Name:Dhaneshwar.V

NOTEBOOK

NASA Image Colorizer

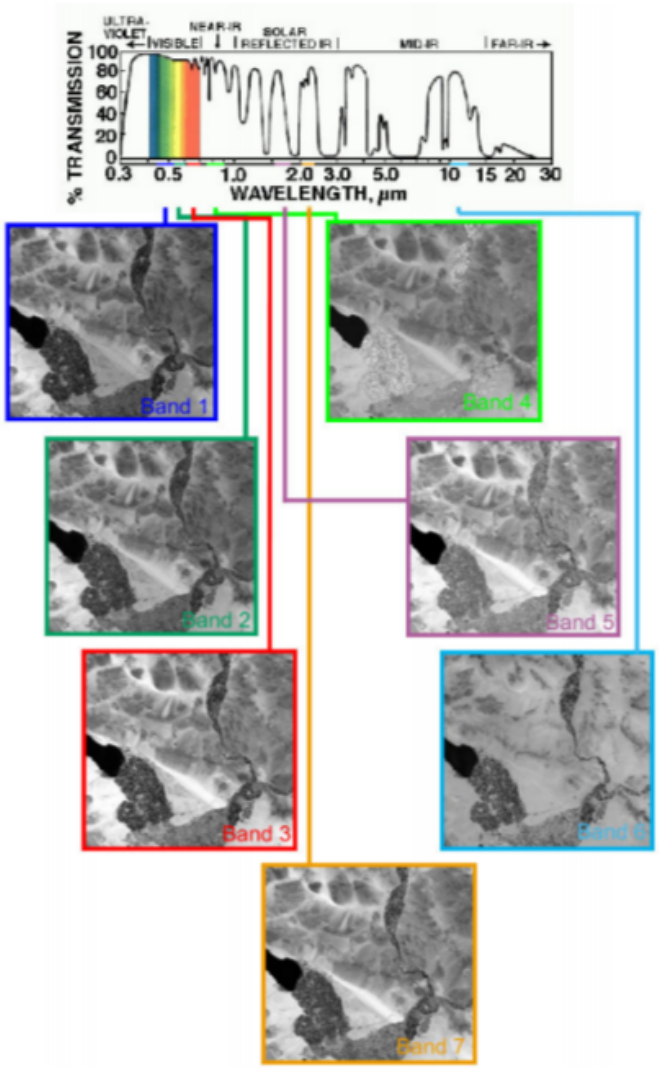
Colorize black and white satellite images

Learn

#1

Introduction

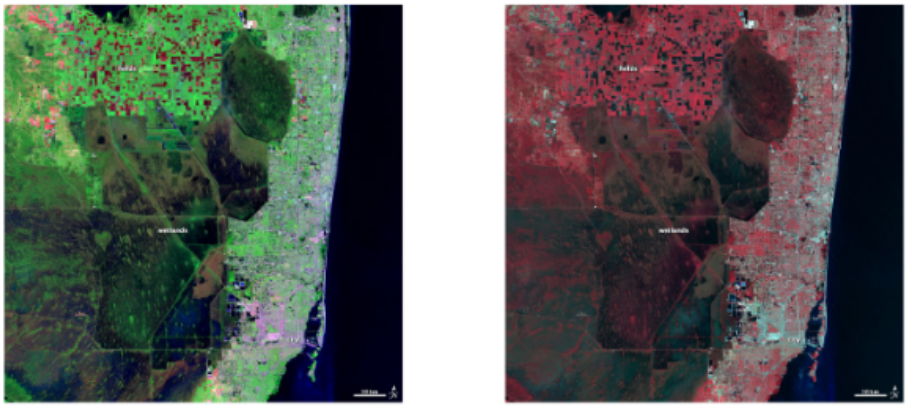
NASA and other space agencies have numerous satellites in earth's orbit which capture earth's and other planets' pictures. But none of them have digital cameras like we carry each day. They rely on various filters that are present in them to capture different wavelengths that are not visible to the human eye. Like in the image shown, Landsat satellites capture 7 pictures of the same place at a time. These images captured through different wavelength filters are used in different scenarios, some of them are used for detecting vegetation, and dome to detect water bodies.



#1

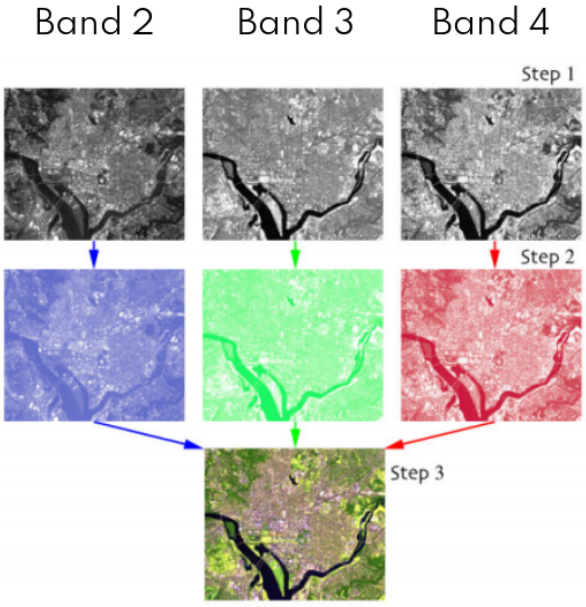
Introduction

As we have seen on the previous page that images captured have only intensity values, so scientists use these intensity values to colorize these images, these colors are called false colors.



False Colors

To get the true image they combine specific three images from different bands, as shown in picture below.



#2

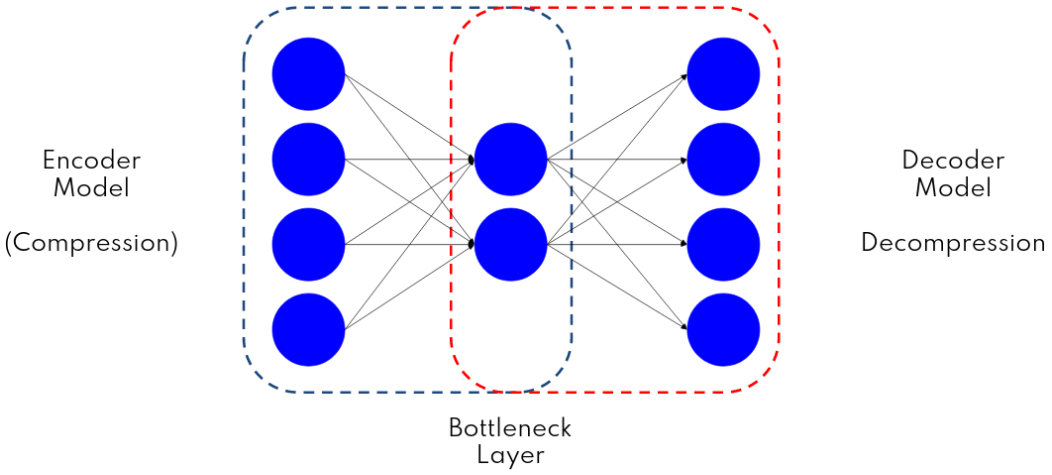
Can it be automated?

Colorizing satellite images of each region is a hectic task, as it involves capturing, and handling multiple images of different bands. Can't we create an algorithm or software that will take only single black and white image of a region and colorize it? Yes, it's possible through AI. We can use special neural networks called autoencoder.

Autoencoders

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation for a set of data, The bottleneck layer makes the autoencoder learn the representation of the input. An autoencoder consists of two networks, the Encoder model, and the decoder model. The encoder model encodes the input and the decoder model tries to decode the encodings.

We will feed a black and white image to the encoder model, and it will learn its representation. Then the decoder model will generate colorful image using this representation.

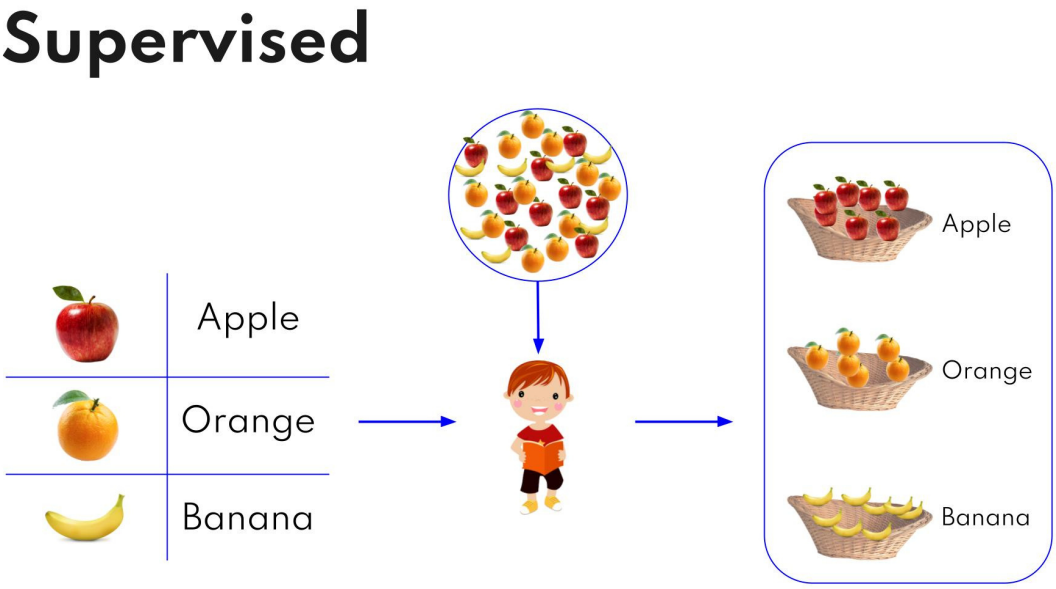


#3

Types of learning algorithms

Supervised Learning

As the name suggests, this type of learning algorithm has a supervisor or teacher. Basically, a well labeled data is the supervisor or teacher, and the algorithm uses this data to learn patterns or features. Like in the fruit sorting problem, a child who can read was given a table of fruits and their names. He was later asked to sort the fruits in a basket. With the provided table, he was able to sort them and knows which basket contains which fruit.

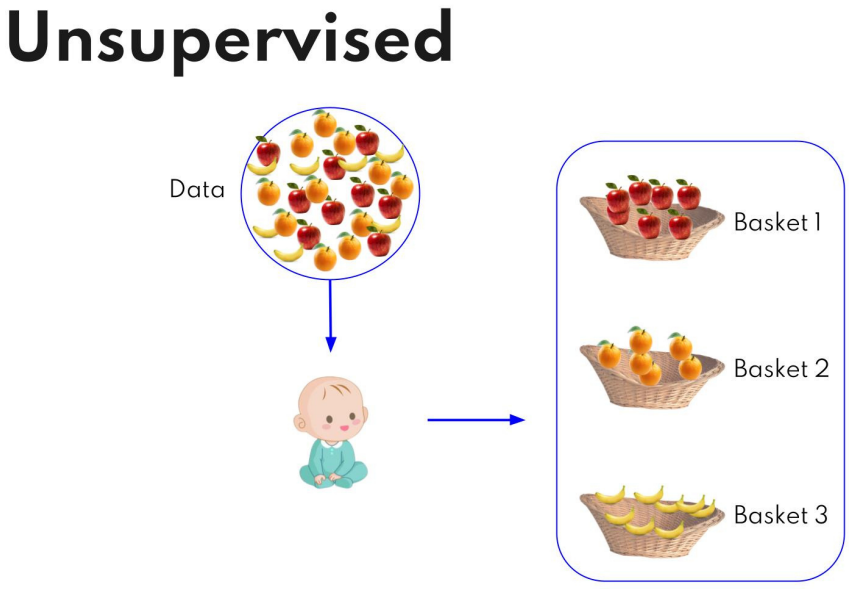


#3

Types of learning algorithms

Unsupervised Learning

Unlike supervised learning, there are no supervisor or teacher present i.e. algorithms use unlabelled data to find features or patterns. Consider a newborn child who can't read. We cannot give the same table that we gave to the previous child. But when this child will be asked to sort the fruits, he/she can sort on the basis of shape, color, or weight. One thing to notice is that at the end, he was able to sort the fruits in a basket but he/she still doesn't know which basket contains which fruit.



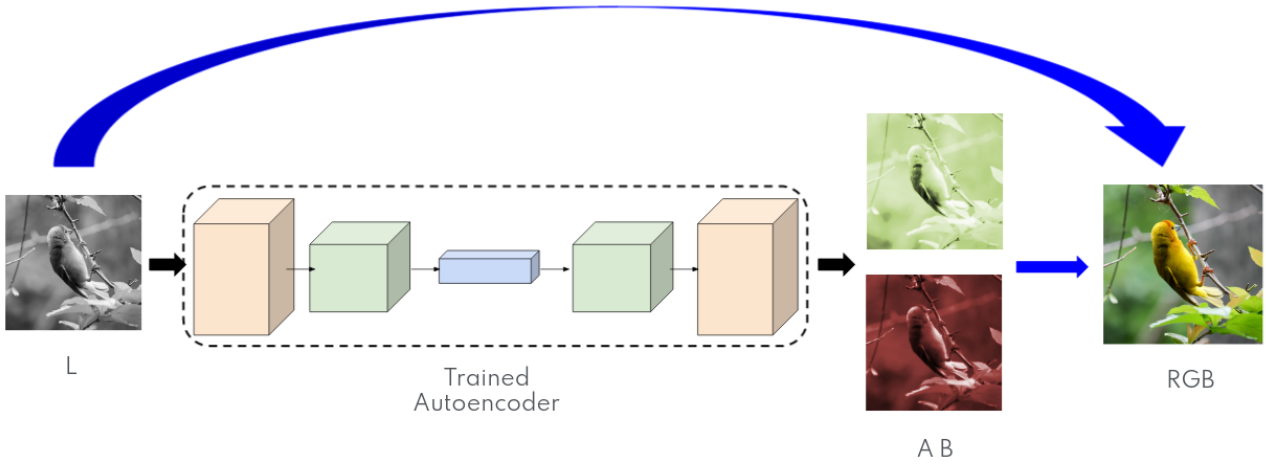
#4

Color Space

An image have three channels for color (Red, Green, Blue). This means that our image shape will be (Width X Height X 3), and a black and white image will have shape of (Width X Height X 1) because it only have one channel. A very complex and deep autoencoder will be required to generate color RGB image from grayscale image. So we need a different color space to predict color of image. We will be using LAB color space.

LAB Color Space

A LAB image also has 3 channels, L for lightness, A for red/green value, and B for blue/yellow value. L channel is a grayscale image and A, B channels are for color. So now we will predict only two channels (A, B) instead of 3 channels (R, G, B) using the L channel (grayscale) of the image. Later we will change the LAB image to an RGB image.



#5

Autoencoder Architecture

The zip that we have shared contains the folder "model", this folder contains the trained autoencoder. In this section, we will see the architecture of that model.

We have created two models, encoder and decoder. Using these two models we have created our final autoencoder. As we are dealing with images, we will only be using convolution layers which are better than dense layers.

Encoder Model

We have defined the input shape as (128x128x1). So for testing, we have to resize the test image to (128x128). We have created a 3 layer encoder. One thing we can observe that after each layer our output shape is decreasing, and at the last convolution layer we have compressed our data to (16x16). This layer will be our bottleneck layer and image representation will be learned in this layer

```
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 1)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 64, 64)        640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 256)       295168
_________________________________________________________________
flatten (Flatten)            (None, 65536)             0
=================================================================
Total params: 369,664
Trainable params: 369,664
Non-trainable params: 0
_________________________________________________________________
```

#5

Autoencoder Architecture

Decoder Model

Our input shape of the decoder model will be the same as the output shape of the encoder model. As now we have to decompress our data/image, we have used a convolution transpose layer. We have created a 4 layer decoder, and you can observe that at the last layer we will be getting an output of shape (128x128x2) i.e A and B channel of our image.

```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
decoder_input (InputLayer)   [(None, 65536)]           0
_________________________________________________________________
reshape (Reshape)            (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 32, 32, 256)       590080
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 64, 64, 128)       295040
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 128, 128, 64)      73792
_________________________________________________________________
decoder_output (Conv2DTransp (None, 128, 128, 2)       1154
=================================================================
Total params: 960,066
Trainable params: 960,066
Non-trainable params: 0
_________________________________________________________________
```

Autoencoder Model

For creating our autoencoder we just have to combine encoder and decoder model.

```
Model: "autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 1)]     0
_________________________________________________________________
encoder (Functional)         (None, 65536)             369664
_________________________________________________________________
decoder (Functional)         (None, 128, 128, 2)       960066
=================================================================
Total params: 1,329,730
Trainable params: 1,329,730
Non-trainable params: 0
_________________________________________________________________
```

#6

Dataset

We have trained our autoencoder on the UC Merced Land Use dataset. This dataset consists of 21 classes like buildings, beach, airplane, runway, freeway, etc. And there are 100 images of each category. This dataset is perfect for our autoencoder as we will be colonizing satellite images. Some of the sample images from this dataset are shown below.

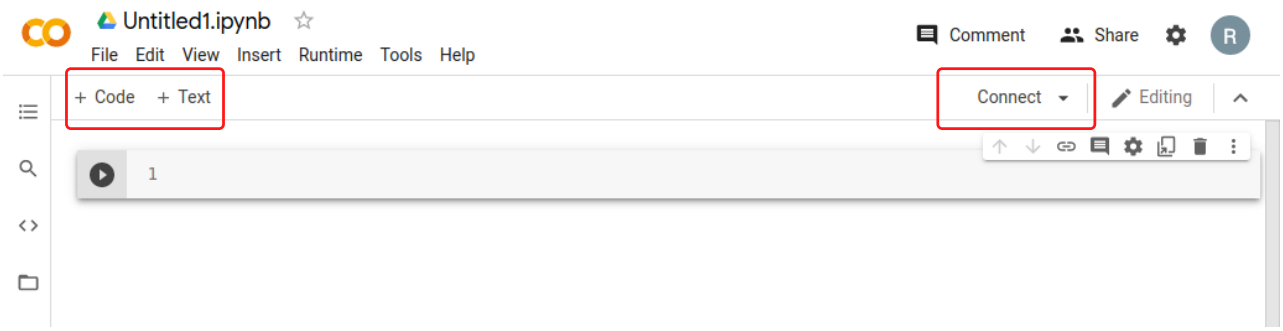
You can check out it on their website.
http://weegee.vision.ucmerced.edu/datasets/landuse.html

#Extra

Introduction to Colab

Nearly all machine learning and deep learning algorithms require good hardware. What if ... you don't have good hardware? Should you drop your dream to be a data scientist? No, there's an alternative Let us introduce you to Colab.

Colab is a service provided by Google which lets you access a virtual machine hosted on google servers. These virtual machines have dual-core Xeon processors, with 12GB of RAM. You can even use GPU for your neural networks. Colab is an interactive python notebook (ipynb), which means that with writing python code, you can also write normal text, include images.
To create a new colab notebook, just go to "https://colab.research.google.com", and create a new notebook. You will get something like below



You can connect to runtime by clicking the "Connect" button in the right corner. You can add a new Code cell, or text cell using respective buttons in the toolbar.

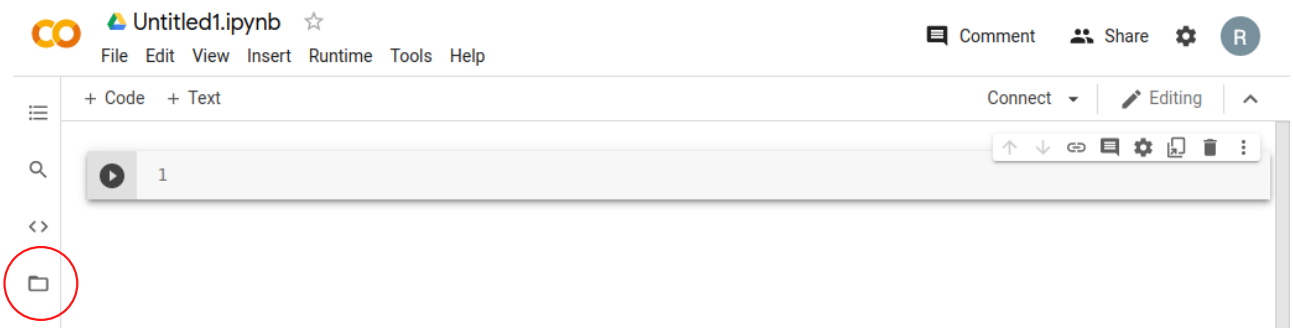
#Extra

Introduction to Colab

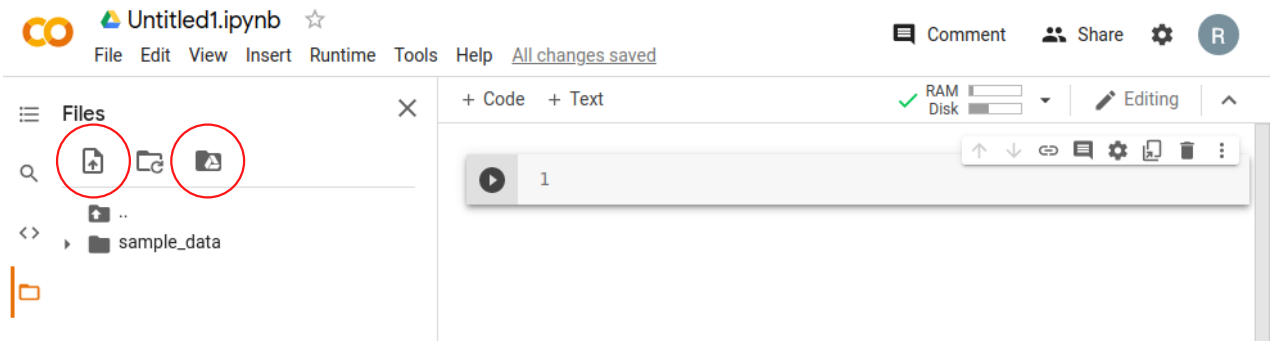
Uploading files

Sometimes you need to use a file from your PC. For that, you can upload the required files to google colab. Colab provides 100 GB in a colab session. If you want to access some files from your drive, you can even do so by connecting the drive to colab.

To upload something, open file pane from left toolbar.



Now, just upload the file using first button, and you can also mount google drive using last button



#7

Code

Upload and unzip

We have also shared a zip folder, upload the same to colab memory and unzip it using the following command

```
!unzip -q Satellite\ Image\ Colorizer.zip
```

After unzipping you will get the following 4 files and folders.

model/ - Folder with trained autoencoder
test_image/ - Folder containing some images for testing
requirements.txt - libraries required for this task
utils.py - python file with functions to convert RGB to LAB image and LAB to RGB image

#7

Code

Importing libraries

Libraries to be used:

1. NumPy: All images are stored as NumPy arrays
2. SkImage: Used to read and apply operations on image
3. Tensorflow/Keras: For loading tensorflow/keras trained model
4. Matplotlib: Plotting images
5. os: To get all the list of images present in a directory
6. utils: utils.py has two functions to convert RGB to LAB and LAB to RGB.

```
import numpy as np
from skimage.io import imread
from skimage.transform import resize
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
import os
import utils
```

Defining Constants

We need to define image dimension as (128, 128) because the trained model can only take the image with this size.

```
IMAGE_DIMENSION = (128, 128)
```

#7

Code

Loading trained model

The trained model is inside the "model" folder. Loading a model consists of two steps:

1. Loading model structure
2. Loading model weights

Keras models can be loaded using models.load_model function.

```
autoencoder = load_model('model/colorizer')
```

Searching images in a directory

All images are in the "test_imags" folder. We need to find their names, or you can upload your own image that you want to test. Make sure image_file_name variable have correct filename of image.

```
test_image_dir = 'test_images'
testing_files = os.listdir(test_image_dir)
print('FOUND {} IMAGES'.format(len(testing_files)))

CHOSEN_IMAGE_INDEX = 9 # change to the image you want to test
print('Testing on ', testing_files[CHOSEN_IMAGE_INDEX])
image_file_name = testing_files[CHOSEN_IMAGE_INDEX]
```

#7

Code

Reading Image

Now we will be using skimage to read the image, and matplotlib to plot that image. We also have to resize our image to required size of (128, 128)

```
image = imread('test_images/'+image_file_name)
image = image[:,:,:3]
image = resize(image, IMAGE_DIMENSION)

plt.imshow(image)
```

Converting to L (grayscale)

This step is required as the image is in color and we need to test the performance of the model.

```
# converting to lab color
l, ab = utils.RGB2L_AB(image, IMAGE_DIMENSION)

# plotting l channel (black and white)
plt.imshow(l.reshape(IMAGE_DIMENSION), cmap='gray')
```

#7

Code

Predicting color using model

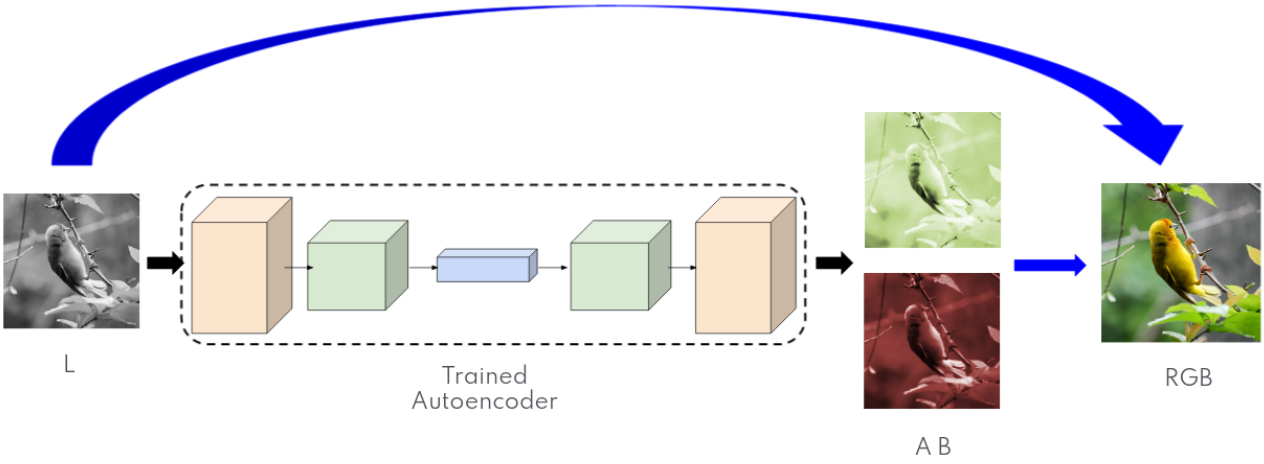
Now we will be using a trained model to predict color i.e A, B channel of LAB image.

```
predicted_ab = autoencoder.predict(np.expand_dims(l, axis=0))
```

Creating RGB from LAB

Now we will be using L (input) A, B (output) to create RGB image.

```
colorized_image = utils.L_AB2RGB(l, predicted_ab[0], IMAGE_DIMENSION)
print("FInal color image shape ",colorized_image.shape)
```



#7

Code

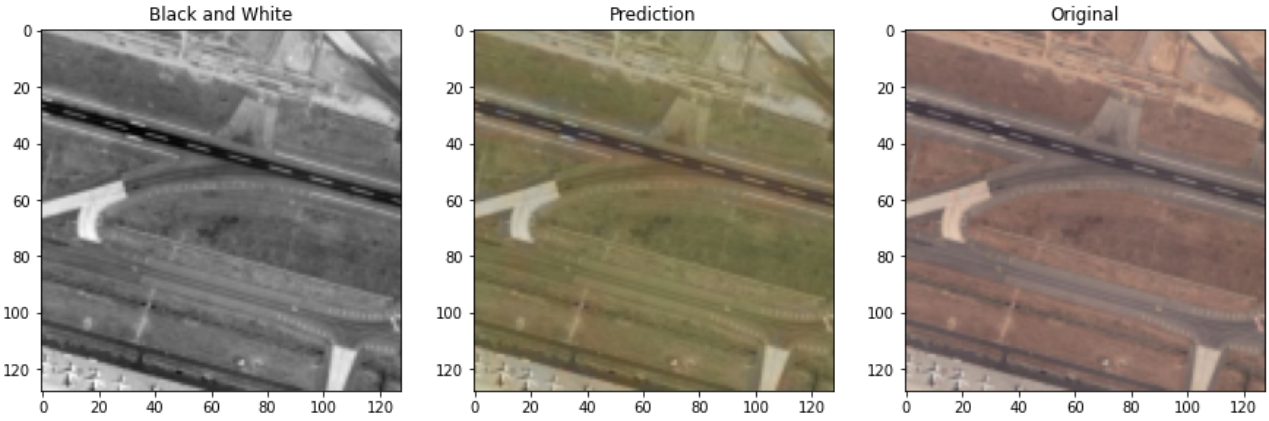
Result Comparision

Now we will be plotting all the images (grayscale image, predicted color image, original color image) for comparision.

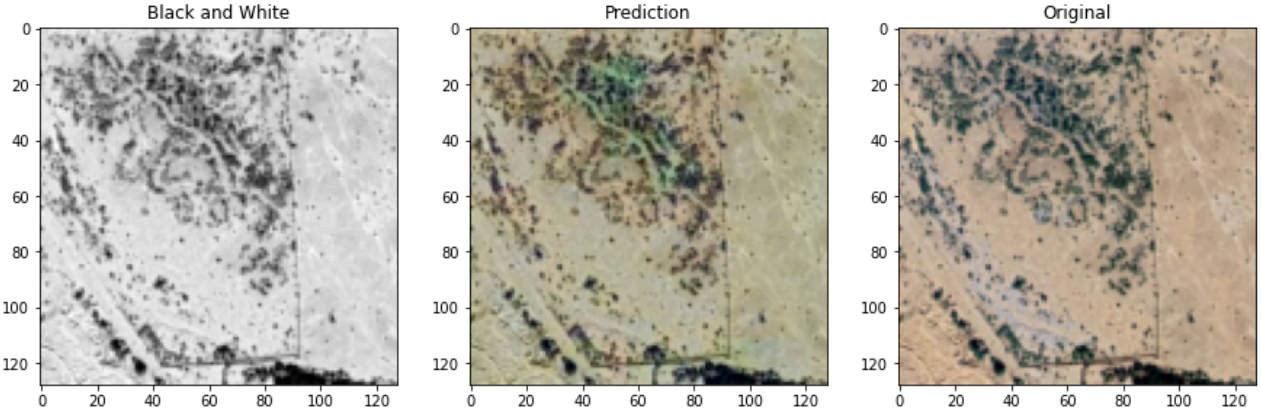
```
plt.figure(figsize=(15,18))
plt.subplot(1,3,1)
plt.imshow(l.reshape(IMAGE_DIMENSION), cmap='gray')
plt.title('Black and White')

plt.subplot(1,3,2)
plt.imshow(colorized_image)
plt.title('Prediction')

plt.subplot(1,3,3)
plt.imshow(image)
plt.title('Original')

plt.show()
```

#8

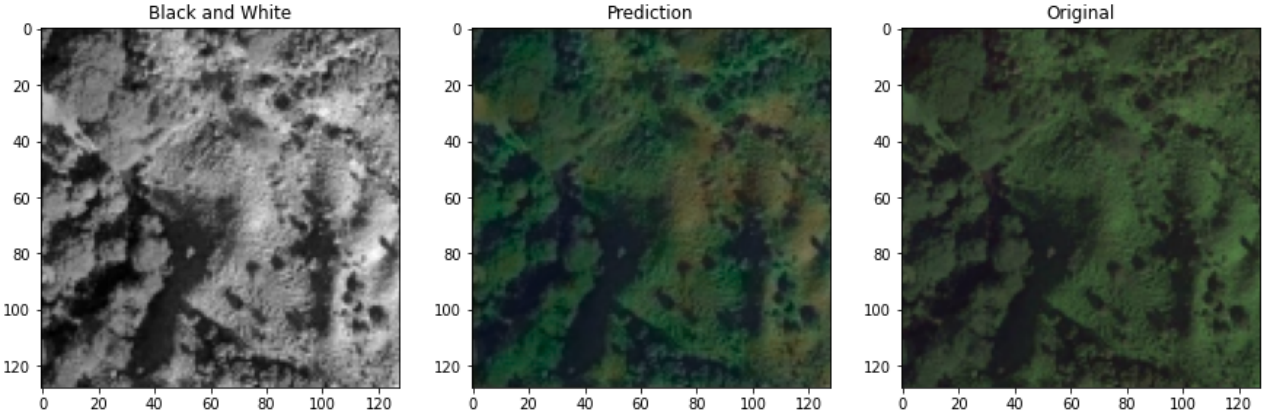
Result and Conclusion



The model correctly predicted runway and taxiway regions. But instead of colonizing land as brown, model colorized as green grass. But still image seems to be original.



In this image also the model was successful in predicting desert and green regions.



Learn

Knowledge Discovery through Brand Stories

Visit Us at :
techlearn.live

Email : support@techlearn.live
Phone : +91-9154796743