




# RAMAIAH

## Institute of Technology

Department of Information Science  
and Engineering



# Colorization of grayscale images using deep learning

Under the guidance of Prof. Shruthi G

Pavana H A 1MS19IS086  
Pradyumna 1MS19IS089

# Introduction:

Image colorization is the process of assigning colors to a grayscale image to make it more aesthetically appealing and perceptually meaningful.

In this presentation we will show our model which converts black and white images to colour images using deep learning.



# Problem statement:

image colorization is usually done by hand in Photoshop. However, using Photoshop for this purpose requires more energy and time. One solution to this problem is to use machine learning / deep learning techniques.

This is helpful in converting the black and white movies of the past to colour.

It is useful for solving of crimes from cctv footage using b&w footage to colour





# Dataset:

## **COCO dataset:**

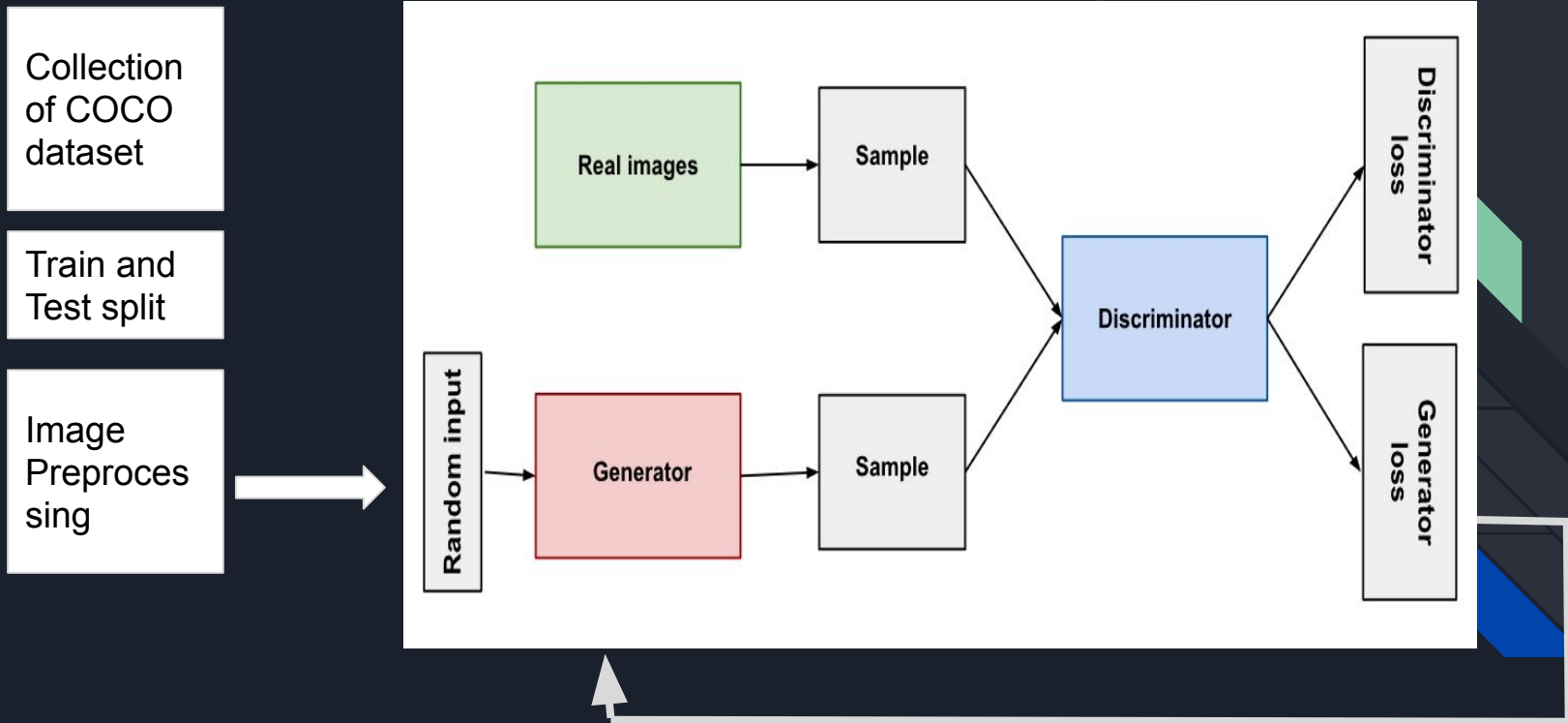
1. COCO dataset is a large-scale object detection, segmentation, and captioning dataset published by Microsoft.
2. COCO stands for Common Objects in Context, as the image dataset was created with the goal of advancing image recognition. The COCO dataset contains challenging, high-quality visual datasets for computer vision, mostly state-of-the-art neural networks.
3. It has over 200000 images and 80 different classes.
4. We will be using 8000 images for training our model and 2000 images for testing.

URL for dataset:

<https://cocodataset.org/#download>

# System block diagram:

Generative adversarial network (GAN)



# Data preparation:

- Download part of COCO dataset using fastai library it will download 20000 images.
- In that we are using 8000 images for training and 2000 images for testing.
- Resizing all images to  $256 * 256$ .
- Converting images from RGB colour space to Lab colour space.
- Separating L and a b channel
- From  $L * a * b$  colour space use L channel as input to train the model.



```
1 from fastai.data.external import untar_data, URLs
2 coco_path = untar_data(URLs.COCO_SAMPLE)
```

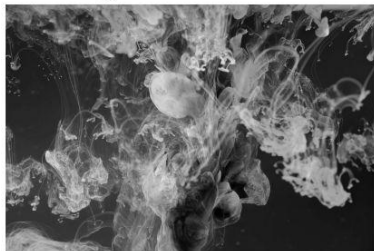
# Lab colour space:

- In  $L^*a^*b$  color space, we have three numbers for each pixel.
- The first number (channel),  $L$ , encodes the Lightness of each pixel and when we visualize this channel (the second image in the row below) it appears as a black and white image.
- The  $a^*$  and  $b^*$  channels encode how much green-red and yellow-blue each pixel is, respectively.

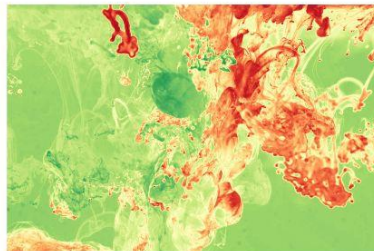
Main Image



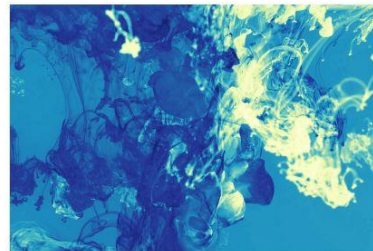
Lightness Channel



$a^*$  Channel



$b^*$  Channel







## generative adversarial network:

A generative adversarial network, or GAN, is a deep neural network framework which is able to learn from a set of training data and generate new data with the same characteristics as the training data.

For example, a generative adversarial network trained on photographs of human faces can generate realistic-looking faces which are entirely fictitious.

Generative adversarial networks consist of two neural networks, the generator and the discriminator.

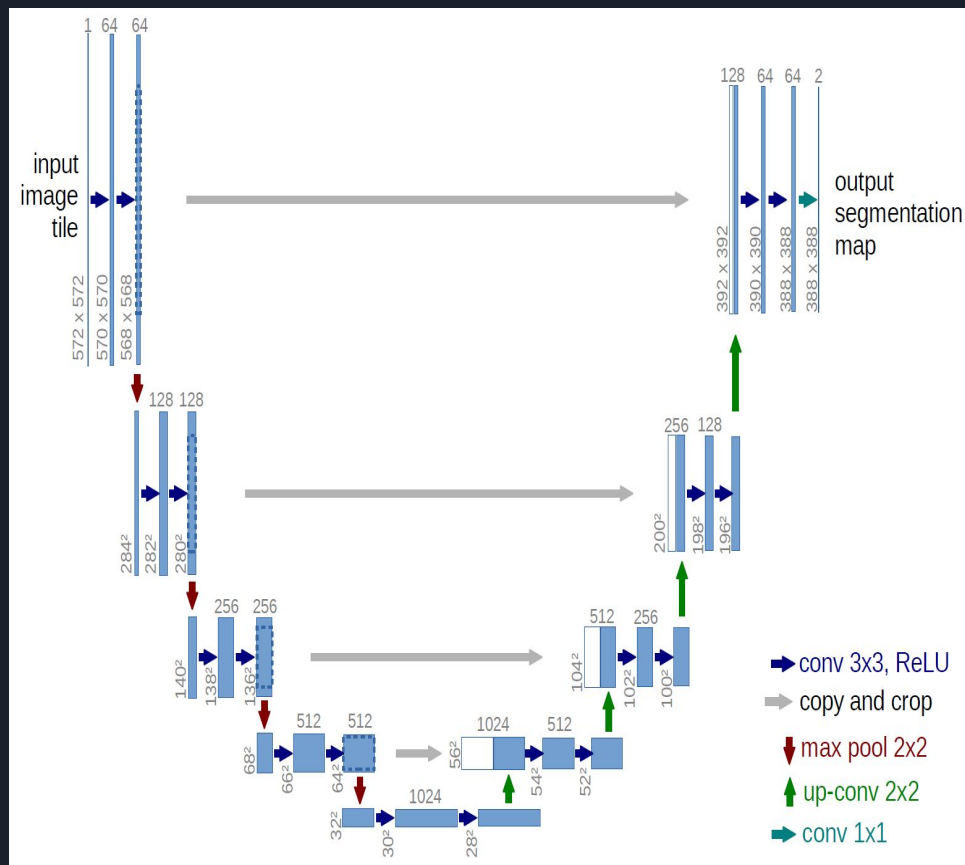
Generator: generator model takes a grayscale image (1-channel image) and produces a 2-channel image, a channel for  $a$  and another for  $b$ .

Discriminator: the discriminator takes these two produced channels and concatenates them with the input grayscale image and decides whether this new 3-channel image is fake or real.

# U-net:

We are using u-net as generator for our GAN

- U-Net is an architecture for semantic segmentation. It consists of a contracting path and an expansive path.
- The contracting path follows the downsampling.
- The expansive path follows upsampling

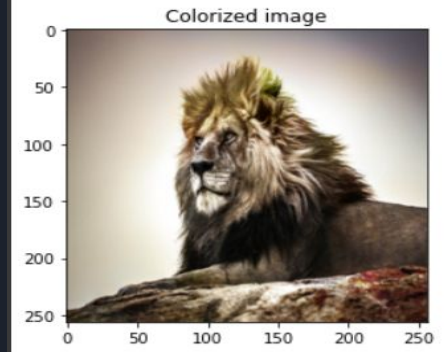
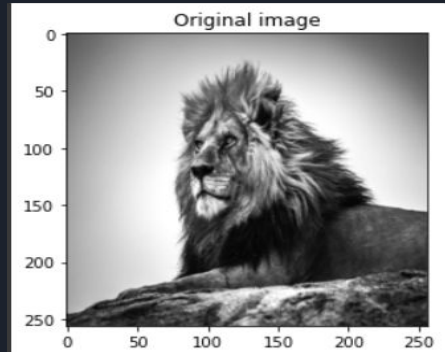
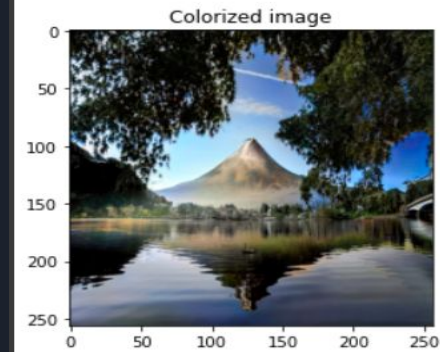
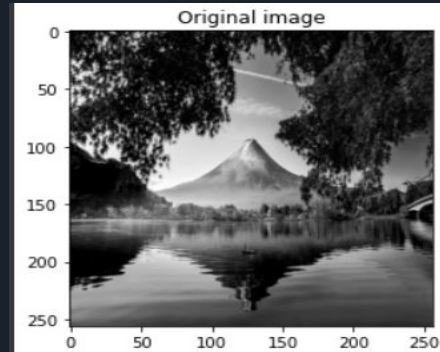
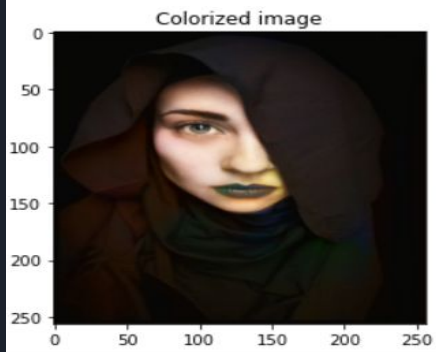
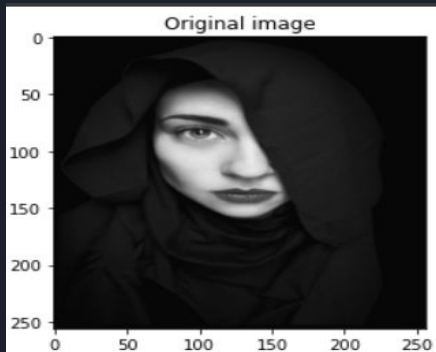
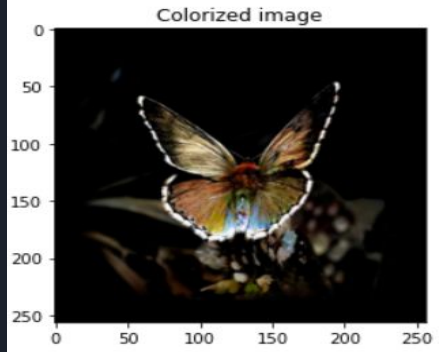
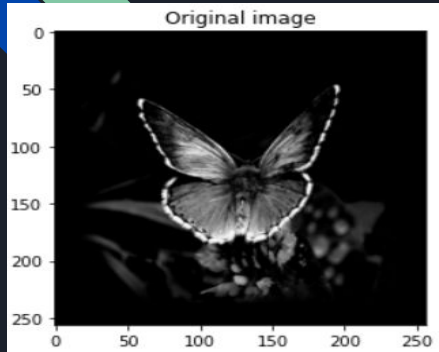



## Discriminator:

This code implements a model by stacking blocks of Conv-BatchNorm-LeakyReLU to decide whether the input image is fake or real.

```
1 class PatchDiscriminator(nn.Module):
2     def __init__(self, input_c, num_filters=64, n_down=3):
3         super().__init__()
4         model = [self.get_layers(input_c, num_filters, norm=False)]
5         model += [self.get_layers(num_filters * 2 ** i, num_filters * 2 ** (i + 1), s=1 if i == (n_down-1) else 2)
6                   for i in range(n_down)]
7
8         model += [self.get_layers(num_filters * 2 ** n_down, 1, s=1, norm=False, act=False)]
9
10        self.model = nn.Sequential(*model)
11
12    def get_layers(self, ni, nf, k=4, s=2, p=1, norm=True, act=True):
13        layers = [nn.Conv2d(ni, nf, k, s, p, bias=not norm)]
14        if norm: layers += [nn.BatchNorm2d(nf)]
15        if act: layers += [nn.LeakyReLU(0.2, True)]
16        return nn.Sequential(*layers)
17
18    def forward(self, x):
19        return self.model(x)
```

# Results and discussion:



- 
- We trained our model using the Tesla T4 GPU used in Google Colab.
  - This model took 9 hours to train .
  - We trained our model for 100 epochs.

Link for our trained model:

<https://drive.google.com/file/d/1-7kuZiW3B03Tq8ZSd0CvfYwcWCPzCCr3/view?usp=sharing>

Link for google colab:

[https://colab.research.google.com/drive/1Qd52Z9jW8Y2vjYi\\_fbGr8j6nV2osXSEI?usp=sharing](https://colab.research.google.com/drive/1Qd52Z9jW8Y2vjYi_fbGr8j6nV2osXSEI?usp=sharing)



## References:

- Dataset: <https://cocodataset.org/#download>
- Paper: <https://arxiv.org/pdf/1611.07004.pdf>
- <https://paperswithcode.com/method/u-net>
- Drive link:  
[https://drive.google.com/drive/folders/1p9AVXuy9WDMgPJ-7C4sSRKNTJl\\_yLktY?usp=sharing](https://drive.google.com/drive/folders/1p9AVXuy9WDMgPJ-7C4sSRKNTJl_yLktY?usp=sharing)



# Thank You

