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 which generates an image and a discriminator which discriminate images. 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. The code is available at https://github.com/nirbhay-design/dlops-project

- The wandb link for RGB-format model is available at: RGB-wandb
- The wandb link for LAB-format models is available at: LAB-wandb

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.

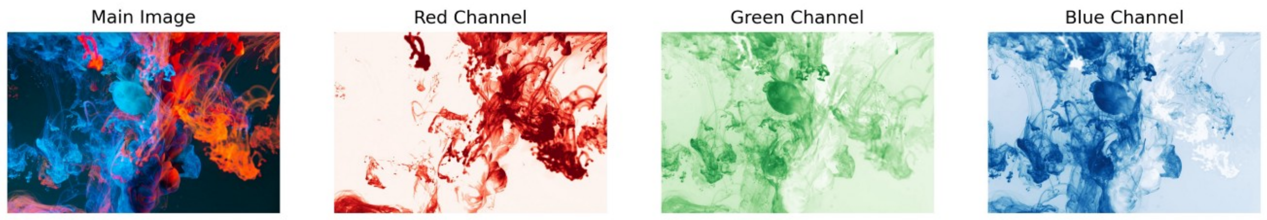
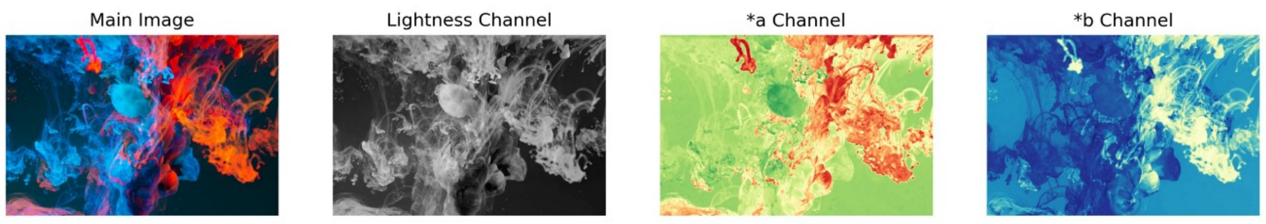


Image coloration using LAB format

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 yellowblue 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 We use a UNET-GAN model to convert $256 \times 256 \times 1$ grayscale images to $256 \times 256 \times 2$ 'AB' format images. We use batch size of 8 and trained the model for 60 epochs. We made a custom optimizer to optimize our models which computes the gradient and update the weights. Our loss function for generator is weighted sum of MSE (mean squared error) and BCEWithLogitsLoss. For discriminator, we use Binary-Crossentropy as the loss function. Our GAN model uses Batch-normalization, LeakyReaLU and L1 regularization to facilitate back-propagation and prevent overfitting.

RGB-format We are using pix2pix architecture to train on the gray scale images with an initial learning rate of $2e^{-4}$, with Adam optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$, we are using a minibatch size of 16,32 (in some cases 16, and in some 32) with image size as $(3 \times 256 \times 256)$ and trained for 30-50 epochs depending on the previous iteration result, we are using naive GAN loss and l1 loss for image generation and Automatic mixed precision training using GradScaler.

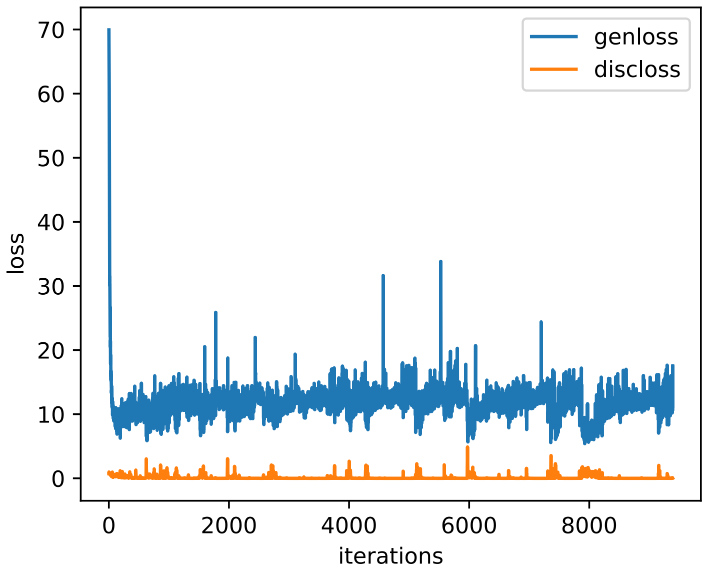
Optimizations

- 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

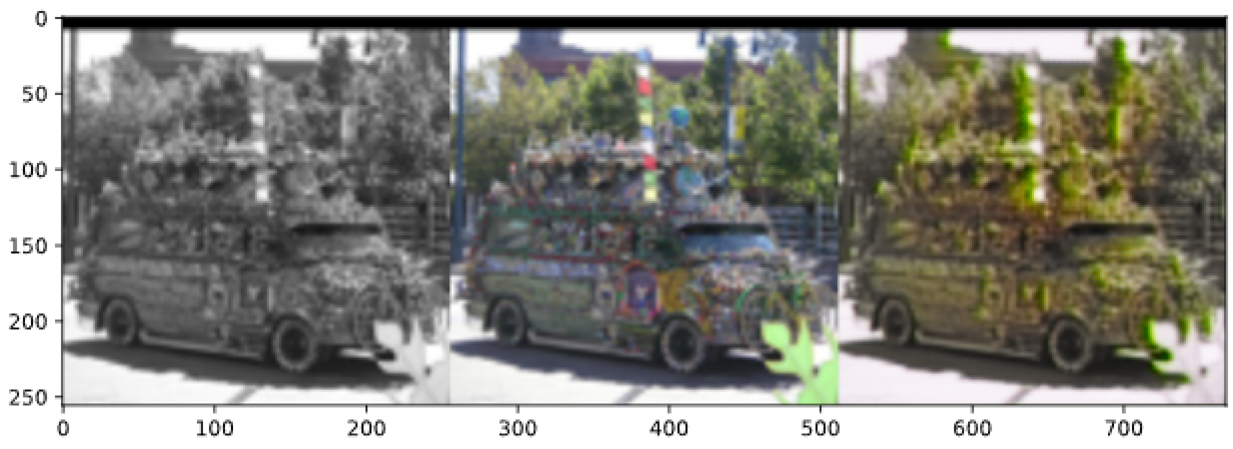
RGB Results:

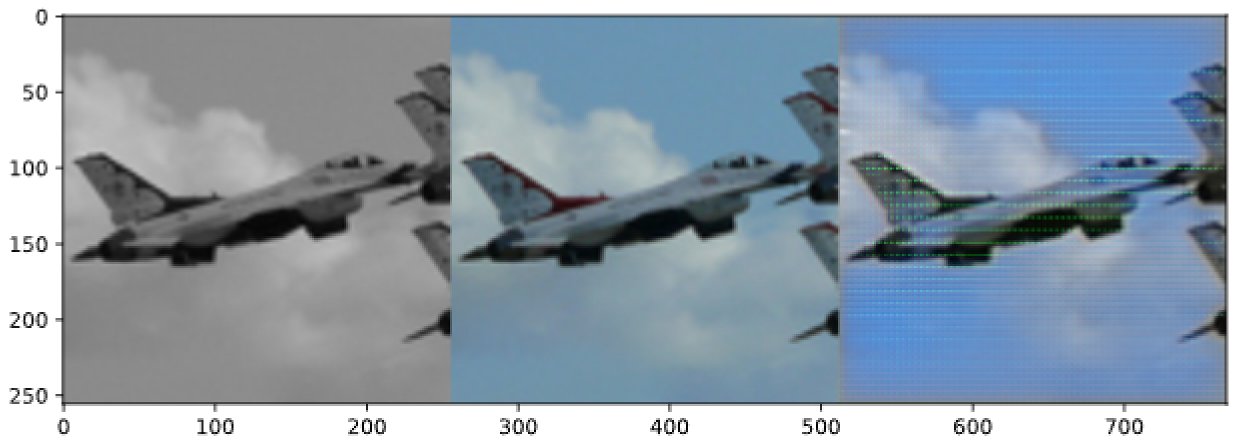
Loss vs Iterations curve



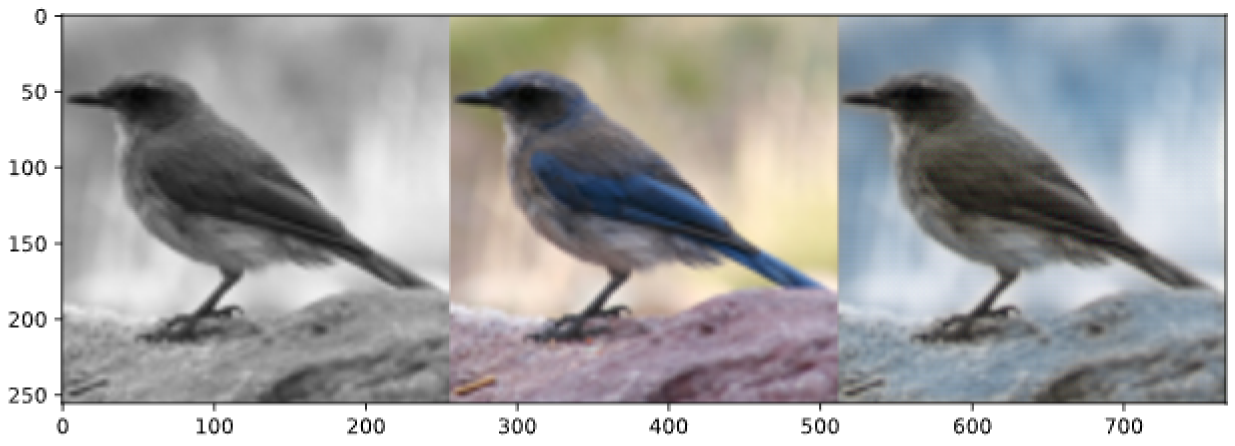
sample unseen test images

The images are in the format [Gray_image, Ground_truth, Generated_image]

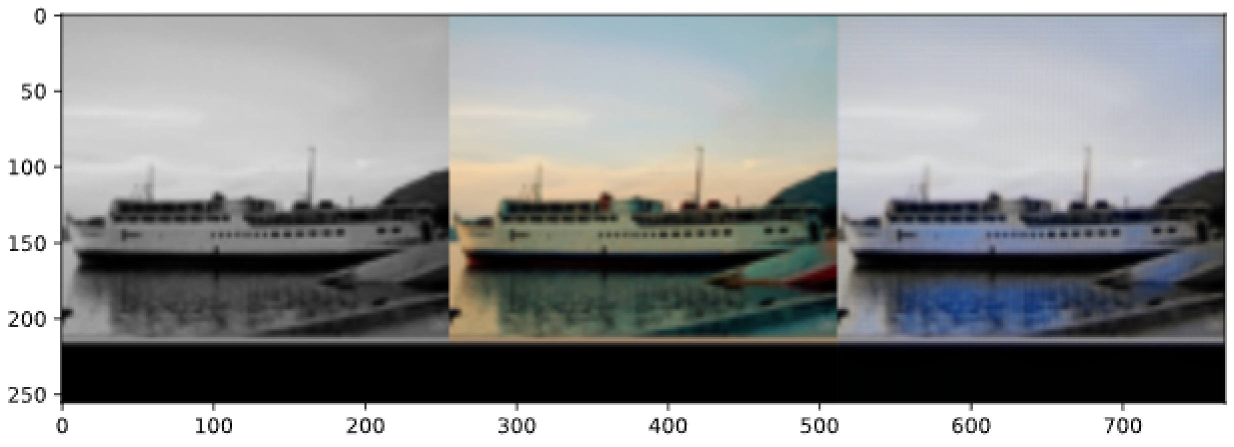




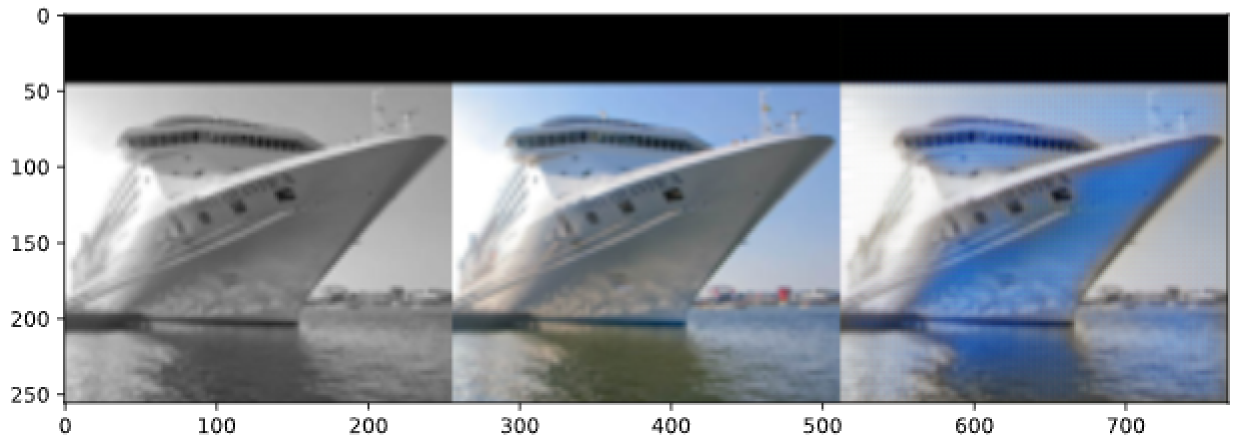
0
50
100
150
200
250
0
100
200
300
400
500
600
700



0
50
100
150
200
250
0
100
200
300
400
500
600
700



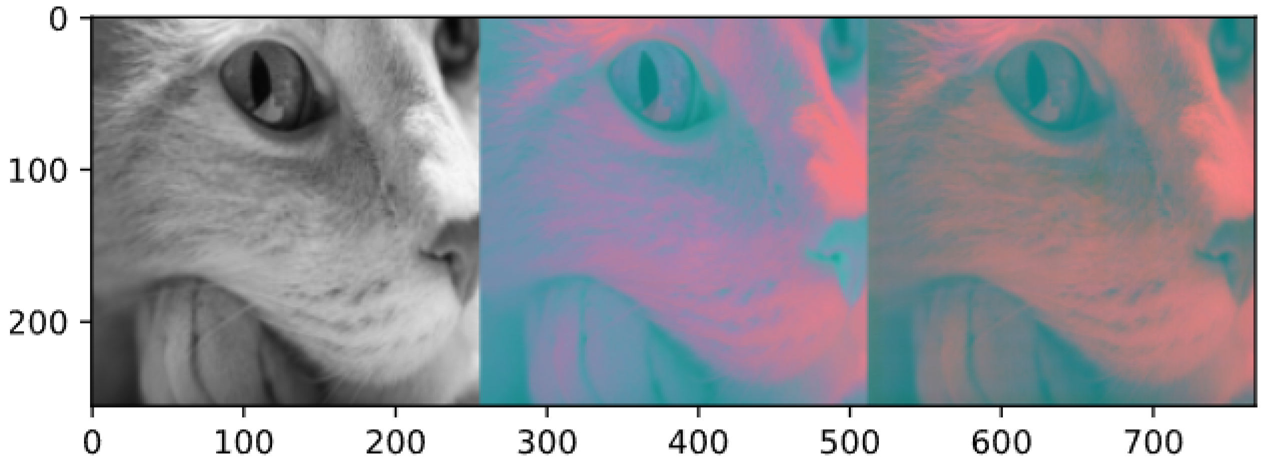
0
50
100
150
200
250
0
100
200
300
400
500
600
700

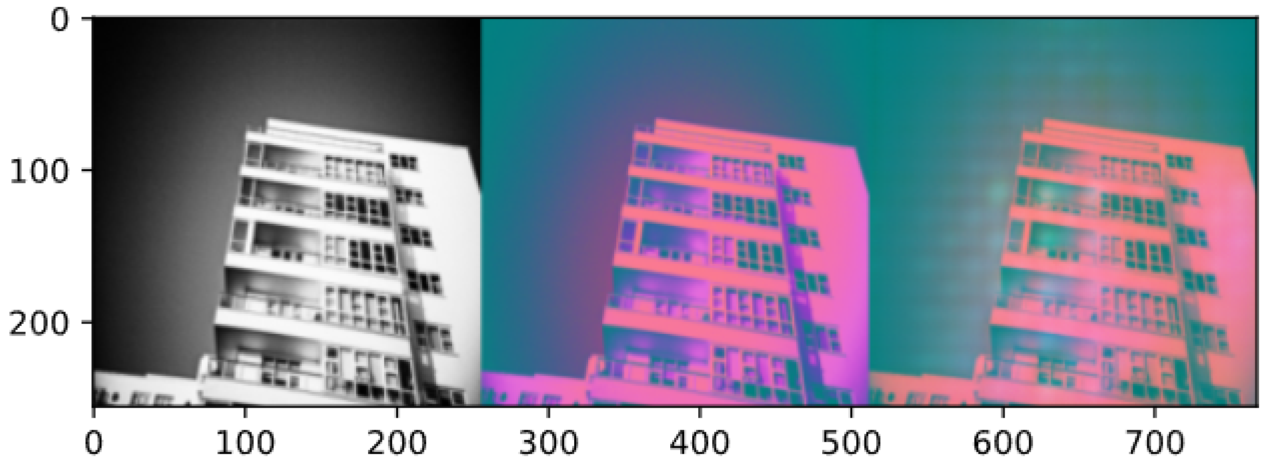


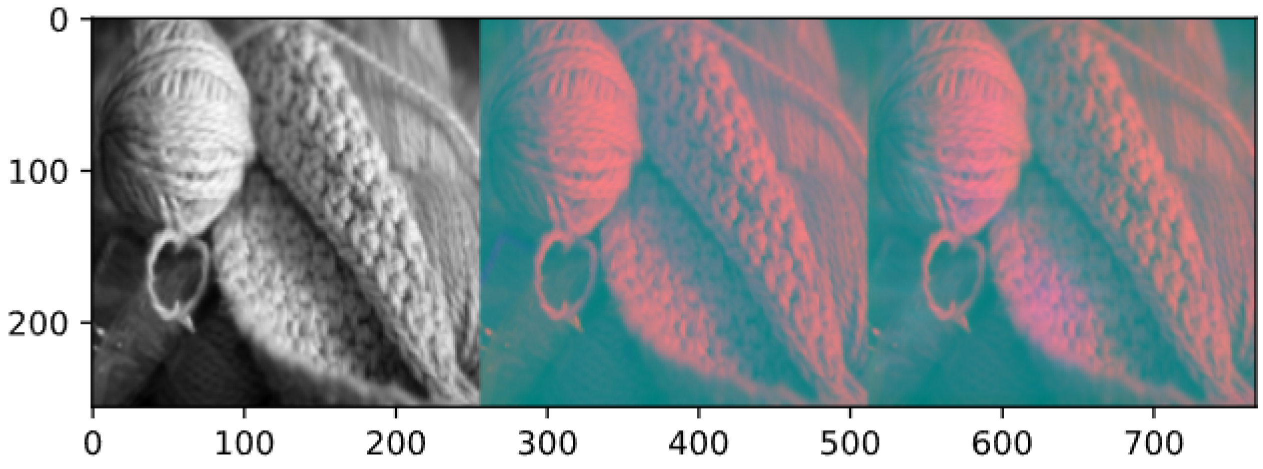
LAB Results:

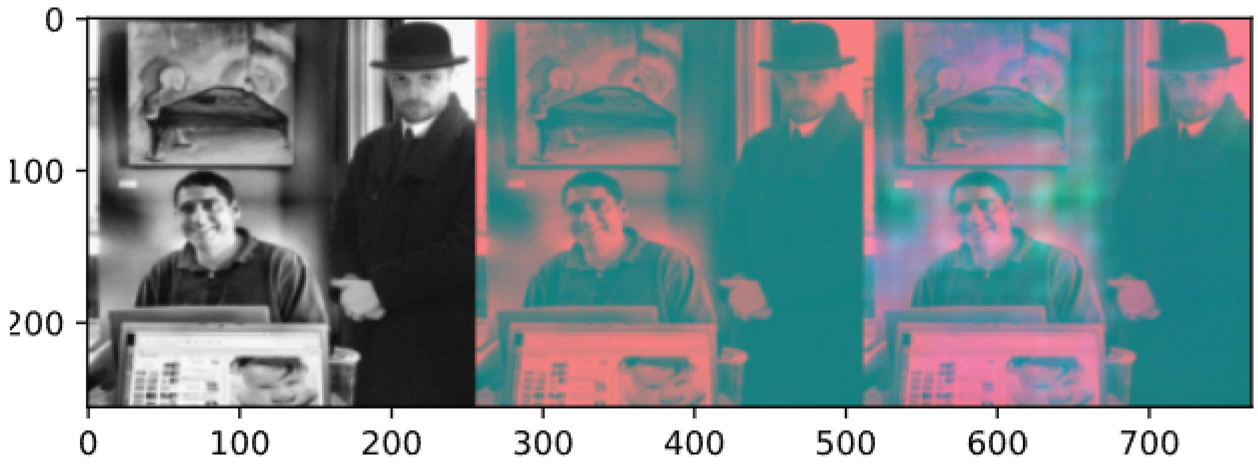
sample unseen test images

The images are in the format [Gray_image, Ground_truth, Generated_image]









Analysis On Results

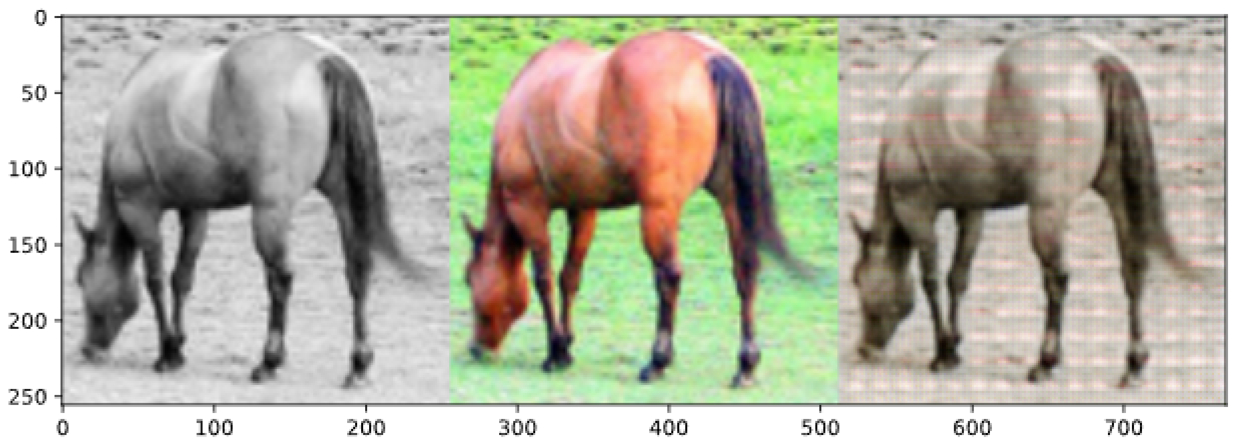
RGB-format results The results shown above are nice and some of the image above are actually really good while some of them has drawback of the model. So as we can infer from the images that it kind of learns to fill the blue color much so as we can see it tries to fill blue color to the images and thus the images which are related to sky, water are looking really nice but some images which does not have water, sky etc doesn't looks that good.

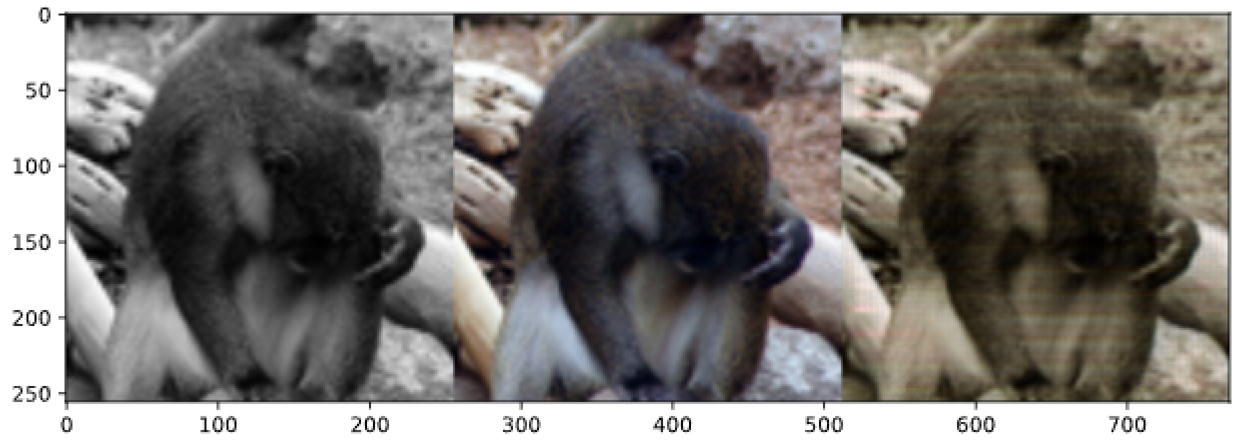
LAB-format results Our GAN model successfully learns to map grayscale images to LAB format images. However, it also adds some shine in the produced images. This glow is added wherever there is a possibility of presence of luminous object like reflection of sunlight, shining eyes, reflection from sea water etc. This additional effect actually improves the quality of reproduced image in some cases. In nutshell, we can say that the models learnt the mappings in an optimal manner.

ONNX conversion and ONNX-inferencing

- scripts are written to convert the .pt model to .onnx format using torch.onnx.convert() method and the results for onnx inferencing code is shown below

ONNX-inferencing





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 & future works

The Generative Adversarial networks are now the most popular deep learning architectures for image generation tasks and there are immense posibilities to work in the area of GAN's, we are just doing one use of GAN's i.e the Image coloration which is pix2pix architecture and future extensions are a lot in this domain as well, we can generate faces, animals etc, whatever we need to generate we can generate with the help of GAN's and so the future works includes to work upon these architectures more and apply to various deep learning areas.