VGG19 U-Net Image Colorizer With Perceptual Loss

Hunter Adrian, John Farrell, Tyler Gurth, Jania Vandevoorde
Brown University

Abstract

We implemented a convolutional neural network (CNN) to colorize grayscale images using a U-Net architecture with the VGG-19 model. U-Net is a popular deep learning architecture known for its effectiveness in image segmentation tasks. VGG-19 is a large model with almost 150 million parameters that is pre-trained. It is traditionally used for feature detection and was adapted for colorizing in our project. Our model is trained using the MIT Places365 dataset, which contains 365,000 images of scenes (which we split into 328,500 train and 36,500 test images, a 90/10 split). Moreover, the model makes use of a custom Perceptual Loss function for a higher level chromatic evaluation of the CNN. Our results show that the model produces vibrant and realistically colored images. This project reinforces the potential of deep learning in creative image processing.

1. Introduction

For quite a span of time, historians, archivists, and the occasional Reddit user, have tried to use manual means of coloring historical (or otherwise) black and white photos. While possible for someone with enough time on their hands, proper references, and skills in Photoshop, it is useful to automate this process, making use of large image models to color photos with pretrained weights.

Thus, our research group (The RGBaddies) sought to create a model architecture and algorithmic process to take grayscale photos, and color them to a high efficacy. Through proper research, we determined that large Convolutional Neural Networks (CNNs) would be the most proficient in this difficult task. Although often used for recognition and classification, a basic understanding of a CNN is that it takes an input and an output, and tries its best to create weights that make that conversion.

This is not novel - many publications exist in the greater scheme of coloring grayscale images to various accuracies (and, on various methodologies), however, we implement a strategy that combines the best thoughts from several difficult publications to make a quick and accurate model.

We utilize a UNET modeled CNN for our project, combining a base VGG-19 pretrained network with our own custom upsampling process. More unique though, is that we combine our VGG-19 model with a perceptual loss function, to increase our results and be more resistant to variation that doesn't necessarily affect perceptual color expectations. Thus, our strong model takes insights from accurate object detection in juxtaposition to human perception of grouped hues across pixel variation.

We train our model on the MIT Places365 dataset, which contains a myriad of environments, people, and other varied scenes. The large (and accessible) dataset provided us with a broad learning base, and reasonable scenes for color prediction.

In our project, we thoughtfully explore the uses of image colorization, and how it might impact different people, researchers, developers, and communities. We build an interactive tool for users to try our model themselves, and discover how they might interact with an automatic tool for their own use cases. Our results are confident and show promise; we believe that our implementation is not just a highly accurate model for predicting color, but is a scalable backbone for more complex models to come.

2. Related Work

The 2016 Zhang et al. [6] study was the first paper we consulted to develop our approach to colorization. The researchers used a series of convolution blocks totaling about 20 convolution layers and used spatial downsampling and upsampling between blocks to form their network. They used rebalanced classification loss to promote vibrance and diversity of colors in their recoloring. Zhang's paper compared its results to the 2016 Dahl colorization study [2] which used 4 layers from VGG16 in a small U-Net-like architecture. The latter study also used a custom L2-inspired loss function by averaging the mean squared error with two blur distances of 3 and 5 pixel Gaussian kernels. Nguyen et al. [4] built on VGG19 with L2 as the loss function. Based on this, decided to go with a U-Net approach with VGG19 as our pre-trained head (since VGG16 is outdated). We adapted the custom Dahl loss function to the Lab colorspace (whereas that study had originally used it in HSV).

An et al. [1] used a similar approach to Zhang et al. with a series of big convolution layers, ReLU, and batch normalization. They also used a classification loss function – multimodal cross-entropy loss – with 313 color bins.

Researchers Sahay and Choudhary [5] automated the colorization of *videos* using the pre-trained network from Larsson et al. [3], another interesting application of this problem. The hypercolumn-based architecture is inspired by VGG-style architectures, using a similar approach with 7 convolution blocks, ReLU, and batch normalization. They used multinomial cross-entropy loss. A key element of their network is the use of skip-layer connections within the network to get a pixel's color value by reading localized slices from a series of layers. We also implemented skip connections in our model to improve accuracy.

Our choice to implement a custom loss function – perceptual loss – was largely inspired by the array of different loss functions we read about during the research stage of the project. We began with L2 loss as that seemed straightforward to interpret in the early stages, but the implementation of a custom loss function significantly increased the quality of our results. We attribute this to the extensive research we completed before beginning to build our architecture. The Dahl paper [2] was pivotal during this process.

3. Method

In trying to color images, we understood that at a baseline level, the method should involve taking some gray image, and outputting a 2 or 3 channel color image. Thorough research suggested that the best way of doing this was to break an image down from RGB into the Lab color space, where L represents perceptual lightness and a and b denote the four unique colors of human vision: red, green, blue and yellow. This was a simple (and, admittedly popular) way to represent our inputs and hopeful outputs into distinct categories.

With this in mind, our goal was to build a model that accurately took the L channel of an image, and outputted its associated a and b channels, which we would then combine with the input L channel to form the *now colored* image.

Our process can be broken down into 3 key stages: pre-processing, defining a loss function, and constructing an appropriate architecture.

Pre-processing efforts, albeit cyclic, a reflection of (at the time, predicted) later efforts in constructing the CNN architecture. Pre-processing started with separating and parsing the train / test images. Then, all images went through a augmentations (for vertical and horizontal flipping), and a custom process function, which brought all images into a floating point 0-1 range and resized them to be $224x224$ (the size of image that VGG-19 was trained on with Imagenet).

Finally, images were converted into the LAB colorspace with the following function (which took in the tensor objects from the TensorFlow flowfromdirectory function):

```
def rgb_to_lab(self, dat):
    for im in dat:
        im = color.rgb2lab(im[0])
        # We return a light image (we
            just copy the same channel 3
             times for VGG), and AB
            channel image.
        yield (im[:, :, :, [0, 0, 0]],
            im[:, :, :, [1, 2]])
```

What may not seem intuitive is the definition of the the lightness image (the first item in the returned tuple, while the second item is the the ab space channels. The 0th channel of the converted image, the L space, is copied over three times solely because VGG-19 expects an input with 3 channels. Thus, we give it a pseudo 3 channel input, that is, the L channel copied thrice, to which it predicts a returned "image" of the same size, composed of an a and b channel.

Defining a loss function was simple enough, in fact, our naive implementations just used MSE (Mean Squared Error) to calculate the differences between predicted and ground truth ab channel pairs. That being said, research guided us to define a more complex perceptual loss function, to consider images that had variations, but on a high level, should produce a similar smoothed color scheme.

We started with the MSE:

$$\mathrm{MSE}((\mathrm{Y_i}, \hat{\mathrm{Y}}_\mathrm{i}) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{1}$$

We used this MSE as a component for finding what we call a Gaussian Filtered MSE (GD, or Gaussian Distance), (where f is the filter size, and G is the Gaussian filter function, which also takes a filter size.), (Eq. 2).

$$\mathrm{GD}((\mathrm{Y_i}, \hat{\mathrm{Y}}_\mathrm{i}, \mathrm{f}) = \sqrt{MSE(G(Y_i, f), G(\hat{Y}_i, f))} \tag{2}$$

We used our GMSE function at several filter sizes to define our loss function (where, here M is the MSE function), (Eq. 3).

$$L(Y_i, \hat{Y}_i) = \frac{\sqrt{\mathrm{M}(\mathrm{Y_i}, \hat{\mathrm{Y}}_\mathrm{i})} + \mathrm{GD}(\mathrm{Y_i}, \hat{\mathrm{Y}}_\mathrm{i}, 3) + \mathrm{GD}(\mathrm{Y_i}, \hat{\mathrm{Y}}_\mathrm{i}, 5)}{3} \tag{3}$$

Our loss function, (Eq. 3) acts as a more complex, perceptual function with higher level smoothing, leading to faster convergences due to its somewhat underfitting nature. Although it is less sensitive to small variability, it resulted in smoother (and, more optimized) outputs.

Implemented in Python, (Eq. 3) takes on a simple form (adapted from Ryan Dahl's LUV space perceptual loss function [2]):

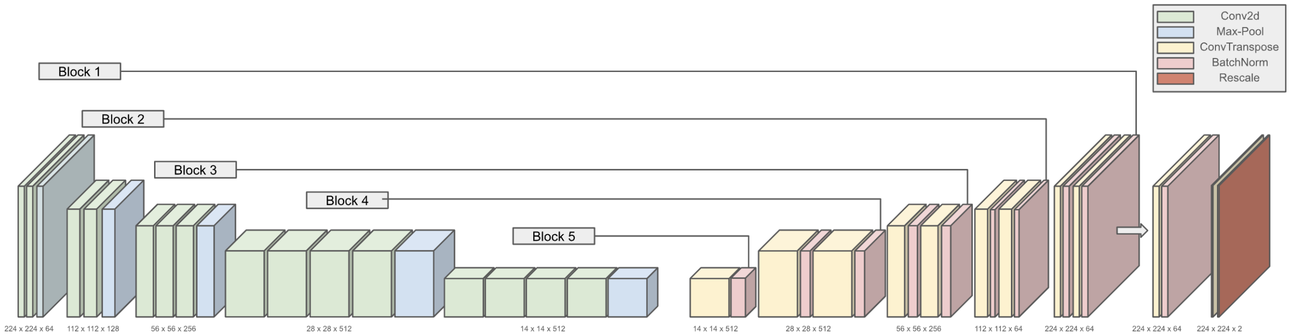

Figure 1. Vgg-19 U-Net Architecture implemented.

```
def percept_loss_func(self, truth,
    predicted):
     truth_blur_3 = blur(truth, (3,3))
     truth_blur_5 = blur(truth, (5,5))

     predicted_blur_3 = blur(predicted,
         (3,3))
     predicted_blur_5 = blur(predicted,
         (5,5))

     dist = mse(truth, predicted) ** 0.5
     dist_3 = mse(truth_blur_3,
         predicted_blur_3) ** 0.5
     dist_5 = mse(truth_blur_5,
         predicted_blur_5) ** 0.5

     return (dist + dist_3 + dist_5) / 3
```

With the perceptual loss function in mind, the architecture takes in a 3 (same) channel L image, and returns 2 channels, the ab channels for that given image. Specifically, our architecture, visualized in (figure. 1) is a U-NET based architecture, making use of the VGG-19 pretrained model. At a glance, the architecture takes the outputs of the VGG-19 model after each block, and upsamples them with Conv2dTranspose layers to maintain the desired output size across the entire model. In the naive solution, one could run the pretrained model (which, brings forth an output of 14x14x512), and then upsample with several Conv2dTranpose layers back up to the desired output shape. In fact, this was our first naive solution. However, because the input to all the upsampling layers is just a 14x14 block, the upsampling is heavily marked by this original output, and struggles to define detail and clarity in a lower dimensional space.

Using a UNET architecture allows us to upsample "along the way" preventing us from needing to grapple with a very small outputted image, and somehow blow it up to size in an accurate fashion. Our preliminary efforts were obviously marked in a 14x14 grid, and lacked sharpness in detail (obviously lost in this extreme down - upsampling process). To form the UNET architecture, we studied the VGG-19 architecture with great attention, and then "matched" its block outputs with the same size Conv2dTranspose layers to undo the shrinking convolutions. We also made uses of BatchNormalization layers to normalize outputs and to create a more stable model.

Our architecture finishes with sizing a final convolution to be 224x244x2 (to match the 2 ab channels). Then, it makes use of a Rescaling layer to take the output values (all 0-1), and map them to a range of (-128, 128), which is the range of the LAB colorspace.

The architecture over 13.39 million trainable parameters, and 33 million total parameters, and ran fairly quickly on Colab GPUs.

Finally, with weights in hand, we use a special visualization script to take the weights, predict ab channels, and match them back up with the L channel of the original image.

4. Results

After training our model on 50 epochs of 160 images each, we reached a training MSE of 147.9 and validation MSE of 135.2. We were quite pleased with these results, as they were comparable to the results of other papers. The progression of our mean square error (MSE) and perceptual loss can be seen in figures 1 and 4. Both MSE and loss improve quickly over the first 5 epochs, but seem to plateau by epoch 20. There is a clear correlation between MSE and our perceptual loss values over epochs. This correlation is expected because our perceptual loss is an average of euclidean distances between truth and predicted pixel values, similar to MSE between truth and predicted pixel values.

Our model seems to predict the ab color space of grayscale images best for grass and blue sky landscapes.

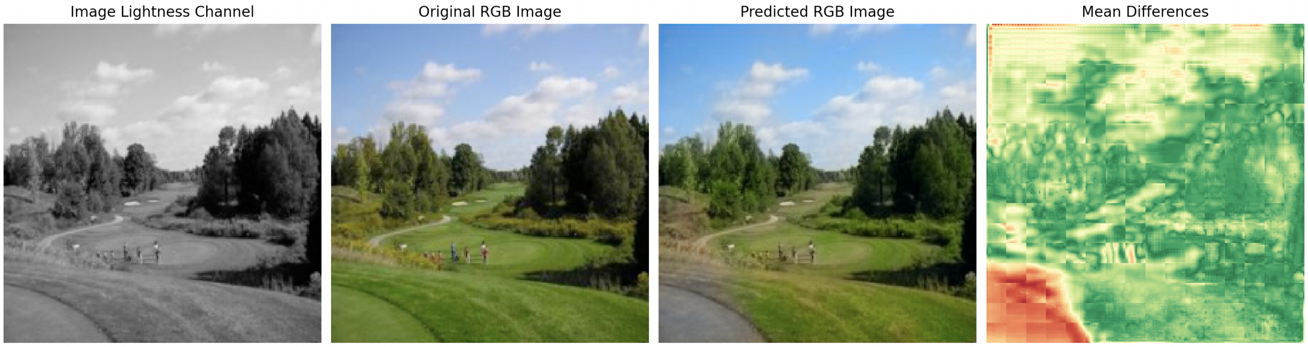

Figure 2. Result on Test Image.

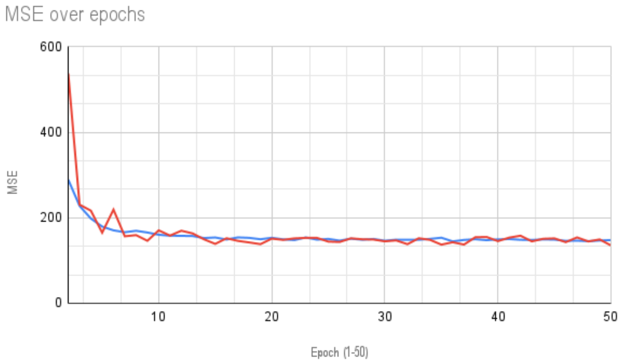

Figure 3. MSE over epochs.

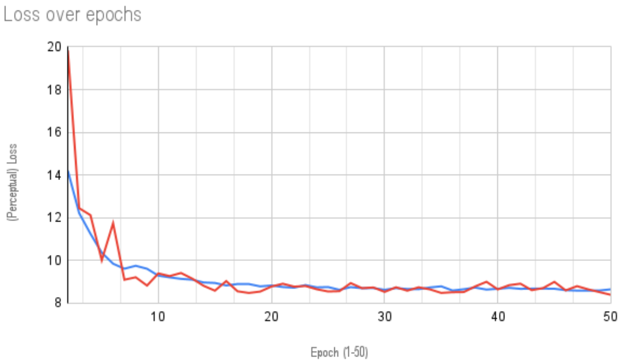

Figure 4. Loss over epochs.

Figure 2 is an example of this. Although the results of figure 2 are quire good, we can see the results differ for other photos. Figure 5 shows more results. In the third row of figure 5, we see where our model struggles to understand sunsets and sunrises, always choosing to color the sky blue. And in the last row of figure 5, the baby's blue pants are colored brown. This is a phenomenon where our model is biased towards bland colors that are common in the dataset (included dirt, grass, and gray clouds) [6]. Although the baby's pants were actually blue, the brown coloring is also natural to the human eye. It is important to point out the goal of our model is to predict colors that seem natural, not necessarily replicating a truth value. Therefore, the model's perceptual loss function prioritizes natural coloring [2]. Our perceptual loss function doesn't account for color re-balancing, leading to biased unsaturated coloring. To account for bland coloring, we increase the contrast of our results by scaling the predicted *ab* color space.

4.1. Technical Discussion

A future improvement to consider is further image augmentation. As our input must be a 224 × 224 pixel image, any image not of this shape must be resized. It would be expected that resizing an image will distort its perception and erode the colorization quality. To be robust against dimension shifting, including image augmentation that that distorts the original image to simulate this reshaping could improve the outputs of our model. At the moment, the only augmentation prepossessing is flipping the image on the x-axis.

Choosing a loss function is a crucial part of colorization models. There are papers covering many different loss functions, including a color classification [1]. The classification includes binning the ab color space into 313 bins [5]. The bins are a somewhat arbitrary creation, of the ab color space split into 10×10 grid squares. There are originally more than 313 bins, but only 313 are in-gamut [6]. This categorical binning allows such colorization models to utilize cross-entropy loss, helping with color rebalancing. Due to the high quantity of grey clouds, dirt, roads, and other bland colors, the results without when using an L2-like loss function are often unsaturated, biased towards these greys and browns [1]. This is clear in the last two rows of our truth to predicted comparisons, where vibrant colors are replaced with browns (figure 5). This bias is solved by reweighing bins in the cross entropy loss function by rarity [1]. To improve our model, quantizing the ab color space could improve our results, as the more intelligent cross entropy loss function can account

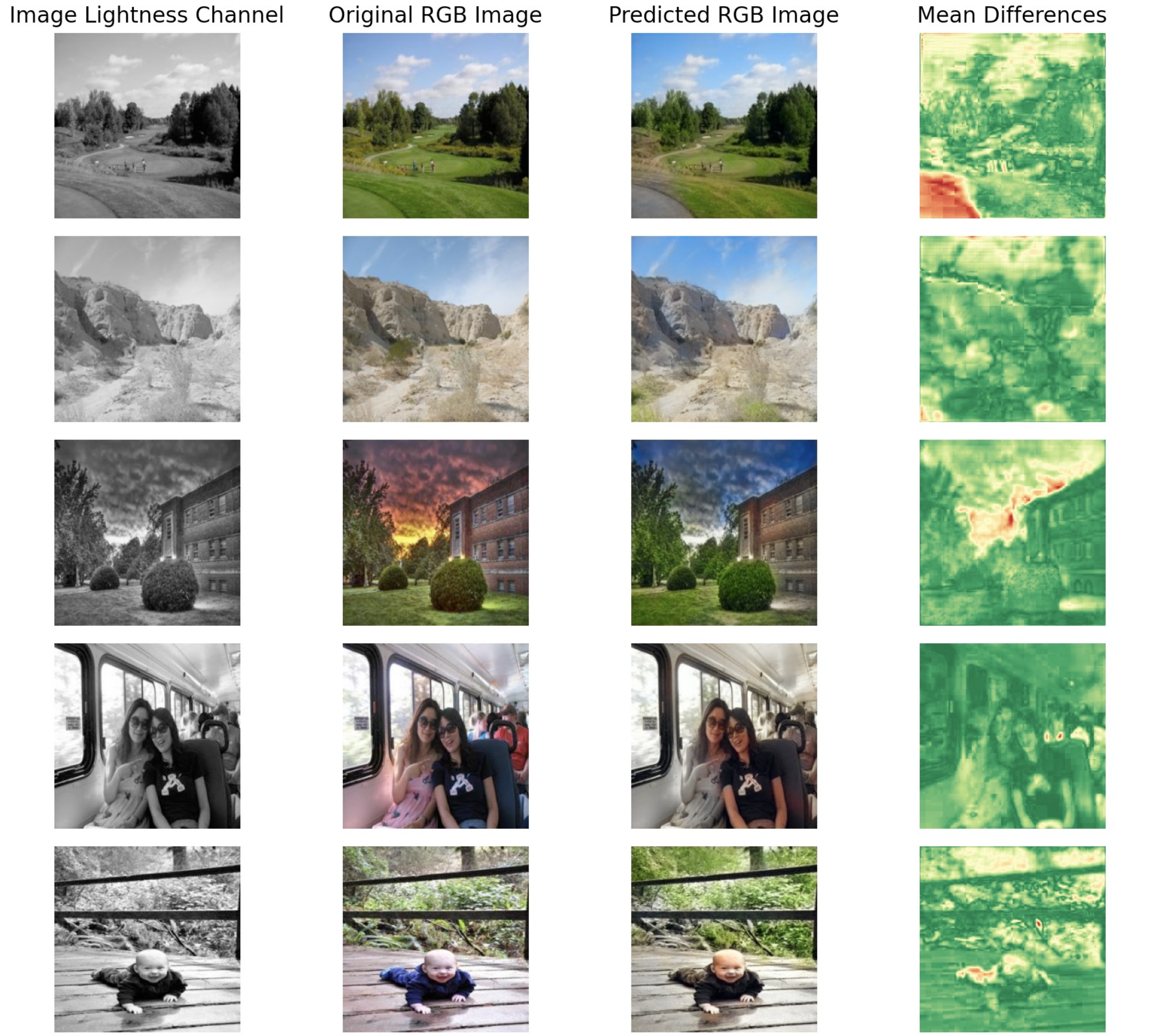

Figure 5. Comparison of different types of test images.

for color biases. Such an implementation requires clustering existing pixel values for every image to the existing 313 bins, which can be time and resource exhaustive. Our perceptual loss function sacrifices some accuracy and vibrancy for the sake of resources and natural-feeling coloring.

Many previous works use a subset of ImageNet as their dataset. This dataset provides a wide variety of objects, scenary, and life, categorized under thousands of labels. This model was trained on Places365, a set of 365,000 images ranging from photos of people, to landscapes and cars. This dataset worked well for training, and was a manageable size for cloud computing. Filtering grayscale images from the dataset would likely improve our existing results. Using a larger dataset would be tricky due to storage limits on our training platform, Google Colab. Nevertheless, expanding to a larger dataset, such as ImageNet, would give us a larger scope of scenery and life, following many of the mainstream colorization papers.

5. Social Impact

Besides the popularity for image colorization requests on a Reddit Thread (`/r/Colorization`), historical image colorization is a vital tool needed by researchers, archivists, and many people worldwide who wish to color black and

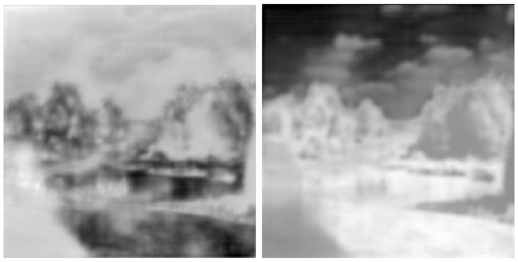

Figure 6. *Left:* predicted A channel of test image. *Right:* predicted B channel.

white images they may own.

The fact is, the ability to colorize black-and-white images can have significant impacts on society, although care and nuance is required in the training, implementation, and interpretation of these systems. Thus, there are both positive and negative impacts on using automatic colorizing models:

On the positive side, our project can enhance educational experiences. Colorizing historical photos can make past events feel more immediate and relatable, potentially making learning more engaging for students. Moreover, coloring these older images can offer a more accurate lens into "what once was", providing clarity and context to older materials in a modern viewpoint. It may help remove the guesswork in understanding societal, fashion, and cultural trends. Most folks see the world in color, and so color presentations of historical media helps humanize subjects and bring forth clarity beyond the black and white.

Additionally, colorization can deepen individuals' connections to their heritage by bringing old family photographs to life, thus fostering a greater understanding of their ancestry. Many individuals wish to use colorizing to revitalize images of passed loved ones, and reconnect to the past. Thus, automatic coloring systems such as ours have a lot of personal benefit to many impassioned would-be users.

That being said, automatic color depictions should always be taken with a grain of salt. Inaccurate colorization might misrepresent historical realities, misleading viewers unaware of the modifications. Ethical concerns arise regarding the alteration of historical records and the authenticity of colorized images compared to their original forms. If the training data for the colorization AI model is biased, these biases could manifest in the outputs, affecting the portrayal of skin tones and potentially perpetuating stereotypes or misrepresentations of certain demographic groups. Furthermore, an excessive reliance on colorized images might lead some to undervalue original artworks and photographs, potentially diminishing their historical importance and the artistic intent of the original creators.

To mitigate these risks, careful curation of training data and vigilant testing for biased outputs are crucial to maintain the integrity and accuracy of historical representations. It is also important to clearly label colorized images as modified to prevent confusion and misinterpretation. Used responsibly, colorization can be a powerful educational and engagement tool, but it requires careful handling and transparency.

6. Conclusion

In this paper, we experimented with using perceptual loss and VGG-19 U-Net architecture to predict the colors of grayscale images. The results of perceptual loss showed our model and architecture is viable for creating naturally looking colorful photos, but doesn't correctly account for unique coloring and saturation. Colors returned are plausible and look natural to the human eye. The model can be used to color any grayscale image, but has best use-cases for naturally existing photos, such as old black and white photography or night vision goggles (figure 7).

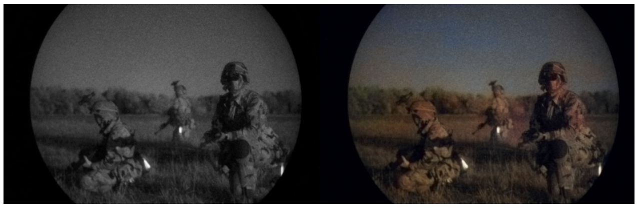

Figure 7. Recoloring of military night vision.

References

[1] Jiancheng An, Koffi Gagnon Kpeyiton, and Qingnan Shi. Grayscale images colorization with convolutional neural networks. *Springer-Verlag GmbH Germany, part of Springer Nature*, 2020. Published online: 24 February 2020. 1, 4

[2] R. Dahl. Automatic colorization, 2016. Accessed: May 12, 2024. 1, 2, 4

[3] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. *CoRR*, abs/1603.06668, 2016. 2

[4] Tung Duc Nguyen, Kazuki Mori, and Ruck Thawonmas. Image colorization using a deep convolutional neural network. *CoRR*, abs/1604.07904, 2016. 1

[5] Tanvi Sahay and Ashutosh Choudhary. Automatic colorization of videos, 2017. 2, 4

[6] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. *Proceedings of the University of California, Berkeley*, 2016. 1, 4

Appendix

Team contributions

John Farrell Developed web interface and Google Colab pipelines for our model. Researched broader methods of colorization, including quantizing colorspaces. Implemented perceptual loss function using gaussian blur.

Tyler Gurth Model architecture, visualization scripts and visualizations.

Hunter Adrian Researching effective methods and social impacts. Work on UNET architecture implementation and image pre-processing.

Jania Vandevoorde Imported and modified `main.py` and `tensorboard_utils.py` from HW5 for our use case. Read relevant research papers and helped build CNN architecture.