Dlops - Group Project

Project-title - Image coloring using Generative adversarial networks

---

Team Members

- Gautam Kumar (B19EE031)
- Mayank Raj (B19CSE053)
- Nirbhay Sharma (B19CSE114)

Introduction

Color is power which directly influences the soul -Wassily Kandinsky

Generative Adversarial Networks or GANs are deep learning architectures which is used for generative modelling. In terms of image data,GANs consist of a generator who generates an image and a discriminator which discriminate whether the image is real or fake. The aim of the generator is to fool discriminator by generating a fake image which the discriminator cannot distinguish. While discriminators aim to distinguish the fake images. There is a minmax game between generator and discriminator.Image-to-Image translation is the controlled conversion of a given source image to a target image.The pix2pix GAN is a general approach for image-to-image translation. In this project, we are using pix2pix Generative Adversarial Network (GAN) for the task of coloring black and white images. We have prepared two methods : one for RGB images and one for LAB images.

Image coloration using RGB format

In $R \times G \times B$ image format we have three pixels R (Red), G(Green), B(Blue), all the 3 pixels when combined, generates a colored image, and so to train a pix2pix model on this format we have to pass in 1 channels image (gray-scale) and predict 3 values (RGB) for each pixel i.e we need to make 1 channel image to 3 channels, which is a bit tough and computational heavy task for model. below also shown the RGB format image.

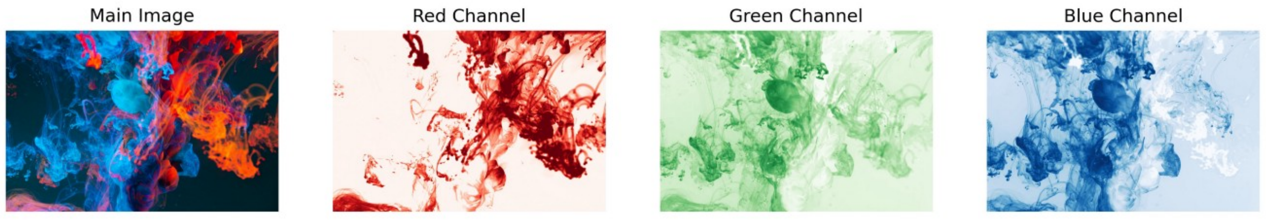
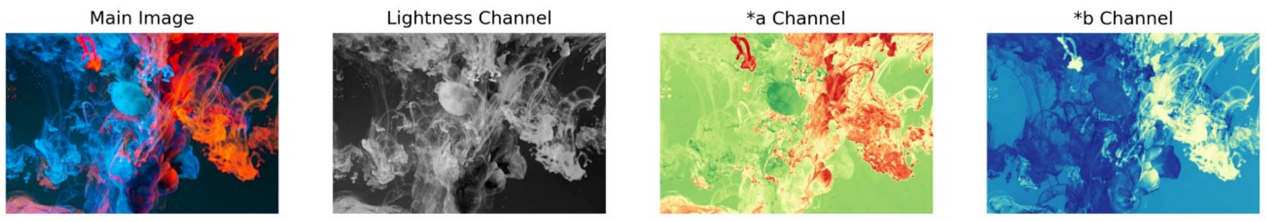


Image coloration using LAB format

In $L \times A \times B$ color space, we have 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 seco 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, so to train a pix2pix model on this we need to pass in L image and predict A and B and so the prediction has to be less and reduced by 1 from the case in RGB image. In the following image you can see each channel of LAB color space separately.



Methods and Setup

The dataset we are using for training the model using RGB format is random natural images from stl-10 dataset, so the training pipeline is as follows for RGB

- first we are importing the colored image from stl-10 dataset
- then we are concatenating the gray scale image with itself 3 times to make a 3 channel gray scale image
- then we are using a tuple of image as follows (gray-scale-image, colored-image) both are 3 channel images
- then passing to the generator for further training the model

The dataset we are using for training the model using LAB format is the Image coloration dataset from kaggle https://www.kaggle.com/datasets/shravankumar9892/image-colorization which is the dataset in the $L \times A \times B$ format, the training pipeline is as follows for LAB format

- first we are importing the colored image from the dataset in the ab format
- then we have a tuple of images (L-format, AB-format)
- then we are passing it to generator to generate an AB image format and so the model is getting trained

setup

LAB-format

RGB-format

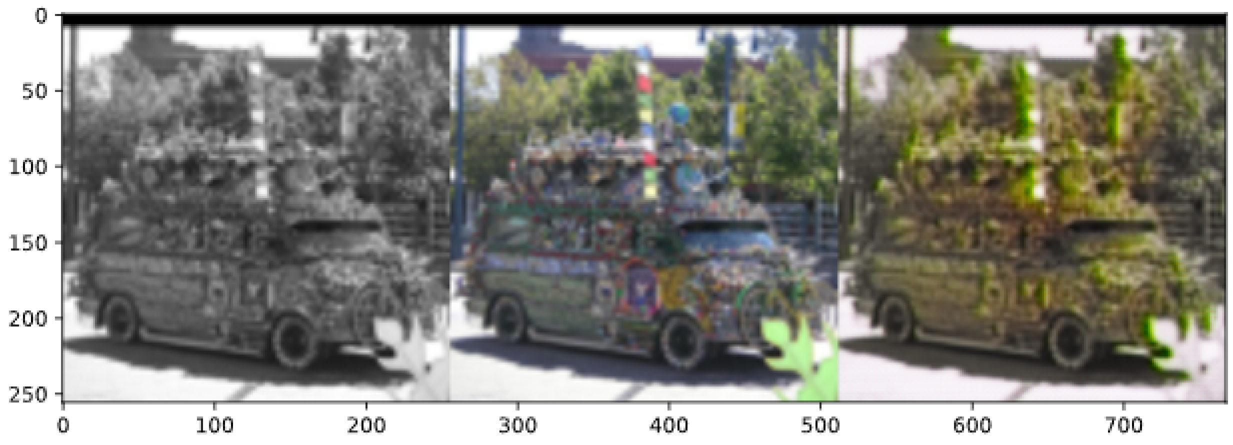
optimizations

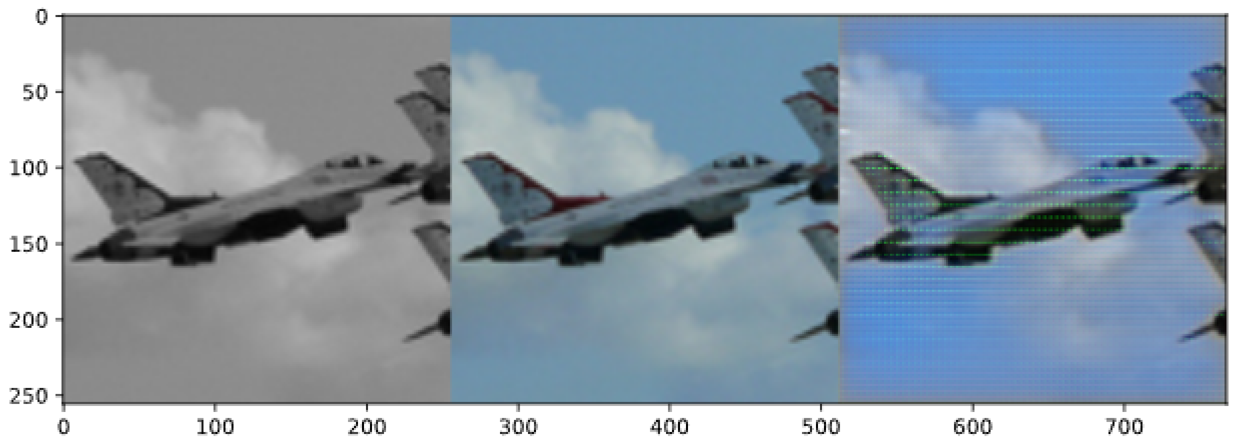
- we are applying optimizations such as pin_memory = True, n_workers = 2/4/8 etc in the dataloader which will make the data loading very fast
- next we are using some optimizations like torch.backends.cudnn.benchmarks = True torch.backends.cudnn.deterministic = True torch.backends.cuda.matmul.allow_tf32 = True torch.backends.cudnn.allow_tf32 = True, which will pick the best algorithms for the convolution operations
- next we are using amp(automatic mixed precision) using torch.cuda.amp.GradScaler() which is basically using the fp_16 for operations and hence makes the code much faster to

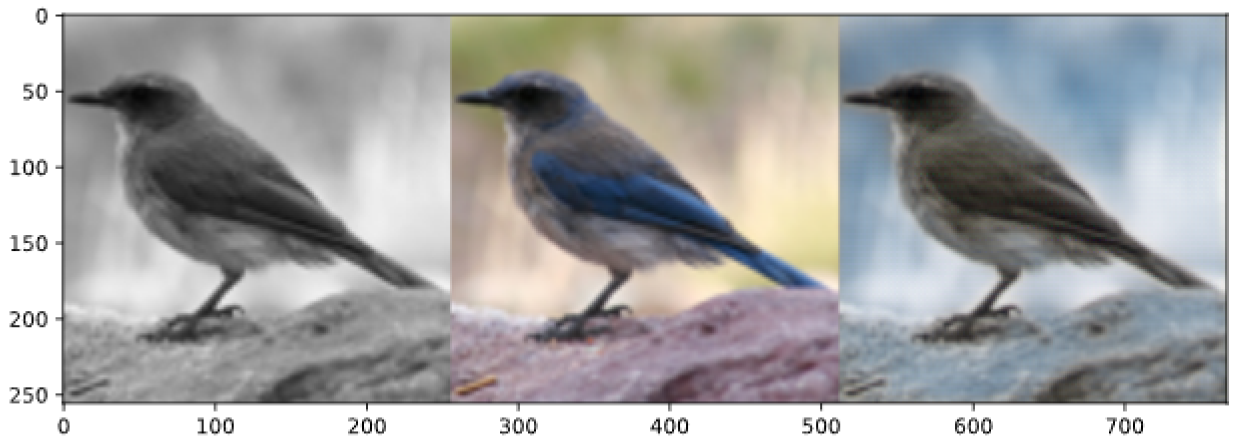
run computationally

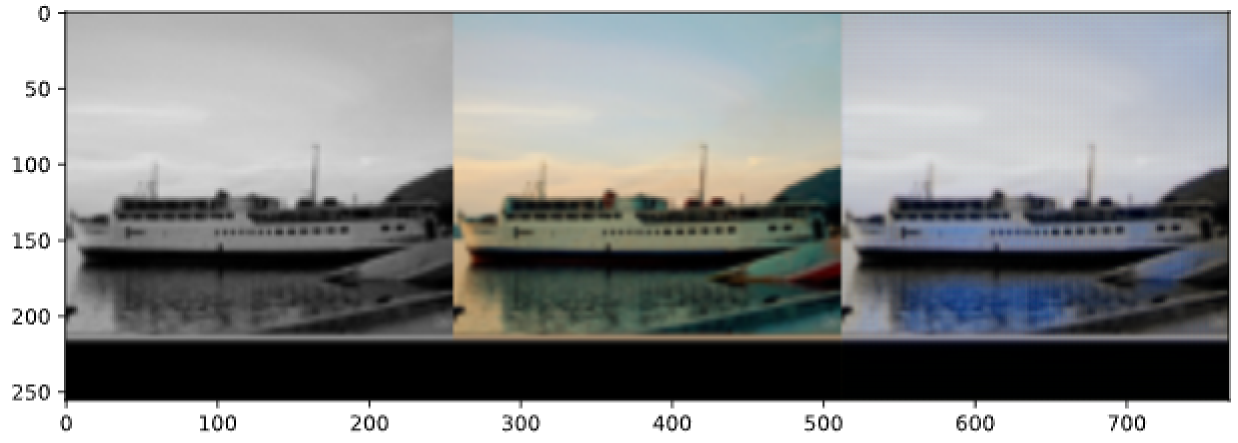
Results

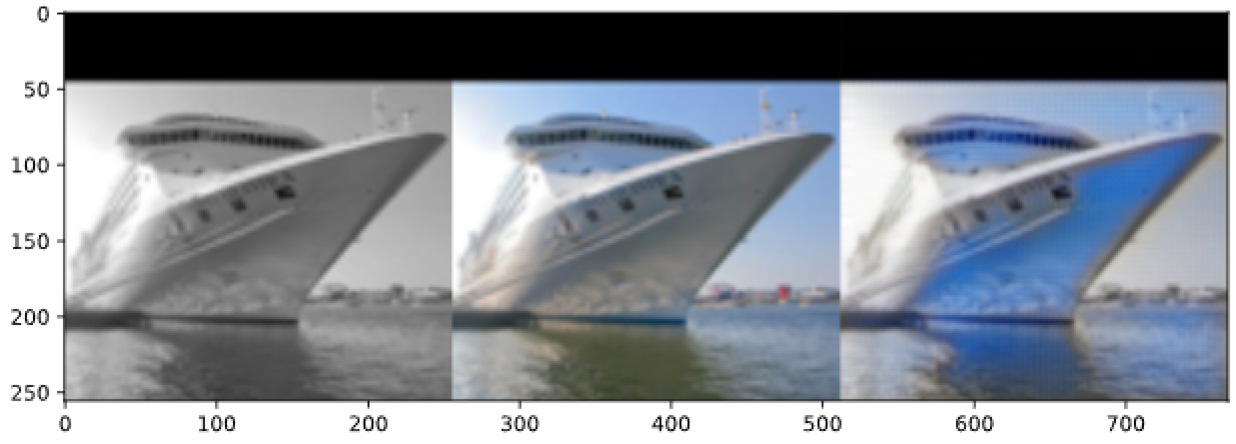
RGB Results:











LAB Results:

Analysis On Results

Effect of optimizations

- the only effect of optimization is on training time of the models the training was so fast using the above techniques that the models are trained very fast on large datasets as well will huge image size

Conclusion
