Image Colorization with GANs and Image Captioning

Ivan Villa-Renteria
Stanford University
Stanford, CA
ivillar@stanford.edu

Tom Starshak
Stanford University
Stanford, CA
starshak@stanford.edu

Ruben Rodriguez Buchillon
Stanford University
Stanford, CA
coconutruben@cs.stanford.edu

Abstract

Image colorization has gained notoriety due to its promise to colorize photos from the past, when camera technology was too underdeveloped to feature color in pictures. However, it remains a hard problem largely because it is under constrained, and hard to formulate mathematical interpretations of what a good colorization is after a certain point. There have been many approaches aiming to tackle this problem, such as text-based networks and diverse colorization networks. We propose a method which combines both by using both image captioning and GANs to colorize grayscale images. In this report we outline why our proposed method yielded poor results, why the standard metrics of the problem space - peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) - are inadequate after a certain point of colorization quality, and finish by proposing a focus on the sub-problem of color vibrancy and segmentation as a possible avenue to tackle quantifiable colorization improvements.

1. Introduction

Image colorization is a difficult problem because it is under constrained; multiple image colorizations can correspond to the same grayscale image. When an image is depleted of its color channels and mapped into a grayscale channel, there is a loss of chromatic information. Previous approaches to colorization almost always produced desaturated colorizations [15]. However, there has been an emergence of multiple deep learning approaches in computer vision to tackle this task, ranging from different strategies and architectures, such as plain networks, user-guided networks, domain-specific networks, and other methods which are covered more in detail in [1].

Fundamentally, the problem can be split into a quantifiable part and a perceptive part. While it's easy to detect very poor colorization approaches, such that yield unnatural colors, or bleed colors across edges in the image, after a certain quality, common metrics stop being informative, and people's perception of the images starts to play a larger role. This is illustrated easily in the domain of objects that can justifiably have a multitude of colors. This limitation has led to multiple papers [1][13] to use a human test to judge naturalness of color, rather than mathematical (and differentiable) approaches.

However, there are formulations that allow us to generate at least very plausible colorizations, albeit with some drawbacks. One approach of interest that has been explored is the use of GANs as a technique for image colorization via diverse colorization; the goal here is to generate different colorized images, rather than just just restore the original color. More specifically, GANs attempt to generate colorizations adversarially, wherein the Generator attempts to fool the Discriminator with generated colors, and the Discriminator tries to correctly differentiate between the ground-truth and generated colorizations.

In addition, another approach that is of interest is the use of text-based colorization; in these networks, deep learning models are able to colorize images based on text inputs, alongside image inputs. One example of this approach is given in [4], where the authors employ an existing language-agnostic architecture to provide captions as additional input.

Our approach aims to combine both GANs and captioning in order to achieve superior colorizations of grayscale images. We feed images and their captions to our model, which contains a GAN, and then output a colorization.

2. Related Work

In [4], the authors employ an architecture called FCNN which takes images as input; it is an architecture composed of eight blocks, each of which is a sequence of convolutional layers followed by batch normalization. In addition to this, the caption text t is encoded into a continuous representation $\mathbf{h}$ by using the last hidden state of a bi-directional LSTM. Let $\mathbf{Z}_n$ be the n^{th} convolutional block of FCNN.

They then compute two vector γ_n and β_n:

$$\gamma_n = \mathbf{W}_{n_\gamma}\mathbf{h},$$
$$\beta_n = \mathbf{W}_{n_\beta}\mathbf{h},$$

where $\mathbf{W}_{n_\gamma}$ and $\mathbf{W}_{n_\beta}$ are learned weight matrices. They modify FCNN by modulating the resulting feature maps of the convolution blocks by computing

$$\mathbf{Z}'_{n_{i,j}} = (1 + \gamma_n) \odot \mathbf{Z}_{n_{i,j}} + \beta_n.$$

This newly constructed architecture is denoted as FILM.

Meanwhile, in [13], the authors implement a GAN that takes 224 x 224 x 1 CIE *Lab* colorspace images as input where the only channel that is fed to the GAN is the luminance channel, and outputs a 224 x 224 x 2 image, where the two channels correspond to the predicted chrominance channel elements in the CIE *Lab* color space of the original image . More specifically, an RGB image is converted into CIE *Lab* color space, fed to the GAN to generate an output, the input and output are concatenated so that each location has all three corresponding channels, and said image is then converted back in to RGB color space. The network's Generator contains a pretrained VGG-16 model which then splits into two tracks, one which produces a class distribution vector, and another which produces chrominance information. Both branches are then fused by concatenating the output features and then a single branch of Conv-Relu layers with up-sampling in between follows, predicting the chrominance channels of the input image. The discriminator takes either a ground truth image or a reconstructed image, predicts whether it is a true image or a generated image.

The Loss function is based on discriminator and generator loss, the loss based on how the colors of the generated image differ from the ground truth, and a loss based on whether class prediction matches what VGG-16 would predict. Loss function is sum of the three terms.

The total loss is formulated as

$$\mathcal{L} = \mathcal{L}_e + \gamma_g \mathcal{L}_g + \gamma_s \mathcal{L}_s,$$

where $\mathcal{L}_e$ is the color error loss, $\mathcal{L}_g$ is the WGAN loss, and $\mathcal{L}_s$ is the class distribution loss. γ_g and γ_s are weight terms for the WGAN loss and class distribution loss, respectively.

3. Problem Statement

Our work is closely related to the problem of image colorization; take a color image and remove all color information, leaving only the color channel; how do we colorize the image such that it is as close to the original color image as possible? A formulation of this problem can be rewritten as

$$I_g = \Phi(I_{rgb}),$$

where $\Phi(\cdot)$ is a function that converts an RGB image I_{rgb} to a grayscale image I_g.

In our work we relax the problem statement above by introducing text information that accompanies the grayscale image I_g. Our proposed method receives a grayscale image I_g and a text caption t accompanying the image as inputs, and outputs a colorized image in RGB color space. Essentially, we attempt to create a mapping Ψ such that

$$\hat{I}_{rgb} = \Psi(I_g, t),$$

where $\hat{I}_{rgb}$ is a colorized image that attempts to be as close to I_{rgb} as possible. The metrics of similarity in question that our work will be using are peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM), which are discussed in further detail in Section 4.3.

4. Methods

4.1. First Baseline Approach

We built a very simple baseline to get an understanding of the dataset, the challenges in the space, and understand evaluation metrics.

The first baseline, baseline0, takes the MS-COCO [3] dataset, and retrieves a correspondence mapping from gray scale to RGB color channels. It calculates for each channel the average color that produced a given gray-scale tone. This produces a M_{Cx256} matrix where each column indexes the gray value from the pixel of the image, and each row indexes the color channel for the prediction output.
This baseline is simplistic, but allows us to get an initial understanding of issues that can arise, and how we can classify good and bad color mappings, and build out basic tooling.
This baseline limits itself to producing one color prediction (one per channel) for each pixel from the gray scale image, considering the gray tone value only. To see why this very frequently fails, consider that that standard mapping [6] to gray scale is

$$Y = 0.299R + 0.587G + 0.144B$$

For the majority gray tones, there are a large number of original 3 channel combinations that it could have originated from, whereas this approach limits itself to always choosing the same one.

4.2. Experimental Models

Our approach can be summarized as starting from the ChromaGAN implementation as a reference, and building up onto it with the aim of reaching the model architecture described in the previous section.

4.2.1 Model A

The reference ChromaGAN implementation [14], ported to tensorflow 2, and adjusted to run on a distributed TPU system. This required solving migration problems, to use tensorflow_gan [7] implementations of the losses, and reformatting the model to drop the a combined model of discriminator and generator in favor of an explicit generator and discriminator.

4.2.2 Model A.2

An iteration on Model A, that replaces the Weissman formulation of losses, with a binary cross entropy loss [8] for the discriminator and generator loss portions. It maintains the KLD loss for the class distribution, and the MSE loss (vs the ground truth color) for the generato . This implementation was used to attempt to move beyond training instability with the Model A implementation.

4.2.3 Model B

An evolution on Model A, to see the impact of captions, rather than class labels. This version takes as input captions in addition to the images. Those get embedded into a 50 dimensional caption word vector, and transformed and reshaped to the same size as an image channel (224x224), then added to the first layer of the generator (so the first layer of convolutions learns how to weigh and embed them). This formulation removes the class labels, both generation in the generator, generation of the ground truth from a pretained VGG model, and their loss contribution in the generator loss.

4.2.4 Model C

This model combines the text captioning approach detailed in [4] and the GAN with supplemental information approach from [13] fully.

We plan to combine both approaches by implementing the affine modulation computation from FILM into the convolution layers of the Generator network in ChromaGAN. Please see Section 2 for details on the transformation. This gives the model multiple entry points to weave the LSTM's output into the hidden layers, allowing it to learn at which point in the CNN funnel that information is most useful. This model uses the same loss calculation from the ChromaGAN paper which was stated above, except that the class distribution loss is replaced by the LSTM's classification loss across the entire caption.
This model again has two sub flavors, (1) whether to go through the LSTM caption generation based on the colorized ground-truth, or (2) the grayscale representation. Model C.2 being effective would show that previous formulations were not extracting enough context from the grayscale if only trained against the colors and/or class label, and that more contextual information to make better colorizations can be retrieved when training against their objectives explicitly.

This model was unfortunately not implemented.

4.3. Metrics

In addition to the mean squared error (MSE) of the colors, we also decided to use peak signal to noise ratio (PSNR) and structural similarity index measure (SSIM) as metrics to evaluate the colorization.
This is motivated in part by the metrics being ubiquitously used in the literature [1].
PSNR allows us to understand how much (potential) noise the model is adding in an attempt to get edges or individual pixels right, while SSIM helps weigh the error in a perceptual sense - weighing things differently depending on texture and luminance of an area to quantify the impact of the mistakes on perceived error.
Both are common metrics used in image compression evaluation as well [2] and one could re-frame colorization as a (poor) compression and de-compression process.

5. Dataset

We used the MS-COCO dataset [3] because it has captions. We originally tried to use ImageNet, but it is not captioned. The dataset has multiple images sizes. The initial implementation of the baseline takes random crops, while the ChromaGAN [13] implementation uses a resizing method. In the end we ran with the resizing method to keep results comparable, though Section 7.1 offers some thoughts on the potential impact this might have had especially on classification and object detection.
Notably, we do not process the images to remove the mean image initially, as we worry that might compromise the ability to pick up the color distributions, which is exactly what we care about.
Further, we note that the ChromaGAN implementation works with class labels, while MS-COCO does not provide class labels, but rather captions for the images. This is however not an issue, as ChromaGAN uses a pretained VGG-16 as the 'oracle' of correct labels, and thus the objective is for the generator pipeline to learn to *get as close to the pretrained VGG-16 outputs* as it can. This architecture tweak is what allows the model to be flexible and accept a wide variety of color image datasets.
The 2017 MS-COCO dataset provided us with 118K test images, and a validation and test set. For captions we chose one of the human captions for each image as the reference caption in a one-time pre processing to generate the vocabulary. We took the 231N formulation from Assignment 3

	exp0	exp1	exp2
MSE	444.81	454.17	473.35
SSIM	0.9318	0.9397	0.9385
PSNR	24.73	24.28	24.39

Table 1: Mean metrics for baseline experiments - (model that only returns mean colors)

PSNR	SSIM
23.39	0.902

Table 2: Mean Imagenet-trained ChromaGAN metrics

Spring 2021, and created a vocabulary with 1500 words, replacing others with an $< UNK >$ token.

6. Experimental Results

6.1. Baseline

The data from the baseline experiments can be found in Table 1. This information helps highlight that even a very naive baseline can achieve high SSIM and PSNR scores. Further discussion on the impact of those metrics can be found in Section 7.5

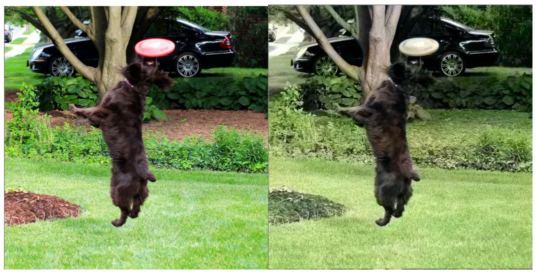
6.2. Reference ChromaGAN

The reference ChromaGAN implementation pretrained on Imagenet was used to colorize grayscale MS-COCO image in order to observe what a high quality colorization model should produce. When colorizing the MS-COCO test set, it achieved moderate PSNR and SSIM values, in line with our other results, but never achieved the PSNR values (25dB) cited in the ChromaGAN paper.

While the metrics for the Imagenet-trained model and our models that were trained from scratch on MS-COCO are similar, the colorization itself seemed much better for the Imagenet-trained model, which influenced our belief that PSNR and SSIM, while wildly used, are inadequate to compare colorization schemes after a certain quality bar. This seems to start in the 0.9 range for SSIM, and 22+dB for PSNR, at which point other formulations need to be used and developed to distinguish colorization quality.

A ChromaGAN trained from scratch on MS-COCO is able to extract some color information, green for plants, and blue for sky are common results, but in general the output was muted.

The ChromaGAN paper claimed to gain better colorization results when they included semantic information, that is they made a class prediction on the image. We found that leaving off the class prediction head of the GAN yielded higher PSNR and SSIM values than including it.



(a) ground truth (b) model output

Figure 1: Example image from ChromaGAN pretrained on Imagenet. Note the model was not able to distinguish between mulch and grass, and the desaturated (faded) Frisbee color.

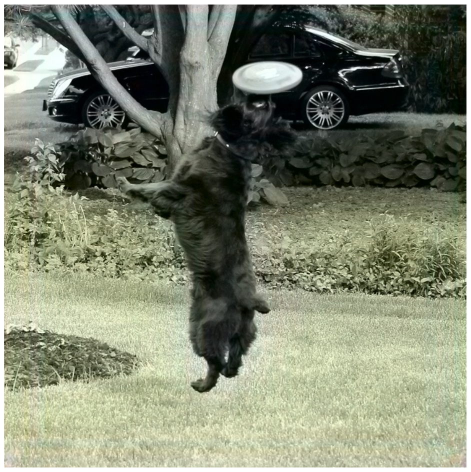


Figure 2: Example colorization output from ChromaGAN trained from scratch on MS-COCO

There is some indication that continued training would yield better results; PSNR and SSIM values increased after each epoch, up to 10 epochs, but computational constraints precluded training to a sufficient level.

6.3. Our Models

Fundamentally, we were unable to reach the performance of the ChromaGAN paper with our formulation (model A) and distributed training. The original formulation and code runs for 8h/epoch on a standard GPU on the MS-COCO 2017 dataset.

As mentioned in Section 4.2 our reformulation made experimentation possible due to TPU hardware. This made running an epoch in 15 minutes possible, however, the results are poor as shown in Section 6.3. The Model A, Model A.2, and Model B formulations all produced very unstable training, quickly reducing the loss for both generator and discriminator, before diverging, and often not converg-

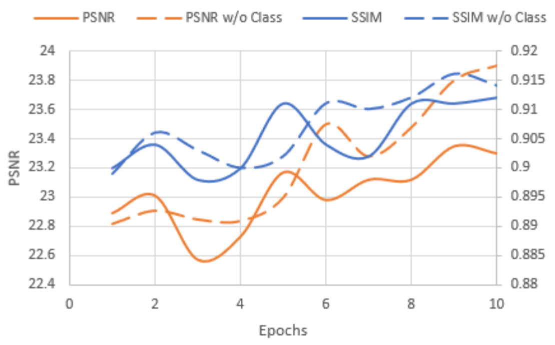


Figure 3: Training metrics for the reference ChromaGAN with and without classification head up to 10 Epochs.

name	model	epochs	best epoch
exp3	Model A	1	n/a
exp4	model A	5	1
exp5	model A.2	5	4
exp6	model B	10	8

Table 3: experiments on own models overview

	exp3	exp4	exp5	exp6	exp5.d
MSE	0.0081	0.0078	0.0067	0.0088	2.788
SSIM	0.919	0.921	0.925	0.913	0.527
PSNR	23.28	22.21	22.57	23.32	0.349

Table 4: experiments on own models key metrics

ing again. Model B produced the most stable formulation, though it often required 10 epochs to get to meaningfully small loss formulations (0.7 for the discriminator, 10.3 for the generator). As shown in Section 6.3 even those formulations do not guarantee meaningful colorizations.

The reported results in Table 4 are on the test set, with the weights from the best epoch (as described in Table 3). Please also note that while the MSE below is with respect to a floating point scale (values are taken to be between 0 and 1.0 in RGB), in the baseline0 they are with respect to 8bit 255 values, accounting for the large amplitude difference. Note that exp5.d is the same experiment as exp6, but with the worst epoch (and losses) chosen rather than the best. This is to highlight the divergence once the discriminator becomes too good at distinguishing outputs, and the generator loss starts continuously increasing. (as with Figure 6)

Our formulation produced mostly one of two categories of outputs - conservative, barely colorized images (Figures 4, 5) or hyper-saturated images (once the generator started diverging, Figure 6). Notably, the metrics (PSNR, SSIM) are reasonably good on the barely colorized images. This is something we observed in our toy baseline as well. So it seems that our formulations all run into the same issue - the generator is quickly punished when trying to add color, which in turn leads to almost non-existent colorization in stable cases.

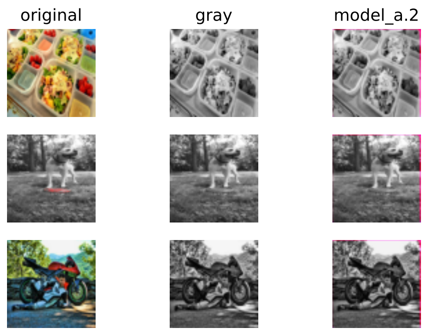


Figure 4: Model A.2 validation set images. L: original, M: gray scale, R: predicted. PSNR(22.1), SSIM (0.923) (bottom image)

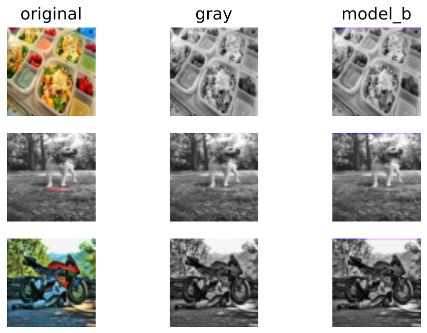


Figure 5: Model B validation set images. L: original, M: gray scale, R: predicted. PSNR(22.3), SSIM (0.911) (bottom image)

As Figure 5, and the data in Table 4 show, these results held true even when changing the formulation of the GAN, or changing the supplemental data being fed in from a class label, to an embedded caption. The lack of meaningful colorizations meant that we could not explore the impact of model B properly, or saw any benefit in implementing model C flavors, as it would likely have run into the same issues.

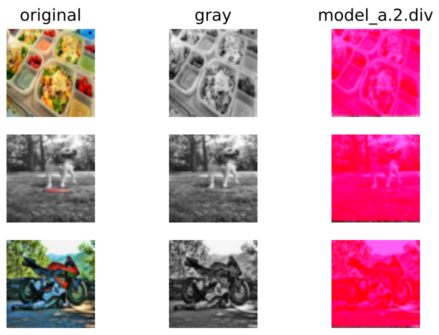


Figure 6: Model A.2 validation set images, generator had diverged. L: original, M: gray scale, R: predicted. PSNR(1.32), SSIM (0.43) (bottom image)

6.4. Hyper parameters

In addition to the model changes, we used two Adam optimizers, with a learning rate of $2 * 10^{-5}$ and a beta1 of 0.9 (ChromaGAN uses 0.5). The change is motivated by the distributed system learning in parallel and then fusing the results together again. Some experiments were also conducted with a learning-rate of $2 * 10^{-5} * 8$ (the number of parallel batches), though this showed similar results. Batch size was 256, with 32 images per TPUv2 core (8 cores per system). All experiments on our own models ran for 5 epochs, on the entire training set. The degree of instability in the training, and lack of meaningful results indicate that at least on our own distributed formulation, the hyper-parameters around updates (learning rate, betas, steps before applying gradients) need to be tuned much more, though the experimental overhead here made that difficult (see Section 7.2). Additionally, the results on the reference ChromaGAN contrasting pretained ImageNet weights with Xavier initialized weights show that potentially our distributed formulation would have produced stable, more colorful results, if run for many more epochs (e.g. 100 epochs, to 150 epochs, to roughly cover the same number of images looked at as ImageNet with 10 - 15 epochs).

7. Discussion

7.1. Dataset impact

While we had few meaningful results with our own model formulations, the baselines we ran on the reference implementation of ChromaGAN with MS-COCO showed worse results and colorizations after 10 epochs than the paper. These results fall below the quality (PSNR) of the reported results in the paper, both the ChromaGAN and ChromaGAN without classes formulations. The size of MS-COCO ($\approx \frac{1}{10}$ of the ImageNet images used) might be partially to blame here. There is some indication that a combination of dataset size and variance plays a key role as using the pretained ImageNet weights produced better outputs than scratch training on MS-COCO, even with the same epoch length.
Additionally, we observe that resizing on MS-COCO might lead to significant distortions, and might make learnt shapes and patterns for colorization less meaningful. While we did not get to experiment on this, a subsequent step would be to contrast the model with selective crops rather than resized images.

7.2. TPU pipeline optimization

While the speedups on the TPU pipeline where impressive, the model still spent 73% of the time idle, waiting for the next batch of data. The lowest hanging improvement is optimizing the TPU pipeline and moving towards potentially a larger batch size on TPUv3. This would allow for simpler iteration on hyper-parameters. Notably, the data retrieval overhead on the cloud TPU was so large, that using a small dataset (1/10 or 1/5 of the original dataset) lead to insignificant speedups (only 2x faster epochs), and thus provided a barrier to quick experimentation on hyper parameters.

7.3. GAN approach tradeoffs

We note that the failure of most of our experiments hints at the known issue [9] of GANs being problematic to train and how easy it is for the discriminator to quickly become so good, that the generator can no longer meaningfully learn. This raises the question of whether GANs for this space are the right approach to begin with. Putting aside training stability, it's not obvious that the large amounts of freedom in the GAN formulation are necessary for plausible colorizations, nor that the discriminator is discriminating in meaningful ways the perceived color quality. While especially the discriminator could also be adjusted to account for segments and saturation (as explored in Section 7.6 for the generator), it might be more stable, and quicker to iterate on a simpler, deep CNN network, than forcing this problem statement into the GAN model.

7.4. Impact of input formulation

The ChromaGAN paper formulates the problem as a 1 channel to predict 2 additional channels problem. Luminance is fed in, and the color channels are retrieved. However, in practice, and with our implementation too, the actual supply to the generator is a 3 channel image with 3 greyscale channels stacked. This is to accommodate the pretained VGG formulation for the classification learning. We observed that on our 'good' (read - non-failed) results, an average of 82% of pixels are the same across all 3 channels (in RGB space) - thus producing the grey results we see. We suspect that potentially the initial feed of 3 identical channels might contributing to the GAN in most of our cases producing outputs that barely color. In the results we saw that the MSE is very low for a grey image compared to the correctly colored ground-truth, and grows very large when the generator tries to color (and fails, as our data on the generator divergence shows). Thus this formulation might be giving the generator too strong a prior hint on what the other two channels might need to look like, and might be negatively impacting learning.

7.5. SSIM and PSNR metrics quality

Furthermore, we note that in our experiments, both with our toy baseline, our models (model A, model A.2, model B), as well as the pertained, reference implementation of ChromaGAN, PSNR (and SSIM, though not used in the ChromaGAN paper) are an imperfect metric after a point. Namely, once SSIM reaches the high 90s, and PSNR reaches the 22+dB, we notice that they fail to correlate strongly with perceived (our own perception) image colorization quality (or ground truth MSE for that matter). So we conclude that that while they can be helpful metrics, they usage should be constraint to differentiating between models that produce 'credible' results and models that fail to colorize properly (e.g. our model A once the generator starts diverging). They fail to be a highly informative metric once that threshold is passed, and motivate the formulation of more expressive metrics.

7.6. Vivid colors as a loss, explicit segmentation

In our experiments, and in the literature[13][1], we notice that after the point of credible colorization, one of the main obstacle is capturing realistic 'vivid' colors, rare colors, and capturing colors for objects that plausibly can have multiple color distributions. When analyzing the images, and their respective metrics, we notice that images tend to suffer when generic pieces take over the majority of the image - e.g. beaches, lawns, fields, skies. Note Figures 1, 2 outputs in particular. In those cases, a failure to saturate or even color components is easily averaged out in losses and metrics. This leads us to think that the next step should be exploring explicit segmentation in the pipeline, and saturation of the colors in the segments. By forcing the model to segment, classify, and then colorize objects, we could explicitly punish a faded colorization of an object on a naturally unsaturated background - e.g. the tail-lights on the car, or the Frisbee in those examples. This could be formulated as another loss component, or replace the current formulation in ChromaGAN of the MSE loss with respect to the original image. Naturally, segmenting and learning small features like the tail lights come with its own computation and resolution concerns in an already computationally taxing pipeline, which might further motivate moving towards a simpler formulation of the colorization generation backbone.

8. Conclusion

In conclusion, we were unable to reproduce the quality of the ChromaGAN paper in our implementation, and also suspect that the dataset, and length of training might have played a role in even the reference implementation of ChromaGAN on our dataset performing worse than the paper's reported results. There are a multitude of areas where the system might be improved and experimented with, though we suspect that the most promising approach to the interesting problem of color vibrancy and bleeding of surrounding colors would be to take a simpler model, and build a pipeline that explicitly encodes a notion of color vibrancy and explicitly segments the images.

9. Contributions

Name	Contributions
ivillar@	Captioning understanding
	Model C design
starshak@	ChromaGAN baseline local and GPU results
	ChromaGAN PSNR, SSIM experiments
	ChromaGAN 5 epoch MS-COCO experiment
rubenr2@	baseline0 experiments
	Model A(.2), Model B experiments
	TPU acceleration and cloud experimentation
combined	dataset and pre-processing pipeline
	MS-COCO 2017 vocab generation
	report and presentation

10. Acknowledgements

We leveraged Colab and Google Cloud for experimentation extensively. Additionally, we leveraged the ChromaGAN GitHub [14] reference to conduct experiments on the reference, and serve as a starting point for our own variations on it. We also used Google's tutorials on TPUs and distributed learning [12], on a basic GAN implementation [10], and on image processing on tensorflow [11] as starting point for our implementations. We also leveraged the open implementations on tensorflow gan [7] for the Wasserstein loss formulations. Lastly, we leveraged the GitHub reference for learning to color from language [4] [5] to understand how to implement the captioning layers.

References

[1] Saeed Anwar, Muhammad Tahir, Chongyi Li, Ajmal Mian, Fahad Shahbaz Khan, and Abdul Wahab Muzaffar. Image colorization: A survey and dataset. *arXiv preprint arXiv:2008.10774*, 2020. 1, 3, 7

[2] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010. 3

[3] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. 2, 3

[4] Varun Manjunatha, Mohit Iyyer, Jordan Boyd-Graber, and Larry Davis. Learning to color from language. *arXiv preprint arXiv:1804.06026*, 2018. 1, 3, 7

[5] Varun Manjunatha, Mohit Iyyer, Jordan Boyd-Graber, and Larry Davis. Learning to Color from Language - Github. https://github.com/superhans/colorfromlanguage, 2018. [Online; accessed 21-May-2021]. 7

[6] opencv. Miscellaneous Image Transformations. https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#, 2019. [Online; accessed 10-May-2021]. 2

[7] Joel Shot. TensorFlow GAN support package. https://github.com/tensorflow/gan, 2020. [Online; accessed 25-May-2021]. 3, 7

[8] Google TensorFlow. TensorFlow Binary Cross Entropy Loss implementation. https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy, 2021. [Online; accessed 20-May-2021]. 3

[9] Google TensorFlow. TensorFlow GAN problems. https://developers.google.com/machine-learning/gan/problems, 2021. [Online; accessed 28-May-2021]. 6

[10] Google TensorFlow. TensorFlow GAN tutorial. https://www.tensorflow.org/tutorials/generative/dcgan, 2021. [Online; accessed 26-May-2021]. 7

[11] Google TensorFlow. TensorFlow Image tutorial. https://www.tensorflow.org/tutorials/images/classification, 2021. [Online; accessed 20-May-2021]. 7

[12] Google TensorFlow. TensorFlow TPU tutorial. https://www.tensorflow.org/guide/tpu, 2021. [Online; accessed 24-May-2021]. 7

[13] Patricia Vitoria, Lara Raad, and Coloma Ballester. Chromagan: An adversarial approach for picture colorization. *CoRR*, abs/1907.09837, 2019. 1, 2, 3, 7

[14] Patricia Vitoria, Lara Raad, and Coloma Ballester. ChromaGAN github implementation. https://github.com/pvitoria/ChromaGAN, 2019. [Online; accessed 21-May-2021]. 3, 7

[15] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016. 1