COLORIZATION OF GRAYSCALE IMAGE

INTRODUCTION:

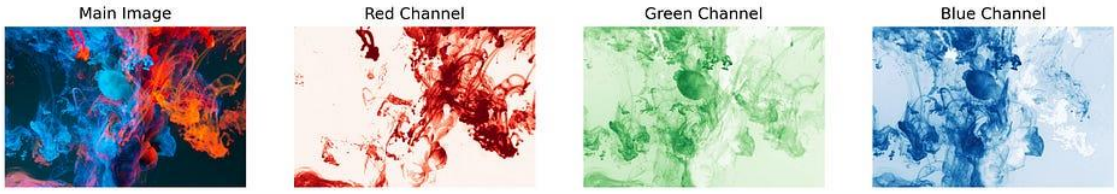
- The idea of colouring a grayscale image is a task that is simple for us, we learn from our childhood to fill in missing colours in colouring books, by remembering that grass is green, the sky is blue with white clouds or that an apple can be red or green. And the from an image. Like in the year around 1980, many movie creators resist the idea of colourizing their black and white films and thought of it as destruction of their art. But today it is accepted as an amplification to their work of art. So, an AI based model is best suited for this conversion.
- One of the most exciting applications of deep learning is colorizing greyscale 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**. In this article, I'm going to explain what we did to make this happen, and the strategies that helped and those that were not useful. Before that, we will explain the colorization problem and give you a short review of what has been done in recent years.

Introduction to colorization problem:

About a month ago, we didn't know much about the problem of image colorization, so we started to study deep learning papers related to this task. At the beginning, it seemed difficult but by doing a lot of Google searches, and studying a lot more papers on the problem, we gradually felt more comfortable with the colorization problem. Here I'm going to give you some basic knowledge that you may need to understand what the models do.

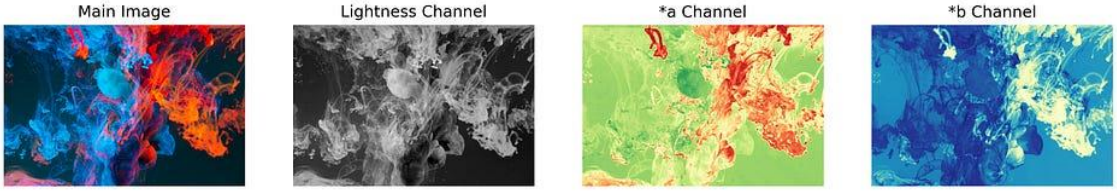
RGB vs L*a*b

- 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 ****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.



- In all papers we studied and all codes we checked out on colorization on GitHub, people use L*a*b color space instead of RGB to train the models. There are a couple of reasons for this choice, but I'll give you an intuition of why we make this choice. To train a model for colorization, we should give it a grayscale image and hope that it will make it colorful. When using L*a*b, we can give the L channel to the model (which is the grayscale image) and want it to **predict the other two channels** (*a, *b) and after its prediction, we concatenate all the channels, and we get our colorful image. But here we had used RGB, we had first **converted our image to grayscale**, feed the grayscale image to the model and hope **it will predict 3 numbers.** We'll now parse the images (RGB images to be precise) one by one and transform each one to a grayscale image using PIL's. convert ('L') method. So, our dataset will have samples of (grayscale image, RGB image) We used only a part of our dataset, determined by dataset split, as Collab's computational power would cease on providing a large number of images.

The Strategy:

Generative Adversarial Network

- In 2014, Goodfellow et al. [1] proposed a new type of generative model: generative adversarial networks (GANs). A GAN is composed of two smaller networks called the

generator and **discriminator.** As the name suggests, the generator's task is to produce results that are indistinguishable from real data. The discriminator's task is to classify whether a sample came from the generator's model distribution or the original data distribution. Both subnetworks are trained simultaneously until the generator is able to consistently produce results that the discriminator cannot classify. The architectures of the generator and discriminator both follow a multilayer perceptron model. Since colorization is a class of image translation problems, the generator and discriminator are both convolutional neural networks (CNNs). The generator is represented by the mapping G(z; θG), where z is a noise variable (uniformly distributed) that acts as the input of the generator. Similarly, the discriminator is represented by the mapping D(x; θD) to produce a scalar between 0 and 1, where x is a color image. The output of the discriminator can be interpreted as the **probability** of the input originating from the training data. These constructions of G and D enable us to determine the optimization problem for training the generator and discriminator: G is trained to minimize the probability that the discriminator makes a correct prediction in generated data, while D is trained to maximize the probability of assigning the correct label. Mathematically, this can be expressed as

$$\min \theta G \, J\,(G)\,(\theta D, \theta G) = \min \theta G \, Ez\,[\log(1 - D(G(z)))] \tag{1}$$

$$\max \theta D \, J\,(D)\,(\theta D, \theta G) = \max \theta D\,(Ex\,[\log(D(x))] + Ez\,[\log(1 - D(G(z)))]) \tag{2}$$

- The above two equations provide the cost functions required to train a GAN. In literature, these two cost functions are often presented as a single minimax game problem with the value function V (G, D):

$$\min G \max D \, V\,(G, D) = Ex\,[\log D(x)] + Ez\,[\log(1 - D(G(z)))]\,. \tag{3}$$

In eqn -1, the cost function is defined by the minimizing the probability of the discriminator be correct. However, this arises two issues in this approach:

1.) If the discriminator performs well during training stages, the generator will have a near-zero gradient during back-propagation. This will tremendously slow down convergence rate because the generator will continue to produce similar results during training.
2.) The original cost function is a strictly decreasing function that is unbounded below. This will cause the cost function to diverge to $-\infty$ during the minimization process.

- To address the above issues, we have redefined the generator's cost function by **maximizing the probability of the discriminator being mistaken**, as opposed to minimizing the probability of the discriminator being correct.

$$\max \theta G \, J\,(G) * (\theta D, \theta G) = \max \theta G \, Ez\,[\log(D(G(z)))]\,, \tag{4}$$

- In addition to these the cost function was redefined using L1 norm in the regularization term. This produces an effect where the generator is forced to produce results that are like the ground truth images. This will theoretically preserve the structure of the original images and prevent the generator from assigning arbitrary color to pixels just to **fool**

the discriminator. The cost function takes the form. (Here λ is a regularization parameter and y is the ground truth color labels)

$$\min \theta G\, J(G) * (\theta D, \theta G) = \min \theta G - Ez\,[\log(D(G(z)))] + \lambda kG(z) - yk1 \quad (5)$$

Conditional GAN

In a traditional GAN, the input of the generator is randomly generated noise data z. However, this approach is not applicable to the automatic colorization problem because grayscale images serve as the inputs of our problem rather than noise. This problem was addressed by using a variant of GAN called conditional generative adversarial networks. Since no noise is introduced, the input of the generator is treated as zero noise with the grayscale input as a prior, or mathematically speaking, G(0z|x). In addition, the input of the discriminator was also modified to accommodate for the conditional network. By introducing these modifications, our final cost functions are as follows:

$$\max \theta D\, J(D)\,(\theta D, \theta G) = \max \theta D\,(Ey\,[\log(D(y|x))] + Ez\,[\log(1 - D(G(0z|x)|x))]) \quad (6)$$

$$\min \theta G\, J(G)\,(\theta D, \theta G) = \min \theta G - Ez\,[\log(D(G(0z|x)))] + \lambda kG(0z|x) - yk1 \quad (7)$$

The discriminator gets coloured images from both generator and original data along with the grayscale input as the condition and tries to decide which pair contains the true coloured image.

Methodology

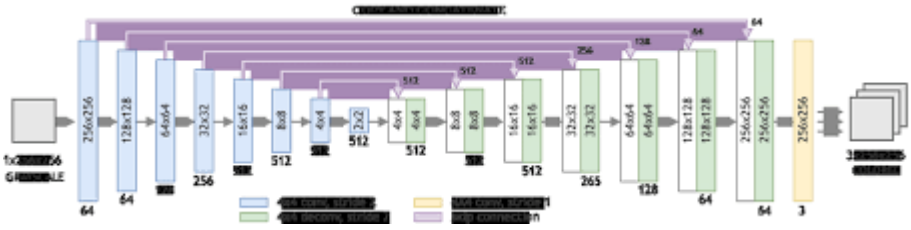
Image colorization is an image-to-image translation problem that maps a high dimensional input to a high dimensional output. It can be seen as pixel-wise regression problem where structure in the input is highly aligned with structure in the output. That means the network needs not only to generate an 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 up sampling instead of pooling operators. This idea is based on **encoder decoder networks** where input is progressively **down sampled** 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 4 train the model end-to-end without consuming large amounts of memory. Note that the subsequent down sampling 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

Problem: However, there is an information bottleneck that prevents flow of the low-level information in the network in the encoder-decoder architecture.

Solution: To fix this problem, features from the contracting path are concatenated with the up sampled output in the expansive path within the network. This also makes the input and output share the locations of prominent edges in grayscale and coloured 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 down sampling, each followed by batch normalization and Leaky-ReLU activation function with the slope of 0.2. The number of channels is doubled after each step. Each unit in the expansive path consists of a 4 × 4 transposed convolutional layers with stride 2 for up sampling, 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. 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.

Convolution 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 [4] 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 down sampling. 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 coloured 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 proposed by [2/16]. We used initial learning rate of 2 × 10−4 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 λ we followed the protocol from [5] and chose $\lambda = 100$, which forces the generator to produce images like ground truth.

We followed a set of constraints and techniques, to encourage convergence of our convolutional GAN and make it stable to train.

- Alternative Cost Function:
- 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 regularize for convolutional networks.
- Batch Normalization
- All Convolutional Net
- Reduced Momentum
- LeakyReLU Activation Function

Benefits of GAN over another model:

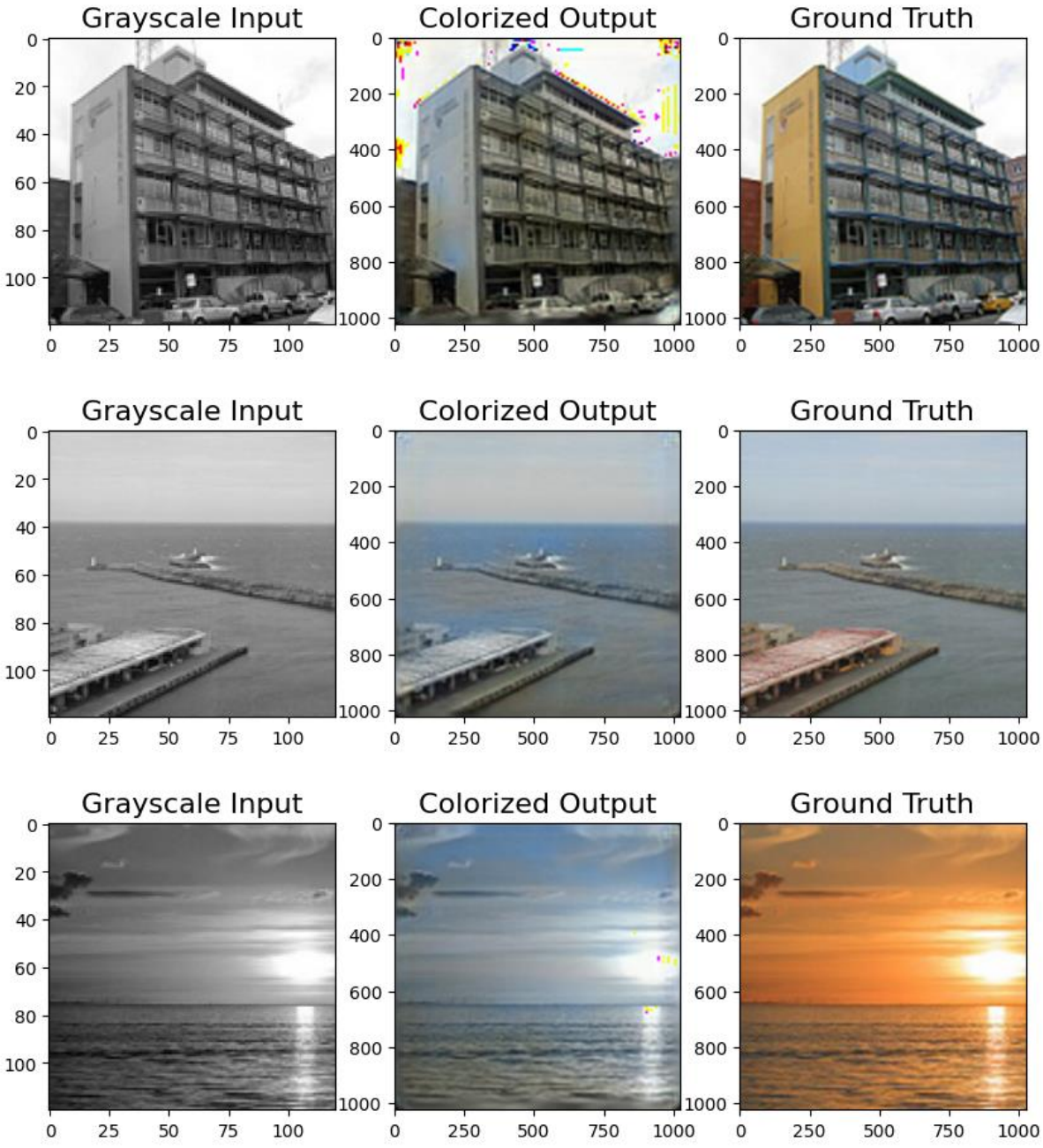
GANs can generate highly realistic and diverse colorization of greyscale images. They are well suited for to create new and diversing sample, but they are more difficult to train and require more computational resources then CNNs.

Dataset	Size	Network	Batch Size	EPOCHs	Accuracy(5%)	Delta-E2000 (0.0 means identical)
Oxford Paintings (10k images)	32x32	GAN	32	666	66.63%	10.9
Oxford Paintings (18k images)	224x224	ConvNet	64	220	60.87%	13.91
CIFAR-10 (50k images)	32x32	GAN	32	200	36.64%	14.82

Some Proven Results for GAN to be better than other Model.

Simulation Results:

At 150 epochs



Therefore, the overwhelming results depicts the power of GANs and the disruption which can be brought through them.

Links:

https://github.com/AbhinavM71/Chromify

https://colab.research.google.com/drive/1bWJDYqGsuMe7VYnBH_V-hECombi-e3rc#scrollTo=4pM5xHCEBueF

References:

[1]. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014.

[2]. Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images. In ACM TOG, volume 21, pages 277–280, 2002

[3]. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, 2015.

[4]. Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. 2017.

[5]. Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization.

[6]. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning, 2015.

[7]. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. 2016.