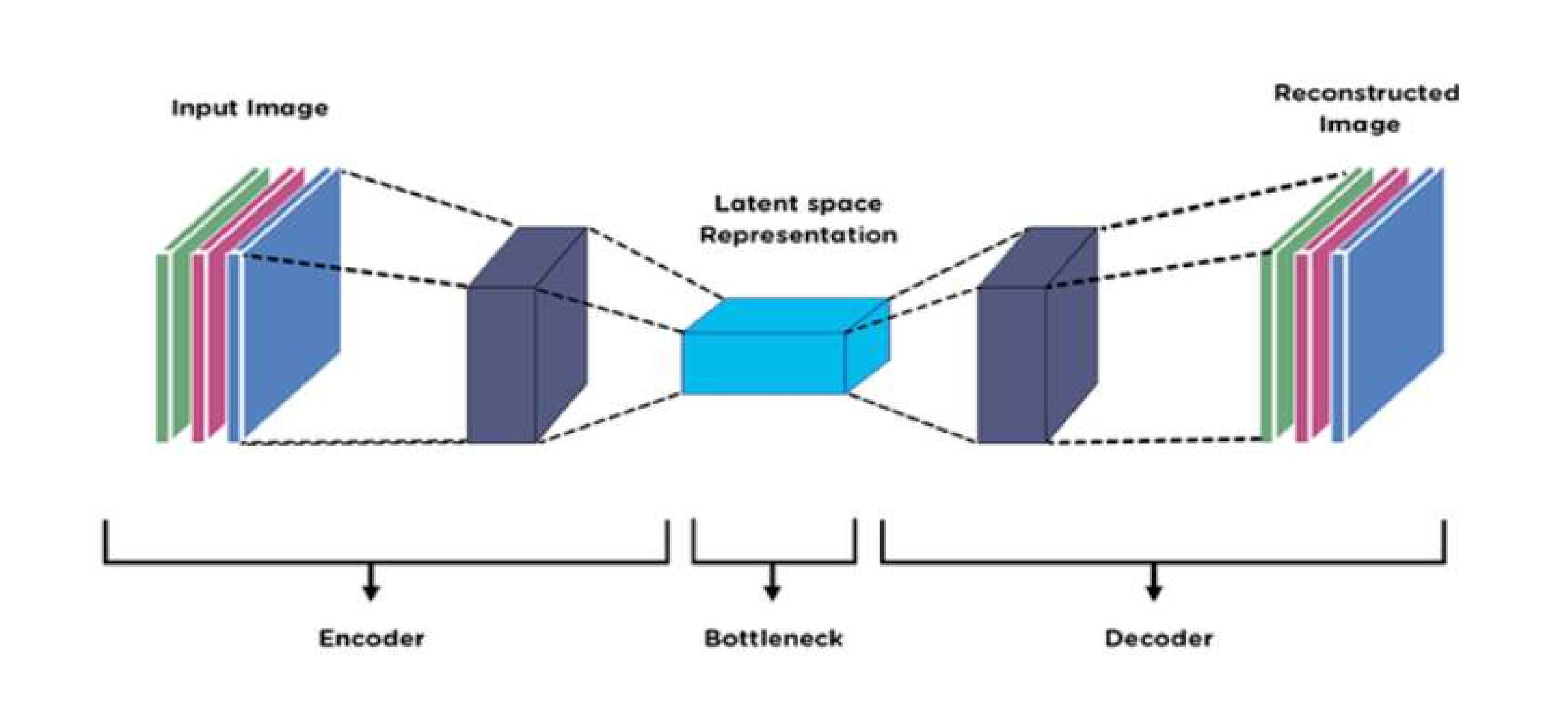
Autoencoder:Grayscale to color image

AM202126802
오지훈

Image Colorization

Image colorization using different softwares require large amount of human effort, time and skill.But special type of deep learning architecture called autoencoder has made this task quiet easy. Automatic image colorization often involves the use of a class of convolutional neural networks (CNN) called autoencoders. These neural networks are able to distill the salient features of an image, and then regenerate the image based on these learned features.

Introduction



Autoencoder are special type of deep learning architecture that consist of two networks encoder and decoder. The encoder, through a series of CNN and downsampling, learns a reduced dimensional representation of the input data while decoder through the use of CNN and upsampling, attempts to regenerate the data from the these representations. A well-trained decoder is able to regenerated data that is identical or as close as possible to the original input data. Autoencoder are generally used for anamoly detection, denoising image, colorizing the images. Here, i am going to colorize the landscape images using autoencoder.

Import necessary libraries

```
import numpy as np
import tensorflow as tf
import keras
import cv2
from keras.layers import MaxPool2D,Conv2D,UpSampli
ng2D,Input,Dropout
from keras.models import Sequential
from keras.preprocessing.image import img_to_array
import os
from tqdm import tqdm
import re
import matplotlib.pyplot as plt
```

Code_Getting landscape image data,resizing them and appending in array

```
def sorted_alphanumeric(data):
    convert = lambda text: int(text) if text.isdigit() else text.lower()
    alphanum_key = lambda key: [convert(c) for c in re.split('([0-9]+)',key)]
    return sorted(data,key = alphanum_key)
# defining the size of the image
SIZE = 160
color_img = []
path = '../input/landscape-image-colorization/landscape Images/color'
files = os.listdir(path)
files = sorted_alphanumeric(files)
for i in tqdm(files):
    if i == '6000.jpg':
        break
    else:
        img = cv2.imread(path + '/'+i,1)
        # open cv reads images in BGR format so we have to convert it to RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        #resizing image
        img = cv2.resize(img, (SIZE, SIZE))
        img = img.astype('float32') / 255.0
        color_img.append(img_to_array(img))
```

Sorted_alphanumeric,OpenCV
To get the image in sorted order

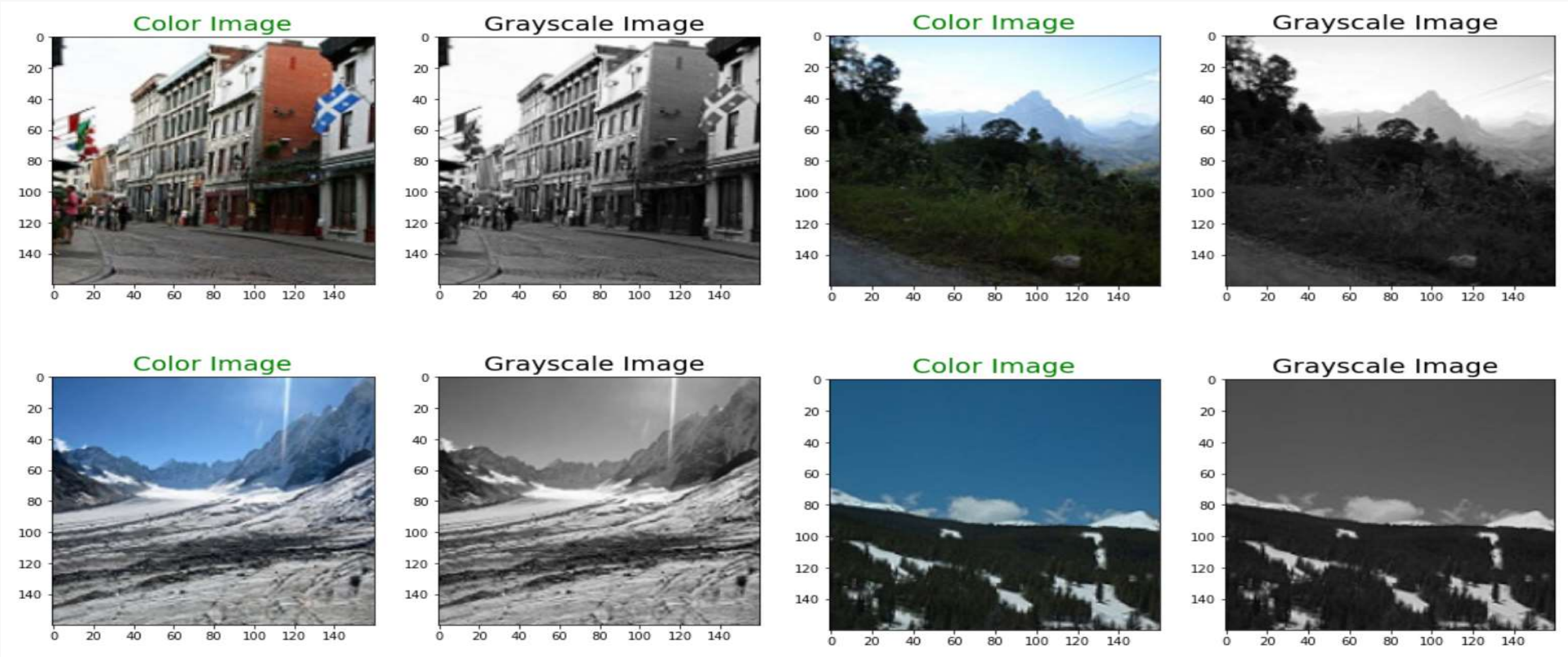
DataSources
Landscape color and gratscale images
(each 6000 files)

Code_Getting landscape image data,resizing them and appending in array

```
gray_img = []
path = '../input/landscape-image-colorization/landscape Images/gray'
files = os.listdir(path)
files = sorted_alphanumeric(files)
for i in tqdm(files):
    if i == '6000.jpg':
        break
    else:
        img = cv2.imread(path + '/'+i,1)

        #resizing image
        img = cv2.resize(img, (SIZE, SIZE))
        img = img.astype('float32') / 255.0
        gray_img.append(img_to_array(img))
```

```
84%|████████▍ | 6000/7129 [00:35<00:06, 171.07it/s]
84%|████████▍ | 6000/7129 [00:31<00:05, 191.33it/s]
```

Code_Plotting Color image and it's corresponding grayscale image



Code_Slicing and reshaping

```
train_gray_image = gray_img[:5500]
train_color_image = color_img[:5500]

test_gray_image = gray_img[5500:]
test_color_image = color_img[5500:]
# reshaping
train_g = np.reshape(train_gray_image,(len(train_gray_image),SIZE,SIZE,3))
train_c = np.reshape(train_color_image, (len(train_color_image),SIZE,SIZE,3))
print('Train color image shape:',train_c.shape)


test_gray_image = np.reshape(test_gray_image,(len(test_gray_image),SIZE,SIZE,
3))
test_color_image = np.reshape(test_color_image, (len(test_color_image),SIZE,S
IZE,3))
print('Test color image shape',test_color_image.shape)
```

```
Train color image shape: (5500, 160, 160, 3)
Test color image shape (500, 160, 160, 3)
```

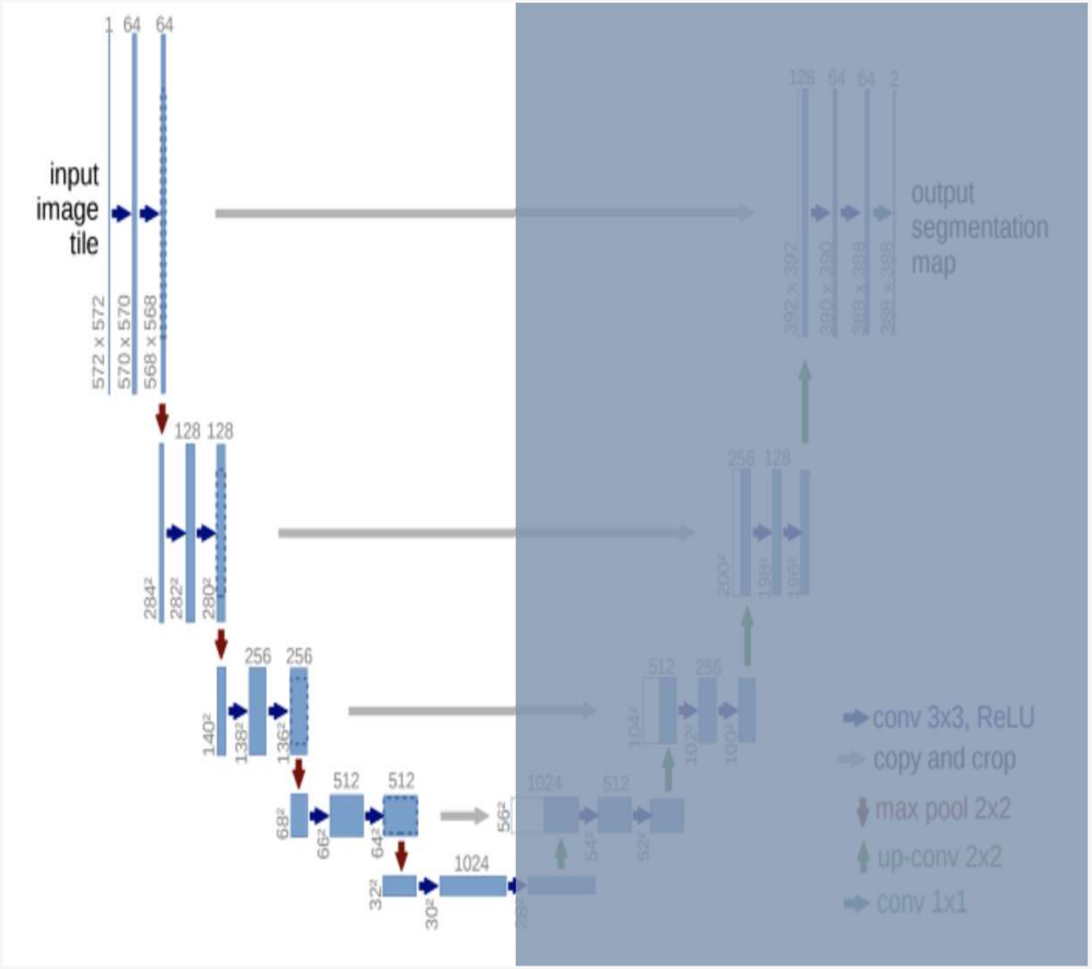
Code_Defining model

```
from keras import layers
def down(filters , kernel_size, apply_batch_normalization =
True):
    downsample = tf.keras.models.Sequential()
    downsample.add(layers.Conv2D(filters,kernel_size,padding
= 'same', strides = 2))
    if apply_batch_normalization:
        downsample.add(layers.BatchNormalization())
    downsample.add(keras.layers.LeakyReLU())
    return downsample


def up(filters, kernel_size, dropout = False):
    upsample = tf.keras.models.Sequential()
    upsample.add(layers.Conv2DTranspose(filters, kernel_siz
e,padding = 'same', strides = 2))
    if dropout:
        upsample.dropout(0.2)
    upsample.add(keras.layers.LeakyReLU())
    return upsample
```

```
def model():
    inputs = layers.Input(shape= [160,160,3])
    d1 = down(128,(3,3),False)(inputs)
    d2 = down(128,(3,3),False)(d1)
    d3 = down(256,(3,3),True)(d2)
    d4 = down(512,(3,3),True)(d3)

    d5 = down(512,(3,3),True)(d4)
    #upsampling
    u1 = up(512,(3,3),False)(d5)
    u1 = layers.concatenate([u1,d4])
    u2 = up(256,(3,3),False)(u1)
    u2 = layers.concatenate([u2,d3])
    u3 = up(128,(3,3),False)(u2)
    u3 = layers.concatenate([u3,d2])
    u4 = up(128,(3,3),False)(u3)
    u4 = layers.concatenate([u4,d1])
    u5 = up(3,(3,3),False)(u4)
    u5 = layers.concatenate([u5,inputs])
    output = layers.Conv2D(3,(2,2),strides = 1, padding = 's
ame')(u5)
    return tf.keras.Model(inputs=inputs, outputs=output)
```

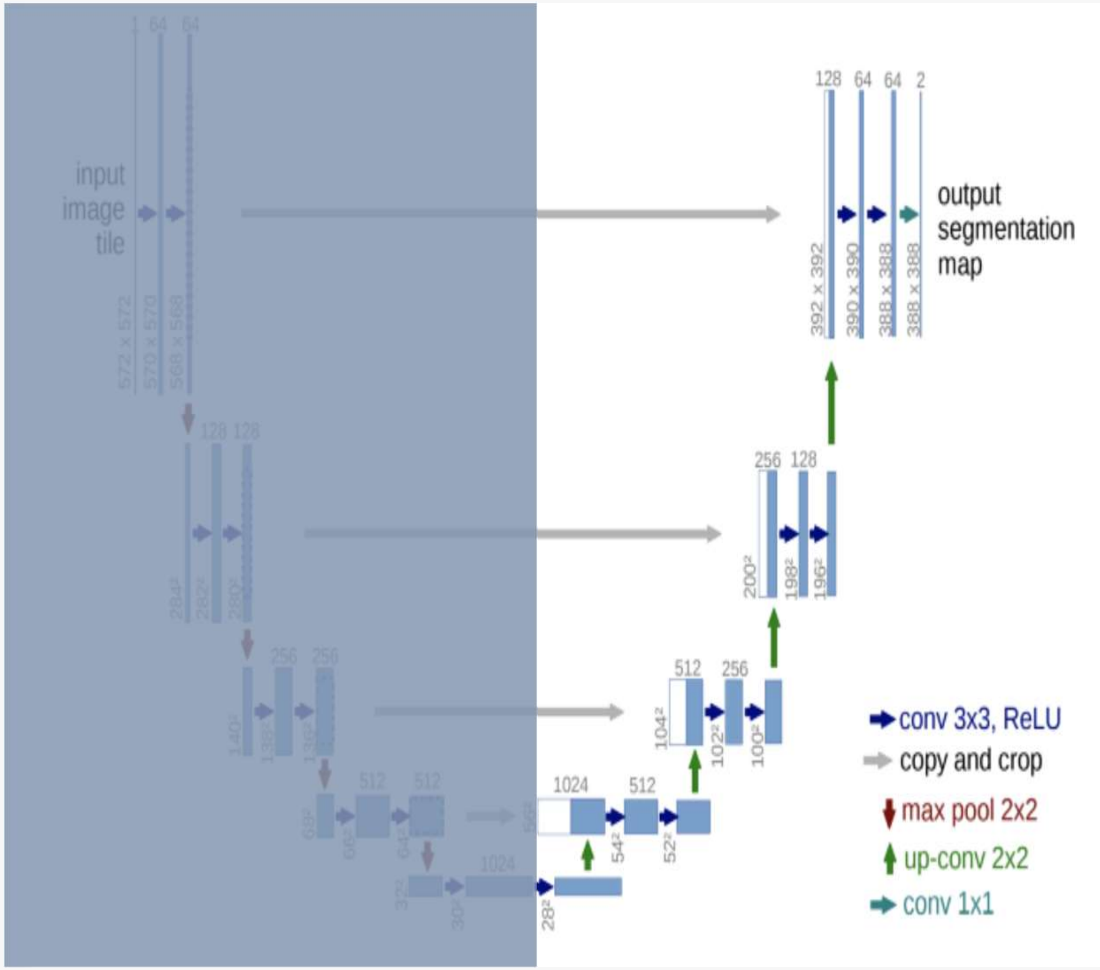
Code_Defining model_U-net Architecture

```
Model: "functional_1"
________________________________________________________________
________________________________________
Layer (type)                    Output Shape         Para
m #     Connected to
================================================================
========================================
input_1 (InputLayer)            [(None, 160, 160, 3) 0
________________________________________________________________
________________________________________
sequential (Sequential)         (None, 80, 80, 128)  3584
input_1[0][0]
________________________________________________________________
________________________________________
sequential_1 (Sequential)       (None, 40, 40, 128)  1475
84      sequential[0][0]
________________________________________________________________
________________________________________
sequential_2 (Sequential)       (None, 20, 20, 256)  2961
92      sequential_1[0][0]
________________________________________________________________
________________________________________
sequential_3 (Sequential)       (None, 10, 10, 512)  1182
208     sequential_2[0][0]
________________________________________________________________
________________________________________
sequential_4 (Sequential)       (None, 5, 5, 512)    2361
856     sequential_3[0][0]
________________________________________________________________
________________________________________
sequential_5 (Sequential)       (None, 10, 10, 512)  2359
808     sequential_4[0][0]
```

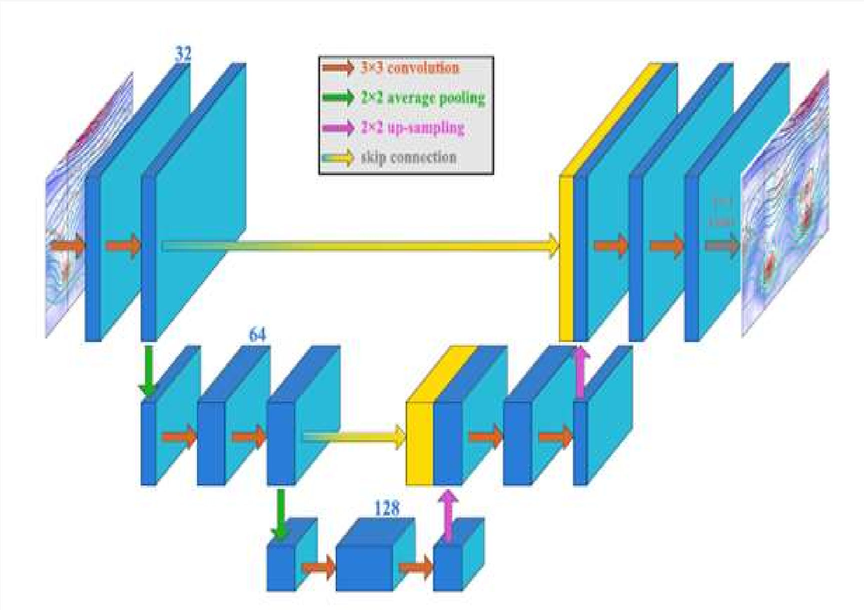


Code_Defining model_U-net Architecture

```
concatenate (Concatenate)       (None, 10, 10, 1024) 0           sequential_5[0][0]
                                                                 sequential_3[0][0]
----------------------------------------------------------------------------------------------
------
sequential_6 (Sequential)       (None, 20, 20, 256)  2359552     concatenate[0][0]
----------------------------------------------------------------------------------------------
------
concatenate_1 (Concatenate)     (None, 20, 20, 512)  0           sequential_6[0][0]
                                                                 sequential_2[0][0]
----------------------------------------------------------------------------------------------
------
sequential_7 (Sequential)       (None, 40, 40, 128)  589952      concatenate_1[0][0]
----------------------------------------------------------------------------------------------
------
concatenate_2 (Concatenate)     (None, 40, 40, 256)  0           sequential_7[0][0]
                                                                 sequential_1[0][0]
----------------------------------------------------------------------------------------------
------
sequential_8 (Sequential)       (None, 80, 80, 128)  295040      concatenate_2[0][0]
----------------------------------------------------------------------------------------------
------
concatenate_3 (Concatenate)     (None, 80, 80, 256)  0           sequential_8[0][0]
                                                                 sequential[0][0]
----------------------------------------------------------------------------------------------
------
sequential_9 (Sequential)       (None, 160, 160, 3)  6915        concatenate_3[0][0]
----------------------------------------------------------------------------------------------
------
concatenate_4 (Concatenate)     (None, 160, 160, 6)  0           sequential_9[0][0]
                                                                 input_1[0][0]
----------------------------------------------------------------------------------------------
------
conv2d_5 (Conv2D)               (None, 160, 160, 3)  75          concatenate_4[0][0]
==============================================================================================
======
Total params: 9,602,766
Trainable params: 9,600,206
Non-trainable params: 2,560
```

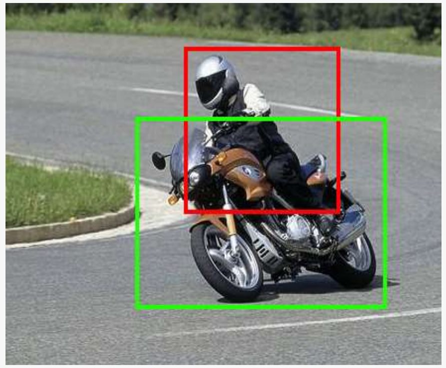


U-net Architecture

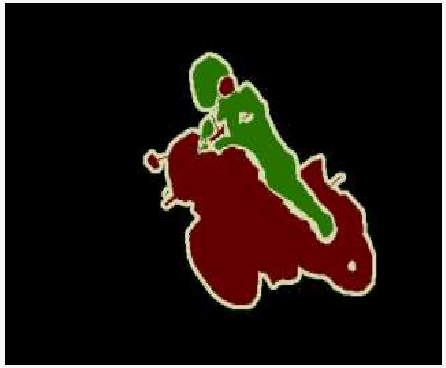


U-net Architecture

U-Net enhances the standard CNN architecture by adding the corresponding extended path (also referred to as a decoder) with the aim of generating semantic predictions of overall resolution. That is, it generates a split image that emphasizes a specific shape and object found in the image.



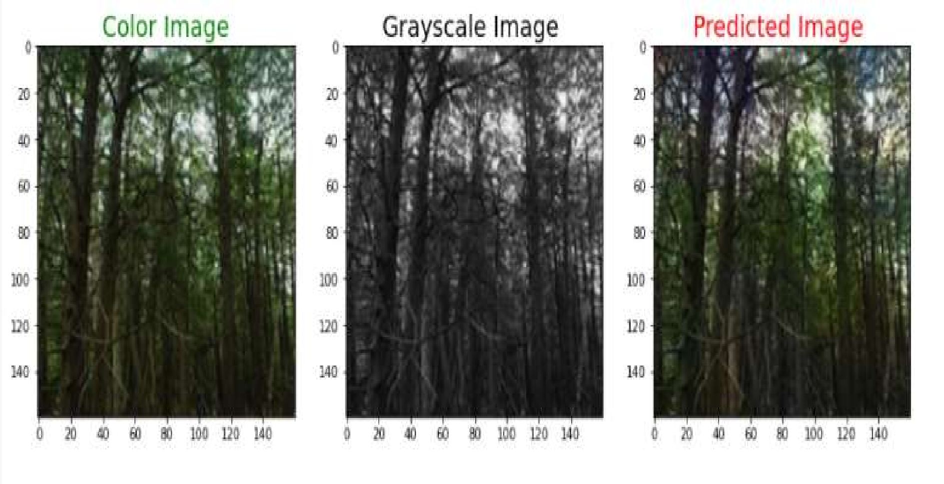
CNN



U-Net

Code_plotting colorized image along with grayscale and color image

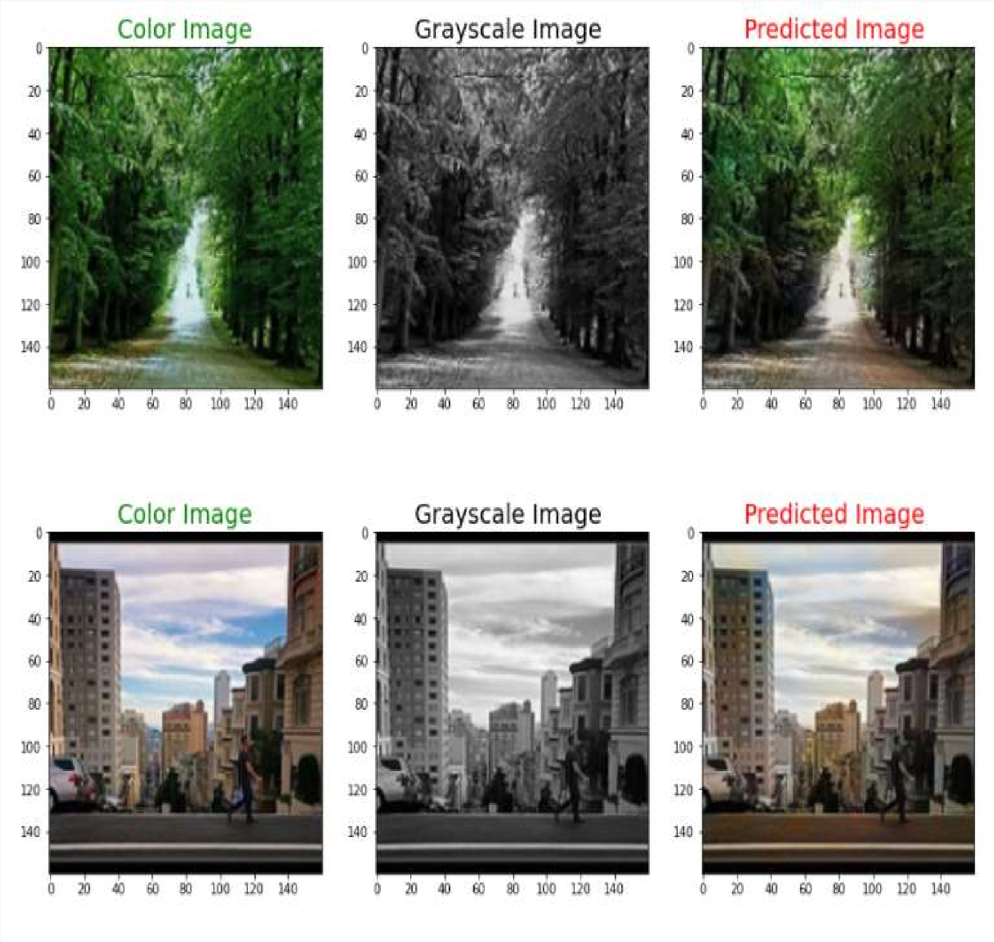
```
def plot_images(color,grayscale,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('Color Image', color = 'green', fontsize = 20)
    plt.imshow(color)
    plt.subplot(1,3,2)
    plt.title('Grayscale Image ', color = 'black', fontsize = 20)
    plt.imshow(grayscale)
    plt.subplot(1,3,3)
    plt.title('Predicted Image ', color = 'Red', fontsize = 20)
    plt.imshow(predicted)

    plt.show()

for i in range(50,58):
    predicted = np.clip(model.predict(test_gray_image[i].reshape(1,SIZE, SIZE,
3)),0.0,1.0).reshape(SIZE, SIZE,3)
    plot_images(test_color_image[i],test_gray_image[i],predicted)
```

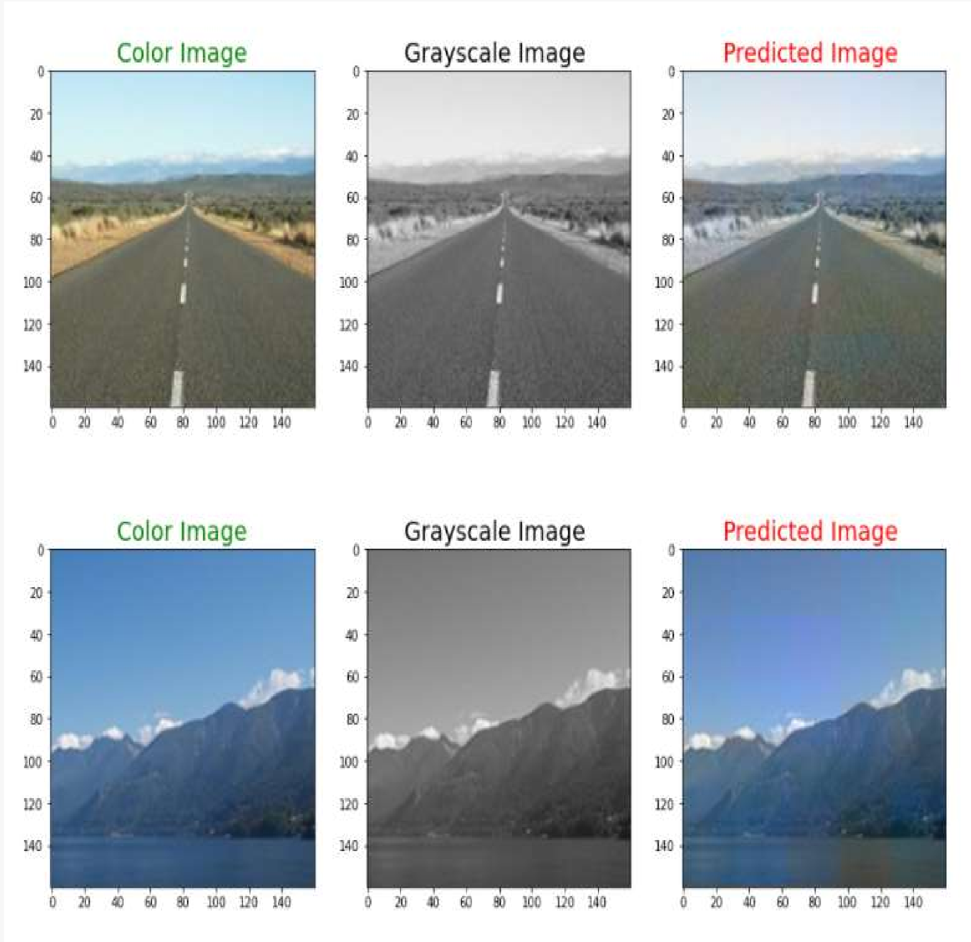



```
model.evaluate(test_gray_image,test_color_image)
```

```
16/16 [==============================] - 1s 38ms/step - loss: 0.0481 - acc:
0.5307

[0.04813535511493683, 0.5306968092918396]
```

Code_plotting colorized image along with grayscale and color image





Reference

-https://www.kaggle.com/theblackmamba31/autoencoder-grayscale-to-color-image

-https://www.kaggle.com/shiratorizawa/pix2pix-gan-for-image-colourisation/notebook

- U-Net: Convolutional Networks for Biomedical Image Segmentation

Thanks