get the working idea [5][6][7][8][9], and among them, we have narrow down ourself to [9] for super-resolution task. The open-source implementation of this paper is also done by the community to save our time we have directly used their weights to build our model [10]. The author of the paper [5], Chao Dong et al. had done a comparative study of the usage of Deep learning techniques for super-resolution of images. In that paper, Chao Dong et al. have implemented Super-Resolution Convolutional Neural Network (SRCNN) model on different channels like YCbCr, RGB, only Y, etc. In the paper [6], the author has implemented Deep learning-based and Transfer learning-based super-resolution reconstruction from only one image of the medical field where paper intended to make medical images more clear and give the doctor a better understanding of MRI and CT-scan images. Shreyas in [8] have done a comparative study on interpolation techniques used for super-resolution task and compared the results. Equation 1 and 2 shows the formula of Bi-Linear and Bi-cubic interpolation techniques respectively.

$$V(x,y) = ax + by + cxy + d \tag{1}$$

$$V(x,y) = \sum\sum a_{ij}x^i y^j \ ,(i,j = 0 \ to \ 3) \tag{2}$$

As we have a lack of computation power to deal with a huge dataset like Imagenet and DIV2K for training our GAN

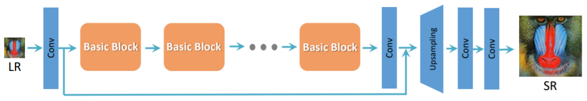


Figure 1: Image represenation of ESRGAN[9]

we used existing weights from [10]. Within the implementation part, they have 4 pre-trained models, and among we have used RRDN GAN which is based on a residual block where there is a skip connection among the layers. With that, we have also used the noise-cancel module as well for basic enhancement purposes.

As a part of image colorization, we have found an interesting paper where they have done a brief survey of each type of Generative Models and simple Regression models used for colorizing images with a detailed study of each type with a good comparison [11]. The paper was published in 2020 so we have the latest literature survey for the image colorization process through GAN and regressions. Among them, we have chosen a simple regression-based model for quick implementation and fast computation which is derived from VGG-16 consist of 16 neurons made by Oxford [12].

3. Proposed Solution

To approach this problem we will use the GAN-based algorithm. The GAN consist of two networks namely the Generative network and the Discriminative network. The job of the first network is to create fake images which look realistic in nature. And the job of the second neural network is to check whether the images are fake or real image. Both networks try to optimize themselves by conflicting results of each other by the following process of nash equilibrium [13]. For our task, there will be one generative network used for image super-resolution and the other will be a VGG-16 based regression model. Figure 1 is the generator for image super-resolution. To save computational power while processing, firstly the image will be a pass for colorization and after that, we will pass the newly created image to a super-resolution network. As discussed above we have used the open-source implementation for image super-resolution contributed by [10]. And for image colorization, we have used VGG-16 based regression model optimized using Adam for Mean Squared Error loss. MSE is calculated for each channel and sum it up to get the whole loss. Figure 2 represents the procedure carried out by our algortihm. Below is the mathematical equation for custom MSE as:

$$MSE_{Red\ Channel} = \frac{1}{i*j}\sum_{r_1=0}^{i}\sum_{r_2=0}^{j}\left(\hat{p}_{(r_1,r_2)} - p_{(r_1,r_2)}\right)^2 \qquad (3)$$

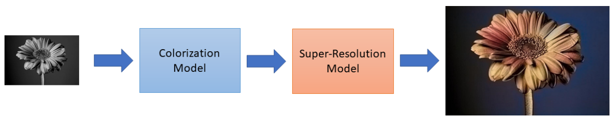


Figure 2: Algorithmic flow.

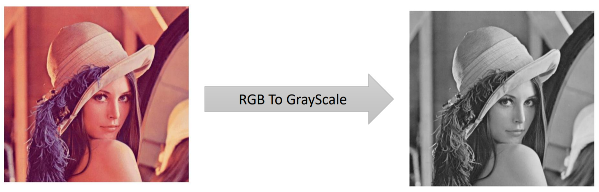


Figure 3: RGB to GrayScale Conversion

$$MSE_{Green\ Channel} = \frac{1}{i*j}\sum_{g_1=0}^{i}\sum_{g_2=0}^{j}\left(\hat{p}_{(g_1,g_2)} - p_{(g_1,g_2)}\right)^2 \qquad (4)$$

$$MSE_{Blue\ Channel} = \frac{1}{i*j}\sum_{b_1=0}^{i}\sum_{b_2=0}^{j}\left(\hat{p}_{(b_1,b_2)} - p_{(b_1,b_2)}\right)^2 \qquad (5)$$

$$Total\ Loss = [Eq\,1\ +\ Eq\,2\ +\ Eq\,3] \qquad (6)$$

where $p_{(i,j)}$ represent the pixel value than i,j represents the number of rows and column and final loss is the combination of losses for each channel from RGB. And this loss was minimized using Adam optimizer.

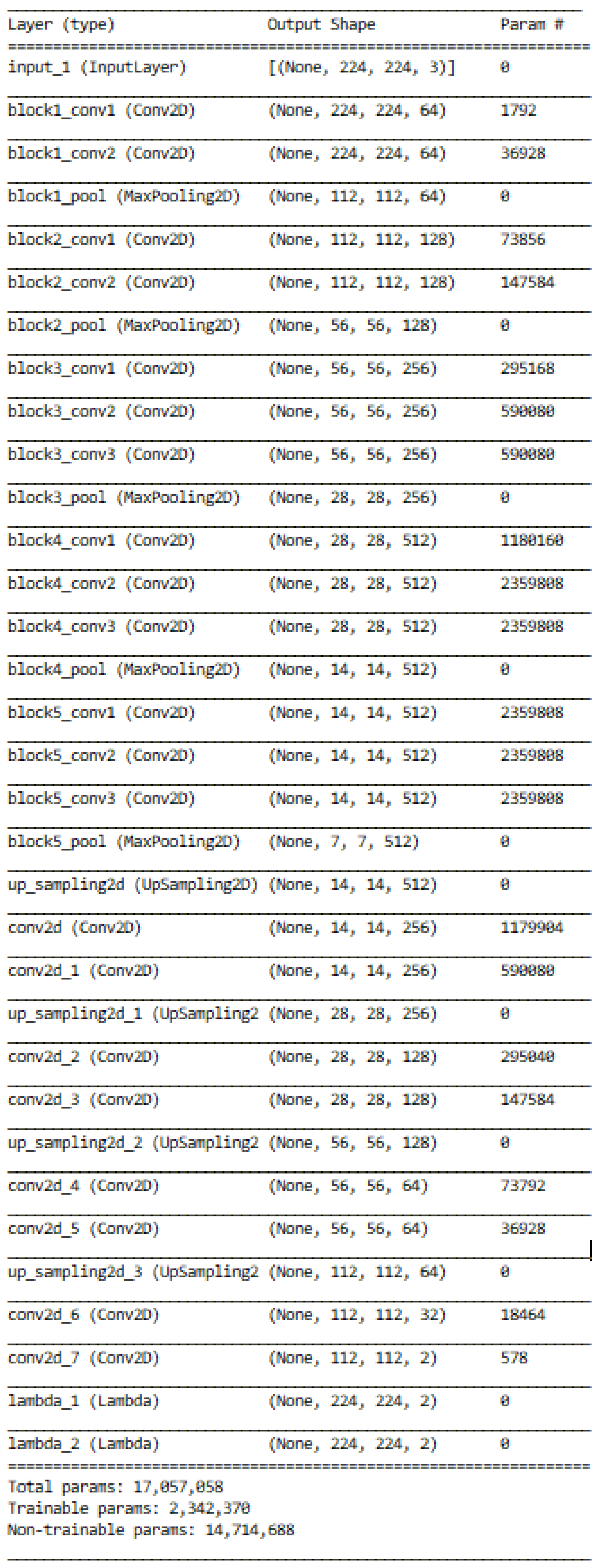
For preparing the dataset to train the regression-based model. We have selected random 1000 images from Imagenet which have 3 channels i.e. RGB. We have created x,y tuple where y is the original image i.e. colorized image from model and x is the black and white version of the original image converted using [14]. Figure 3 is the example for creating a dataset using the OpenCV module. We have explored nearly 4 different model the biggest model has nearly 148 million parameters. The summary of the smallest regression based model is shown in Figure 4 becuase of page constraint.

4. Implementation Results

In this section, we will analyze the performance of our model. There cannot be any statistical significance until the results are appealing in the context of Image enhancement. As in image enhancement, visual appealing is the most significant. The minimum MSE loss for our colorization model is 856.92. Below is the output for the model. The images used to generate output are out of the training set so there can be the proper justification of the model in a real-world scenario.

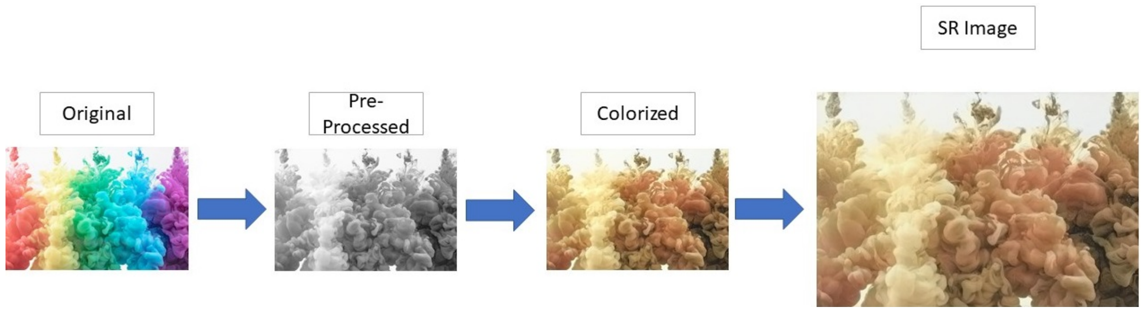
5. Practical Applications

* *Medicinal area*: Image super-resolution algorithms will be helpful to magnify images of the brain to detect a Tumour or for detecting Pneumonia from X-ray images.

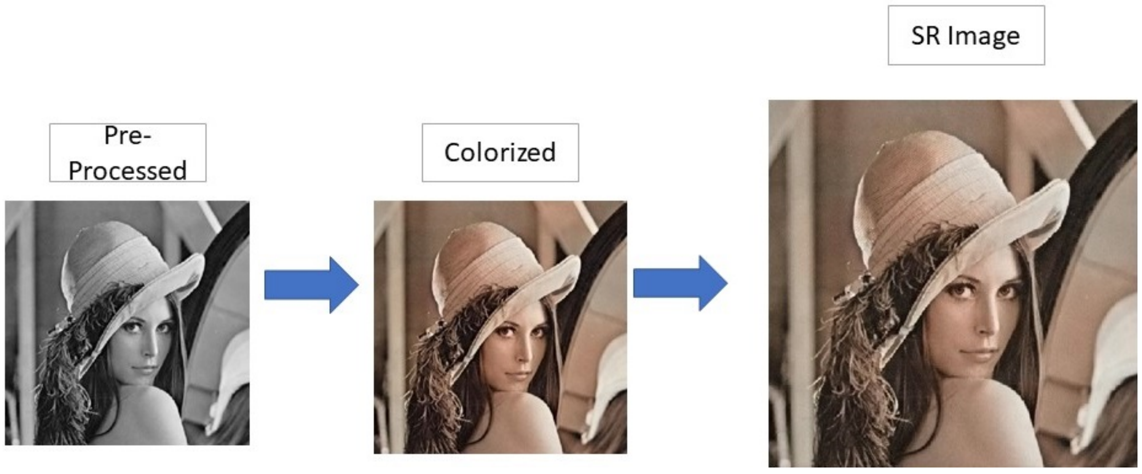


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0         
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 14, 14, 512)       0         
_________________________________________________________________
conv2d (Conv2D)              (None, 14, 14, 256)       1179904   
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 256)       590080    
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 28, 28, 256)       0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       295040    
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 128)       147584    
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 56, 56, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 56, 56, 64)        73792     
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 64)        36928     
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 112, 112, 64)      0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 112, 112, 32)      18464     
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 112, 112, 2)       578       
_________________________________________________________________
lambda_1 (Lambda)            (None, 224, 224, 2)       0         
_________________________________________________________________
lambda_2 (Lambda)            (None, 224, 224, 2)       0         
=================================================================
Total params: 17,057,058
Trainable params: 2,342,370
Non-trainable params: 14,714,688
_________________________________________________________________
```

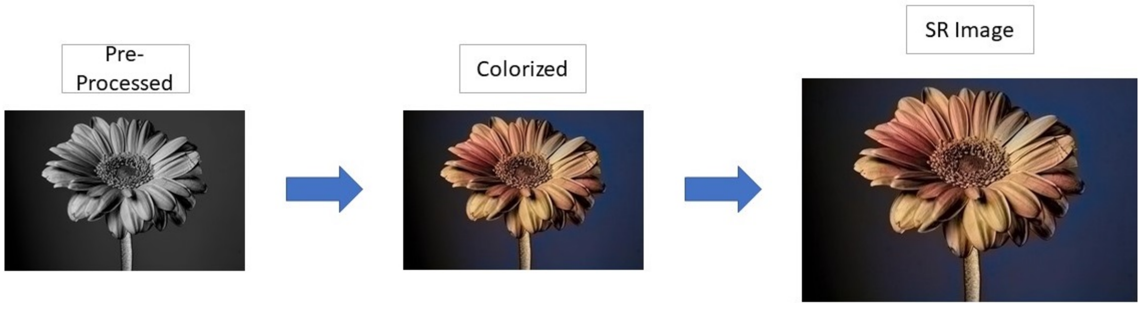
Figure 4: Model Summary



(a) Image colorization and super resolution of complex colored image from net [15]



(b) Image colorization and super resolution of Lenna Image



(c) Image colorization and super resolution of random image for google images.

Figure 5: Practical Outputs from our model

∗ *Storage aspects*: Images can be stored in form of grayscale and low-resolution format and while its usage we can retrieve our original image.

∗ *Restoration of old images*: With the combination of noise removal, image colorization can help to colorized images of the childhood of our parents, and it's a fun activity among family members in this hard situation and create a slideshow type of video.

6. Conclusion

In this report, we discussed new image enhancement algorithms i.e. Image enhancement and colorization. Enhancement is a must in today's scenario as a small mobile device is supporting 1920*1080 images i.e. Full HD and future is of 4K, so this type of enhancement can improve the performance of the camera, as images can be capture in low resolution and after processing high-resolution image can be given to the user. The super-resolution algorithm is an unsupervised GAN-based algorithm, and we have a short study related to it. After that, we have a brief discussion on image colorization. We have used a regression-based algorithm on 1000 images dataset for this task. The algorithm of modern image enhancement first converts b/w image to color image and then performs a super-resolution algorithm on it. At last, we have an overview of implementation results and its practical applications.

7. Future Scope

The task of colorization was carried out using 1000 images only. So to improve it we can use large datasets like DIV2K or Imagenet to train our model. As well the model of colorization was regression-based which can be improved using GAN-based architecture which further can be improved with segmentation+generation. Because of lack of resources we have used open-source implementation of ESRGAN this algorithm can also be improvised by learning on custom datasets like X-ray images where super-resolution can be helpful.

8. Acknowledgement

The project and it's report is build under the guidance of Dr. Ankit Sharma. This work was supported by the Department of ICE and CSE of Nirma University, Ahmedabad Gujarat India.

9. Code Screenshot

Preprocessing code for converting colored to grayscale image for training process is shown through Figure 6a. Generated image will be saved at required directory for further process.

Utitlity code like loading images, making tuples and resizing of image can be done through the fuction shown in Figure 6b.

Code for loading model and using it for colorizing black and white images is shown in 6c. Code for accessing open source Image Super Resolution model in shown in Figure 6d.

Code for making model of 148 million parameters is shown through Figure 7.

References

[1] S. Haldar, "Chapter 6 - photogeology, remote sensing and geographic information system in mineral exploration," in *Mineral Exploration* (S. Haldar, ed.), pp. 95–115, Boston: Elsevier, 2013.

[2] Open-CV, "Geometric Image Transformations." `https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html`, 2021.

[3] K. Vijayaraghavan, K. Parpyani, S. A. Thakwani, A. D, and N. Iyengar, "Methods of increasing spatial resolution of digital images with minimum detail loss and its applications," in *2009 Fifth International Conference on Image and Graphics*, pp. 685–689, 2009.

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[5] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[6] Y. Zhang and M. An, "Deep learning- and transfer learning-based super resolution reconstruction from single medical image," *Journal of Healthcare Engineering*, vol. 2017, 07 2017.

[7] Z. Zhang, Z. Wang, Z. Lin, and H. Qi, "Image super-resolution by neural texture transfer," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 7974–7983, IEEE Computer Society, jun 2019.

[8] S. Fadnavis, "Image interpolation techniques in digital image processing: An overview," *International Journal Of Engineering Research and Application*, vol. 4, pp. 2248–962270, 11 2014.

[9] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, C. C. Loy, Y. Qiao, and X. Tang, "Esrgan: Enhanced super-resolution generative adversarial networks," 2018.

[10] F. C. et al., "Isr." `https://github.com/idealo/image-super-resolution`, 2018.

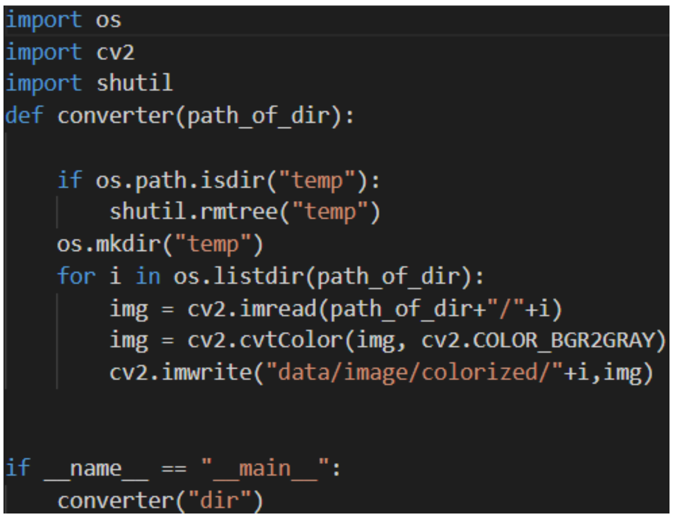
[11] S. Anwar, M. Tahir, C. Li, A. Mian, F. S. Khan, and A. W. Muzaffar, "Image colorization: A survey and dataset," 2020.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[13] J. Hui, "Gan — why it is so hard to train generative adversarial networks!." `https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b`, 2018.

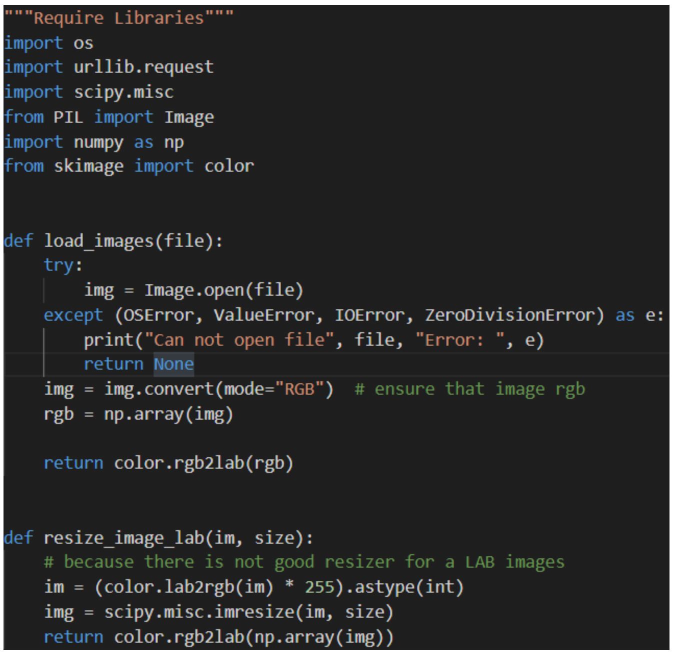
[14] Open-CV, "Color conversions." `https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html`, 2021.

[15] Unsplash. `https://www.google.com/url?sa=i&url=https\%3A\%2F\%2Funsplash.com\%2Fs\%2Fphotos\%2Fcolor&psig=AOvVaw1EHWRFkGGgrhDEl-0-jNJE&ust=1619370585826000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCLivh9avl_ACFQAAAAAdAAAAABAD`.



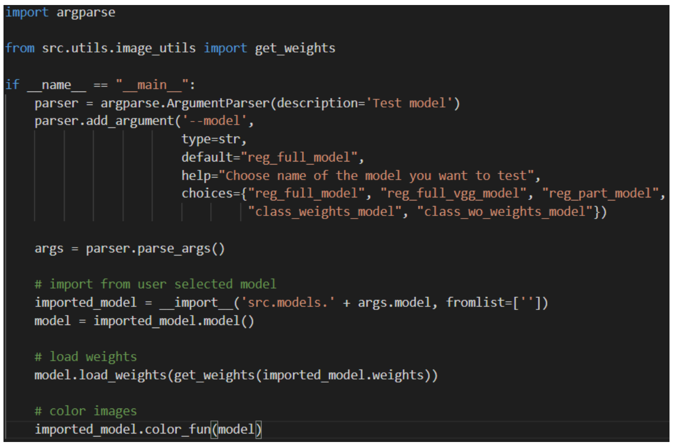
```python
import os
import cv2
import shutil
def converter(path_of_dir):

    if os.path.isdir("temp"):
        shutil.rmtree("temp")
    os.mkdir("temp")
    for i in os.listdir(path_of_dir):
        img = cv2.imread(path_of_dir+"/"+i)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        cv2.imwrite("data/image/colorized/"+i,img)


if __name__ == "__main__":
    converter("dir")
```

(a) Image pre-processing code



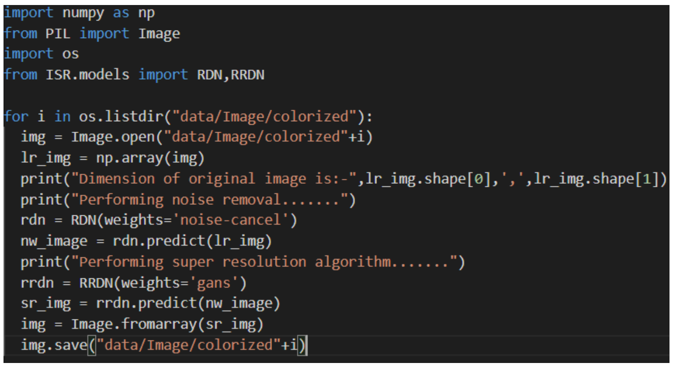
```python
"""Require Libraries"""
import os
import urllib.request
import scipy.misc
from PIL import Image
import numpy as np
from skimage import color


def load_images(file):
    try:
        img = Image.open(file)
    except (OSError, ValueError, IOError, ZeroDivisionError) as e:
        print("Can not open file", file, "Error: ", e)
        return None
    img = img.convert(mode="RGB")  # ensure that image rgb
    rgb = np.array(img)

    return color.rgb2lab(rgb)


def resize_image_lab(im, size):
    # because there is not good resizer for a LAB images
    im = (color.lab2rgb(im) * 255).astype(int)
    img = scipy.misc.imresize(im, size)
    return color.rgb2lab(np.array(img))
```

(b) Image utils code



```python
import argparse

from src.utils.image_utils import get_weights

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Test model')
    parser.add_argument('--model',
                        type=str,
                        default="reg_full_model",
                        help="Choose name of the model you want to test",
                        choices={"reg_full_model", "reg_full_vgg_model", "reg_part_model",
                                 "class_weights_model", "class_wo_weights_model"})

    args = parser.parse_args()

    # import from user selected model
    imported_model = __import__('src.models.' + args.model, fromlist=[''])
    model = imported_model.model()

    # load weights
    model.load_weights(get_weights(imported_model.weights))

    # color images
    imported_model.color_fun(model)
```

(c) Colorization code



```python
import numpy as np
from PIL import Image
import os
from ISR.models import RDN,RRDN

for i in os.listdir("data/Image/colorized"):
  img = Image.open("data/Image/colorized"+i)
  lr_img = np.array(img)
  print("Dimension of original image is:-",lr_img.shape[0],',',lr_img.shape[1])
  print("Performing noise removal.......")
  rdn = RDN(weights='noise-cancel')
  nw_image = rdn.predict(lr_img)
  print("Performing super resolution algorithm.......")
  rrdn = RRDN(weights='gans')
  sr_img = rrdn.predict(nw_image)
  img = Image.fromarray(sr_img)
  img.save("data/Image/colorized"+i)
```

(d) ISR code

Figure 6: Screenshots of Code

```
 from keras.applications import VGG16
 from keras.engine import Model
 from keras import backend as K, Input
 from keras import optimizers
 from keras.layers import Conv2D, Lambda, Dense, concatenate,
 regularizers, add, Conv2DTranspose, MaxPooling2D
 from src.image_colorization.test import color_images_full
 input_shape = (224, 224, 1)
 weights = "data/weights/imp9-full.h5"
 color_fun = color_images_full

 def model():
     main_input = Input(shape=input_shape, name='image_part_input')
     x = Conv2D(64, (3, 3), padding="same", activation="relu",
                kernel_regularizer=regularizers.l2(0.01),
 name="conv1")(main_input)
     x = MaxPooling2D((2, 2), strides=(2, 2))(x)
     x1 = Conv2D(128, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01), name="conv2")(x)
     x = Conv2D(128, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01), name="conv3")(x1)
     x = Conv2D(128, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01), name="conv4")(x)
     x = add([x, x1])
     x = Conv2D(128, (3, 3), padding="same", activation="relu",
                kernel_regularizer=regularizers.l2(0.01), name="conv5")(x)
     x = MaxPooling2D((2, 2), strides=(2, 2))(x)
     x1 = Conv2D(256, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01), name="conv6")(x)
     x = Conv2D(256, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01), name="conv7")(x1)
     x = Conv2D(256, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01), name="conv8")(x)
     x = add([x, x1])
     x = Conv2D(256, (3, 3), padding="same", activation="relu",
                kernel_regularizer=regularizers.l2(0.01))(x)
     x = MaxPooling2D((2, 2), strides=(2, 2))(x)
     x1 = Conv2D(512, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01))(x)
     x = Conv2D(512, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01))(x1)
     x = Conv2D(512, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01))(x)
     x = add([x, x1])
     x = Conv2D(512, (3, 3), padding="same", activation="relu",
 kernel_regularizer=regularizers.l2(0.01))(x)
     main_output = Conv2D(256, (3, 3), padding="same", activation="relu",
                          kernel_regularizer=regularizers.l2(0.01))(x)
     # VGG
     vgg16 = VGG16(weights="imagenet", include_top=True)
     vgg_output = Dense(256, activation='relu',
 name='predictions')(vgg16.layers[-2].output)
     def repeat_output(input):
         shape = K.shape(x)


        return K.reshape(K.repeat(input, 28 * 28), (shape[0], 28, 28,
256))
    vgg_output = Lambda(repeat_output)(vgg_output)
    # freeze vgg16
    for layer in vgg16.layers:
        layer.trainable = False
    # concatenated net
    merged = concatenate([vgg_output, main_output], axis=3)
    last = Conv2D(128, (3, 3), padding="same")(merged)
    last = Conv2DTranspose(64, (3, 3), strides=(2, 2), padding="same",
activation="relu",

kernel_regularizer=regularizers.l2(0.01))(last)
    last = Conv2D(64, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.01))(last)
    last = Conv2D(64, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.01))(last)
    last = Conv2DTranspose(64, (3, 3), strides=(2, 2), padding="same",
activation="relu",

kernel_regularizer=regularizers.l2(0.01))(last)
    last = Conv2D(32, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.01))(last)
    last = Conv2D(2, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.01))(last)
    def resize_image(x):
        return K.resize_images(x, 2, 2, "channels_last")
    def unormalise(x):
        # outputs in range [0, 1] resized to range [-100, 100]
        return (x * 200) - 100
    last = Lambda(resize_image)(last)
    last = Lambda(unormalise)(last)
    def custom_mse(y_true, y_pred):
        return K.mean(K.square(y_pred - y_true), axis=[1, 2, 3])
    model = Model(inputs=[main_input, vgg16.input], output=last)
    opt = optimizers.Adam(lr=1E-4, beta_1=0.9, beta_2=0.999, epsilon=1e-
08)
    model.compile(optimizer=opt, loss=custom_mse)
    model.summary()
    model.name = "reg_full"
    return model
```

Figure 7: Regression Model