EXPLORATORY PROJECT REPORT

PROJECT AIM

The aim is to colorize black and white images using tensorflow, Keras and a deep learning framework known as Generative adversarial network (GAN's).

Submitted by:

Anurag Chaudhary (20095012)

Dipanshu Chaudhary (20095134)

Jubin Banerjee (20095048)

Tirumani Eshwar Sai Teja (20095117)

Under the Guidance of

Dr. shivam verma

Department Of Electronics Engineering

IIT (BHU) Varanasi

CERTIFICATE

This is to certify that this project report **"colorize black and white images using tensorflow and a deep learning framework known as Generative adversarial network (GAN's)"** is submitted by Anurag Chaudhary, Dipanshu Chaudhary, Jubin Banerjee, Tirumani Eshwar Sai Teja who carried out the project work under the supervision of

Dr. shivam verma

We approve this project for submission of the Exploratory Project, IIT(BHU) Varanasi.

Signature of Supervisor
Dr. shivam verma
Department of Electronics Engineering
IIT(BHU) Varanasi

Acknowledgment

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide, Dr. Shivam verma , for his valuable guidance, encouragement, and help for accomplishing this work. His useful suggestions for this whole project are sincerely acknowledged.
We would also like to express our sincere thanks to all others who helped us directly or indirectly during this project work.

Students Name : **Date : 25-04-2022**

Anurag Chaudhary - 20095012
Dipanshu Chaudhary - 20095134
Jubin Banerjee - 20095048
Tirumani Eshwar Sai Teja - 20095117

ABSTRACT

Have you ever thought about colorizing an old black and white photo of your parents, grandparents? Well doing it manually (using photoshop) is absolutely painstaking. In deep learning we have studied about Generative Adversarial Networks (GANs) which is an approach of generative modeling. We have made a Deep Learning Project "Image colorization using GANs" in which we input a grayscale image and the GAN will output the colorized image of it. We have used GANs because GANs are effective in learning the loss function. GANs have two major parts Generator and Discriminator. Generator takes b/w input images and generates a synthetic color image of it. Discriminator distinguish from the real data, which one is real and which one is generated/fake. When we trained our model, it gave us very promising results with a very slight difference between a real image and a synthetic image. This model can be used in the case of colorizing old photos as it is quite accurate and effective. Thanks to GANs.

The project went through 2 phases:

Phase 1:

Learning about Generative adversarial network, GAAN's are turning more and more popular these days due to its high accuracy and easy handling so we created our own network.

Phase 2:

In phase 2nd we optimized our dataset which is a set of 3000 images and then we trained our model by adding several convolutional and pooling layers. we also made our model dense to increase the accuracy.

INTRODUCTION

One of the most exciting applications of deep learning is colorizing black and white images. This task needed a lot of human input and hardcoding several years ago but now the whole process can be done end-to-end with the power of AI and deep learning. You might think that you need huge amount of data or long training times to train your model from scratch for this task but in the last few weeks we worked on this and tried many different model architectures, loss functions, training strategies, etc. and finally developed an efficient strategy to train such a model, using the latest advances in deep learning, on a rather small dataset and with really short training times.

Image-to-Image Translation with Conditional Adversarial Networks paper, which you may know by the name pix2pix, proposed a general solution to many image-to-image tasks in deep learning which one of those was colorization. In this approach two losses are used: L1 loss, which makes it a regression task, and an adversarial (GAN) loss, which helps to solve the problem in an unsupervised manner (by assigning the outputs a number indicating how "real" they look!).

Results we are expecting:

A little piece of history on coloring black and white images

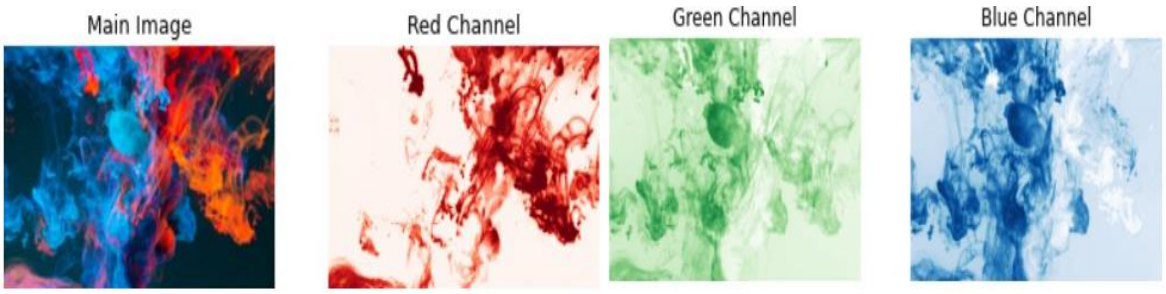
Models for the colorization of grayscales began back in the early 2000s. In 2002, Welsh et al. proposed an algorithm that colorized images through texture synthesis. Colorization was done by matching luminance and texture information between an existing color image and the grayscale image to be colorized.
However, this proposed algorithm was defined as a forward problem, thus all solutions were deterministic. Levin et al. proposed an alternative formulation to the colorization problem in 2004. This formulation followed an inverse approach, where the cost function was designed by penalizing the difference between each pixel and a weighted average of its neighbouring pixels. Both of these proposed methods still required significant user intervention which made the solutions less than ideal.

In 2016, a colorization method was proposed by comparing colorization differences between those generated by convolutional neural networks and GAN.
The models in the study not only learn the mapping from input to output image, but also learn a loss function to train this mapping. Their approach was effective in ill-posed problems such as synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images. We aim to extend their approach by generalizing the colorization procedure to high resolution images and suggest training strategies that speed up the process and greatly stabilize it.

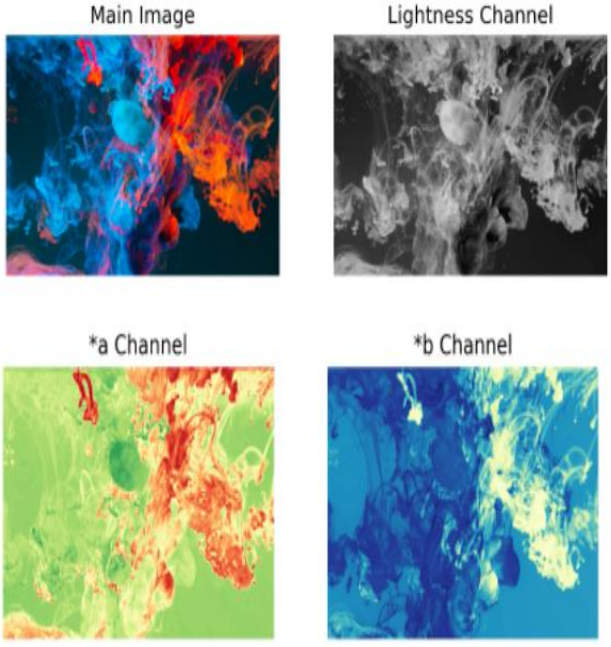
Basic problems we need to understand to make our model

RGB vs L*a*b:

As you might know, when we load an image, we get a rank-3 (height, width, color) array with the last axis containing the color data for our image. These data represent color in RGB color space and there are 3 numbers for each pixel indicating how much Red, Green, and Blue the pixel is. In the following image you can see that in the left part of the "main image" (the leftmost image) we have blue color so in the blue channel of the image, that part has higher values and has turned dark.



In L*a*b color space, we have again three numbers for each pixel but these numbers have different meanings. 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. In the following image you can see each channel of L*a*b color space separately.



Generative adversarial network:

In a GAN we have a generator and a discriminator model which learn to solve a problem together. In our setting, the generator model takes a grayscale image (1-channel image) and produces a 2-channel image, a channel for *a and another for *b. 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. Offcourse the discriminator also needs to see some real images (3-channel images again in Lab color space) that are not produced by the generator and should learn that they are real.

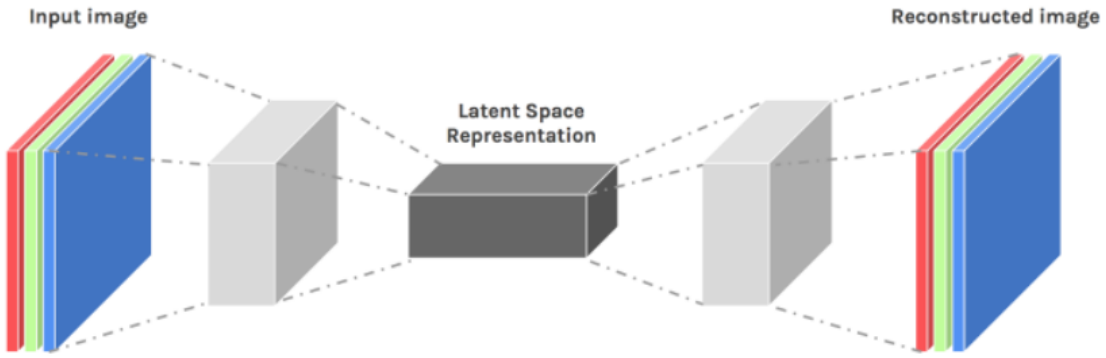
So what about the "condition" we mentioned? Well, that grayscale image which both the generator and discriminator see is the condition that we provide to both models in our GAN and expect that the they take this condition into consideration.

Let's take a look at the math. Consider x as the grayscale image, z as the input noise for the generator, and y as the 2-channel output we want from the generator (it can also represent the 2 color channels of a real image). Also, G is the generator model and D is the discriminator. Then the loss for our conditional GAN will be:

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))]$$

Notice that x is given to both models which is the condition we introduce two both players of this game. Actually, we are not going to feed a "n" dimensional vector of random noise to the generator as you might expect but the noise is introduced in the form of dropout layers (there is something cool about it which you will read in the last section of the article) in the generator architecture.

The Generator



Loss function we optimize:

The earlier loss function helps to produce good-looking colorful images that seem real, but to further help the models and introduce some supervision in our task, we combine this loss function with L1 Loss (you might know L1 loss as mean absolute error) of the predicted colors compared with the actual colors:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[||y - G(x,z)||_1]$$

If we use L1 loss alone, the model still learns to colorize the images but it will be conservative and most of the time uses colors like "gray" or "brown" because when it doubts which color is the best, it takes the average and uses these colors to reduce the L1 loss as much as possible (it is similar to the blurring effect of L1 or L2 loss in super resolution task). Also, the L1 Loss is preferred over L2 loss (or mean squared error) because it reduces that effect of producing gray-ish images. So, our combined loss function will be:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

where λ is a coefficient to balance the contribution of the two losses to the final loss (of course the discriminator loss does not involve the L1 loss).

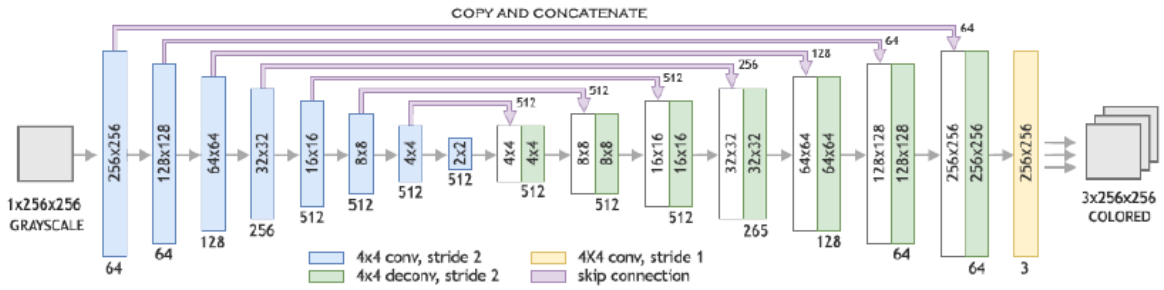
Method to create our Model

Image colorization is an image-to-image translation problem that maps a high dimensional input to a high dimensional output. It can be seen as pixel-wise regression problem where structure in the input is highly aligned with structure in the output. That means the network needs not only to generate an output with the same spatial dimension as the input, but also to provide color information to each pixel in the grayscale input image. We provide an entirely convolutional model architecture using a regression loss as our baseline and then extend the idea to adversarial nets.
In this work we utilize the L*a*b* color space for the colorization task. This is because L*a*b* color space contains dedicated channel to depict the brightness of the image and the color information is fully encoded in the remaining two channels. As a result, this prevents any sudden variations in both color and brightness through small perturbations in intensity values that are experienced through RGB.

Baseline Network:

For our baseline model, we follow the “fully convolutional network" model where the fully connected layers are replaced by convolutional layers which include upsampling instead of pooling operators. This idea is based on encoder-decoder networks where input is progressively downsampled using a series of

contractive encoding layers, and then the process is reversed using a series of expansive decoding layers to reconstruct the input. Using this method we can train the model end-to-end without consuming large amounts of memory.
Note that the subsequent downsampling leads to a much more compact feature learning in the middle layers. This strategy forms a crucial attribute to the network, otherwise the resolution would be limited by GPU memory.
Our baseline model needs to find a direct mapping from the grayscale image space to color image space. However, there is an information bottleneck that prevents ow of the low level information in the network in the encoder-decoder architecture. To fix this problem, features from the contracting path are concatenated with the upsampled output in the expansive path within the network.
This also makes the input and output share the locations of prominent edges in grayscale and colored images. This architecture is called U-Net , where skip connections are added between layer i and layer n-i.
The architecture of the model is symmetric, with n encoding units and n decoding units. The contracting path consists of 4-4 convolution layers with stride 2 for downsampling, each followed by batch normalization and Leaky-ReLU activation function with the slope of 0.2. The number of channels are doubled after each step. Each unit in the expansive path consists of a 4 - 4 transposed convolutional layer with stride 2 for upsampling, concatenation with the activation map of the mirroring layer in the contracting path, followed by batch normalization and ReLU activation function. The last layer of the network is a 1 -1 convolution which is equivalent to cross-channel parametric pooling layer.
We use tanh function for the last layer as proposed by . The number of channels in the output layer is 3 with L*a*b* color space.



U-Net architecture (256 - 256 input)

We train the baseline model to minimize the Euclidean distance between predicted and ground truth averaged over all pixels:

$$J(x;\theta) = \frac{1}{3n}\sum_{\ell=1}^{3}\sum_{p=1|}^{n} \|h(x;\theta)^{(p,\ell)} - y^{(p,\ell)}\|_2^2$$

where x is our grayscale input image, y is the corresponding color image, p and ` are indices of pixels and color channels respectively, n is the total number of pixels, and h is a function mapping from grayscale to color images.

Convolutional GAN:

For the generator and discriminator models, we followed Deep Convolutional GANs (DCGAN) guidelines and employed convolutional networks in both generator and discriminator architectures.

The architecture was also modified as a conditional GAN instead of a traditional DCGAN; we also follow guideline in and provide noise only in the form of dropout , applied on several layers of our generator. The architecture of generator G is the same as the baseline. For discriminator D, we use similar architecture as the baselines contractive path:

a series of 4 - 4 convolutional layers with stride 2 with the number of channels being doubled after each downsampling. All convolution layers are followed by batch normalization, leaky ReLU activation with slope 0.2. After the last layer, a convolution is applied to map to a 1 dimensional output, followed by a sigmoid function to return a probability value of the input being real or fake. The input of the discriminator is a colored image either coming from the generator or true labels, concatenated with the grayscale image.

Training Strategies:

For training our network, we used Adam optimization and weight initialization. We used initial learning rate of 2 *10^34 for both generator and discriminator and manually decayed the learning rate by a factor of 10 whenever the loss function started to plateau. For the hyper-parameter we followed the protocol and chose lambda= 100, which forces the generator to produce images similar to ground truth.
GANs have been known to be very difficult to train as it requires finding a Nash equilibrium of a non-convex game with continuous, high dimensional parameters . We followed a set of constraints and techniques to encourage convergence of our convolutional GAN and make it stable to train.

Alternative Cost Function:
This heuristic alternative cost function was selected due to its non-saturating nature; the motivation for this cost function is to ensure that each player has a strong gradient when that player is “losing" the game.

One Sided Label Smoothing
Deep neural networks normally tend to produce extremely confident outputs when used in classification. It is shown that replacing the 0 and 1 targets for a classifier with smoothed values, like .1 and .9 is an excellent regularizer for convolutional networks . Salimans et al demonstrated that one-sided label smoothing will encourage the discriminator to estimate soft probabilities and reduce the vulnerability of GANs to adversarial examples. In this technique we smooth only the positive labels to 0.9, leaving negative labels set to 0.

Batch Normalization
One of the main difficulties when training GANs is for the generator to collapse to a parameter setting where it always emits the same output .

This phenomenon is called mode-collapse, also known as the Helvetica scenario . When mode-collapse has occurred, the generator learns that a single output is able to consistently trick the discriminator. This is non-ideal as the goal is for the network to learn the distribution of the data rather than the most ideal way of fooling the discriminator. Batch normalization is proven to be essential to train both networks preventing the generator from collapsing all samples to a single point . Batch-Norm is not applied on the first layer of generator and discriminator and the last layer of the generator .

All Convolutional Net

Strided convolutions are used instead of spatial pooling functions. This effectively allows the model to learn its own downsampling/upsampling rather than relying on a fixed downsampling/upsampling method. This idea was proposed in and has shown to improve training performance as the network learns all necessary invariances just with convolutional layers.

Reduced Momentum

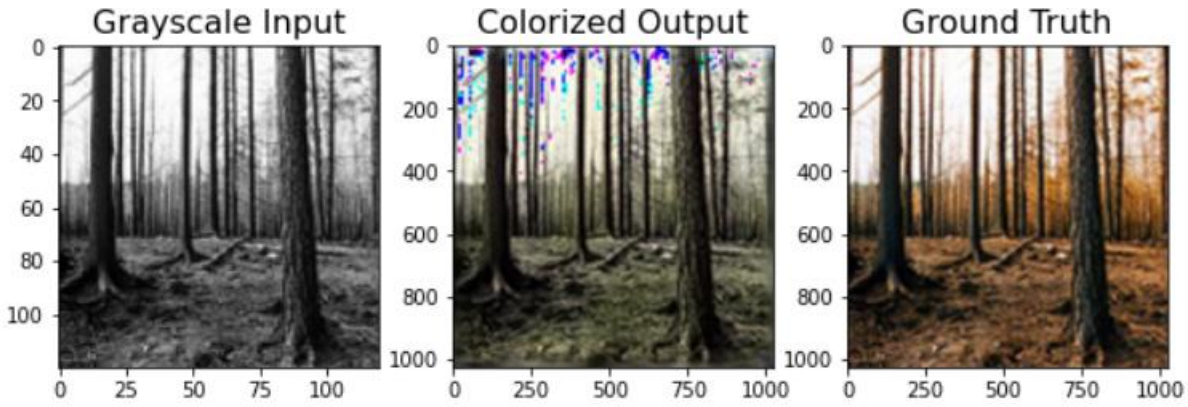
We use Adam optimizer for training both networks. Recent research has shown that using a large momentum term Beta1 (0.9 as suggested), could result in oscillation and instability in training. We followed the suggestion in to reduce the momentum term to 0.5.

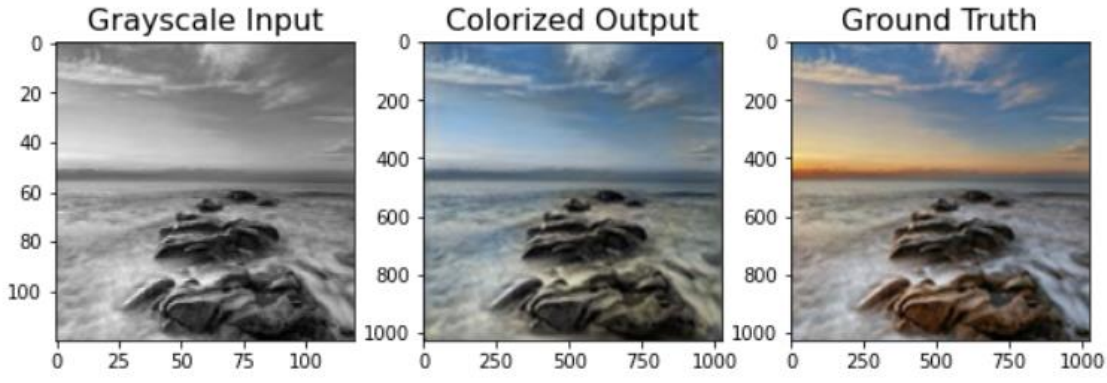
LeakyReLU Activation Function

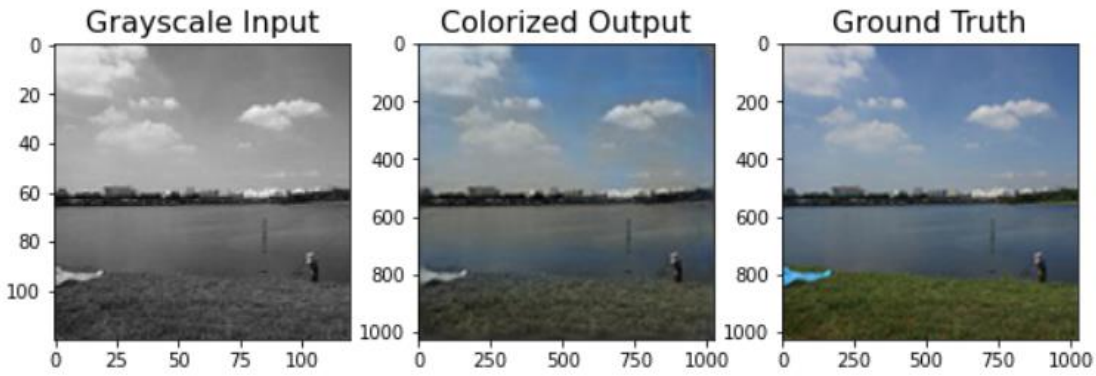
Radford et al. showed that using leaky ReLU activation functions in the discriminator resulted in better performance over using regular ReLUs. We also found that using leaky ReLU in the encoder part of the generator as suggested by works slightly better.

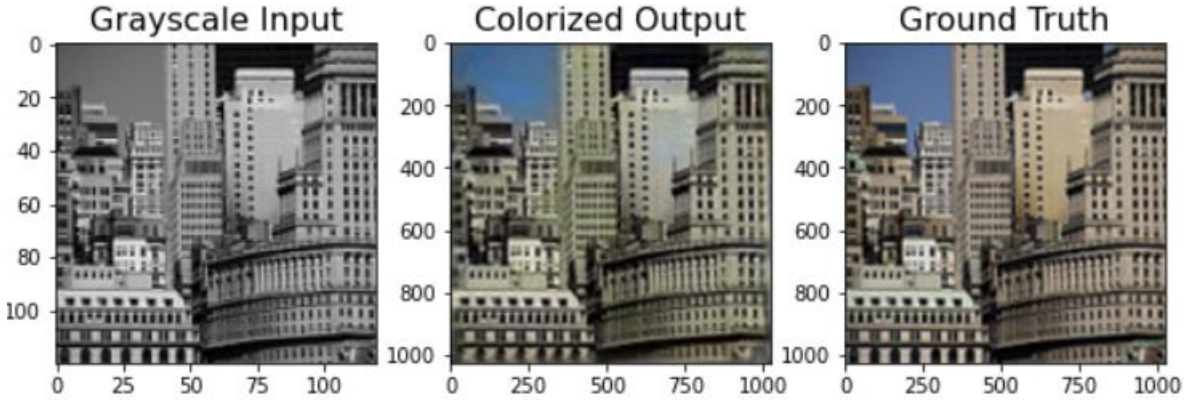
Results

The result is shown in three columns, in the first column we have gray scale image then in the second column we have generated output using the model the in the third column we have original image.









In this study, we were able to automatically colorize grayscale images using GAN, to an acceptable visual degree.
Many of the images generated by U-Net had a brownish hue in the results known as the “Sepia effect" across L*a*b* color space. This is due to the L2 loss function that was applied to the baseline CNN, which is known to cause a blurring effect.
Mis-colorization was a frequent occurrence with images containing high levels of textured details. This leads us to believe that the model has identified these regions as grass since many images in the training set contained leaves or grass in an open field.
Overall the results look promising and the accuracy seems pretty good.

References

1. Dataset Link: https://drive.google.com/drive/folders/1dXnWUuM98b6KuoaO9tCw_1Y1ShPZ-XcT?usp=sharing
2. Google Colab notebook with code: https://colab.research.google.com/drive/1wLaQDNk-ipAv0ugCvXYxQPzCK0fVDH1y?usp=sharing
3. Based on research paper: https://www.researchgate.net/publication/325803914_Image_Colorization_Using_Generative_Adversarial_Networks?enrichId=rgreq-714ac41b812d5d19b4c6dbb19a389fd9-XXX&enrichSource=Y292ZXJQYWdlOzMyNTgwMzkxNDtBUzo3MTc3MzY0Njg1NTM3MjhAMTU0ODEzMzExNzQ1Mg%3D%3D&el=1_x_3&_esc=publicationCoverPdf