Stable Image Colorization via Reference Image Cross-Attention

Anshu Bansal
Stanford University
anshub@stanford.edu

Abstract

Existing colorization methods typically encode all relevant priors in their learned weights. Previous work by Vondrick et al. [19] demonstrated that attending to prior video frames is a powerful method for frame colorization. In this work, we combine the attention mechanism introduced by Vondrick et al. with a U-Net-based colorization network to encode priors not captured by prior frames. Specifically, we aim to allow the network to use reference images that are temporally distant from the target image. Experimental results show that the trained model performs poorly in practice, however it shows significant improvements over a similarly architected and trained non-attention colorization network. Additionally, we detail construction of a high-quality image-pair dataset for reference colorization.

1. Introduction

Image colorization is the problem of inferring colors from greyscale images. In addition to its prevalence as a self-supervised task for image embedder training, it has independent applications in art and animation, as well as historical archiving. [15]

Existing colorization techniques largely rely on memorizing object-color relationships in the network parameters. This presents several problems: the network must have enough capacity and enough training data to memorize these, the model may generalize poorly (especially given that in colorization, an object can have many equally plausible colors. For example, absent any other context, a T-shirt can be any color.), and the model may colorize the same object with different colors in slightly different contexts. For example, a person's hair may be colorized red in one picture of them, but black in another. These issues are seen in existing works. [15] [22]

Reference colorization is a relaxation of the general colorization problem. In reference colorization, we are supplied a reference image in addition to the target image. The reference image is assumed to show similar objects to the target image, and provides grounding for colorization choices (for example, supplying the model with color information for a T-shirt in the image). Vondrick et al. [19] showed that supplying a reference frame in video colorization led to substantially improved results over prior techniques without reference images. Their approach extracted a feature embedding for each pixel (in a low-resolution version of the reference and target image) and used a simple softmax attention mechanism to retrieve the color directly from the reference image. This was then assumed to be the color of the pixel in the target image. Note that colorization was not the primary focus of Vondrick et al's work – rather, colorization was a self-supervised task in service of performing object tracking. [19].

This presents two major problems for colorization. First, objects which are present in the target image but not in the reference image may have a color that is not in the reference image at all. Thus, coloring them by retrieving the most likely color from the reference image would be inaccurate. Second, objects in subsequent frames may change color (for example, if the illumination changes). As before, simply retrieving colors from the reference image would not suffice.

In this work, we fuse Vondrick et al.'s attention-based approach with prior U-Net approaches [17] [23], allowing the model to retrieve colors from the reference image, color objects which lack reference colors according to the model's learned priors, and account for objects which change color across frames. Similar work has been performed by Yoo et al. [21], however their model, data, and full results are proprietary, and thus our work contributes an open implementation for further study. As part of this contribution, we also detail the construction of a high-quality image-pair dataset, which was extremely beneficial in exploring reference colorization models.

2. Related Work

Anwar et al. [15] provide an excellent review of the image colorization problem and existing approaches. Here, we summarize their findings, with particular focus on the approaches which directly inspired our work.

2.1. Single Image Approaches

These approaches attempt to solve the "hard" version of colorization, in which only the greyscale image is provided, without any further context grounding the colors of the image.

2.1.1 Colorful Colorization

Colorful Colorization [24] is an early approach to colorization which trained a simple CNN to predict the "ab" components of the input image's L*ab representation. The model is quite large by the standards of the time, with a maximum of 512 channels and trained for 450k iterations.

2.1.2 U-Net Image Colorization

Zayd et al. [23] developed a U-Net [14] based architecture for image colorization. This is trained similarly to the Colorful Colorization model, however, the "skip" connections in the U-Net architecture grant the model to access the original image's lightness component, allowing it to make better color choices during the upsampling phase. Their work uses ResNet [5] blocks in the downsampling arm of the U-Net architecture, potentially allowing more efficient use of the downsampling component's parameters. Note that even with this more efficient architecture (as compared to a simple CNN), the model still requires roughly 63M parameters.

2.2. User-Guided Networks

These approaches rely on "hints" which disambiguate the color for key segments of the image. These "hints" may be provided by an end user or taken from the environment.

2.2.1 Scribbler

Scribbler [16] is a GAN colorizer which takes as input small strokes of color over segments of the target image. This could be, for instance, a stroke of red over a red T-shirt, indicating that that segment of the output image should be red. The generator is constructed similarly to Zayd et al's U-Net.

2.2.2 Hint-Guided Anime Colorization

Hint-Guided Anime Colorization [22] is similar to Scribbler, in that it takes in small scribbles indicating the true colors of various segments. However, it is trained specifically on anime line art. The original image in this case tends to have very bold patches of color, and thus may be more amenable to colorization by propagating the provided colors.

2.3. Reference Image Colorization

This category is most similar to our approach, as these provide a similar image, which contains subjects present in the target image, as a reference image for the model. Thus, the model is able to disambiguate color choices by referring to the supplied image.

2.3.1 Colorization via Attention

Vondrick et al. [19] determined that a simple attention mechanism (originally designed for object tracking in video) performed well when trained against a colorization self-supervision task. However, colorization was not the primary focus of Vondrick's work; rather, the attention maps learned as part of colorization were effective in tracking pixel motion on a frame-by-frame basis. Viewed from the perspective of colorization, there are significant drawbacks to this approach. Because all pixel colors had to be derived from attending to the previous frame, new objects that weren't present in the previous frame could not be accurately colored. Furthermore, the model was only shown to work on a frame-by-frame basis, where the target and reference image had very little difference due to their close temporal proximity.

2.3.2 Memo Painter

MemoPainter [21] is an approach which augments a U-Net-inspired GAN with a memory mechanism. This memory mechanism holds learned representations of individual training examples, allowing the generator to reference specific objects. Thus, it does not receive an explicit reference image during training, instead retrieving reference features from its memory. However, the model, data, and training pipeline used by Yoo et al. are not open, and thus we are unable to corroborate their results.

2.3.3 Deep Exemplar-based Colorization

Deep Exemplar-based Colorization [6] solves reference colorization by using the luminance channels from the L*ab representations of the reference and target images to explicitly align pixels between the two images. This is then used to explicitly "stack" the chrominance channels of the reference image with the target image's luminance channel as a preprocessing step to a U-Net. The model is trained as a GAN.

2.4. Classical Approaches

2.4.1 Color Transfer

Welsh et al. [20] introduced a reference-coloring algorithm which matches 5x5 pixel neighborhoods in the target image against neighborhoods in the reference image. Pixels in the

target image are matched to the reference image by comparing the mean and standard deviation of the luminance channels in their neighborhood, and the chrominance channels are taken from the most matching reference pixel.

3. Proposed Method

Given a reference full-color image and a target greyscale image, we train a deep network to synthesize a full-color version of the target greyscale image.

3.1. Learning Framework for Reference Colorization

The reference image for our network is an image $X_{\text{ref}} \in \mathbb{R}^{3\times H_{\text{ref}}\times W_{\text{ref}}}$. The target image is a greyscale image $X_{\text{target}} \in \mathbb{R}^{3\times H_{\text{target}}\times W_{\text{target}}}$. Note that in this framework, the target image also has three channels, even though it is greyscale. In our learning framework, we keep images in the RGB color space, so for X_{target} all channel values will be the same per-pixel. For training and validation, X_{target} is derived by converting an original image $X_{\text{orig}} \in \mathbb{R}^{3\times H_{\text{target}}\times W_{\text{target}}}$ to greyscale according to the standard PyTorch [13] `Grayscale` functionality.

The original image forms the ground-truth for colorization, as detailed in the "Loss Function" section below. The reference image is assumed to be "similar" to the target image. That is, it should show many of the same subjects as the target image. This allows the model to extract relevant color information from the reference image. We detail this further in the "Datasets" section below.

3.2. Loss Function

Ballester et al. [1] provide a comprehensive overview of the various loss functions commonly (and uncommonly) used in colorization. To reduce GPU consumption, we chose a loss function for simplicity and computational efficiency. Thus we use mean-squared error as the sample loss between a colorized image and the ground-truth image.

Given images $u, v \in \mathbb{R}^{C\times H\times W}$, the mean-squared error (MSE) is defined as

$$\text{MSE}(u,v) = \sum_{i=1}^{C}\sum_{j=1}^{H}\sum_{k=1}^{W}(u_{i,j,k} - v_{i,j,k})^2 \qquad (1)$$

Ballester et al. note that MSE is not robust to outliers in the data, and is tolerant to small absolute per-pixel errors over a large region of the image. Our experiments did show this failure case. However as our goal is to evaluate the relative lift in performance across differing model architectures, the quality of the loss function is less relevant to our analysis than the relative performance of the different loss functions on it. We discuss the potential pitfalls and corrections to this in the "Further Work" section.

During training and validation, we compare the colorized image, $f(X_{\text{target}})$ against the original image, X_{orig}. Thus, the loss incurred on a single image, with colorizer function f, is $\text{MSE}(f(X_{\text{target}}), X_{\text{orig}})$

3.3. Model Architecture

We evaluate two distinct model architectures. Inspired by Zayd et al. [23] we train a simple U-Net colorization model as a baseline model. This does not use the reference image, relying on the learned network parameters to encode information about what colors are relevant for objects in the target image. We also train StableColorizer, which augments the U-Net model with a cross-attention mechanism to attend to features extracted from the reference image. We extract these features with a simple encoder derived from an autoencoder which we also train.

3.3.1 Baseline U-Net Architecture

Our implementation of a U-Net colorizer is significantly smaller than the implementation provided by Zayd et al. Specifically, this implementation uses a total of 177,132 trainable parameters, in contrast to the roughly 63M trainable parameters in the U-Net trained by Zayd et al. [23]. This is done to optimize for computational efficiency, as the StableColorizer will have a similarly small number of parameters.

Our U-Net colorizer consists of four stride-2 down-convolutional blocks. Each down-convolution uses a 3x3 kernel, and is followed by batch-normalization [9] and a GELU nonlinearity.

Following the down-convolutional layers we up-convolve the image back to its original height and width, using the same number and configuration of up-convolutional blocks as we had down-convolutional blocks. As in the U-Net architecture, we introduce skip-connections between each down-convolutional block and its corresponding up-convolutional block. We do this by concatenating the down-convolved image with the previous up-convolutional layer's output along their channel dimension. As with the down-convolutions, we follow each up-convolution with a batch-normalization and GELU nonlinearity.

At the end, we concatenate the greyscale input image with the final upconvolved image, and apply a 1x1 convolution to map it back to 3 RGB channels. Figure 1 shows the architecture of our baseline model.

3.3.2 StableColorizer Architecture

StableColorizer augments the baseline U-Net architecture with an attention mechanism to attend to features extracted from the reference image. Specifically, we use a frozen encoder (trained as part of an autoencoder) to featurize the reference image, and use the final down-convolutional layer

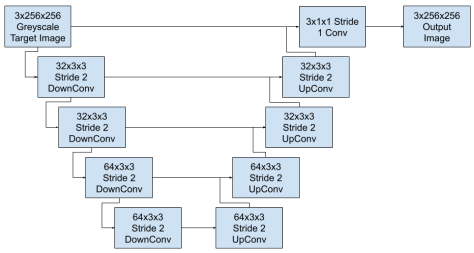

Figure 1. Architecture of our baseline colorizer. Note that skip connections are implemented by concatenating along the channel dimension, and all convolutions are followed by batch normalization and GELU nonlinearity

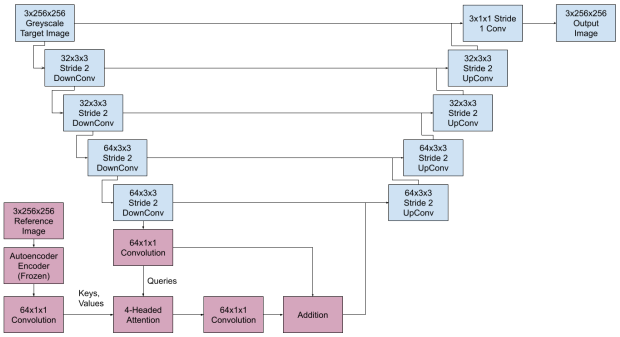

Figure 2. Architecture of StableColorizer. Note that skip connections are implemented by concatenating along the channel dimension, and all convolutions are followed by batch normalization and GELU nonlinearity. Components not present in the baseline U-Net are highlighted in red.

of the U-Net to compute queries which we use to attend to the autoencoder features. We discuss the autoencoder architecture in the Autoencoder Architecture subsection below. Here we assume that it encodes images into 64-dimensional patches.

We insert a 64x1x1 convolutional block after the encoder to adapt its outputs for the attention module. This includes a batch normalization and GELU nonlinearity after the convolutional layer. We attach an identical adapter block to the output of the final U-Net down-convolutional block. Following this, we use a multiheaded attention module with 4 attention heads, using keys and values from the encoder adapter and queries from the U-Net down-convolution adapter. We finally use another convolutional block with batch normalization and GELU nonlinearity on this output, and add it to the adapted U-Net down-convolutional output. This is then concatenated with the raw U-Net down-convolutional output as input to the first up-convolutional layer. Following this, the network architecture proceeds as in the baseline U-Net. Figure 2 shows the architecture of StableColorizer. The StableColorizer has a total of 309,452 parameters, including the frozen parameters in its reference image encoder.

3.3.3 Autoencoder Architecture

We use a simple encoder, derived from an autoencoder, to featurize the reference image. We chose to train an autoencoder rather than use a pretrained image embedder such as ResNet both to keep the parameter count low and to ensure that color information is preserved in the encoder. Given that ResNet is trained on image classification, we believe that the autoencoder training problem forces the network to preserve color information rather than focusing on higher-level object features.

The autoencoder has effectively the same architecture as the basic U-Net, but without any skip connections between the down-convolutional and up-convolutional blocks. That is, it contains four down-convolutional blocks in the encoder, each with 3x3 kernels applied with a stride of 2, outputting 32, 32, 64, and 64 channels, in order. This reduces the height and width of the image each by a factor of 16. The up-convolutional blocks are inverses of the down-convolutional blocks, and form the entirety of the decoder. Every convolution is followed by a batch normalization and GELU layer. The full autoencoder has 131,721 parameters, with 65,952 in the encoder.

4. Datasets and Data Processing

We required significant image data sources to train and evaluate the autoencoder, basic colorizer, and StableColorizer models. While autoencoding and simple colorization can use any image data, reference colorization requires pairs of related images. In this section, we detail each of the datasets we used, and our process for cleaning and collating the data required for model training and validation.

4.1. Datasets

4.1.1 COCO

In order to train our autoencoder, we simply needed a diverse source of natural images. We used the 2017 COCO validation dataset for this [12]. COCO is an open dataset of images used for a variety of vision model benchmarks. Since autoencoder training does not require labels, we simply used the raw images, and did not use any of the annotation data provided. This COCO dataset contains 5000 images, each at a size above 512x512 pixels.

4.1.2 KoNViD

For reference colorization, we required pairs of images which shared subjects, such that one image provided meaningful context for colorization decisions in the other. We

could not find a ready source of image data for this. Thus, we synthesized image pairs from video data. Specifically, we used the KoNViD-1K [7] [8] and KoNViD-150K-B [3] [4] datasets to provide (reference, target) image pairs. These datasets provide a total of 2,776 8 to 10 second videos, originally intended for video quality assessment. A qualitative assessment indicated that the first and last extracted frames of the video tend to be significantly different from each other, but contain many objects in common. Thus we constructed a dataset of 2776 image pairs by taking the first and last frame from each video. Figure 5 shows an example of frame pairs extracted from KoNViD. These videos (and thus, the frames) are at 960 x 540 resolution.

Figure 3. First Frame

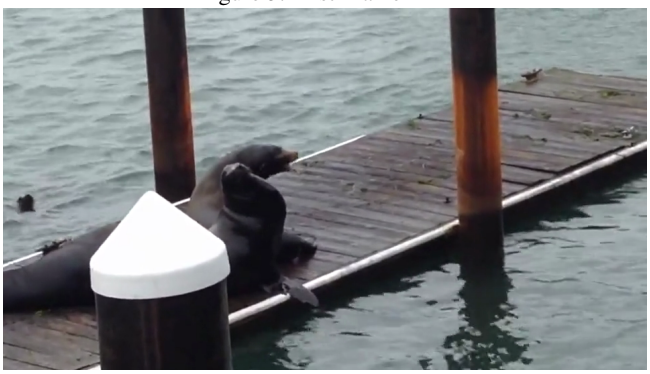

Figure 4. Final Frame

Figure 5. Example frames of videos from the KoNViD dataset. Note that the frames share common objects, such as the white cone.

4.2. Data Processing

In order to reduce the computational expense of training our models, we resized all images to 256 x 256 resolution. This resolution was chosen so that our down-convolutional layers in the autoencoder, basic colorizer, and StableColorizer, which reduced the height and width of the image by a factor of 16, could cleanly reduce the image dimensions. For the target image to colorization, we converted the last frame of each image pair to greyscale, and used the first frame as the reference image.

Conversion from RGB to L*ab color space is frequently performed in colorization tasks. However, we encountered technical difficulties in performing this conversion in PyTorch, and continued with RGB color space.

5. Experiments, Results, and Discussion

5.1. Comparison of Network Sizes and Resource Usage

Our closest point of network size comparison, as measured by number of trainable parameters, is from Zayd et al. [23]. The U-Net constructed in their work was roughly 63M parameters, and significantly underperforms other models mentioned in the Related Work section. By contrast, the baseline colorizer trained in these experiments has 177K parameters, and the StableColorizer has 309K parameters. This led to very quick model training, with each model taking less than an hour to train on readily available Google Colab instances (with a total training cost of roughly $4). However, their relatively tiny size makes them unsuitable to compare directly against state-of-the-art models. Rather, we compare them to each other to understand the performance differences which arise from their different architectures.

5.2. Autoencoder Training

We trained the autoencoder for 20 epochs on the COCO dataset, with an 80/20 training/validation data split. Adam [10] was chosen as the optimizer, with a learning rate of 0.001, and otherwise default optimizer parameters. This learning rate successfully achieved model convergence, as shown by the loss curves in figure 6. Figure 9 shows example images reconstructed by the autoencoder at epochs 5 and 20. We used this to qualitatively verify that our autoencoder training was behaving as expected. These qualitative results indicate that the autoencoder is not maintaining very accurate color representations, especially in rarer colors. For example, it seems to correctly color the sky as white or blue, and water as blue, but the pink umbrella becomes grey. This is significant to our later interpretation of the StableColorizer results.

5.3. Colorizer Training

We trained the basic U-Net and StableColorizer for 30 epochs on the image pairs in the processed KoNViD dataset. Note that the basic U-Net does not actually use a reference image – our implementation still accepts a reference image in order to maintain API parity with the later StableColorizer. Thus, both are trained with the same set of training and validation images. As with the autoencoder, we used an 80/20 training/validation data split. We again used Adam as the optimizer, however we found that the default learning rate of 0.001 led to very slow model convergence. We hypothesize that this is because the loss values for MSE loss

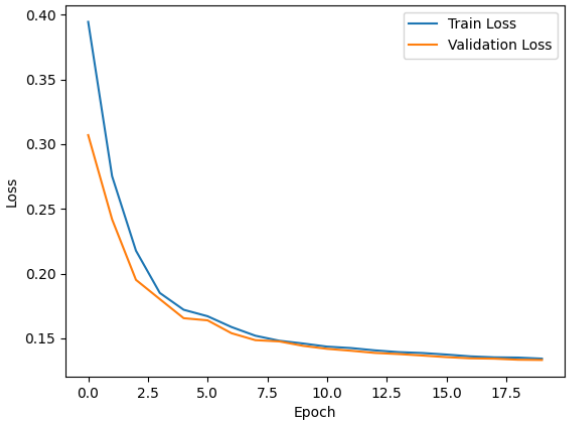

Figure 6. Autoencoder Loss Curves

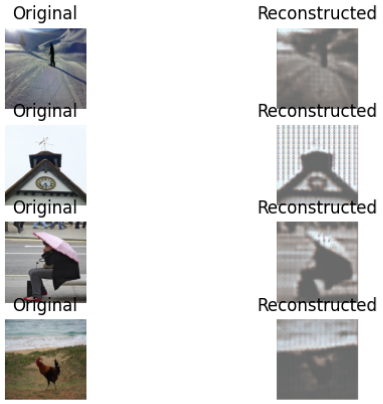

Figure 7. Epoch 5

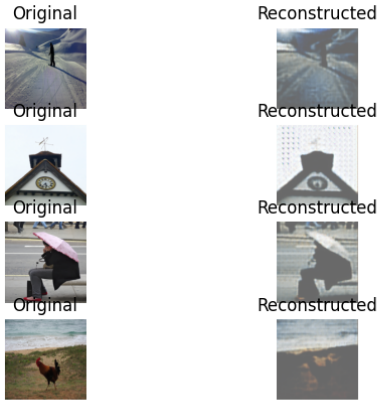

Figure 8. Epoch 20

Figure 9. Validation-set reconstructed images from autoencoder training.

in RGB color space tends to be quite small. Increasing to 0.01 provided a much more rapid and stable convergence, and thus we applied this to both the basic U-Net and to the StableColorizer.

As stated above, we provided the frozen encoder from the autoencoder to the StableColorizer. Its parameters were not fine-tuned for colorization. Figure 12 shows the loss curves obtained by training the basic colorizer and StableColorizer. Figure 15 shows images colorized by the basic colorizer and StableColorizer models after training.

The loss curves demonstrate that in both cases the model has effectively converged, however, the qualitative results indicate that the colorization is not typically high quality in either case. Both colorizers output relatively muted colors, with the basic colorizer in some cases outputting a nearly-monochrome image. By contrast, the StableColorizer seems to much more readily produce more vibrant colors, as can be seen in the sky on the second image. However, it also seems to mis-transfer colors sometimes, as seen in the red colorized background of the first image, where the ground truth is brown. The muted colors may be accounted for by an observation by Ballester et al. [1], that MSE loss tends to prioritize avoiding large losses on a single point; instead, it will often mis-color large regions of the image slightly. Given that these images are represented in RGB color space, the model may be able to get "close enough" by maintaining the same greyscale value as it's given, and only colorizing regions where it's highly likely to get the color correct.

5.3.1 The "Cheating" StableColorizer

The StableColorizer showed qualitative improvements over the basic colorizer, but still produced low-quality images. The "Cheating StableColorizer" is trained identically to the StableColorizer. However, rather than using a similar image to the target image as its reference, it uses the original image. That is, $X_{\text{ref}} = X_{\text{orig}}$, and $X_{\text{target}} = \texttt{Grayscale}(X_{\text{orig}})$. Since the model is provided exactly the desired output as input, this represents the best performance we could hope to obtain from the original StableColorizer model given its architecture, image processing, and training pipeline. Figure 16 shows the loss curve for this model, and Figure 17 shows example images produced after training the Cheating StableColorizer.

We see from these results that the Cheating StableColorizer achieves a final training loss of 0.017, similar to that of the ordinary StableColorizer. However, it has a significantly lower validation loss, at 0.019, vs the StableColorizer's 0.024. This indicates that indeed, the Cheating StableColorizer is able to learn to copy as directly as possible from the reference image. However, examining the re-

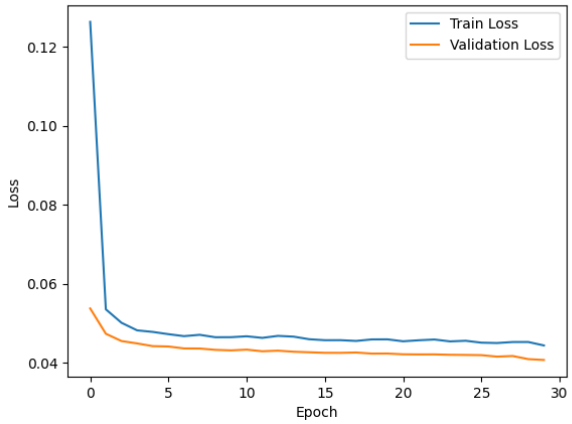

Figure 10. Basic Colorizer

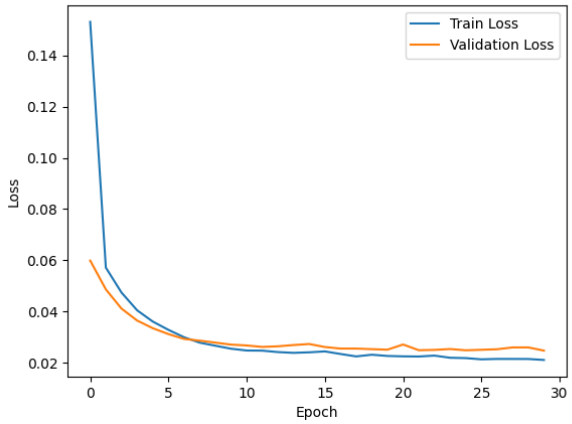

Figure 11. StableColorizer

Figure 12. Loss curves for the Basic and Stable Colorizers

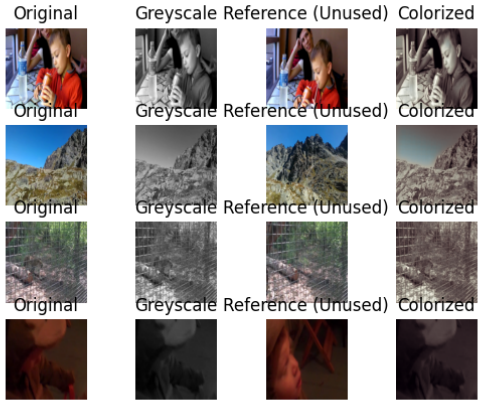

Figure 13. Basic Colorizer

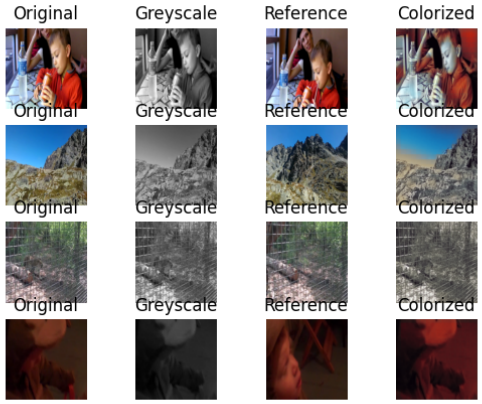

Figure 14. StableColorizer

Figure 15. Validation images taken from the Basic and Stable Colorizers

constructed images from the autoencoder model, it appears that the autoencoder does not maintain color information accurately. This may be the root cause for the Cheating StableColorizer's poor image quality. Note also that both the Cheating StableColorizer and StableColorizer have very low MSE losses (in absolute terms), given their extremely poor colorization quality. This indicates that MSE may be a poor loss metric for colorization.

6. Conclusions and Future Work

The results support our hypothesis that cross-attention to extracted reference image features can improve colorization accuracy, however, given the small size of the models tested, it is unclear if this is necessary in order to improve image quality at larger scales. Overall, the muted color palettes corroborate Ballester et al.'s [1] finding that MSE loss is a poor choice for colorization training.

The KoNViD image-pair dataset prepared as part of this work represents a significant building block for future work on reference image colorization. Previous work in the space, such as those by Vondrick et al. [19] and Yoo et al. [21] either directly used video data or used propriety datasets. With the preprocessing and frame extraction performed on the KoNViD videos, there is now a high-quality set of related image pairs to build future reference colorization models on.

Significant follow-up work is possible to increase the performance of these models. Improvements may be possible from training and evaluating in L*ab space, instead of RGB space, as Anwar et al. [15] suggest. More aggressive loss functions, such as MAE loss, may also improve the perceived color accuracy. Following these improvements, increasing the size of the model to be on-par with existing state-of-the-art models may prove useful in getting an accurate qualitative comparison of the architecture here against

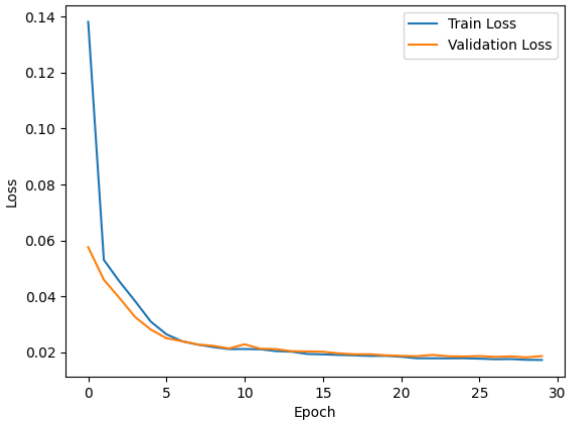

Figure 16. "Cheating StableColorizer" Loss Curves

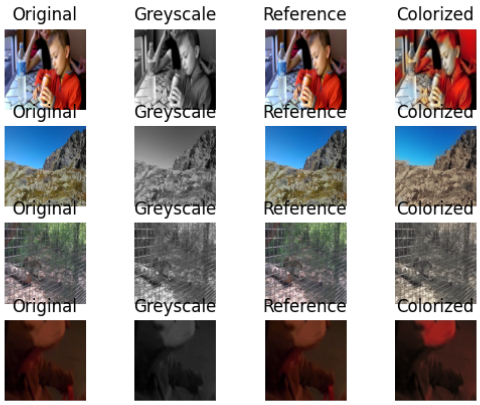

Figure 17. "Cheating StableColorizer" sample validation images after 30 epochs of training

existing methods.

7. Contributions and Acknowledgements

Much of the background for this project was taken from the Stanford CS231N course. We would like to thank the teaching staff of the course for their work.

Jenny Xu served as a research mentor for this project, giving valuable feedback on missing components, and helping us refine our ideas.

References

[1] C. Ballester, A. Bugeau, H. Carrillo, M. Clément, R. Giraud, L. Raad, and P. Vitoria. Analysis of different losses for deep learning image colorization, 2022.

[2] Z. Cheng, Q. Yang, and B. Sheng. Deep colorization, 2016.

[3] F. Götz-Hahn, V. Hosu, H. Lin, and D. Saupe. The konstanz 150k in-the-wild video database (konvid-150k), 2021.

[4] F. Götz-Hahn, V. Hosu, H. Lin, and D. Saupe. Konvid-150k: A dataset for no-reference video quality assessment of videos in-the-wild. In *IEEE Access 9*, pages 72139–72160. IEEE, 2021.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[6] M. He, D. Chen, J. Liao, P. V. Sander, and L. Yuan. Deep exemplar-based colorization, 2018.

[7] V. Hosu, F. Hahn, M. Jenadeleh, H. Lin, H. Men, T. Szirányi, S. Li, and D. Saupe. The konstanz natural video database, 2017.

[8] V. Hosu, F. Hahn, M. Jenadeleh, H. Lin, H. Men, T. Szirányi, S. Li, and D. Saupe. The konstanz natural video database (konvid-1k). In *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2017.

[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[11] M. Kumar, D. Weissenborn, and N. Kalchbrenner. Colorization transformer, 2021.

[12] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[14] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[15] . Saeed Anwar, Muhammad Tahir. Image colorization: A survey and datase. https://arxiv.org/pdf/2008.10774.

[16] P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays. Scribbler: Controlling deep image synthesis with sketch and color, 2016.

[17] M. Shariatnia. Baseline code. https://github.com/moein-shariatnia/Deep-Learning/tree/main/Image

[18] A. Unknown. Autoencoder tutorial. `http://www.eecs.qmul.ac.uk/~sgg/_ECS795P_/papers/WK07-8_PyTorch_Tutorial2.html`.

[19] C. Vondrick, A. Shrivastava, A. Fathi, S. Guadarrama, and K. Murphy. Tracking emerges by colorizing videos, 2018.

[20] T. Welsh, M. Ashikhmin, and K. Mueller. Transferring color to greyscale images. *ACM Trans. Graph.*, 21:277–280, 07 2002.

[21] S. Yoo, H. Bahng, S. Chung, J. Lee, J. Chang, and J. Choo. Coloring with limited data: Few-shot colorization via memory-augmented networks, 2019.

[22] . Yuanzheng Ci, Xinzhu Ma. User-guided deep anime line art colorization with conditional adversarial networks. https://arxiv.org/pdf/1808.03240.

[23] M. H. Zayd, N. Yudistira, and R. C. Wihandika. Image colorization using u-net with skip connections and fusion layer on landscape images, 2022.

[24] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization, 2016.