Course Project Final Report

Kirk Ford
University of Saskatchewan
Saskatoon, Saskatchewan
kwf014@usask.ca

Abstract

Deep Neural Image recolorization and palette-swapping is an image/style transfer technique that is especially useful for improving image quality, vintage photo/video restoration, generating new images with different moods or styles, and enhancing the visual appeal of images for artistic or commercial purposes. As such, it has incredible potential for use in creative domains. In this paper, I propose a modified version of PaletteNet, an existing recolorization model, that is expressly designed and trained for artistic use on a dataset consisting of fine art from 10 distinct styles, and only accepts 4 colors from the user as its palette to create reductive images. I compare the model to a more naïve color transfer algorithm, as well as examine how ablation, heavy hyperparameter tuning and general training mistakes can still lead to a model that has artistic merit and maintains semantic integrity. All code is available on the GitLab link below. https://github.com/KirkFord/CMPT-489-Project

1. Introduction

Creative industries and artists of all kinds play a significant role in today's society, impacting both the economy and culture at large. A major focus within these industries is the production, consumption, control, and assessment of visual content. With recent advancements in technology, there has been a surge of research in utilizing deep learning and computer vision techniques in various aspects of these creative fields. Art is routed in human creativity, and although certain creative processes cannot be accelerated, menial and/or time-consuming tasks that surround these processes can call upon deep learning algorithms to speed up or even revolutionize the overall production and acquisition of art. Recolorization aims to facilitate the issues involved with manually recoloring an image with a new color palette by using deep learning to automatically transfer given colors onto an image while preserving the images' original style. There is, however, several challenges within this field, including but not limited to:

1. Color Ambiguity: The mapping between the original colors and the target colors is often ambiguous, and multiple solutions can exist for a single image.
2. Semantic Preservation: It is important to preserve the semantic content of the image, such as the objects, scenes, and shapes, while changing the colors.
3. Color Consistency: The recolored image should be visually coherent and consistent, with colors that are realistic and blend well with each other.
4. Stylistic Preservation: The image should retain the original style and artistic qualities, such as texture, tone, and contrast.
5. Scalability: The recoloring process should work well for a wide variety of images and color schemes and be able to handle large amounts of data.

For the sake of this project, I will not be doing any colorization (where a greyscale image is given as input and a deep learning model discerns the colours of the image, used in image restoration), rather the end-user will supply an input image and a color palette that they wish to translate the input images' contents into, resulting in the output image. This task is an image-to-image translation problem with elements of color classification, which in an ideal scenario could be applied to any image to produce a new image with a palette of the users choosing. This task is transferrable to any photographic subject since any image could be used as by an artist for their projects, and as such, I am using the ArtBench-10 dataset, which has been created with PyTorch use in mind. The aforementioned challenges of this task are loose metrics for how well my deep learning model. Thus, the models I chose required proper color extraction, semantic segmentation, and recoloring capabilities. For my main model, I am using PaletteNet, a machine learning model that can automatically recolor images with a specified color palette and is closely related to my project [1]. It presents a deep neural network architecture that can be trained to predict new colors for an image based on a given color palette. The model can learn to generate plausible and consistent color modifications. The authors claim that the model can be applied to various creative domains such as graphic design, photography, and fashion. In these areas, the ability to automatically recolor images with a specific color palette can be used to generate new variations of an image or explore different color options. My baseline model is the photo recoloring optimization network [2] – which is a much simpler, non-deep learning algorithm primarily implemented in 2 parts: palette calculation, which divides the color space of the photo evenly and then applies a k-means clustering algorithm to determine the k-most used colors in the palette, and recoloring, which changes the colors of the calculated palette via individual LAB channel transformations and translations. Like PaletteNet, it is also designed to transform the palette of an input image into a new palette of the user’s choice.

2. Methods

The dataset I have chosen to use for this task is Artbench-10, a large-scale benchmark dataset for fine-grained art classification tasks. The dataset comprises 60,000 images of artwork from 10 different artistic styles, with 5,000 training images and 1,000 testing images per style. One of the key benefits of ArtBench-10 is its class-balanced distribution, which addresses the long-tail class distributions commonly found in previous artwork datasets. Each sample in the Artbench-10 dataset is an image of a fine art piece. The dataset offers images of high quality with clean annotations and is provided in three versions with different resolutions (32 x 32, 256 x 256, and original image size), formatted to be easily incorporated by popular machine learning frameworks. The images are in JPEG format, and each sample is associated with a set of labels given in a .csv metadata file, including the artist’s name, title, year of creation (if available), and art movement. An example of this image/label annotation is as such:

```
Image: "The Persistence of Time"
Art style: Surrealism
Artist: Joseph Cusimano
Year: 1963
```

2.1. Input Visualization

The version of Artbench-10 that I have selected has normalized all image dimensions to 256 x 256 pixels as well as standardized the bit depth to 24. However, the dataset does not contain duplicate photos with different colour palettes, which poses a fundamental problem: how will the model be trained to apply different colour palettes onto images without a training reference set of 2 differently coloured images? Luckily, this problem was answered by the researchers and creators of *PaletteNet: Image Recolorization with Given Color Palette* [1], which I have been using as the basis for my model. They proposed that to train PaletteNet to transform a source image into a desired target image, it is necessary to have a corresponding target ground truth image that is a differently colored version of the source image. However, in most cases, such an image does not exist. This is where color augmentation becomes crucial, as it allows us to define both the input and output of the network. Color augmentation involves modifying the pixel values of an image in a particular color space, such as HSV, RGB, or LAB, on a channel-by-channel basis. PaletteNet primarily uses hue-shift in the HSV color space, although this method can cause luminance distortion.

Because HSV does not distinguish between luminance and color characteristics, naive hue-shifting can lead to unwanted luminance distortion. To overcome this, the PaletteNet researchers have proposed a hue-shift algorithm that maintains the original image's luminance during color augmentation.

$RGB \rightarrow LAB$ and cache L

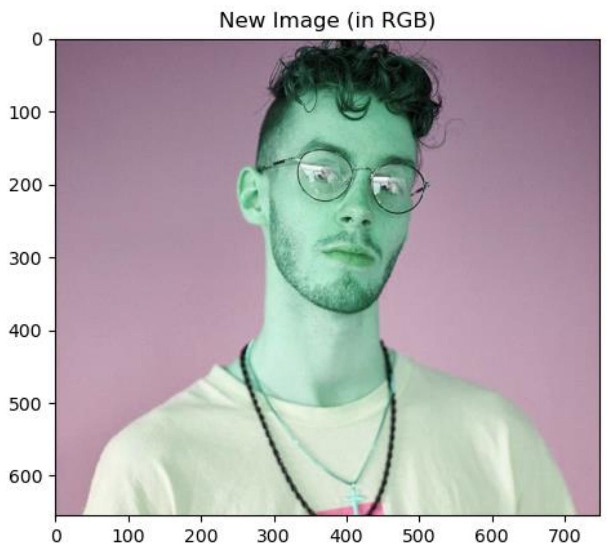
$$RGB \rightarrow HSV \xrightarrow{hue-shift} H^*SV \rightarrow L^*A^*B^* \qquad (10)$$

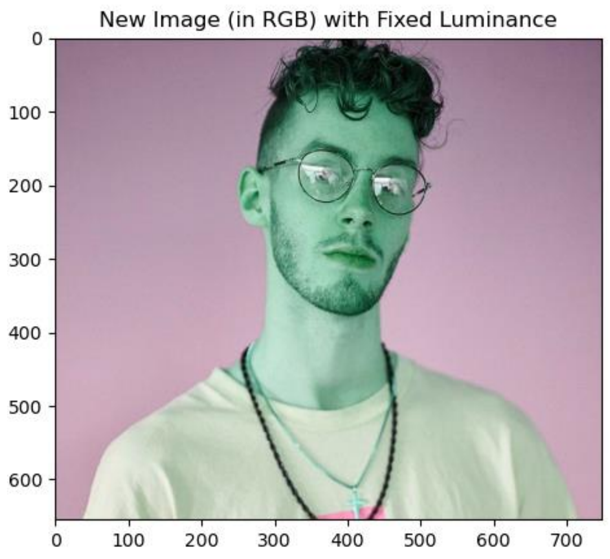
Final hue-shifted image: LA^*B^*.

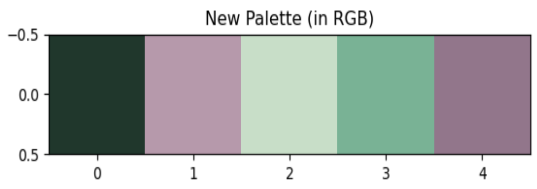
From *PaletteNet: Image Recolorization with Given Color Palette* [1].

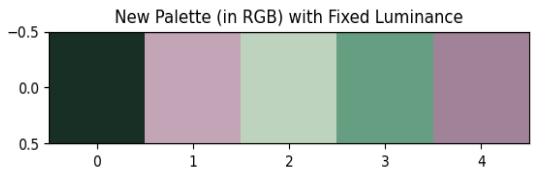
This technique successfully alters the color concept of the image while minimizing luminance distortion. Fixing luminance is important because the goal is to change only the color concept of the image, not its overall brightness. The Assumption is made that the corresponding palette of the hue-shifted image is also hue-shifted by the same amount as that of the original image.

I use this luminance-correcting hue-shift method to generate ground truth images, as it allows me to generate a fully specialized dataset for my task out of any pre-existing dataset. Below is a comparison of a ground truth image/palette vs. the same ground truth image/palette with its luminance corrected. Luminance distortion can severely disrupt the color balance of hue-shifted photos with high saturation, brightness and/or exposure if not properly accounted for.



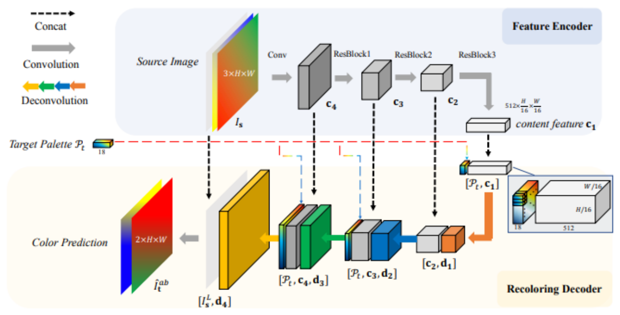






2.2. Main Model Description

The main model I am using is that of “PaletteNet: Image Recolorization with Given Color Palette” [1]. The model architecture is based on a modified U-Net, a type of convolutional neural network commonly used for image segmentation and image-to-image translation tasks. The model consists of an encoder and a decoder, with skip connections between corresponding encoder and decoder layers to enable the model to preserve fine details during the recolorization process. Afterwards, the model is put through an adversarial neural network discriminator that assigns image-pixel/palette pairs as either fake (not related to the input-palette) or real (related to the input palette) to further help the model distinguish luminance and semantics within an image. The authors also introduce a loss function that combines both perceptual and color similarity measures to improve the quality of the recolorized images and demonstrate that their model outperforms several state-of-the-art image recolorization methods in terms of both visual quality and color fidelity.



The proposed framework for “PaletteNet: PaletteNet: Image Recolorization with Given Color Palette” [1].

The PaletteNet model proposed in the paper "PaletteNet: Image Recolorization with Given Color Palette" has several hyperparameters that can affect its performance. Here are some of the main/relevant hyperparameters of the model:

1. Learning rate: The learning rate determines the step size taken during gradient descent optimization. A higher learning rate can result in faster convergence but may also cause the optimization to oscillate or diverge. The authors of PaletteNet used an initial learning rate of 0.0002.
2. Batch size: The batch size determines the number of samples used in each iteration of gradient descent. A larger batch size can result in faster convergence but may also require more memory and slow down the training process. The authors of PaletteNet used a batch size between 8 and 12 depending on the GPU.

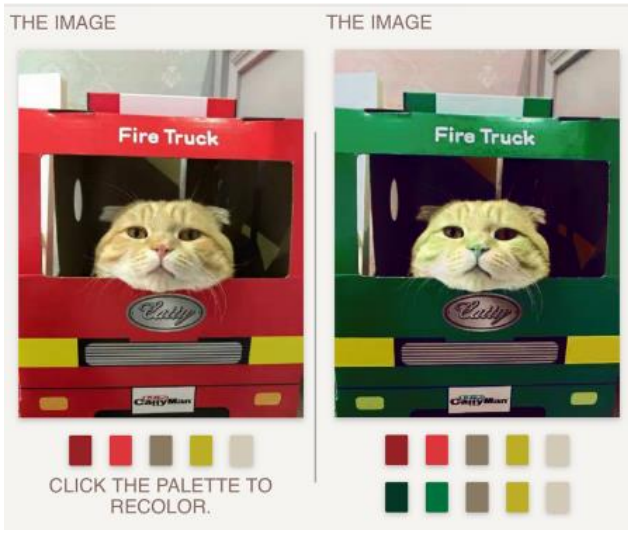
3. Number of epochs: The number of epochs determines the number of times the entire training dataset is passed through the model during training. Increasing the number of epochs can improve the model's performance but may also lead to overfitting. The authors of PaletteNet trained their model for 1000 epochs.
4. Number of filters: The number of filters determines the depth of the model and can affect its ability to capture complex features. The authors of PaletteNet used a modified U-Net architecture with 64 filters in the first layer of the encoder and decoder, and gradually increasing the number of filters in subsequent layers.
5. Loss function weights: As previously mentioned, the authors of PaletteNet used a novel loss function that combines both perceptual and color similarity measures. The weights assigned to each component of the loss function can affect the balance between preserving fine details and ensuring accurate color representation in the recolorized images.
6. Data augmentation: Data augmentation techniques such as random cropping, flipping, and rotation can help to increase the size of the training dataset and improve the model's ability to generalize to new images. The authors of PaletteNet used random horizontal flipping as a data augmentation technique.

These hyperparameters were chosen based on empirical experimentation and can be adjusted. As such, I have changed the way colours are extracted from images, opting into a KNN approach as described in my dataset plan. I decreased the number of colours in an input palette from 6 to 4, being able to experiment with increasing/decreasing the batch size to maximum 32 and minimum 8.

2.3. Baseline Model Description

For my baseline model, I am using the approach presented in the paper "palette-based photo recoloring" by Huiwen Chang et al [2]. The authors propose a technique that takes as input an image and a user-defined color palette and generates a new image where the colors have been replaced with colors from the given palette while preserving the overall appearance of the original image. The method is based on an optimization framework that iteratively updates the colors of the pixels in the image to minimize a cost function that measures the difference between the new color and the original color.

The authors evaluate their method on a dataset of images and show that it is able to produce visually pleasing results that are consistent with the user-provided color palette. They also compare their method with other standard recoloring techniques and show that it exceeds them in terms of both quality and computational efficiency.



These images were taken from the palette-based photo recoloring GitHub page: (`https://github.com/b-z/photo_recoloring`).

This model does not have any explicit elements of machine learning within its implementation. The method is based on an optimization framework where the cost function is designed to take into account both the perceptual similarity between the original and new colors, as well as the overall smoothness of the recolored image.

While the method does not use any machine learning algorithms explicitly, it does incorporate some elements that are similar to machine learning techniques. For example, the method uses a color difference measure that is based on the perceptual similarity of colors, which is similar to the idea of using a distance metric in machine learning. Additionally, the method uses spatial regularization terms to encourage smoothness in the recolored image, which is also similar to the idea of using regularization in machine learning to avoid overfitting.

While the paper does not explicitly mention hyperparameters, the optimization algorithm has several parameters that can affect the performance of the method. These parameters include:

1. The initial color palette: The user provides a set of colors that the algorithm uses to recolor the image. The choice of colors can greatly impact the final result.
2. The weight of the color difference measure: The cost function used in the optimization framework involves a color difference measure that weighs the difference between the original and recolored colors. The weight of this measure can be

adjusted to control the balance between color fidelity and color variety.

3. The weight of the spatial regularization term: The cost function also includes a term that encourages smoothness in the recolored image. The weight of this term can be adjusted to control the balance between smoothness and detail preservation.
4. The number of iterations: The optimization process iteratively updates the colors of the pixels in the image to minimize the cost function. The number of iterations can be adjusted to control the convergence of the optimization process and the quality of the final result.

The specific values of these parameters may depend on the characteristics of the input image, the chosen color palette, and the desired level of color fidelity and smoothness in the output image. Since I will be using this as my baseline model, this method will serve as an overall benchmark against a deep learning model that utilizes semantic recognition and preservation, as well as luminance preservation and colour diffusion techniques to better translate the input palette onto the image in question. I decreased the size of the input palette of this approach from 5 to 4.

2.4. Training Plan and Evaluation Metric

For my main model, I initialized the model using the He initialization method for the convolutional layers and the Glorot initialization method for the fully connected layers. The PaletteNet authors use a combination of two regularization schemes: L2 weight regularization and dropout, which I did not modify at this given time. The training algorithm I will use is the Adam optimization algorithm. The authors of PaletteNet also use a learning rate scheduler that decreases the learning rate during training to help the model converge more smoothly and avoid getting stuck in local minima. Specifically, they use a step decay scheduler that reduces the learning rate by a factor of 10 after a certain number of epochs. This helps to improve the stability and generalization performance of the model.

I used a CUDA-enabled implementation of PyTorch on an x64 Windows distribution to train this model. During training, the model was trained on the training set and its performance is evaluated on the self-created validation set at the end of a 100-epoch training session. The performance metric used for evaluation is the mean squared error (MSE) between the predicted and ground-truth palettes for each image in the validation set.

Since the baseline model is not a deep learning model, there will be no training plan required for it. An alternative evaluation metric that I am researching to compare my baseline and main models is the Structural Similarity Index (SSIM).

The SSIM is a widely used metric for evaluating the similarity between two images, which takes into account both their luminance and structural information. The SSIM index ranges from -1 to 1, where 1 indicates perfect similarity between the two images, 0 indicates no similarity, and -1 indicates complete dissimilarity.

To compute the SSIM between the predicted and ground-truth images for a given input image, the predicted and ground-truth images are first converted from the LAB color space to the RGB color space. Then, the SSIM is computed over the luminance and chrominance channels of the images.

Using the SSIM as an alternative evaluation metric in addition to the MSE can provide a more comprehensive evaluation of the model's performance. While the MSE measures the pixel-wise difference between the predicted and ground-truth color palettes, the SSIM measures the perceptual similarity between the predicted and ground-truth images, taking into account factors such as brightness, contrast, and structure.

3. Results

After realizing that training my model for 1000 epochs was somewhat overambitious, I trained the modified 4-color model on the testing set for 100 epochs on feature encoder/ recoloring decoder and a separate 100 epochs on the adversarial discriminator. Overall, it took between 10-15 minutes per epoch, resulting in an overall training session of just under 2 days. While this was somewhat lengthy, the average epoch time on the training set was around 3.5 hours, and as such, I could not feasibly complete 100 epochs on my home setup. I spent 2 days transferring my models and dataset into tux storage, and the GPU farm/ Blacktip netbook servers were giving me similar times per epoch. I tried changing batch size and learning rate, but to no avail. After I realized that the tux netbook servers could not reasonably handle this task either, I decided to focus my tests specifically on the test set, and by doing so I was able to obtain some very interesting results.

To preface, the baseline model did not need any training, as it is not a deep learning/ machine learning model. With my main model being a modified version of palettenet, it required double the training time, as I needed to train the encoder/decoder as well as adversarial neural network/discriminator. I noticed that my total loss was quite high while training the discriminator and would slowly whittle down per epoch. By the time 100 epochs were met, it still had a total loss value over 200, and created very interesting images but was ultimately unsuccessful at the task at hand. Continuing onward, I realized that my ablated model which only went through encoder/decoder training was performing quite well. This will be later visited in the ablation study.

Overall, this main model created through the test set creates reduced, pleasing and nearly semantically identical photos, even with an outrageous color palette. Because of PaletteNet's training, it can introduce colors in a much more subdued fashion than the naïve color algorithm. By using a SSIM tool created by Nathancy (more info under acknowledgements on GitLab) I was able to quickly analyze the structure of the original photo against the newly created photo. Using the same palette for both models (orange, brown, purple, green), the SSIM tool reported that the Image similarity score was 0.936519987490214 (Figure 1), which is quite good when compared to SSIM value of the naïve algorithm, which produced an Image similarity score of 0.48873317084460277 (Figure 2).

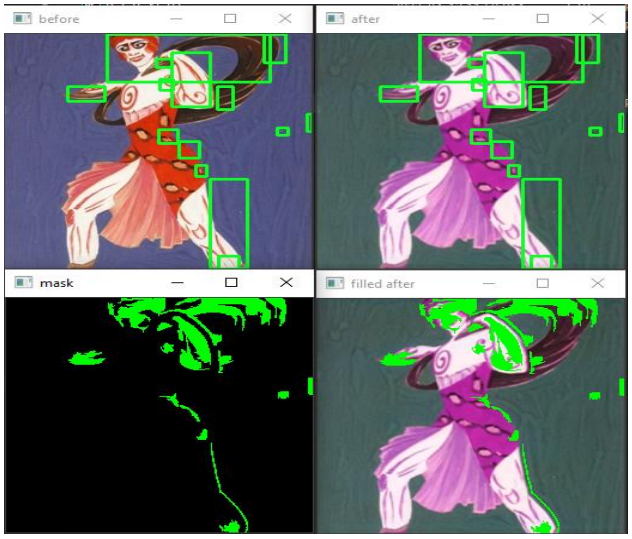


Figure 1: Original image vs. Main Model Image.

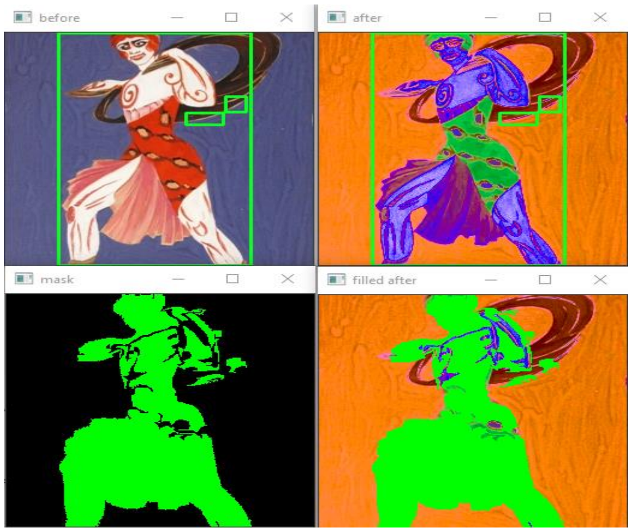


Figure 2: Original image vs Baseline Model Image.

3.1. Hyper Parameter Experiment

After the initial training session, I knew I needed to tune hyperparameters to minimize my total loss, especially after seeing the loss on the adversarial network. To achieve that, I modified my main model to use TensorBoard to track scalars such as loss, as well as the weights/biases of convolutional layers. To come up with an optimal configuration, I changed the learning rate, batch size, and image shuffle configurations values into dynamic values, having 2 different learning rates (0.01, 0.0002), 2 different shuffle states (on and off) and 3 different batch sizes (8, 16, and 32), which gave me 12 different total combinations of hyper parameters. Using my new TensorBoard setup, I ran each hyper parameter combination for 5 epochs, and tracked the loss at the end of each run. Figure 3 visually demonstrates all combinations final loss after their respective runs, and Figure 4 contains the raw data on how well each combination of hyper parameters performed regarding minimizing loss.

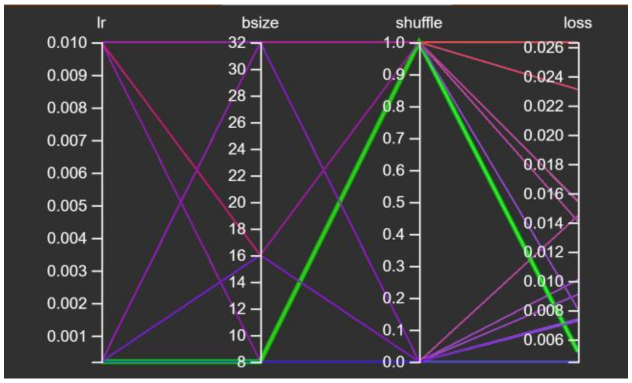
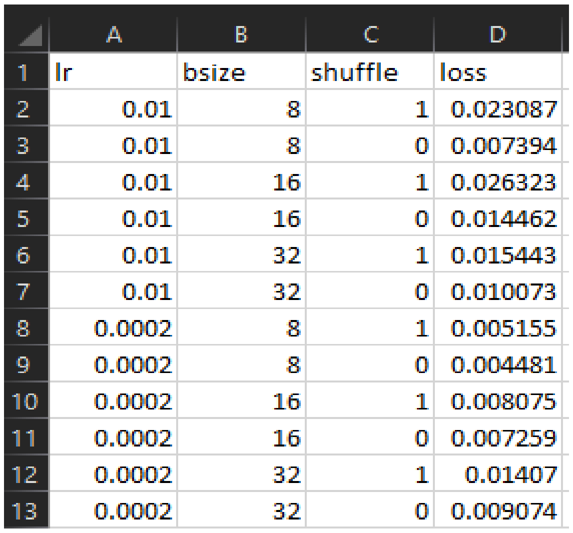


Figure 3: TensorBoard Visualization of all 12 runs total loss. The data for these runs are available on GitLab.



	A	B	C	D
1	lr	bsize	shuffle	loss
2	0.01	8	1	0.023087
3	0.01	8	0	0.007394
4	0.01	16	1	0.026323
5	0.01	16	0	0.014462
6	0.01	32	1	0.015443
7	0.01	32	0	0.010073
8	0.0002	8	1	0.005155
9	0.0002	8	0	0.004481
10	0.0002	16	1	0.008075
11	0.0002	16	0	0.007259
12	0.0002	32	1	0.01407
13	0.0002	32	0	0.009074

Figure 4: Raw data on each hyper parameter run. lr = 0.0002, bsize = 8, shuffle = 0 had the lowest total loss.

All these hyper parameters generally affected loss rate, with shuffle-on consistently raising the loss of its non-shuffled counterpart, and higher batch sizes/ learning rates generally resulting in an increased total loss value. lr = 0.0002, bsize = 8, shuffle = 0 had the lowest total loss, and lr = 0.01, bsize = 16, shuffle = 1 had the highest total loss. Higher batch sizes/and learning rates consistently finished faster than lower sizes/rates but lack the fine-grained precision that lower/sizes/rates provide to the models weights/biases.

3.2. Ablation Study

In the original PaletteNet paper, the training plan for the model involved two phases: Pretraining Feature Encoder (FE) and Recoloring Decoder (RD) with Euclidean loss, Freezing the parameters of FE and training RD with additional Adv-loss. This split training stabilizes the learning of the recolorization process with Adv-loss (Figure 5). For the ablation study, I decided to save my model both after the encoder/decoder training and after the adversarial training, resulting in two separate models, one with and one without adversarial training/ concatenation.

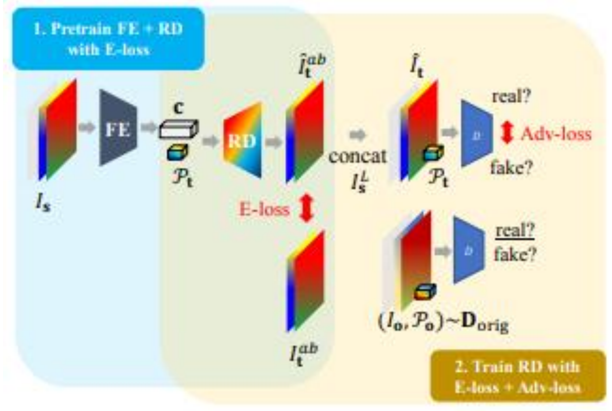


Figure 5: the original training plan for PaletteNet Model.

I chose to cut out all adversarial training as my ablated model because, interestingly enough, my original model that included the adversarial loss performed worse in terms of adversarial total loss versus Euclidean total loss and worse on the SSIM tests than my ablated model which only trained 100 epochs on FE/RD. Figure 6 shows the SSIM tests of the original image versus the adversarial model image generation, on which it scored an image similarity of 0.6427447836424582, much less than the ablated model's score of 0.936519987490214 in figure 1, although these results were still better than that of the photo recoloring baseline model.

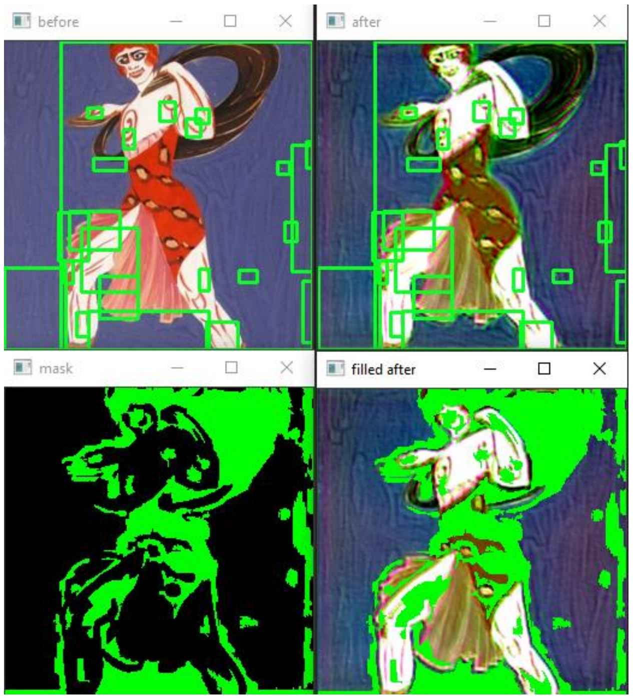


Figure 6: Original image vs non-ablated model.

Originally, I was unsure if the ablated model would perform well at all, as I initially began my image testing with the full model, but as once I realized that the ablated model gave better results, I switched all my testing to that model. The main reason I believe the non-ablated model performs worse is due to FE/RD being trained on less data/epochs than the original PaletteNet-documented model, thus when those weights/parameters are frozen to train the adversarial network, it has less of a tuned model to as its foundation, and then only gets trained for a fraction of what the team behind PaletteNet trained their model on.

4. Discussion

Because this task is focused on deep learning for creative domains, subjective measures such as image appeal, style, and quality are just as important as objective measures such as performance and task efficacy, which has led to some unusual but welcome discoveries. My fully trained adversarial model produces images that are trying to discriminate each image-pixel/palette pair as real or fake, which results in a sort of distortion between the input palette and the original color palette. This was an unintended effect, but nevertheless generates a new stylistic rendition of an image. Had more time been put into training both the FE/RD and the adversarial models, I believe that this issue would have been solved, or at the very least somewhat alleviated. Upon showing some colleagues the faulty model, they preferred the images that it generated over the images generated from the more accurate ablated test model. This shows that, as with all art, preference plays a huge role on the actual level of appeal that an artistic product can have, and as such, mistakes in artistic models can lead to some novel uses and applications of deep learning models.

5. Conclusion

In conclusion, deep neural image recolorization and palette-swapping is a powerful technique that has immense potential for improving image quality and generating new images with different styles and moods. The modified version of PaletteNet proposed in this paper is an excellent example of the adaptability of this technique for artistic purposes. The model successfully created reductive images via the 4-color palette training that maintain semantic integrity. The study compared and demonstrated the superiority of the proposed model against a more naïve color transfer algorithm and demonstrated that through ablation, hyperparameter tuning, and general training mistakes, it was still possible to create a model with artistic worth, even if that model does not do what it was originally intended to do.

References

[1] Cho, J., Yun, S., Lee, K., & Choi, J. (2017). PaletteNet: Image recolorization with given color palette. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. https://doi.org/10.1109/cvprw.2017.143

[2] Chang, H., Fried, O., Liu, Y., DiVerdi, S., & Finkelstein, A. (2015). Palette-based photo recoloring. *ACM Transactions on Graphics*, *34*(4), 1–11. https://doi.org/10.1145/2766978