CS584 ML

Project - Report

Image Colorization

Akshay Jain | A20502846

Sarvesh Shroff | A20488681

Department of Computer Science Illinois Institute of Technology

Dec 02, 2022

Abstract

We reviewed some of the most recent approaches to colorize gray-scale images using deep learning methods. Inspired by these, we selected a paper (Baldassarre et al.) in which the model combines a deep Convolutional Neural Network trained from scratch with high-level features extracted from the VGG16 pre-trained model. Along with it, deep learning encoder-decoder models can process images of any size and aspect ratio. Other than presenting the training results, we assess the "public acceptance" of the generated images by means of a user study.

Problem Statement

Coloring gray-scale images can have a big impact in a wide variety of domains, for instance, re-master of historical images and improvement of surveillance feeds. The information content of a gray-scale image is rather limited, thus adding the color components can provide more insights about its semantics. In the context of deep learning, models such as Inception , ResNet or VGG are usually trained using colored image datasets. When applying these networks on grayscale images, a prior colorization step can help improve the results. However, designing and implementing an effective and reliable system that automates this process still remains nowadays as a challenging task. The difficulty increases even more if we aim at fooling the human eye. In this regard, we propose a model that is able to colorize images to a certain extent, combining a AutoEncoder architecture with VGG 16 the model. While the deep CNN AutoEncoder architecture is trained from scratch, VGG 16 is used as a high-level feature. Due to time constraints, the size of the training dataset is considerably small, which leads to our model being restricted to a limited variety of images. Nevertheless, our results investigate some approaches carried out by other researchers and validates the possibility to automate the colorization process.

Challenges Involved with Image Colorization

The image colorization problem, like many other machine learning problems in computer vision, is generally framed as an “inverse problem” of recovering high dimensional data (a color image) from its low-dimensional representation (a grayscale image). It’s also a pixel-to-pixel problem: we want to predict a label (color) for each pixel.

These two qualities make it quite similar to two highly studied vision problems: semantic segmentation and depth estimation. Semantic segmentation asks for an algorithm to segment an image: group the pixels based on the objects to which they belong, and then label each segment with a semantic category, indicating which kind of object they depict. Depth estimation asks for an algorithm to reason about the 3D nature of a scene and, for each pixel, identify how far away the object producing that pixel was from the camera.

These problems are all ill-posed. Unlike other ill-posed problems like linear regression which are over-constrained (they are kind of like systems of equations with more equations than unknowns), colorization, semantic segmentation and depth estimation are under-constrained (we have a relatively small number of equations, one per pixel, and too many unknowns, three per pixel). We can’t shortcut the problem by combining all of our pixels together, either, as multiple pixels in the same image with the same grayscale value might map to different colors. This means we need to turn to machine learning and use training images.

Proposed Solution

We propose this by implementing an end-to-end learning-based approach that uses an AutoEncoder to color a black-and-white image.

Input Image

We consider images of size H X W in the CIE L*a*b* color space[1]. Starting from the luminance component $\mathbf{X_L} \in \mathbf{R}^{HxWx1}$, the purpose of our model is to estimate the remaining components to generate a fully colored version $\hat{X} \in \mathbf{R}^{HxWx3}$. In short, we assume that there is a mapping F such that

$$: \mathbf{X_L} \rightarrow (\hat{\mathbf{X}}_a , \hat{\mathbf{X}}_b)$$

where $\hat{X}_a$, $\hat{X}_b$ are the a*, b* components of the reconstructed image, which combined with the input give the estimated colored image $\hat{\mathbf{X}} = (\mathbf{X_L}; \hat{\mathbf{X}}_a; \hat{\mathbf{X}}_b)$. In order to be independent of the input size, our architecture is fully based on CNNs, a model that has been extensively studied and employed in the literature. In brief, a convolutional layer is a set of small learnable filters that fit specific local patterns in the input image. Layers close to the input look for simple patterns such as contours, while the ones closer to the output extract more complex features [23].

We choose the CIE L*a*b* color space to represent the input images since it separates the color characteristics from the luminance that contains the main image features. Combining the luminance with the predicted color components ensures a high level of detail on the final reconstructed image.

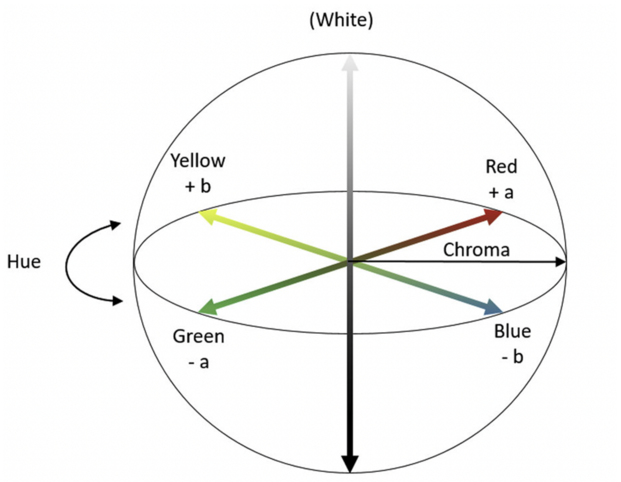
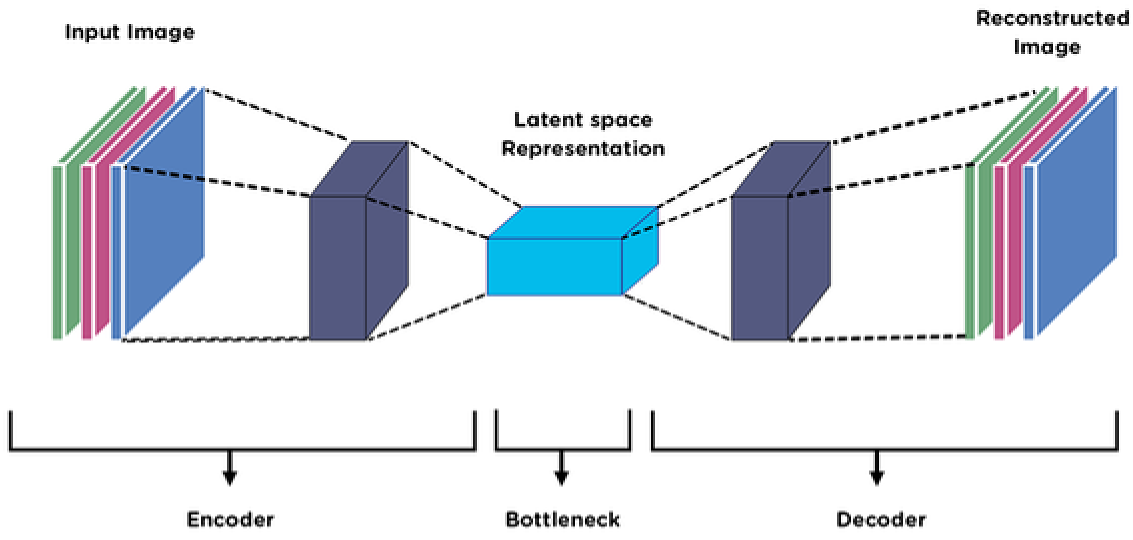


Figure: “CIELAB, or CIE L* a* b, color system diagram” (represents quantitative relationship of colors on three axes: L value indicates lightness, and a* and b* are chromaticity coordinates)[3]

We start with reading the image and conv



erting its size to shape of 150x150x3 where all the 3 channels are the black and white image replicated. As the VGG16 model needs the input shape to be 150x150x3 we do these changes.

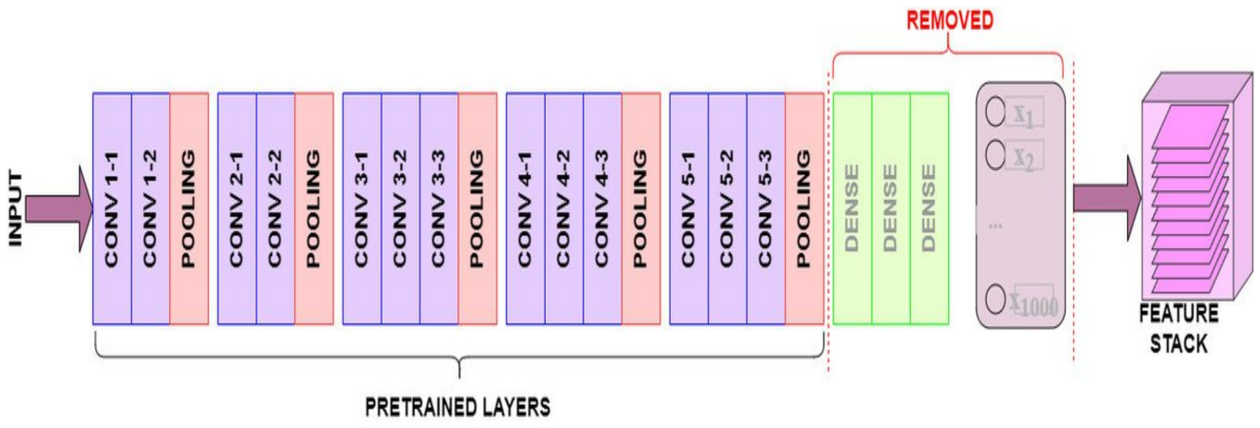
AutoEncoder

Autoencoder are special type of deep learning architecture that consist of two networks encoder and decoder. The encoder, through a series of CNN and downsampling, learns a reduced dimensional representation of the input data while decoder through the use of CNN and upsampling, attempts to regenerate the data from the these representations. A well-trained decoder is able to regenerated data that is identical or as close as possible to the original input data. Autoencoder are generally used for anamoly detection, denoising image, colorizing the images. Here, i am going to colorize the landscape images using autoencoder.

AutoEncoder Architecture

Encoder

For the encoder part we have used pre-trained VGG16 using transfer learning. VGG-16 is a convolutional neural network that is 16 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.



VGG16 Architecture[5]

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

In our approach we have removed the top part of VGG16 and used the pre-trained weights of the previous layers and made them untranable so that the weights wont change while training the model. By doing so we obtain an output shape of 4x4x512 after passing the input image through the model.

Decoder

Decoder part is responsible to recreate the final image from the high dimensional representation of the input image. For doing so we start wirth passing the output of the VGG16 through a series of Conv2D layers where the stride is set to 1 followed by upsampling in the order of the following table:

Layer	Filter	Output Shape
Upsampling	Factor of 7	28 x 28 x 512
Conv2D	128 x (3 x 3)	28 x 28 x 128
Upsampling	Factor of 2	56 x 56 x 128
Conv2D	64 x (3 x 3)	56 x 56 x 65
Conv2D	64 x (3 x 3)	56 x 56 x 65
Upsampling	Factor of 2	112 x 112 x 65
Conv2D	32 x (3 x 3)	112 x 112 x 32
Conv2D	2 x (3 x 3)	112 x 112 x 2
Upsampling	Factor of 2	224 x 224 x 2

After going through all these layers we finally obtain an image of shape 224 x 224 x 2 where the 2 channels are the a* and b* channels of the color space and after combining this with the input black and white image we obtain a colored image in La*b* space.

Results and Discussions

Performance Metrics

Image colorization performance can be evaluated on several factors, among them, the most frequently used is SSIM.

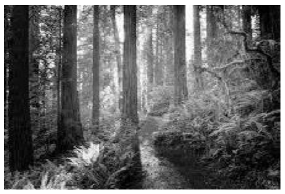
Structural Similarity Index Measure

- Structural Similarity Index Measure (SSIM), measures how similar two images are.
- Its range is from 0-1
- Two identical images will have a SSIM of 1

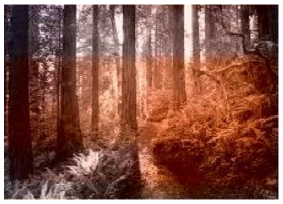
Forest and Sunset Image SSIM Performance Measurement on Different Epochs

SSIM	Epoch 10	Epoch 25	Epoch 50	Epoch 75
Forest	0.77	0.84	0.85	0.85
Sunset	0.85	0.86	0.87	0.87

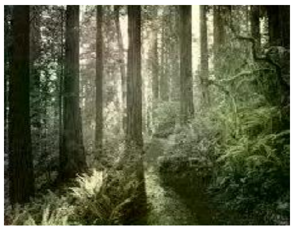
Forest input Image (in grayscale)



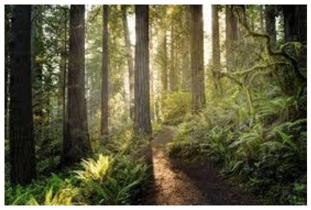
Epoch 10



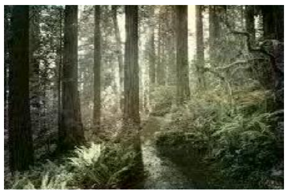
Epoch 25



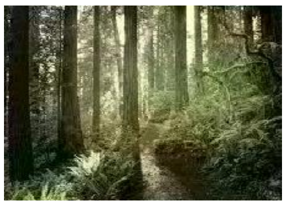
Ground Truth



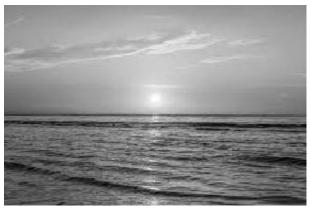
Epoch 50



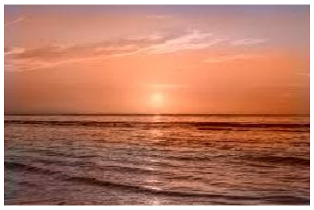
Epoch 75



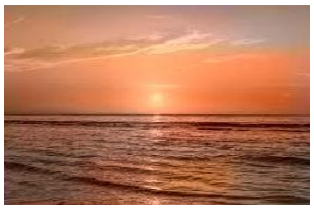
Sunset input Image (in grayscale)



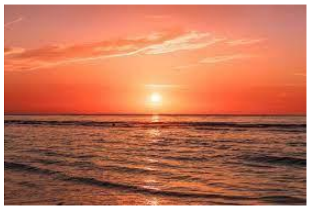
Epoch 10



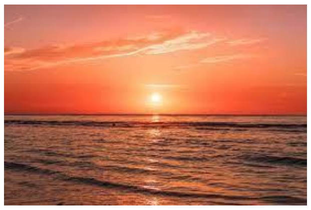
Epoch 25



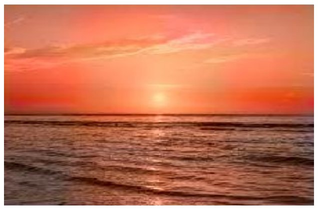
Ground Truth



Epoch 50



Epoch 75



Conclusions and Future Work

This project validates that an end-to-end deep learning architecture could be suitable for some image colorization tasks. In particular, our approach is able to successfully color high-level image components such as the sky, the sea or forests. Nevertheless, the performance in coloring small details is still to be improved. As we only used a reduced subset of ImageNet, only a small portion of the spectrum of possible subjects is represented, therefore, the performance on unseen images highly depends on their specific contents. To overcome this issue, our network should be trained over a larger training dataset. In this regard, a probabilistic approach in the spirit of seems more adequate. We believe that a better mapping between luminance and a*b* components could be achieved by an approach similar to variational autoencoders, which could also allow for image generation by sampling from a probability distribution. Finally, it could be interesting to apply colorization techniques to video sequences, which could potentially re-master old documentaries. This, of course, would require adapting the network architecture to accommodate temporal coherence between subsequent frames. Overall, we believe that while image colorization might require some degree of human intervention it still has a huge potential in the future and could eventually reduce hours of supervised work.

Learnings from the project

Specifically, we learned how to colorize grayscale images using AutoEncoder and Transfer Learning through this project. gained expertise in a variety of architectural designs, including but not limited to VGG16, Inception, and others. faced difficulties with transfer learning, but with time, research, and strong hands-on practice in these strategies, we were able to overcome them.

Work Assigned

Task	Akshay Jain	Sarvesh Shroff
Data Collection/ Creation	60	40
Data Preprocessing	40	60
Data Modeling	40	60
Performance Measures	50	50
Debugging/testing	60	40
Code Optimization	40	60
PPT / Report	50	50

GitHub Repository: link (https://github.com/akshayjain777/CS584-Final-Project)

References

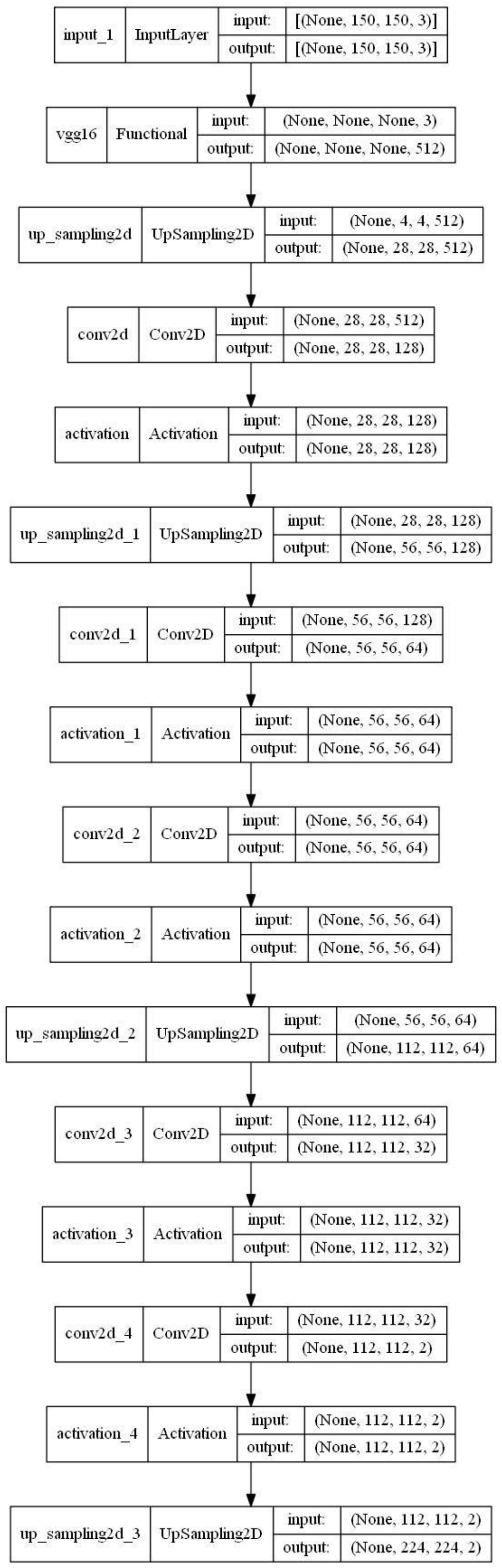
1. Baldassarre, F., Morín, D. G., & Rodés-Guirao, L. (2017). Deep koalarization: Image colorization using cnns and inception-resnet-v2. arXiv preprint arXiv:1712.03400. (link)

2. Robertson, A.R.: The cie 1976 color-difference formulae. Color Research & Application 2(1) (1977) 7-11

3. VGG16 https://doi.org/10.48550/arXiv.1409.1556

4. E. Agustsson and R. Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, July 2017.

Appendix


input_1 | InputLayer | input: [(None, 150, 150, 3)] | output: [(None, 150, 150, 3)]
vgg16 | Functional | input: (None, None, None, 3) | output: (None, None, None, 512)
up_sampling2d | UpSampling2D | input: (None, 4, 4, 512) | output: (None, 28, 28, 512)
conv2d | Conv2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 128)
activation | Activation | input: (None, 28, 28, 128) | output: (None, 28, 28, 128)
up_sampling2d_1 | UpSampling2D | input: (None, 28, 28, 128) | output: (None, 56, 56, 128)
conv2d_1 | Conv2D | input: (None, 56, 56, 128) | output: (None, 56, 56, 64)
activation_1 | Activation | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
conv2d_2 | Conv2D | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
activation_2 | Activation | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
up_sampling2d_2 | UpSampling2D | input: (None, 56, 56, 64) | output: (None, 112, 112, 64)
conv2d_3 | Conv2D | input: (None, 112, 112, 64) | output: (None, 112, 112, 32)
activation_3 | Activation | input: (None, 112, 112, 32) | output: (None, 112, 112, 32)
conv2d_4 | Conv2D | input: (None, 112, 112, 32) | output: (None, 112, 112, 2)
activation_4 | Activation | input: (None, 112, 112, 2) | output: (None, 112, 112, 2)
up_sampling2d_3 | UpSampling2D | input: (None, 112, 112, 2) | output: (None, 224, 224, 2)