FALL 2023 COMP411

Computer Vision with Deep Learning

Term Project

Enhancement of Image Colorization Models

Instructor: Yücel Yemez

Group Members: Birkan Çelik

Burhan Karahasan

Demet Tümkaya

Table of Contents

Introduction **3**

Background Information **4**

Exploring Existing Models 4

Implementation & Results **5**

Transfer Learning - Building a Decoder 5

UNet Implementation 6

Modifications on the Architecture 8

Hope: Hybrid Model of U-Net and VAE 11

Not All Deeper Models Are the Best 12

Validation **14**

Conclusion **14**

References **15**

Introduction

Models work with known data for the task of image colorization, considering the transformation from gray-scale images to colorful ones. This can be classified as recovering high-dimensional outputs in terms of color channels from low-dimensional representation. In this task, the particular challenge is reducing the information loss and matching the patterns by learning certain distinctions and features in the training set. Therefore, the outcomes depend heavily on model capabilities connected with the complexity and dataset diversity. While colorization was first introduced for realistic purposes, people nowadays use the model's creativity as an artistic perspective.

Historically, these tasks were done without machine learning. Instead, more statistical approaches were introduced, such as histogram matching and pixel correlation. As we see the advancements in learning algorithms and the increase in computational power, these methods are replaced by ML-based approaches like supervised learning and transfer learning, particularly CNNs, GANs, and U-Nets, which are the main actors in image and video colorization.

Our aim in this project is to discover the potential of different architectures and combinations to obtain enhanced results by applying several approaches learned in the scope of this lecture. Consequently, starting with a U-Net model, we have tried leveraging deeper architectures, experimenting with the potential of color spaces, the effect of skip connections, variational autoencoders, and loss functions - besides investigating the artifacts that we see in the generated images during the project timeline.

Background Information

Exploring Existing Models

To understand existing solutions to the problems, we reproduced the results of the model that has the most starts on GitHub, DeOldify, on the issue of image colorization. Then, we see that the model colorizes the images in a way that outputs a lack of vivid colors. We also noticed that this is because of the model's training dataset. The DeOldify model was trained on retro images with a mostly brownish theme. This bias can be easily seen in the test images we generated using DeOldify.

Figure 1: DeOldify, reproducing model outputs.

We used a PyTorch implementation of the DeOldify model to generate the above images. After observing already implemented models, we slowly moved on to our own architecture. As the first step, we only wanted to learn how to decode from latent space. That is why we used transfer learning to create embeddings of the input images.

Implementation & Results

Transfer Learning - Building a Decoder

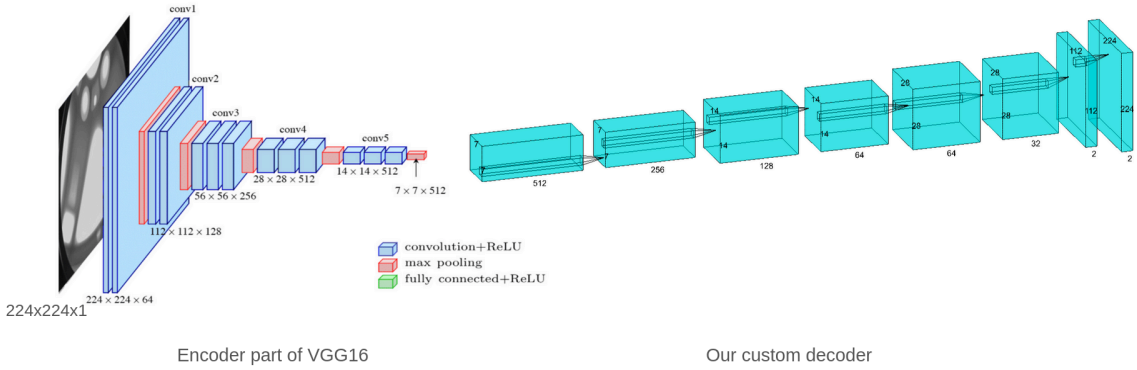

Figure 2: VGG-16 encoder, self-trained decoder model architecture.

Above, you can see our image colorization architecture, which uses VGG-16 as the encoder backbone. The input to the model is a grayscale image that has a size of 224x224x1. Then, the VGG-16 encoder takes the grayscale image and maps it to a 7x7x512 latent space vector. Later, we get this embedding and train our decoder. Before passing on the details of our decoder, we should mention the color space conversion we used. We converted the input image into LAB color space. L stands for luminance, representing the image's gray content, whereas AB channels are the colored content. We first convert colored input images to LAB space from RGB space. Then, we get the L content to feed to our model. At the end of the network, we wanted our model to predict the corresponding AB channel values. Thus, the output matrix has the shape of 224x224x2. After taking the latent vectors of our image, we used a stack of 2D convolution and upsampling layers. We chose the hyperparameters of our model after the hyperparameter search.

In order to test if our models learn properly, we let our model overfit on a small dataset. This way, we became sure that our model learned image colorization. Here are some examples of colorization of VGG-16 back-boned encoder-decoder architecture. We have also tried different encoders like ResNet-18 and ResNet-34 architectures. Yet, we got the best performance on the VGG backboned model, so we reported its outputs below. When we tried the VGG back-boned model on a larger dataset, we got a memory error due to our computational constraints.

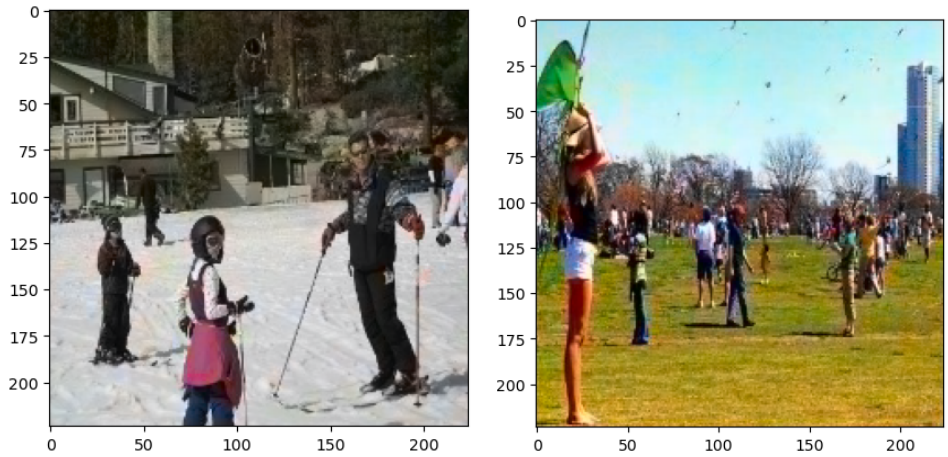

Figure 3: VGG-backboned, small dataset trained model outputs.

UNet Implementation

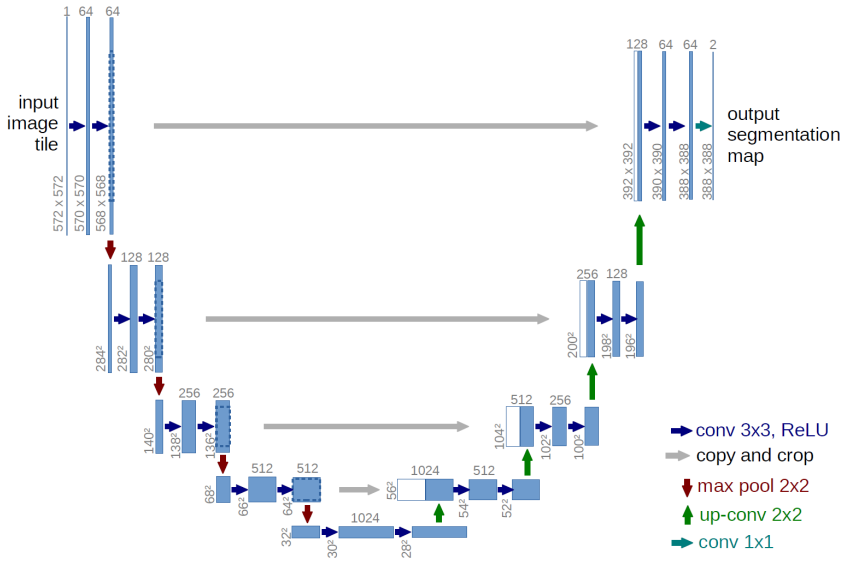

Figure 4: Original UNet Architecture by Ronneberger et. al. [4]

Inspired by the UNet paper, we have developed a similar neural network with a bottleneck shape to maximize the helpful information flow. In our prototypes, the input data is 150-by-150-pixel single-channel (grayscale) images, and the output is three-channel (RGB) images of the exact resolution as in Figure 5.

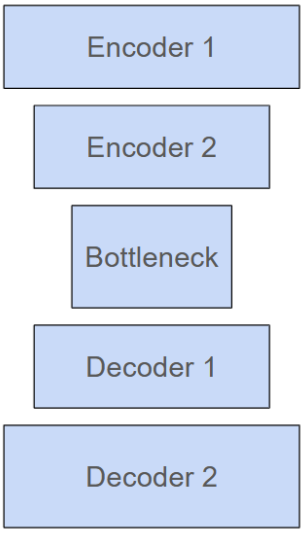

Figure 5: UNet without Skip Connections

Our initial design includes downsampling in the encoder-1 from 1-channel 150 x 150 image to 64-channel 146 x 146-pixel image via convolution layers. Encoder-2 reduces the dimensions from a 64-channel 146 x 146-pixel image to a 128-channel 69x69 pixel image via convolution and pooling layers. In the bottleneck block, a series of convolution, pooling, and transposed convolution has been applied. It yields an output of 64-channel 48 x 48-pixel image. Decoder-1 includes transposed convolution, yielding an output of a 32-channel 88 x 88-pixel image. Decoder-2 includes transposed convolution yielding an output of 32-channel 168 x 168-pixel image. At the end of Decoder-2, a series of convolution operations has been applied to reduce the image size to 3-channel 150 x 150 pixels. This way, the output was comparable to the target images, which have the same dimensionality (3x150x150), and the artifacts caused by the transposed convolution were minimized. All the convolution operations are backed with batch normalization and ReLU activation layers except the final convolution layer. Layer normalization and instance normalization caused a drastic increase in the training time. Thus, we avoided them. A max-pooling layer with stride 2 and kernel size 2 has been adopted as the downsampling method. In the comparison of ReLU and Leaky ReLU after the transposed convolution, we could not find a clear preference in terms of computational efficiency and mean squared error loss.

Modifications on the Architecture

We have slightly played with the architecture (kernel size, stride size, discarding max-pooling, increasing the number of layers, etc), and the following results are a combination of them. While training, we used a combination of landscape datasets from [5,6], 12 thousand images in total, and we included an additional 25 thousand images from [7] for a larger dataset.

On average, this model yielded a 1% mean-squared error in training and test datasets after 10 epochs. Our choice of reconstruction loss is a mean-squared error. The L1 norm was also tried; however, it did not produce any difference in the results.

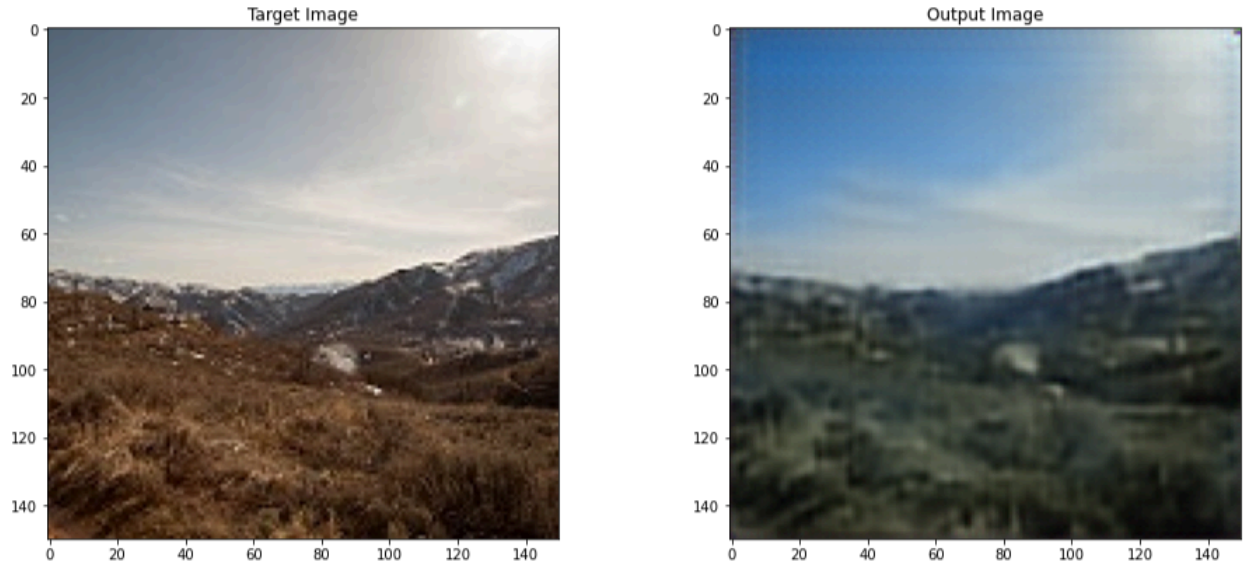

Figure 6: Original image (left) and colorized image (right). Color bias example.

In Figure 6, we see the original RGB image on the left. The colorized version on the right is the neural network result produced from its grayscale copy. Due to the massive sea and ocean content in the landscape datasets, we see significant bias towards blueish colors. This showed the need to adopt a more diverse dataset.

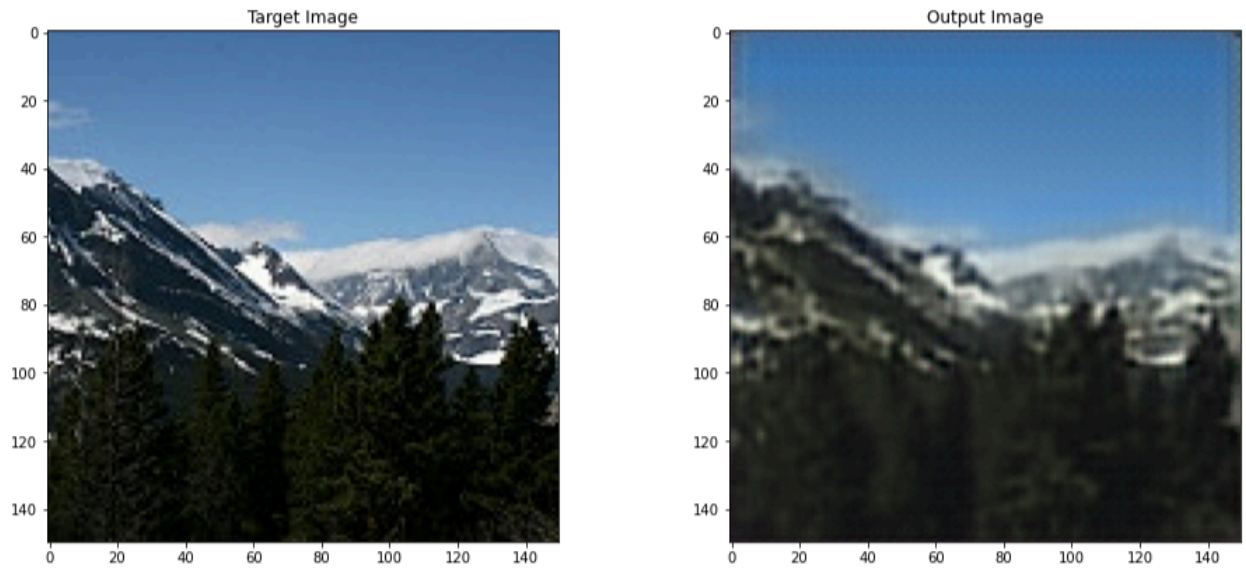

Figure 7: Original image (left) and colorized image (right). Corner and edge artifacts.

Initially, we applied zero-padding convolution in downsampling. We later noticed that it caused information loss in the corners and edges, yielding the artifacts.

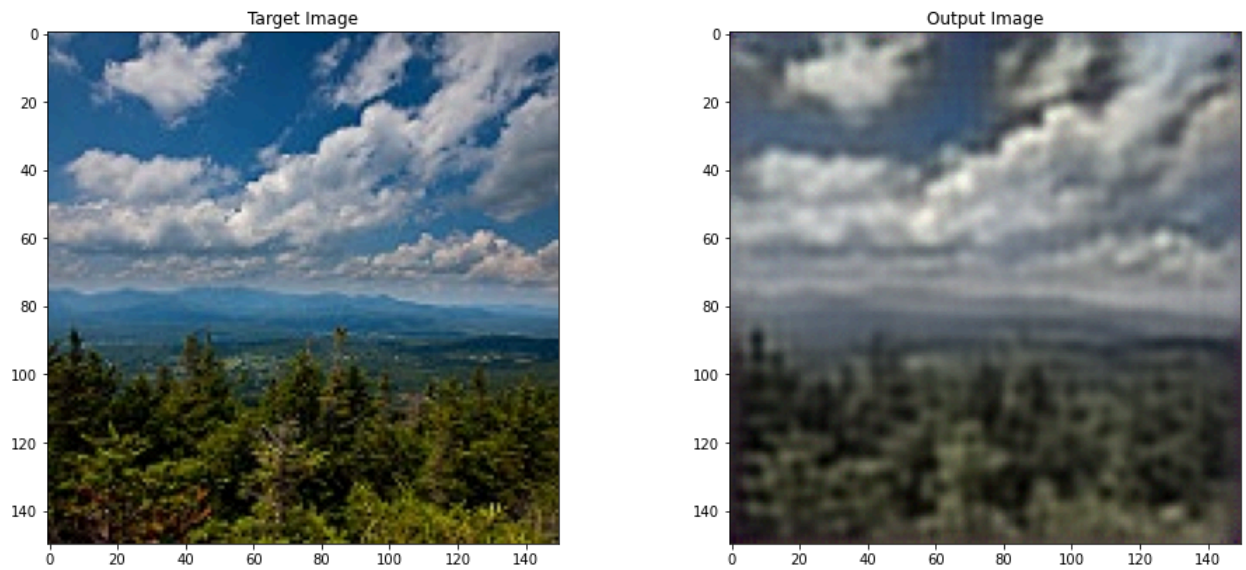

Figure 8: Original image (left) and colorized image (right). Reduced resolution.

In Figure 8, we have seen that our architecture yields visually satisfying results regarding color saturation. However, the resolution was disturbing. Later, in high training epochs, we see the resolution gets better.

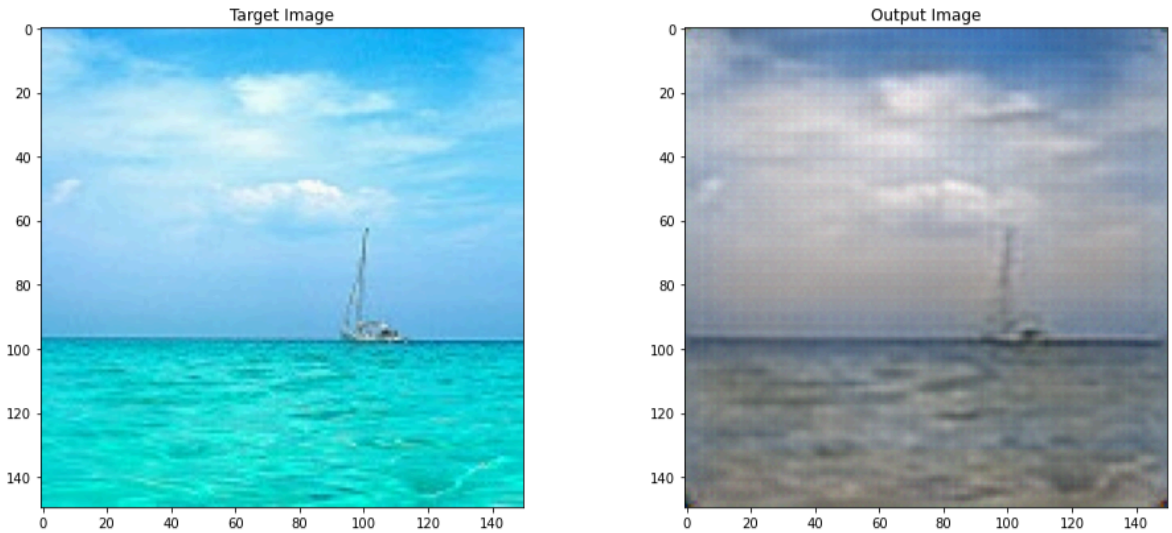

Figure 9: Original image (left) and colorized image (right). Checkerboard artifacts.

In addition to the previous deformations, we also observed checkerboard artifacts in many images. In [8], we see that this is a chronic problem with upsampling using transposed convolution.

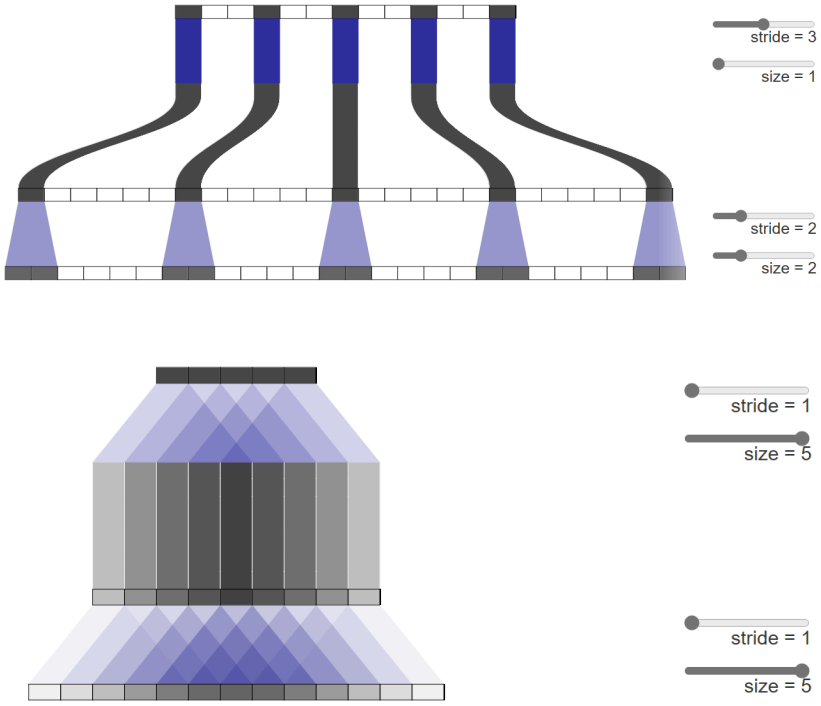

Figure 10: Information transmission to further layers using transposed convolution.

In Figure 10, we can see that kernel size with respect to stride number should be high for more homogeneous data transmission. This implies that using larger kernels and lower strides decreases the checkerboard artifacts. It means a gradual upsampling with more parameters, which means more training time per epoch. We have found a middle ground that serves both our concerns by fine-tuning the parameters. Later, we also saw that higher epoch numbers reduce the artifacts and improve the overall resolution as well.

Hope: Hybrid Model of U-Net and VAE

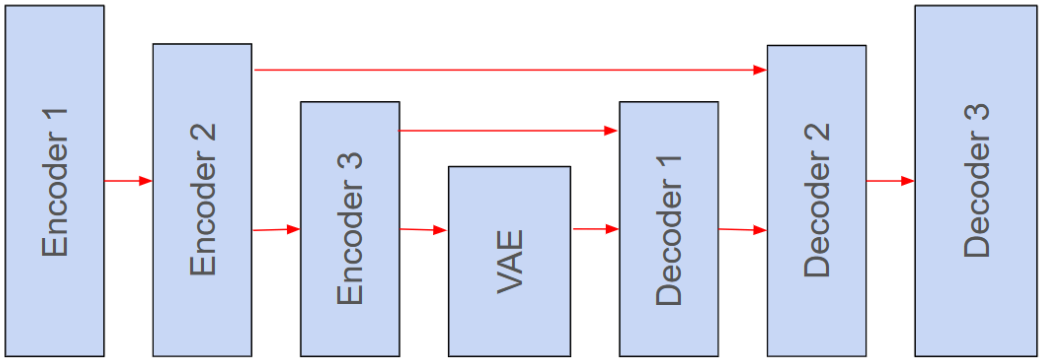

Figure 11: Hybrid UNet and Variational Autoencoder architecture with skip connections.

Lastly, we adopted a hybrid architecture, a combination of UNet with convolutions and transposed convolutions for upsampling and Variational Autoencoder (VAE). The VAE replaced the bottleneck block in the middle. We added skip connections and designed a deeper network than the previous one.

The hybrid network processes a 1x150x150 input image through a sequence of encoding and decoding stages, along with a VAE component. The first encoder reduces the input to a 64-channel 74x74 tensor. Subsequently, the second encoder further decreases dimensions to 64 channels with dimensions 36x36, followed by the third encoder, which results in a 64-channel 17x17 tensor. The VAE operates on this tensor, maintaining the spatial dimensions at 64x17x17. Moving to decode, the first decoder, using a transposed convolutional layer, expands the tensor to 64x38x38. The second decoder increases the dimensions to 64 channels with dimensions 80x80, and the third decoder further expands the tensor to 64 channels with dimensions 164x164.

The final stage employs convolutional operations to yield a 3-channel 150x150 output, completing the UNet-VAE transformation of the input image.

Encoding to and decoding from the latent space introduces some randomness to the network. Fitting a Gaussian distribution and resampling from it slightly jiggles the data about the original values. This kind of approach, hypothetically, would break the dryness in the images. We included the Kullback–Leibler divergence loss caused by the VAE and MSE loss yielded by the reconstruction as a cost function. In test time, we neglected KL divergence and only checked the MSE.

Not All Deeper Models Are the Best

Following our design of U-Net that we have iteratively developed, the best outcomes came with the 3-layered encoder and decoder, connected with a variational autoencoder, in the presence of skip connections. This model was trained for 100 epochs on a computer with GeForce GTX 1650 and yielded the following results. The encoder kernels were 4x4 with stride 2, while the decoder had 6x6 kernels with stride 1.

Figure 12: 3-layered encoder-decoder U-Net output images

As we see a valid understanding of colors and patterns with these results, on top of this model, we wanted to get better colorization for all color channels. Some of the images were constructed better than others because they were mostly landscapes; however, the model could only fade the silhouettes accordingly when it comes to warmer and red color-containing content. To improve the model and results one step further, we have trained a deeper U-Net with 4 layers of encoder and decoder, again including a variational autoencoder and skip connections. In addition to the change in the number of layers, we have also extensively increased the channel size, up to 512, to add complexity and extract a variety of features. Due to resource constraints, this model was trained with Colab on V100 GPU for 70 epochs and ended with a training loss of 0.001051. However, as can be seen in Figure 13, the loss continues to increase, which shows us that the training should continue, and the model may not learn as well as the previous model, resulting in blurry images, but colors are distinct and present at some sense, which was less complex and trained on the same data for a longer time. After seeing the visual results and considering the opportunities we have, we have concluded that we cannot get a better colorization output with a deeper model if the time and resources permit, the results may be improved with more computational power for sure.

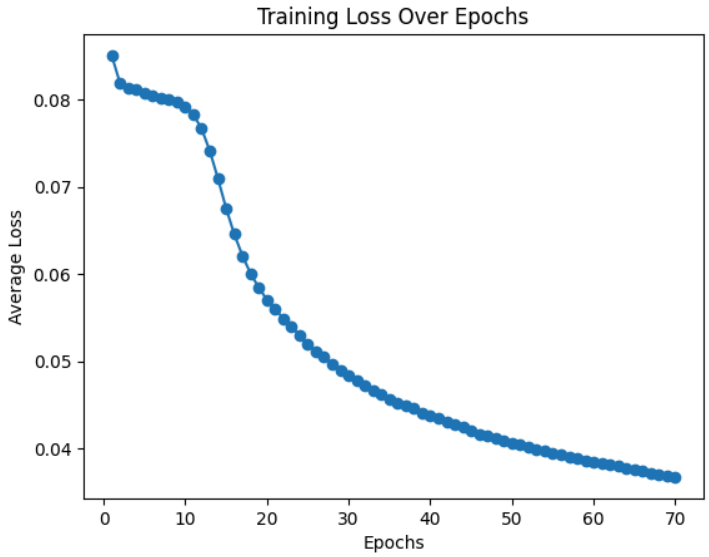

Figure 13: Training loss over epochs for the deeper U-Net

Validation

To validate our results, we mostly used the visual inspection of outputs and MSE loss of our models. We are aware that a systematic evaluation should be provided in terms of loss tracking; however, as we tried many models in limited time and resources, our implementations mostly changed to achieve improvement in the visual structure of images. Thus, investigating artifacts is a valuable contribution to our work besides constructing different architectures. Overall, visual outcomes and necessary loss curves that helped us to interpret the solutions better can be seen throughout the report.

Conclusion

The project started with the aim of investigating image colorization methods in the literature rooted from DeOldify and introducing enhancements with different architectures and strategies to increase the understanding of the details for the task. For this purpose, we first did a brainstorming session to start with certain ideas and valid architectures. Then, we tried iterative modifications after evaluating the result and loss of the models. These modifications and trials include primarily a transfer learning example supported with a custom decoder and enriched U-Net architecture. During this process, besides the model capabilities, results from visual inspection of the outputs were another consideration for enhancing the image colorization. On the way, we have encountered color bias, edge artifacts, reduced resolution, and checkerboard artifacts, and we have reported different methods to solve these problems, such as the influence of kernel size, number of epochs, padding, dataset content, etc. Finally, after this careful investigation and iterative construction of models, the best-performing model is a U-Net and VAE combined architecture trained for 100 epochs on a dataset including 150x150 images. Further improvements can be done with greater computational power and targeted modifications on the parameters changed within the scope of this project.

REFERENCES

[1] Jason Antic. jantic/deoldify: A deep learning based project for colorizing and restoring old images (and video!). https://github.com/jantic/DeOldify, 2019.

[2] A. Salmona, L. Bouza, and J. Delon, “Deoldify: A review and implementation of an automatic colorization method,” Image Processing On Line, vol. 12, pp. 347–368, 2022. doi:10.5201/ipol.2022.403

[3] R. Zhang, P. Isola, and A. A. Efros, “Colorful image colorization,” Computer Vision – ECCV 2016, pp. 649–666, 2016. doi:10.1007/978-3-319-46487-9_40

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015, pp. 234-241. Available: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

[5] TheBlackMamba31. (2022, August). Landscape Image Colorization. Retrieved January 24, 2024 from https://www.kaggle.com/datasets/theblackmamba31/landscape-image-colorization.

[6] Arnaud58. (2022, August). Landscape Pictures. Retrieved January 24, 2024 from https://www.kaggle.com/datasets/arnaud58/landscape-pictures.

[7] Mirflickr. (n.d.). MIRFLICKR - 25k Dataset. Retrieved January 24, 2024 from https://press.liacs.nl/mirflickr/mirdownload.html.

[8] Dumoulin, V., & Olah, C. (2016, October). Deconvolution and Checkerboard Artifacts. Distill. Retrieved January 24, 2024 from https://distill.pub/2016/deconv-checkerboard/.