Exploring Convolutional Neural Networks for Automatic Image Colorization

Mahesh Agrawal*
Stanford University
mahesha@stanford.edu

Kartik Sawhney*
Stanford University
kartiks2@stanford.edu

Abstract

This paper addresses the problem of generating a plausible colored photograph given a grayscale image. Previous approaches to solve this problem have either relied on significant human input or resulted in desaturated colorizations. Inspired by [1], we present a fully automatic approach that uses deep neural networks to color images. We explore several architectures/models, approaches and loss functions to understand the best practices to obtain the most aesthetically pleasing colored images. We train a convolutional neural network, and observe that certain loss functions perform better than others due to their inherent properties. In particular, we find that our classification-based model that uses cross-entropy loss seems to perform the best. We also discuss other state-of-the-art approaches to solve this problem including conditional generative adversarial networks [2]. Finally, we provide a qualitative and quantitative analysis of our results, concluding with avenues for future research.

1. Introduction

1.1. Background

Recently, there has been a lot of interest in colorization of grayscale images without human intervention [1, 2, 3]. This problem is not only interesting from an aesthetics perspective, but also has several important applications, including video restoration and image enhancement for better interpretability. Automatic colorization also allows us to appreciate old photographs, helping us to understand and connect with history in a way that we never could before. Colorization can also be considered as a powerful pretext task for self-supervised feature learning, acting as a cross-channel encoder [1]. Figure 1 shows a sample input and output image for this task. What makes this task especially interesting is the fact that there are multiple 'correct' colorized outputs for the grayscale input image, rendering the problem under constrained and multimodal in nature.

1.2. Convolutional Neural Networks

Humans have the capability of associating grayscale images with certain colors by associating the semantic content in the image with our memory, but humans may take a long time to fill in colors. However, convolutional neural networks (CNN) are very successful at discerning and learning features quickly. In the past years, CNNs have emerged as the dominant technology for a host of image related tasks such as classification, labelling, image generation and style transfer, achieving error rates below 5% on the ImageNet challenge. Given these characteristics, they are clearly a natural choice to explore for the auto colorization task at hand.

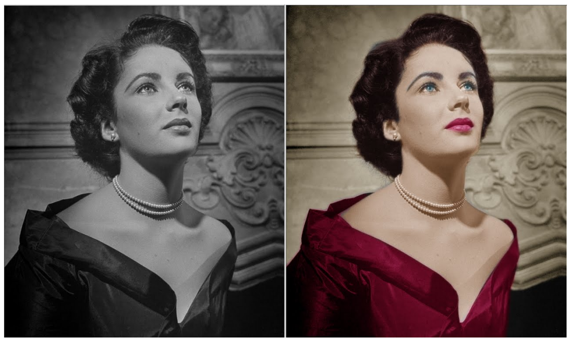


Figure 1: Sample input grayscale image (left) and sample output colorized image (right)

1.3. Model Input/Output

Our approach consists of training a CNN with colored images allowing it to learn color features without any human supervision. More specifically, we convert the RGB image to the CIE Lab color space because the Lab color space provides a Lightness channel that we can use as the grayscale input to our model. It is also known to provide a wider color gamut which should enable more realistic image colorization. We pass in the L (lightness) channel as the input to our CNN which then outputs the A and B channels (which represent colors for the grayscale image) and then concatenate the input and output to generate the full 3 channel image. We then convert this back into the RGB color space. The rest of the paper discusses our methods, experiments and observations in more detail.

2. Related Work

2.1. Human-Assisted Colorization Techniques

For a long time, colorization has been a semi-automated process, relying on hints from the user. Levin et al. [7], for instance, proposed that neighboring pixels in space with similar intensities should have similar colors. Thus, in his work, hints are provided as rough inaccurate "scribbles" on a grayscale image, and the algorithm is able to generate high-quality colorizations based on these hints. Several others have improved this algorithm further, including Huang et al. [8] (by addressing color-bleeding issues) and Qu et al. [9] (by modifying the cost function to account for color continuity over similar textures in addition to similar intensities). Welsh et al. [10], on the other hand, use a full-color example image of similar composition, thereby reducing human dependence even more.

2.2. Convolutional Neural Networks

Dahl [4] was one of the first people to propose a convolutional neural network pre-trained for image classification to generate full color channels for the input image. Trained on the ImageNet database with a L2 loss function (defined later) on the chrominance values, his approach achieved mixed results. While the predicted colors were almost always reasonable, images were desaturated and even brownish, owing to the "averaging effect" of the L2 loss. More recent work in this space has approached this problem either as regression onto a continuous color space [3, 4, 11] or classification of quantized color values [12]. There has also been a lot of innovation in loss functions to capture the multimodal nature of the problem. For instance, Hwang et al. [1] use a loss function tailored to the problem. They predict a distribution of possible colors for each pixel, and re-weight the loss at training time to emphasize rare colors ("class rebalancing"). Larsson et al. [13] and Iizuka et al. [14] have developed similar systems with subtle changes in the architecture and methods. While Hwang et al. [1] use a classification loss, with rebalanced rare classes, Larsson et al. use an un-rebalanced classification loss, and Iizuka et al. use a regression loss. Larsson et al. use hyper-columns [15] on a VGG network [16], Iizuka et al. use a two-stream architecture in which they fuse global and local features. The three papers all produce competitive results with pros and cons for each method.

2.3. Generative Adversarial Networks

GANs serve as an excellent way to not only learn the mappings between the inputs and outputs, but also the loss. This allows a model to be used in a variety of settings. A number of previous works have used GANs in a conditional setting as well, conditioning them on discrete labels [17] and text [18]. Isola et al. [19] extend them for image-to-image translation across several domains such as synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images. More recently, cycle GANs are being used for colorization [20] which eliminate the need for input-output pairs.

3. Approach

Since this problem is an under constrained multimodal problem with no right or wrong answer (i.e. an object can take multiple colorizations), coming up with a loss function that enhances the visual appeal of the final images is an interesting challenge. We thus experiment with different loss functions for our CNN model, including L2 and smooth L1 (which use a regression based approach) and the cross-entropy loss (which uses a classification based approach) to determine the best fit for the task. We also experiment with other CNN techniques such as dropout, different activation functions and batch normalization to observe their effects.

Given our experience with loss functions and the inherent challenges with each of them, as well as CNN based models' tendency to produce sepia-tone colors for objects with ambiguous colors [4], we also explore other approaches that could help solve this problem. One such approach is conditional adversarial generative network, as first introduced by Goodfellow et al. in 2014 [5]. This approach works similar to the CNN approach in that the generative net takes in the L channel as the input, generating the color channels A and B. The discriminator, on the other hand, takes in a generated colorization or a true colorization ("ground truth") and predicts if the input was a true color image. Here, the competition between the discriminator and generator in maximizing the accuracy of the predictions and minimizing the accuracy of the discriminator respectively results in natural loss functions for backpropagation that do not rely on Euclidean distance measures. Our hope is that this leads to brighter and more vibrant colors, since the focus is on generating more realistic colors than those that are close to the training set.

To evaluate our performance with these architectures and loss functions, we use a combination of qualitative and quantitative metrics, including colorization Turing test, "closeness" metrics (using L2 distances), classification accuracy and adversarial success (AdverSuc) [6].

4. Methods

4.1. Baseline Model

Our baseline model involved training a simple CNN model (3 layers of CNN followed by a Fully Connected layer) on 2000 images from CIFAR-10. We used the L2 loss function as the objective, ReLu [22] as the activation function and batch normalization [24] layers to accelerate convergence and improve accuracy. Figure 2 shows some sample outputs from this model. As is evident from these images, the baseline model does not colorize the images well producing very faint/dull colors and muted tonality.

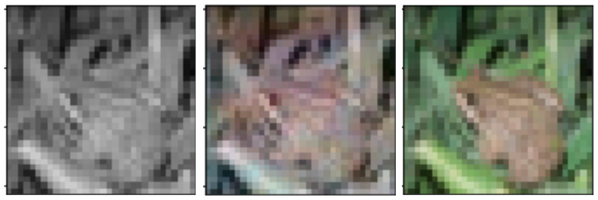


Figure 2: input grayscale image (left), predicted colorized image (middle) and ground truth (right) from our baseline model

4.2. Final CNN Model

4.2.1. Dataset

While we used CIFAR-10 for our baseline, we decided to train our final model on ImageNet since the dataset contains more diverse images, allowing our model to better learn different colorizations. Figure 3 shows a few images from this dataset. Due to memory constraints, we used down sampled ImageNet images with 64x64 resolution. Yet, this was superior to the CIFAR-10 32x32 images and the increased number of input pixels allowed the model to learn more granular features/colorizations. 10,000 images were used to train the model, while the validation and test sets consisted of 1000 images each. Unlike several other works that focus on a specific domain for colorization, we did not explicitly limit our training data to a particular domain. Instead, we purposely sampled data from different synsets to ensure we had a good representation of different image settings, so that we could analyze the categories of images that performed best. As expected, outdoor images and natural landscapes performed well, while indoor images were harder to colorize. Further, we tried subtracting the mean pixel value as a preprocessing step, but this led to the image losing its color information, and thus didn't prove useful for this task with our model setting.

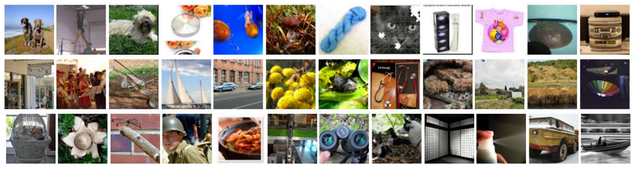


Figure 3: Sample 64x64 images from ImageNet dataset

4.2.2. Flow

Regression-based model: During training, our program reads images of 64×64×3 in the RGB colorspace. We convert these into CIE Lab colorspace where L represents luminance, while A and B channels encode the color information. The CNN receives the L channel as the input (dimension 64×64×1), and outputs the A and B channels (64x64x2). The L channel is then concatenated with the A and B channels and converted back into the RGB colorspace to give the colored output. During testing, we adopt a similar approach where the input is the grayscale image (the L channel) and the outputs are the A and B channels (color information).

Classification-based model: Here, the model input remains the same. However, during training we bin the pixels in the A and B channels for the ground truth image using a bin() functions that takes in a pixel value and determines which bin it belongs to. Each bin represents a 'class' thus reframing this as a classification problem. We use 10 bins such that each bin corresponds to a range of 20 pixels (the A/B channels range from -100 to +100 each). For each pixel, the model then outputs a score for each class that is interpreted as an unnormalized log probability and then it chooses the class with the highest probability. We then convert the bin back to a pixel value using an unbin() function that outputs the mean value of the pixels contained in that bin. The final output that we get thus still continues to be 64×64×2, which when concatenated with the L channel, gives us the colored image.

4.2.3. Model architecture

Our convolutional neural net architecture closely resembles that of [1]. Figure 4 shows our full architecture. We use an entirely convolutional model architecture without any pooling or upscaling layers to prevent loss of spatial information and the only down sampling is done through the convolutional layers themselves. Due to computational constraints, we had to cut down on the number of convolutional layers, but we tried to preserve the

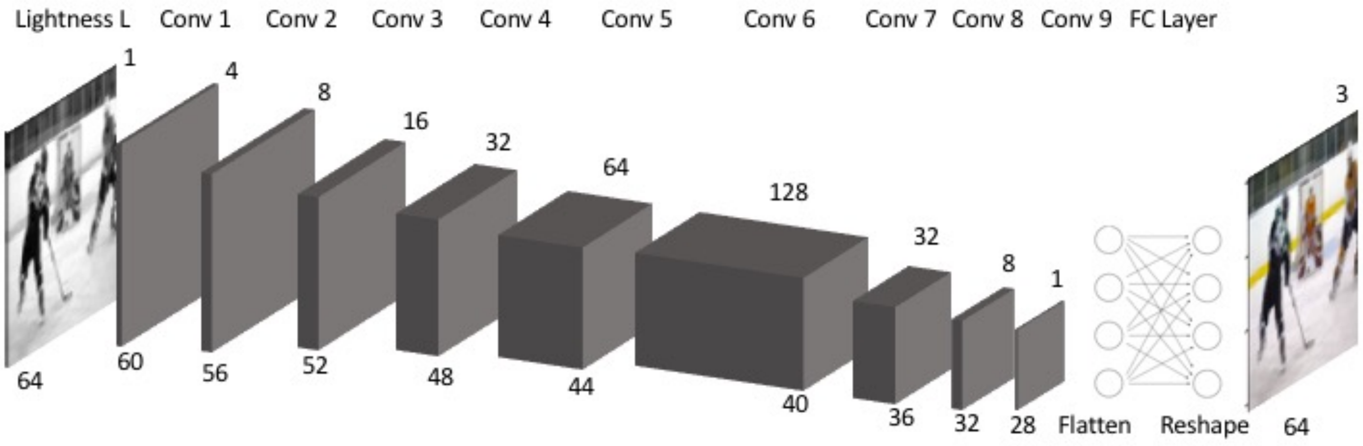


Figure 4: Our CNN Architecture. We feed a 64x64x1 grayscale image through 9 Conv layers, where each Conv layer (kernel size of 5 and stride 1) is followed by a ReLu and BatchNorm layer. The numbers above the Conv layers represent the output channels while the numbers below represent the output spatial resolution of each Conv layer. After the final Conv layer, we flatten the output and feed it to a Fully Connected layer whose output we reshape back to 64x64x2 and concatenate with the input 64x64x1 to get the predicted colorized image

overall architecture in the paper [1]. Our model consists of a series of 9 2D convolutional layers, where each Convolution layer is followed by a ReLu non-linearity [22] and batch normalization layer [24] and after all the Convolution layers we have a fully connected layer that outputs the final model predictions. We tried different settings for our model such as eliminating and changing the position of a few ReLu/Batchnorm layers but found our current model to work best in practice.

4.2.4. Objective Functions

As pointed out in [1, 13, 14], one of the most important challenges in auto-colorization is an objective function that accounts for the multimodal nature of the problem. To investigate this further, we experimented with several loss functions.

L2 loss: The L2 can be mathematically defined as:

$$\frac{1}{n}\sum_{n}(x_i - y_i)^2$$

$where\ x_i\ represents\ the\ predicted\ pixel\ value\ and\ y_i\ the\ ground\ truth\ pixel\ value\ and\ n\ is\ the\ total\ number\ of\ pixels\ across\ all\ input\ images\ to\ the\ model$

Based on the results from the baseline model, we were expecting the L2 loss to give us under-saturated images with muted colors due to its averaging effect. This is because the L2 loss, given various colorizations of an object that can take multiple colors (e.g. a car that can be blue, green or red), will choose the mean colorization to reduce the model loss on the input grayscale image. This predicted mean pixel value causes the output images to have muted colors and appear sepia toned. Thus, while the L2 loss might seem well suited for this task, it does not work well for objects that could take multiple colors and our results in Figure 6 confirm this finding.

Smooth L1 loss: Introduced by Girshick [21], smooth L1 loss can be mathematically given as:

$$\frac{1}{n}\sum_{n}\begin{pmatrix} 0.5 \times (x_i - y_i)^2\ if\ |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5\ otherwise \end{pmatrix}$$

This loss function is primarily used to guard against outliers. However, given our dataset and the problem, outliers is less of a concern. We are more interested in the model capturing finer details, which becomes challenging with the introduction of the L1 loss. This loss penalizes outliers less and this could be bad for the colorization task as pixels that are completely inaccurately predicted (say predicted pixel value 5 instead of ground truth value 100) would not be penalized as harshly and hence would lead to wrong colorings for the images. Therefore, we observe that this performs worse than our L2 objective function and these results can be seen in Figure 6 again.

Cross-entropy loss: This loss is used with our classification approach. Mathematically, cross-entropy loss may be given by:

$$\frac{1}{n}\sum_{n}(-f_{yi} + log\sum_{j} e^{fj})$$

$where\ f_{yi}\ represents\ the\ correct\ class\ score\ and\ j\ is\ the\ number\ of\ classes\ and\ f_j\ is\ the\ corresponding\ class\ score$

This loss function gives us a probability distribution for the class that each pixel can belong to, helping us select the best class for each pixel. We get the most vivid, realistic and statured images using this method (as can be seen in Figure 6), since we are not trying to minimize the difference between the generated image and the ground truth as in the L2 loss, but are instead working with classes that offer the model more flexibility. However, the number of classes is an important and sensitive hyper parameter here. If the number of classes we generate with the bin() function is too large, then there is a high likelihood of the model making an inaccurate prediction as it becomes tougher for the model to choose the correct class amongst the increased class set. An example can help to understand this: let's say Class 1 corresponds to pixel value 5 and Class 10 corresponds to pixel value 100, and the model selects Class 10 over Class 1 by even a small margin, the model will output Class 10 for that pixel, producing a completely garbage image colorization. However, if the number of classes is too small such that each class spans many pixel values, then the model might classify correctly, but the original pixel value might be significantly off from the bin/class value (which is the mean of the pixel values in the bin) again leading to desaturated images. We avoid these issues by treating the number of bins as a hyper parameter and find that 10 classes worked best for our model and dataset.

4.2.5. Activation function

We use the rectified linear unit (ReLu) [22] as the nonlinearity that follows each of our convolutional layers. Mathematically, ReLu may be defined as:

$$f(x) = \max(0, x)$$

In line with [23], we found that ReLu helped accelerate the training convergence. It is also extremely simple to compute this function. One drawback for this function is that the model parameters could be updated in such a way that the function's active region may end up in the zero gradient region which causes a gradient of 0 to backpropagate through the network, effectively 'killing' neurons and preventing the network from training well. However, we did not run into this challenge in practice, and so used ReLu as the activation function for our model.

4.2.6. Batch normalization

Developed by Ioffe and Szegedy [24], batch normalization helps stabilize the learning process by normalizing each of the activation units in the model to have zero mean and unit variance, ensuring that the training is robust against bad weight initialization. It has also been found to improve gradient flow through the network. Introducing a batchnorm layer after every ReLu layer helped improve our training convergence and accuracy and thus was deployed in the final model.

4.2.7. Dropout

Dropout is an extremely effective regularization technique introduced by Srivastava et al. [25]. While training, dropout is implemented by only keeping a neuron active with some probability p (0.4 in our case) or setting it to zero otherwise. During training, Dropout can be interpreted as sampling a Neural Network within the full Neural Network, and only updating the parameters of the sampled network based on the input data. We introduced dropout right after the batchnorm layers in our network, but observed bad results. This is because, given our problem and the dataset size, overfitting is not a challenge. In fact, we want the model to learn as many diverse colorizations as possible. Dropout hinders this process by preventing the model from learning 'too much'. Figure 6 shows the images we obtained with dropout where one can see how Dropout causes even the training images to remain under colored and 'regularized' even after training is complete. Thus, we left Dropout out of the final model. This seems to match the related work we have surveyed as well [1, 3, 4] where Dropout does not seem to be used for this task.

4.2.8. Hyperparameter Tuning

We used the Adam optimization method (default values of β1=0.9 and β2=0.999) [26] with a learning rate of 1e-3 which was decayed to 1e-4 when the loss started to plateau (which usually happened around epoch 150). We trained for 200 epochs using batch sizes of 250 (varied slightly for different models) on a NVIDIA Tesla K80 GPU. Another important hyperparameter was the number of bins in our classification model, where we used 10 bins. All of these hyperparameter values were found after doing a random search over the hyperparameter space. More specifically, we started off with a small training set of 100 images, and used random search to narrow in for a range for these parameters. For instance, for the learning rate, we ran multiple trials of training to see the learning rate that would yield the fastest convergence over a fixed number of iterations. Within the set of learning rates sampled on a logarithmic scale, we found that a learning rate of 1e-3 achieved one of the largest per-iteration decreases in training loss as well as the lowest training loss of the learning rates sampled. Using this information, we then tried to find the learning rate in this range that would work well for the training set as a whole, and settled in on 1e-3.

We also experimented with different update rules, including Adam [26] and Nesterov momentum [27]. The Adam update rule with default β1 and β2 produced slightly faster convergence than Nesterov momentum, so we used it with the default hyperparameters for our final model. We also tried multiple batch sizes, but found fastest and stable convergence with a batch size of 250. Finally, we played around with different kernel sizes and stride values for the Conv layers but found that too small a kernel size did not capture enough information about neighboring pixels while too large a kernel size proved to be too destructive and hence settled on a kernel size of 5 as the optimal value for this task. Figure 5 below shows the Training Loss Convergence for our Classification model using the cross-entropy loss. One can see the learning rate decay from 1e-3 to 1e-4 around iteration 3000 where there is a sudden downward plunge in the loss after a period of relative flatness. The convergence was mostly smooth except we did notice that occasionally the loss would suddenly spike upwards before continuing to converge.

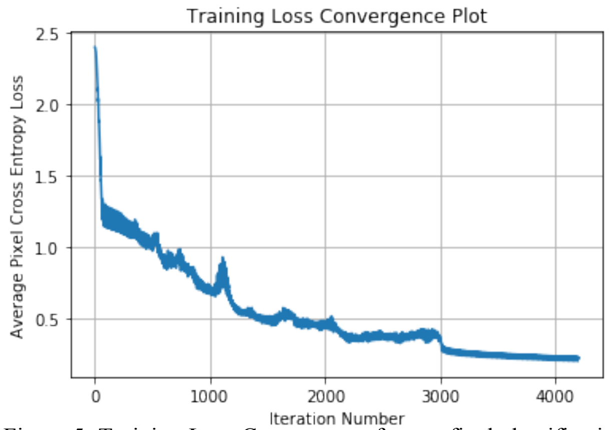


Figure 5: Training Loss Convergence for our final classification model using the cross-entropy loss

4.3. Conditional generative adversarial networks

Given the problem of selecting a suitable loss function and the challenges with each of them, we instead implemented a conditional generative adversarial network with the architecture closely resembling that of [19], using [29] and [30] as references (implementation available at https://github.com/sawhney-kartik/gan-colorization). However, we believe that our model can benefit considerably from more fine tuning and training on larger datasets, so we do not discuss it here. We discuss the implementation details and architecture in Appendix 1.

5. Results

5.1. Comparing Different Loss Functions

Figure 6 shows some of the images (both training and test images) generated with these loss functions. As mentioned before, it is clear from these results that the classification

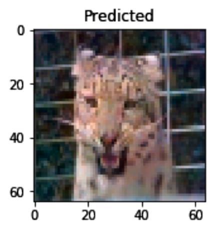
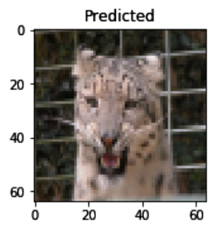
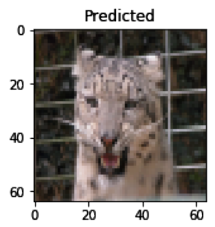
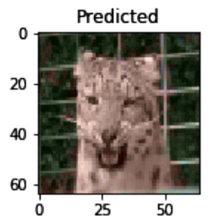
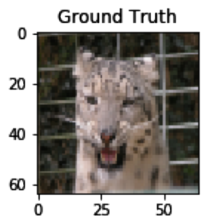
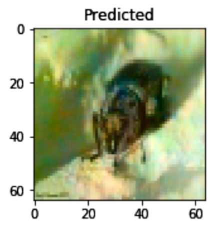
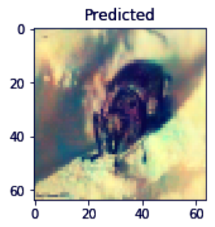
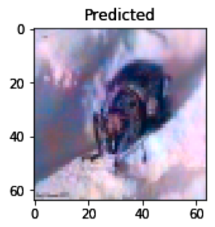
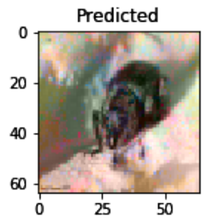
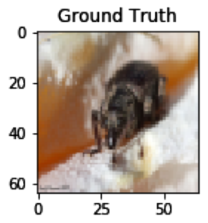
Model	L2 w Dropout	Smooth L1 Loss	L2 Loss	Cross Entropy Loss	Ground Truth
Training Image	 *Sepia Tone on Face*	 *Backdrop not green*	 *Backdrop not green*	 *Backdrop more green than Ground Truth*	
Test Image	 *Too 'regularized'*	 *Sepia Tone*	 *Desaturated and Pale*	 *Closest to Truth*	

Figure 6: Image Results for Training and Test Time. One can see that Dropout causes images to be 'regularized' and hence under colored. Smooth L1 and L2 loss produce sepia toned and muted colors in the images. Cross Entropy Loss produced images closest to ground truth and those that are realistically colored.

approach with the cross-entropy loss function performs the best, since we work with classes of pixels rather than individual pixel values. In this task, the goal is to get an image that can best capture the general colorization in the object and not necessarily produce colorization closest to the ground truth and the classification approach does this best. This can be seen in the Training Image in Figure 6, where the classification model managed to color the grass in the background green even though this was not the case in the ground truth. Nevertheless, L2 loss performs decently, except for the desaturation in images due to averaging of colors. L2 w Dropout and Smooth L1 do not perform as well since they fail to capture the finer details in the image.

5.2. Colorization Turing test

We conducted a colorization Turing test with six Stanford students where we showed them random images and asked them to predict if these images were original images or created by our model. We used adversarial success (AdverSuc) as introduced in [6], which is the fraction of instances in which the students were fooled, as our quantitative metric. Since our model is trained with a relatively small dataset and is much smaller than state-of-the-art models (due to memory constraints), students were easily able to distinguish the original and our generated images. Their observations regarding the aesthetic appeal of images generated using different approaches were consistent with our observations above.

5.3. "Closeness" quantitative metrics

Even though defining a closeness quantitative metric is not very helpful for this problem, we use it as a rough indicator of accuracy. We calculate the Frobenius Norm of the difference matrix between the predicted image and ground truth image which gives us the pixel wise Euclidean distance between the predicted and ground truth. This can give us some sense of how 'close' the predicted colorization is to the actual ground truth colorization. The issue is that this does not account for the multimodal nature of the problem: it is very possible for this difference to be high for a particular predicted image and for this image to still be more realistic than another predicted image for which the difference is lower.

Mathematically, this can be calculated as:

$$\|D\|_F = \sqrt{\sum d^2}$$

$$\textit{where } D \textit{ is the difference matrix computed as the predicted image matrix subtracted from the ground truth image and } d \textit{ is an element of } D$$

Model	L2 Dropout	Smooth L1	L2	Cross Entropy
Euclidean Loss on Test Set	18.67	13.45	15.65	11.27

Table1: Closeness quantitative metric results

We see that, as discussed, this metric is just a proxy for the effectiveness of the model at colorization and is far from the complete truth. While Cross Entropy comes out ahead in this metric indicating it performed the best, based on this metric it would seem that Smooth L1 does better than L2 even though this is not true when the images are assessed on a qualitative basis, which is arguably a more important criterion for this task. Therefore, it is a somewhat useful tool but less important than the qualitative measures mentioned.

5.4. Classification accuracy

For classification, we also use the percentage of binned pixel values that match between the generated image and actual image across both channels A and B as a proxy for the accuracy. Mathematically, this is given by:

$$Accuracy = \frac{1}{N}\sum_{N} 1(bin(Predicted) = bin(Actual))$$

where N is the total number of pixels and bin is our binning functions that translates a pixel value to the corresponding class it belongs to. This metric was only relevant for the Classification model and we show the results on the test set for different values of the number of bins in Table 2. We can see from this table, that increasing the number of bins makes the model more sensitive and hence more likely to output the wrong class since it must now choose between more classes. Even though a greater number of bins is useful as it improves the pixel value accuracy within the bin it would take many more training examples, probably an order of magnitude more, at the very least, to train the model to output the correct class amongst an increased class set. As mentioned previously, this measure is also not 'bullet-proof' in its indication of a successful colorization, given the multi-modal nature of the problem, but it is seems to work better that the Euclidean loss mentioned in Table 1 since it is working on a pixel class accuracy as opposed to raw pixel value accuracy.

Number of Bins	**10**	**30**	**50**
Classification Accuracy (%)	41.5	27.8	18.9

Table 2: Number of Bins in Classification accuracy results

5.5. Best Model

In general, we find that the cross-entropy loss produces full, bright colorized images that don't appear under saturated or in sepia tone. The model is able to learn a collection of higher-order abstract features that help it color different segments of the image differently. This is directly the result of using a classification approach with a CNN based architecture as CNNs can learn different colors, patterns, edges and features within an image given their spatial filters and sliding windows. In Figure 7, we see that the model managed to detect the table, the objects on the table, the person and the background wall in the first image individually and then applied realistic colorizations to each of these semantic segments. In the second image, we see that the model learnt to color trees/grass in green and the ground/earth in brown based on the input images it has learnt from.

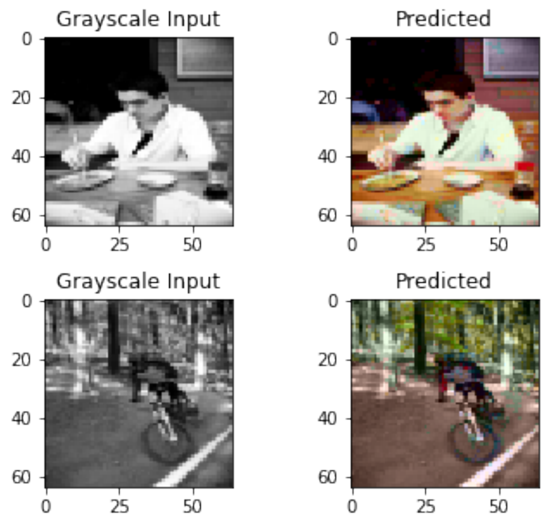


Figure 7: Sample test results from our final Cross Entropy Loss Model

There were nonetheless issues with the cross-entropy loss as well. One particular problem with most of the losses but the cross-entropy loss in particular was it sometimes would produce blotches of incoherent color in the images. In Figure 8 below, one can see that it has interspersed the predicted images with red blotches all over – this is due to the pixel wise prediction of the model. Even though CNNs should learn spatial information regarding neighboring pixels, since the final output is different for every pixel, the model sometimes makes independent and divergent decisions for neighboring pixels, causing the image to appear blotched. This is a well-known problem and is also discussed in [1]. Our model also seems to generally fail at coloring indoor scenes given the multitude of colorings that indoor scenes can take on which probably confuses the model. To better color indoor scenes, the model would need to see several more different indoor settings so that it can learn some sensible colorings.

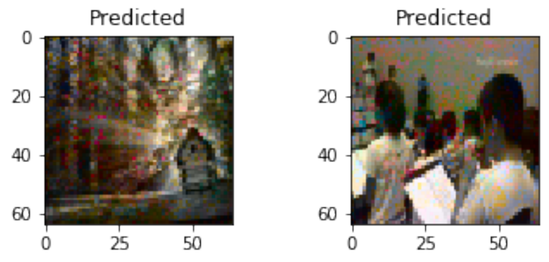


Figure 8: Sample failure test cases from our final Cross Entropy Loss Model

6. Conclusion and Future Work

Based on our experiments, it is clear that convolutional neural networks are very promising for autocolorization. However, the objective function needs to be carefully designed and tailored to this problem to account for the multimodal nature of the problem. For instance, while L2 loss can work well, it often leads to desaturated images due to averaging out the possible colors. Similarly, cross-entropy loss works much better, but the number of bins is a very sensitive hyperparameter that needs to be carefully determined and the model also needs to see several training images before it can coherently color certain settings.

As future extensions of this project, we would like to modify the classification approach to account for the concentration of pixels in certain bins. In Figure 9 below, we see that most of the pixel values, for both the A (blue histogram) and B (orange histogram) color channel were concentrated in a narrow band around 0.

We used bins of equal-width throughout the -100 to +100 range. However, to improve the model accuracy, it might make more sense to have smaller bins where the histogram density is higher (such as around 0) to more precisely capture the mean pixel value within each bin and larger bins in the rarer parts of the distribution. This might lead to less accurate colorizations on rare pixel value but more accurate colorizations on common pixel values that dominate most of the image and hence would likely improve the overall image quality.

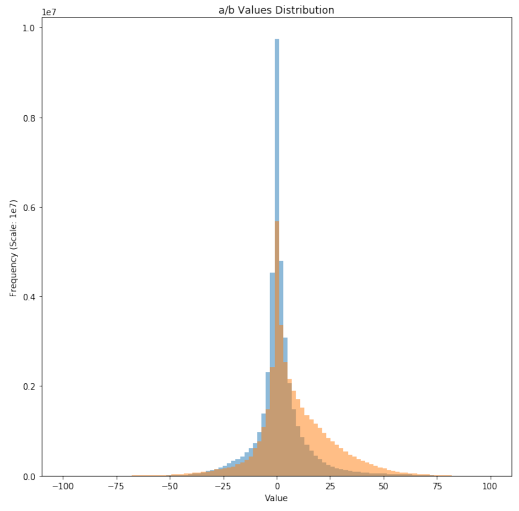


Figure 9: Histogram of A/B color channel values

Another approach we could try is to bin() the A, B value together as a pair. We currently bin A and B values independently and output different classes for each channel per pixel. However, there should exist some correlation between the A and B channel value for each pixel, since they represent colorings for the same pixel and hence the semantic content. Thus, it would make it easier for the model to learn to jointly predict a class for the A, B value pair that we could then un-bin using a 2-D un-binning matrix that computes the individual A, B pixel value given the joint distribution of A and B. If A and B capture enough correlation information, then one might expect this binning to produce A/B channel values that are more coordinated and hence render more realistic colorings.

We would also like to further develop our CGan model to compare it to our CNN findings, and explore CycleGans. Finally, extending this to gifs and videos might be an interesting avenue using a combination of these architectures and recurrent neural networks [28].

Appendix 1

Conditional Generative Adversarial Networks

We discuss our conditional generative adversarial network here. The CGan learns the mapping from an input image (the grayscale image, i.e. L channel) to the output image (the colored image, i.e. the A and B channels) without an explicit loss function for the conversion because the discriminator provides a loss function for the generator. The generator learns how to colorize a grayscale image, while the discriminator tries to predict if the generated image is created by the generator or is the ground truth.

Generator

The generator takes the grayscale image as the input (the L channel). It reduces the image dimensions using a series of encoders (convolutional, activation and batchnorm layers) into a much smaller representation. By compressing the image this way, the hope is that we will have a higher level representation of the data after the final encode layer. The decode layers do the opposite (deconvolution + activation function), and reverse the action of the encoder layers, generating the colored image. Further, as in [19], we use a "U-Net" instead of a strict encoder-decoder model. Here, we have "skip connections" directly connecting encoder layers to decoder layers. The skip connections give the network the option of bypassing the encoding/decoding part if it doesn't have a use for it. Also, instead of feeding in noise directly, we use dropout units for a few layers in the middle.

Discriminator

The discriminator takes in two images as the input-the grayscale image and a colored image. It then determines if the colored image is the target image (ground truth) or the output generated by the generator. Its structure is similar to the encoder section of the generator (a series of convolutional, activation and batchnorm layers), but the output is a 30x30 image where each pixel value (0 to 1) represents how believable the corresponding section of the image is (the authors call this approach "patchGAN").

Training and fine tuning

We use the same data as in the CNN model for our GAN model. However, we observed that this model requires much finer hyperparameter tuning than the CNN model. In fact, our implementation is unable to learn well so far, and is work in progress.

References

1. Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." European Conference on Computer Vision. Springer International Publishing, 2016.
2. Liang, Xiangguo, et al. "Deep patch-wise colorization model for grayscale images." SIGGRAPH ASIA 2016 Technical Briefs. ACM, 2016.
3. Cheng, Zezhou, Qingxiong Yang, and Bin Sheng. "Deep colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.
4. Dahl, Ryan. "Automatic colorization." (2016).
5. Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.
6. Li, Jiwei, et al. "Adversarial learning for neural dialogue generation." arXiv preprint arXiv:1701.06547 (2017).
7. Levin, Anat, Dani Lischinski, and Yair Weiss. "Colorization using optimization." ACM Transactions on Graphics (ToG). Vol. 23. No. 3. ACM, 2004.
8. Huang, Yi-Chin, et al. "An adaptive edge detection based colorization algorithm and its applications." Proceedings of the 13th annual ACM international conference on Multimedia. ACM, 2005.
9. Qu, Yingge, Tien-Tsin Wong, and Pheng-Ann Heng. "Manga colorization." ACM Transactions on Graphics (TOG). Vol. 25. No. 3. ACM, 2006.
10. Welsh, Tomihisa, Michael Ashikhmin, and Klaus Mueller. "Transferring color to greyscale images." ACM Transactions on Graphics (TOG). Vol. 21. No. 3. ACM, 2002.
11. Deshpande, Aditya, Jason Rock, and David Forsyth. "Learning large-scale automatic image colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.
12. Charpiat, Guillaume, Matthias Hofmann, and Bernhard Schölkopf. "Automatic image colorization via multimodal predictions." Computer Vision–ECCV 2008 (2008): 126-139.
13. Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Learning representations for automatic colorization." European Conference on Computer Vision. Springer International Publishing, 2016.
14. Iizuka, Satoshi, Edgar Simo-Serra, and Hiroshi Ishikawa. "Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification." ACM Transactions on Graphics (TOG) 35.4 (2016): 110.
15. Hariharan, Bharath, et al. "Hypercolumns for object segmentation and fine-grained localization." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.
16. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
17. Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).
18. Reed, Scott, et al. "Generative adversarial text to image synthesis." Proceedings of The 33rd International Conference on Machine Learning. Vol. 3. 2016.
19. Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." arXiv preprint arXiv:1611.07004 (2016).
20. Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." arXiv preprint arXiv:1703.10593 (2017).
21. Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE International Conference on Computer Vision. 2015.
22. Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." Proceedings of the 27th international conference on machine learning (ICML-10). 2010.
23. Xu, Bing, et al. "Empirical evaluation of rectified activations in convolutional network." arXiv preprint arXiv:1505.00853 (2015).
24. Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).
25. Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.
26. Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
27. Dozat, Timothy. "Incorporating nesterov momentum into adam." (2016).
28. Medsker, L. R., and L. C. Jain. "Recurrent neural networks." Design and Applications 5 (2001).
29. Stanford's CS231n, Winter 2017. Assignment3. (http://cs231n.stanford.edu/assignments/2017/spring1617_assignment3_v3.zip)
30. Tensorflow port of Image-to-Image Translation with Conditional Adversarial Nets. (https://github.com/affinelayer/pix2pix-tensorflow)