Deep Learning Image Colorization (02456 - Deep Learning E21)

Nikiforos Efthymiadis(s202430), Sixian Zuo(s210365), Renjue Sun(s181294), Abdulstar Kousa(s174360), Malthe Andreas Lejbølle Jelstrup(s184291)
DTU Compute, Technical University of Denmark

Introduction

In this poster, we investigate various methods for automatically converting grayscale images to colorful ones, a process known as **Image Colorization**. We started from scratch and learned the tools and skills we needed along the way. This problem is challenging because it is multimodal -- many plausible colored images can be derived from a single grayscale photograph.

Dataset

Our Dataset consists of 2673 Images from **ImageNet** containing images from 27 different classes. Some examples are flowers, animals, food, people, landscapes. We split it into: **Training Set:** 2133 Images. **Validation Set:** 540 Images.

Color Space

Instead of working with images in the RGB format, we'll use the **LAB colorspace** (Lightness, A, and B). This colorspace provides the same information as RGB, but it makes it easier to distinguish between the lightness and the other two channels (which we call A and B). The A and B channels encode how much green-red and yellow-blue each pixel is, respectively.[1]

Results - Qualitative

Figures beside are from the validation-set. The first line illustrates our bad results and the second line shows our good results.

For the baseline model, the results are quite gray comparing with others.

For the classifier model, as butterflies are normally with similar shapes and colors and the background of butterfly pictures are always with grass and flowers, the model can capture these features easily and do the colorization accordingly. However, birds are quite varied from looks, shapes and colors. This makes the model confused and decides to only go with neutral colors.

Attention U-Net introduces Attention Gates. Models trained with Attention Gates implicitly learn to suppress irrelevant regions in an input image while highlighting salient features useful for a specific task.[2] This characteristic can normally improve model performance but if features are highlighted incorrectly or too much, it can also harm the model performance.

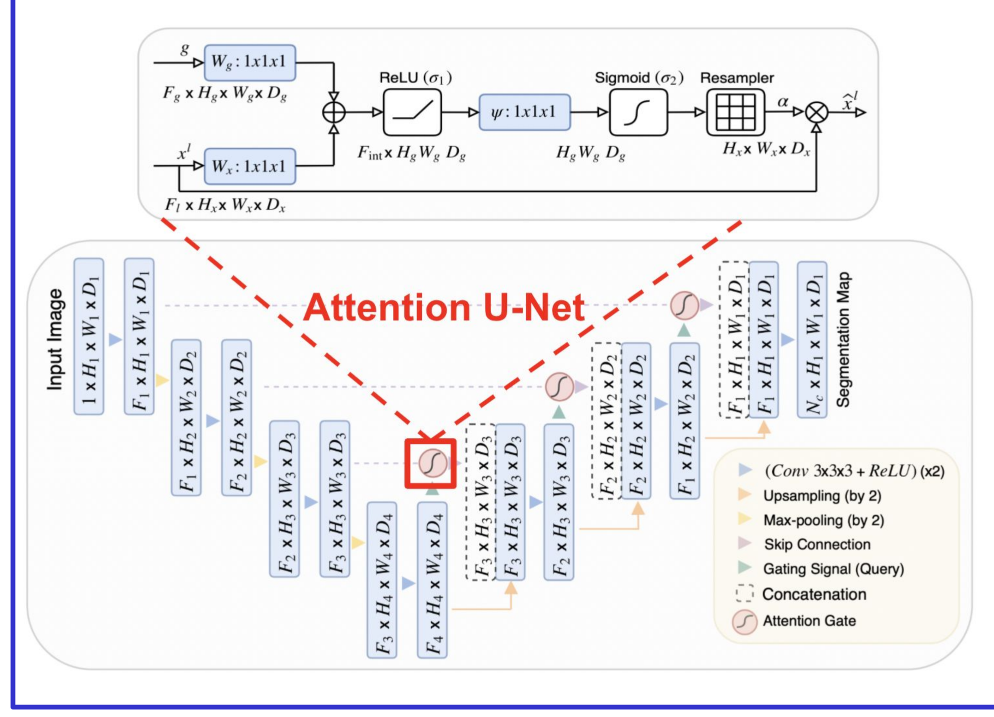
For the Attention model, when it captures some important features correctly, it will normally colorize them with vibrant colors. The color of the butterfly generated by pre-trained UGAN is even more vibrant than the original one. However, the Attention model wrongly highlights the bird's head and body and also some noises on the background. This also results in quite vibrant colorization but a better solution is to go with neutral colors.

When there is an input image with many weighted features, the classifier model tends to become conservative and only uses neutral colors. On the other hand, the Attention model turns out to be bolder and applies quite vibrant colors.

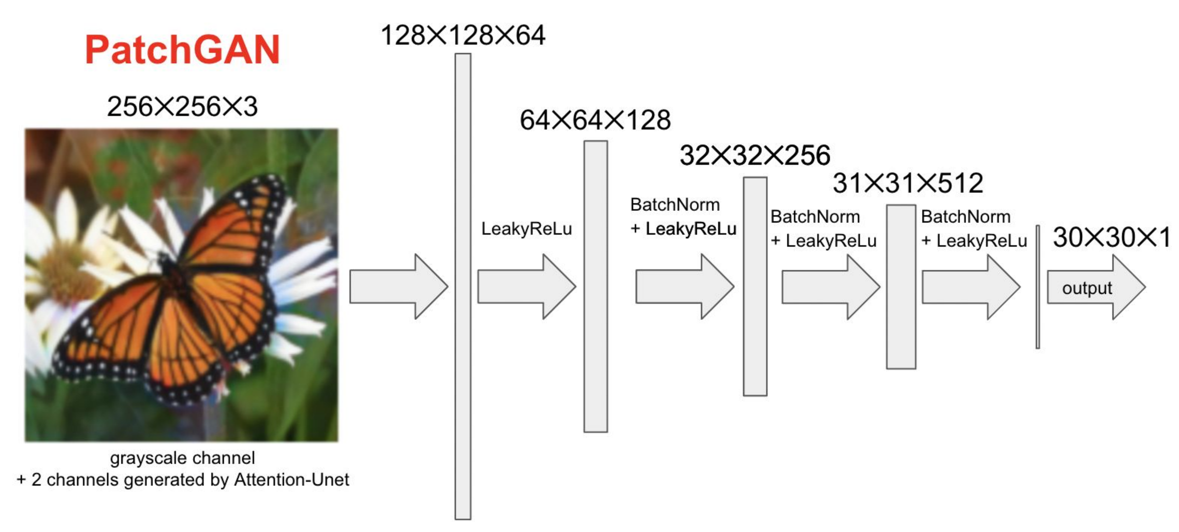
Models

U-GAN

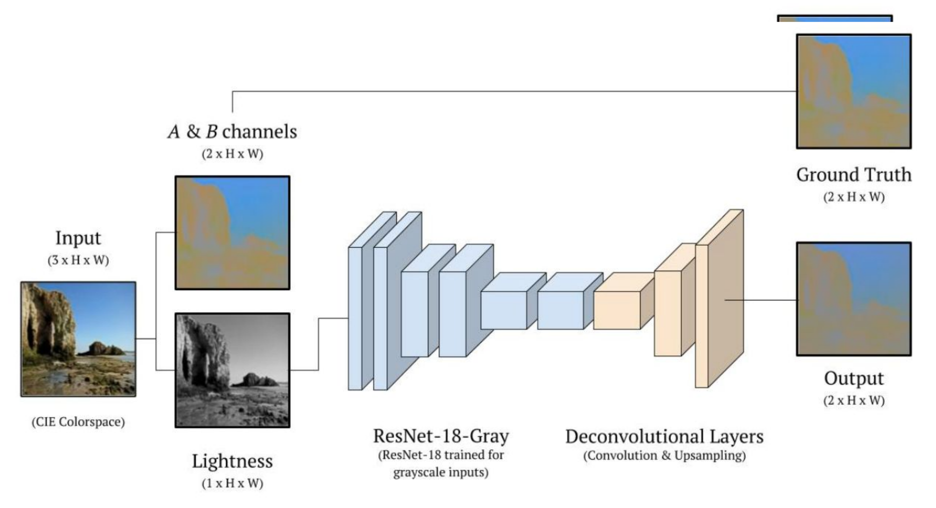
Generator



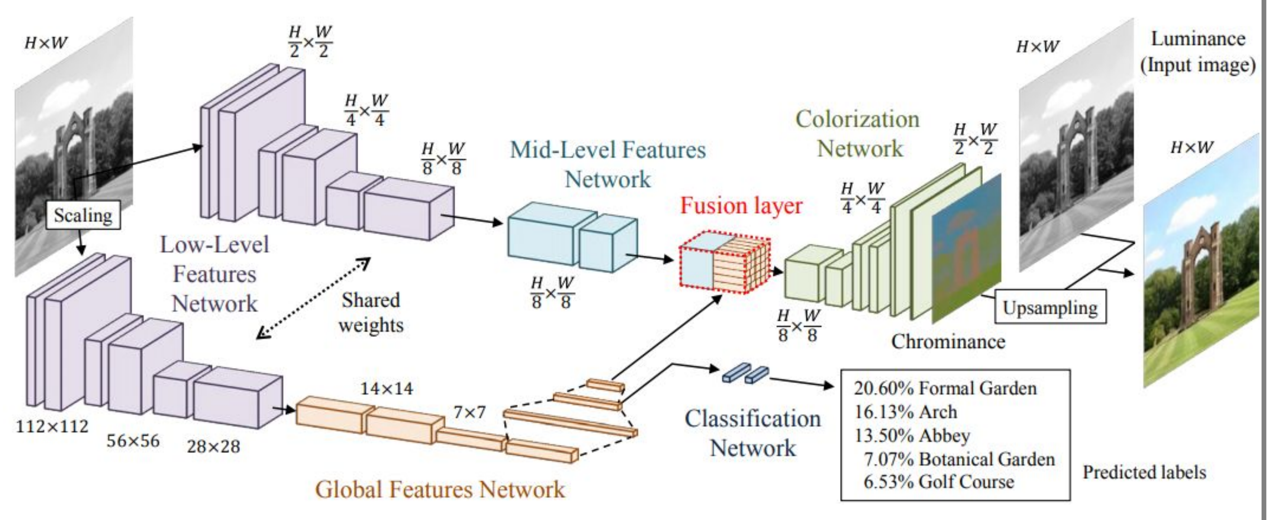
Discriminator



CNN (baseline)



CNN + Classifier



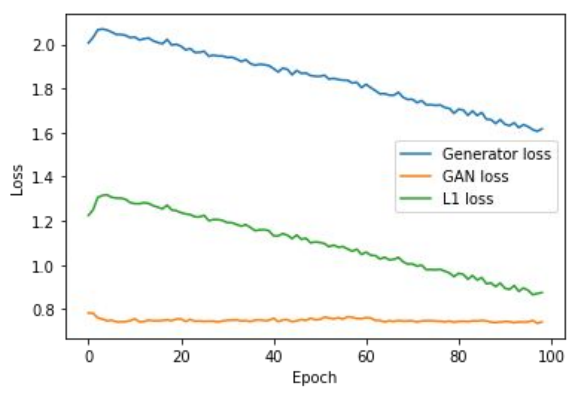
grayscale | CNN (baseline) | CNN + Classifier | Attention UGAN | Pretrained Attention UGAN | original

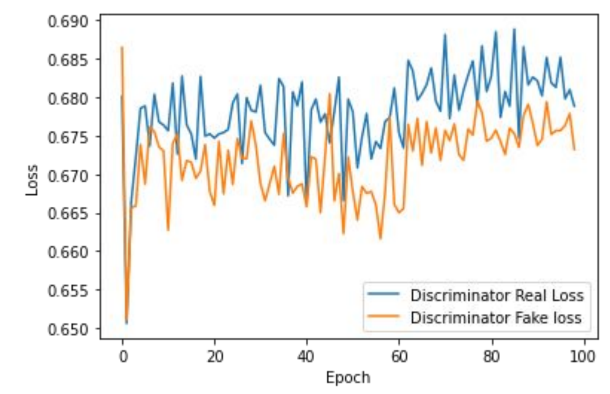
Losses

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

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

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





Results - Quantitative

	PSNR	SSIM	LPIPS
CNN	10.5663	0.5023	0.3370
CNN + Classifier	**20.3523**	**0.9064**	0.1651
Attention UGAN	18.4568	0.8733	0.1899
Baseline UGAN[3]	17.8909	0.8513	0.2001
Pretrained UGAN	19.6081	0.8961	0.1519
Pretrained Attention UGAN	19.9267	0.9002	**0.1483**

References

[1] Luke Melas-Kyriazi, Image Colorization with Convolutional Neural Networks, Blog.
[2] Oktay, Ozan, et al. "Attention u-net: Learning where to look for the pancreas." *arXiv preprint arXiv:1804.03999* (2018).
[3] https://github.com/moein-shariatnia/Deep-Learning/tree/main/Image%20Colorization%20Tutorial
[4] Jizuka, Satoshi, et al. "Let there be color!" - Implementation (https://github.com/kainoj/colnet)