Image Colorization

Benjamin Brown, Michael Byiringiro, Meg Larkin, Jonathan McDevitt, Nadia Saunders
University of Michigan

April 30, 2024

Abstract

Image colorization, the process of adding realistic colors to grayscale images, presents significant challenges in the field of computer vision. This paper details the efforts to develop a model capable of colorizing grayscale images by interpreting the task through classification, regression, and Generative adversarial networks. We propose a novel approach where each pixel in a grayscale image is classified into one of several color categories, similar to methods used in semantic segmentation. Our training utilizes the COCO dataset, with computational resources spanning local machines, and Google Colab, aiming to complete training within one day on a CPU. The efficacy of our model is assessed through user studies, where participants evaluate the colorization results either by comparison or by direct rating. This approach not only offers a practical solution to image colorization but also contributes to the broader understanding of applying classification techniques to complex image processing tasks.

1 Introduction

In the digital age, the visual presentation of images plays a critical role in fields ranging from digital media to historical archives. A particularly compelling aspect of image processing is the colorization of grayscale images, which can enhance the utility and aesthetic appeal of monochrome photographs. This process not only revitalizes historical visuals but also aids in various applications such as medical imaging and artistic reinterpretations.

Our research focused on developing a model that can automatically add color to grayscale images, hoping to make these colorizations appear as realistic as possible. The concept of image colorization is not new; however, our approach reinterprets this task as a classification problem, where each pixel is assigned a color label from a predefined set. This idea was inspired by techniques used in semantic segmentation, a method explored in our class curriculum, suggesting that similar methodologies could be effectively adapted for the task of colorization. We intend to discover the effectiveness of this model by comparing its results against models that interpret this task as a regression problem and as an adversarial-network problem, which are more standard approaches.

The significance of our work lies in its potential to automate a process that traditionally requires extensive manual effort and artistic input, which makes it more accessible and efficient. Additionally, by framing image colorization as a classification problem, we introduce a structured, replicable approach that can be further refined and expanded by the research community. Our method's viability is underpinned by using established datasets like COCO for training, and we leverage both traditional and advanced computational resources to achieve our training goals. The practical implications of our model extend beyond academic interest, offering enhancements in various professional and recreational domains where image interpretation plays a crucial role.

2 Background

2.1 Related Work

Recent advancements in deep learning have significantly contributed to the progress in the field of image colorization. This task is notably challenging due to the high level of ambiguity in colorizing grayscale images since multiple colorizations may be plausible for a single image. In this regard, two related works that we used on this project "Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2" by Baldassarre, Morìn, and Rodés-Guirao and "Colorful Image Colorization" by Zhang, Isola, and Efros, which both propose innovative methods leveraging convolutional neural networks for the task of image colorization. We also leveraged the work of Isola et al. (2018) in "Image-to-Image Translation with Conditional Adversarial Networks". They explore image colorization through a different approach: generative adversarial networks.

In the first paper, Baldassarre, Morìn, and Rodés-Guirao introduce a novel approach by combining the strengths of deep CNNs with the high-level features extracted from the Inception-ResNet-v2 architecture. Their method is distinctive because it uses an encoder-decoder structure that is capable of handling images of varying sizes and aspect ratios. This flexibility is crucial for practical applications where images may not conform to standard dimensions. Moreover, their system includes a user study to evaluate the public acceptance of the colorized images, emphasizing the practical relevance of their research.

On the other hand, the second paper by Zhang, Isola, and Efros (2016) presents a different approach by framing the colorization problem as a classification task over quantized color spaces. Their method, which also employs deep learning, uniquely focuses on the multimodality of the colorization task — acknowledging that one grayscale pixel can correspond to multiple color predictions. They utilize a class-rebalancing technique at the training stage to encourage the model to use more diverse colors, which often results in more vibrant and realistic outputs.

Isola et al. take a completely different approach than the previous two papers. They focus on leveraging generative adversarial models (GANs) — a combination of two models that fight each other for performance gains. They condition their generative model on black and white images, the "L" in LAB color-space, and try to generate the color channels, "A" and "B". They then use a CNN discriminator to weed out generated images.

All methods underscore the potential of deep learning in enhancing the colorization of grayscale images, each introducing significant innovations in terms of network architecture and problem framing. These studies not only push forward the boundary of automatic image colorization but also contribute to broader applications such as historical photo restoration, artistic colorization, and improving visual media for entertainment and education.

2.2 Technical Background

For readers to fully appreciate our research on image colorization using a classification approach, a solid technical background in several key areas is essential. Firstly, a basic understanding of image processing and computer vision is necessary, including knowledge of how digital images are represented and the distinctions between grayscale and color images. Familiarity with common computer vision tasks like image segmentation and object recognition, which share conceptual similarities with image colorization, is also beneficial.

Moreover, a grasp of general machine learning concepts, particularly supervised learning, as our method involves training a model on labeled data. A deeper understanding of deep learning techniques, especially Convolutional Neural Networks (CNNs) used widely in image-related tasks, will aid in comprehending our classification-based approach. Additionally, familiarity with semantic segmentation is important, as it involves classifying each pixel in an image into a category, a concept closely related to how we treat colorization. Finally, an understanding of evaluation metrics and user studies is essential, as these methods involve human participants assessing the model's output, which helps in evaluating the success of image colorization techniques through both technical metrics and subjective measures. This comprehensive

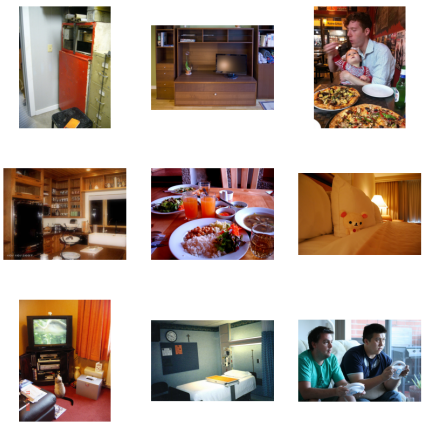


Figure 1: Images from the COCO dataset

technical background will enable readers to fully engage with and understand the innovations and challenges presented in our research.

3 Methodology

3.1 The COCO Dataset

The dataset we chose to use in training is the COCO image dataset (Lin et al., 2015). COCO is a large-scale object detection, segmentation, and captioning dataset. It contains over 300,000 images; Figure 1 shows some images found within the dataset.

3.2 Classification Model

The experimental model we are interested in is a classification model: instead of training a model to predict the continuous color values of each pixel in a greyscale image, we instead defined 256 discrete color classes, and trained the model to predict which of these classes each pixel belongs to.

Preface

This section involves many image-like (3D) matrices. To be clear, in the subsections that follow, H means the images' height, W means their width, and C means their channel count. Their shapes will be presented "CxHxW", and all H and W values are 256 due to resizing (described below). However, we will still say "H" and "W" for clarity's sake.

Training Data Preprocessing

We sampled 10,000 images from the COCO dataset, then resized them to a height and width of 256. We assigned 8,000 of these images to the training dataset, and the other 2,000 to the validation dataset.

We then defined the 256 color classes. In LAB-space, the color information of a pixel is stored in its A and B channels, as a value in the range $[0, 255]$ for each channel. To create classes, we reduced this range to 16 distinct (evenly-spaced) values for each of these channels: $C_{16} = \{8, 24, 40, ..., 248\}$. As each of these 2 channels has 16 possible C_{16} values, there are $16^2 = 256$ color classes.

We then collected the ground-truth classes of each pixel in each image in the training/validation data. For each pixel, we rounded each of its A and B channel values to the nearest values in C_{16}, then assigned it to the 1 out of 256 classes corresponding to its combination of these C_{16} values. For each image, we created a ground-truth file storing the corresponding class for each pixel in it. For a given image, call this file I for future reference; its shape is 1xHxW.

Model Architecture

Next, we constructed our classification model. This model takes a 1-channel greyscale image as input (of shape 1xHxW), and produce a 256-channel output image (of shape 256xHxW). The 256 channels in this output correspond to the predicted probability distribution of a given pixel belonging to each of the 256 color classes.

We used PyTorch and a subsection of the Resnet18 model's architecture/weights for this (He et al., 2015). The full architecture is displayed in Figure 2; note that we added final Softmax layer, so that the 256 channel values of each pixel summed up to 1, as a probability distribution should.

Training the Model

We trained this model using several PyTorch objects, including the Dataset, Dataloader, and Adam (an op-

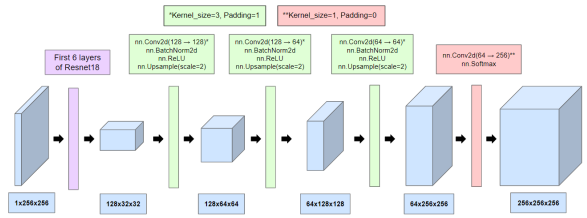


Figure 2: The classifier model architecture

timizer) objects, as well as boilerplate code by Berkay Mayali (see the references section). Our custom hyperparameters were an epoch count of 50, a training batch size of 8, and an Adam learning rate of 0.01; all others were their PyTorch defaults.

We also used a custom cross-entropy loss function. Again, for a given image, the channels in the model output are the classification probability distribution, and each pixel's ground truth class is stored in I. Using this 256xHxW model output and I, we can compute a standard cross-entropy loss.

However, we encountered a problem with this: roughly 34% of the pixels in the entire 10,000 image dataset belonged to just 1 of the 256 color classes (the $A = B = 4$ class), which resulted in the model quickly learning to predict every pixel as likely being in this class. To counteract this, we weighted each pixel in the loss function with a value inversely proportional to how common its ground truth class is. The weights were defined as follows:

$$w_c = \frac{1}{\sqrt{n_c + 10^5}} \tag{1}$$

where w_c is the weight of class c, and n_c is the number of pixels belonging to that class in the entire dataset. With these weights, the more common classes still have slightly more influence on the loss computation, but this influence is less extreme. The specifics of this formula (the square root and $+10^5$) were found via trial and error.

In mathematical terms, this custom cross-entropy loss function for a single image's model output of size 256xHxW = CxHxW is defined as follows:

$$L = -\sum_{i=0}^{H}\sum_{j=0}^{W}\sum_{k=0}^{C}[[k == I_{ij}]]w_{I_{ij}} \ln\left(P_{ijk}\right) \tag{2}$$

P_{ijk} is the model's output at channel k, height i, width j; I_{ij} is the ground truth class of the pixel at height i, width j; $w_{I_{ij}}$ is its corresponding weight; $[[k == I_{ij}]]$ is 1 if channel k matches I_{ij} and 0 otherwise (this is the ground truth "probability distribution").

Producing Model Outputs

For a given greyscale image, we take the channel-wise argmax of the image's 256xHxW model output to get a 1xHxW matrix, representing the predicted color class of each pixel in the image. We then assign each pixel the A and B values of its corresponding class, and combine this with the input greyscale image to get a full LAB-space image.

3.3 Regression Model

In the second iteration of our model, we kept a relatively similar model architecture and instead made alterations to data preprocessing based off of previous work. As mentioned earlier, 10,000 images are sampled from the COCO dataset, 8,000 of which are used in training. Using work by Berkay Mayali, we constructed a pytorch dataloader that converts the images to the LAB color space, normalized the L, a, and b channels to [-1, 1] and converts them to tensors.

Model Architecture

Although the architecture is somewhat similar, there are a few key differences. The first model employs a pre-trained ResNet-18 model for feature extraction but incorporates a classification layer followed by softmax activation to predict pixel-wise color categories, offering a classification-based approach. However this model utilizes a pre-trained ResNet-34 model for feature extraction and upsampling layers to directly predict the 'a' and 'b' channels of the LAB color space, aiming for regression-based colorization. The architecture is inspired by work from Rich Zhang and William Francis. The model was trained on 50 epochs, with a batch size of 16 and a learning rate of 0.001. The figure below depicts the data pipeline.

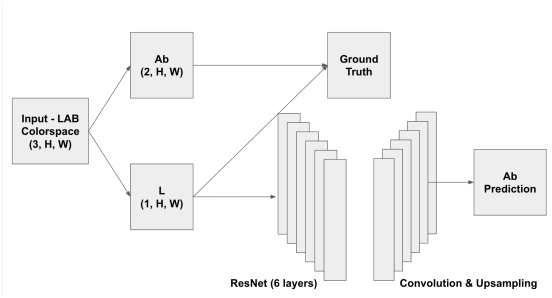


Figure 3: Regression Network

3.4 Adversarial Network

Along with our regression and classification modeling efforts, we developed a model leveraging adversarial loss. Again, we sampled 10,000 imags from the COCO dataset, 8,000 of which were used in training. We leveraged the same dataloader described in section 3.3, converting images to the LAB color space, normalized the L, a, and b channels to [-1, 1] and converts them to tensors. The distinction here lies in the image size; for efficiency and compute concerns with this pipeline we reduced our image size to 64x64. The model stands as an exploratory effort and a comparison metric for our regression and classification models. The majority of the model architecture is inspired by or directly taken from the work of Moein Shariatnia (2020), with several tweaks for our specific pipeline.

Model Architecture

Our pipeline consists of a 6 layer U-Net generative model, inspired by the larger 8 layer U-Net model we worked with in Homework 5. We condition this model on gray-scale images, then use the patch discriminator described in "Image-to-image translation with conditional adversarial networks" (Isola et al., 2018), and implemented in the work of Shariatnia (2020). This discriminator consists of 4 convolutional layers, each followed by batch normalization and a leaky ReLU, and a final convolutional layer. Likewise, we applied a L1 loss to our generator, training with a batch size of 16 over 50 epochs. We initialized weights by sampling from the normal distribution with $\mu = 0$, $\sigma = 0.02$, and used a learning rate of 2×10^{-4}, as Shariatnia does in their work. All code for this section can be found under ImageColorization_UNET.ipynb at our GitHub repo (listed in the appendix). Again, please note the comments indicating any code sourced from other authors.

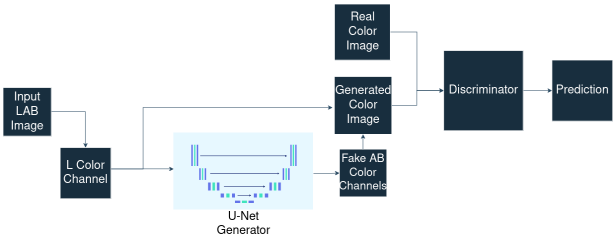


Figure 4: Adversarial Network

4 Results

4.1 Classification Network Results

The quantitative training results are plotted below. At each epoch, we computed the loss for the training set, and the loss, PSNR score, and accuracy score for the entire validation set. Loss was computed using Formula 2, PSNR was computed using its standard formula, and the accuracy score was the proportion of the model's color classifications that were correct.

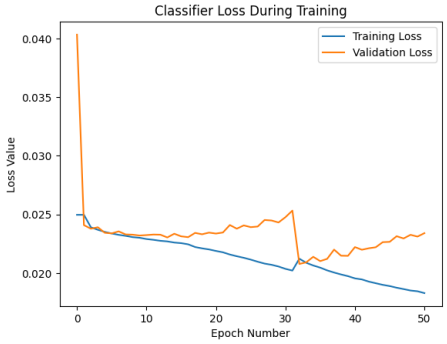


Figure 5: Classifier loss during training

There are two remarkable phenomena in Figures 5 and 6. For one, the training loss spikes up and the

Figure 6: Classifier scores during training

validation loss plummets at Epoch 32, likely indicating some error. Google Colab, the service used to train the models, crashed at this epoch. We restored the model and resumed training, but this anomaly quickly followed. The only reasonable cause of an error like this seems to be the training and validation sets being reshuffled, but this is likely not possible: the random partitioning of the sets is consistently seeded, and if this **did** occur, then the validation accuracy and PSNR scores would increase dramatically, but they did not (see Figure 6). So, this anomaly will remain unexplained and a topic for another report.

For two, the validation performance across all metrics dramatically improves at the first epoch, but then consistently worsens later on as the training loss improves. This suggests that the model begins overfitting to the training data almost immediately, indicating that a different optimizer, a less complex model, or a stronger regularization would have improved the model. However, it is worth noting that this worsening validation performance is not **entirely** the result of overfitting: by inspection, we found that the Epoch 1 model predicted almost all of the pixels as the most common color class (an issue discussed earlier), which is not the goal of this model. So, despite the validation scores worsening, the model was still improving at **actually coloring** the images, so **some** worsening of the validation score may have actually been a beneficial change, strangely. However, the validation score worsening for as many epochs as it did still suggests that there was significant overfitting occurring.

The final model yielded a validation accuracy of 24.65% and PSNR score of 25.31, indicating a moderate/decent performance. Some examples of the final model's outputs from validation images are in Figure 7 below; the top output is randomly selected, while the bottom three were hand-picked outputs that look most accurate compared to the ground truth.

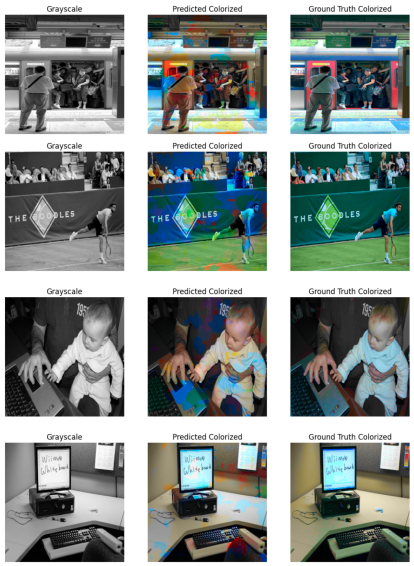
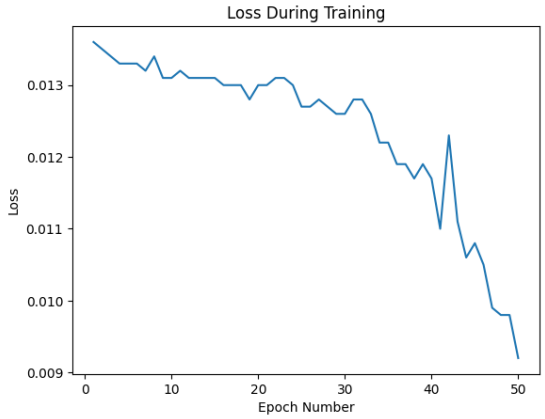


Figure 7: Sample classifier outputs

As shown by the hand-picked outputs, the model is capable of detecting and reasonably coloring several objects, including a human face, a tennis court, a desk, and a computer screen. However, these images, and most especially the randomly-chosen output, also show the model outputting strange, off-color artifacts throughout the image. This makes sense, due to the model design: since this is a classification model, the model learns no similarity between any two colors: for example, black and green are equally "similar" to light green, in the model's interpretation of the image. So, when it doesn't have high confidence about what color class a pixel belongs to, it can't learn to pick a "similar" color to the ground truth; it will likely pick a completely random color class, resulting in strangely-colored artifacts.

4.2 Regression Network Results

The images above show a couple of examples of grayscale images and their corresponding colorized images. We calculated the average PSNR to evaluate our models. The Regression network scored 25.94 dB which indicates it has fair/moderate performance. This is consistent with what we expected since the images tend to skew to the warmer side and have a hard time capturing deeper colors like green and blue. The graph below shows the loss over 50 epochs.



4.3 Adversarial Network Results

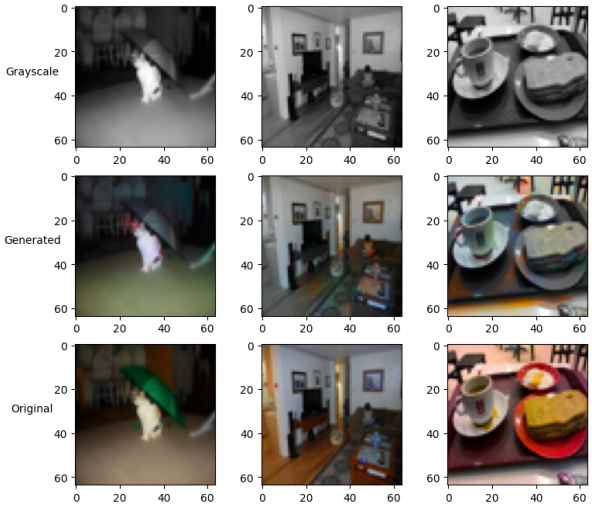


Figure 8: Sample Generated Images

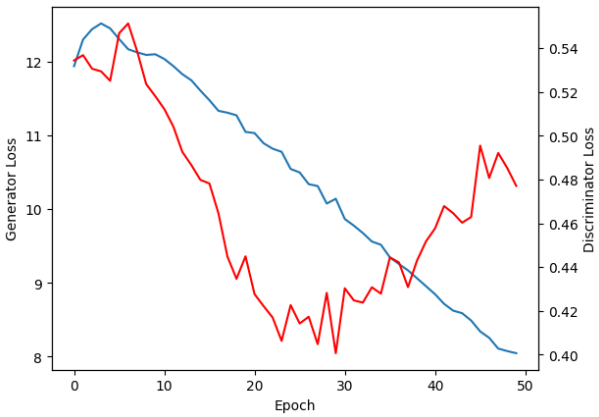


Figure 9: Generative and Adversarial Loss
Blue representing generator loss, red representing discriminator loss

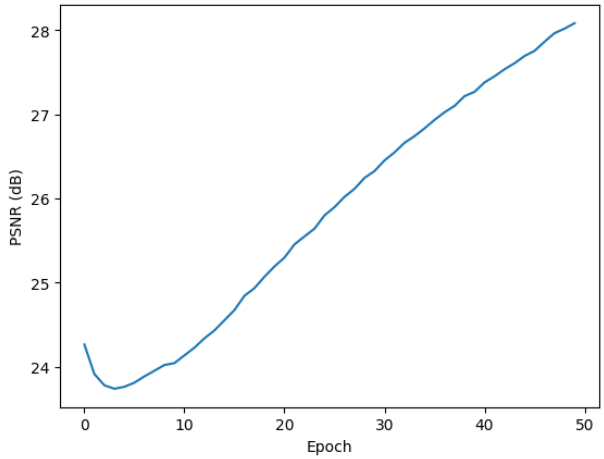


Figure 10: PSNR Values Over Epoch

Above, we have some visual representations of the training process for our generative adversarial network. It is interesting to note the shape of our PSNR curve, as well as our sample images. Clearly, our PSNR results increased over time, whereas our generated sample images are visually acceptable. This is supported by a small-scale user study we conducted. Each member of our team looked over a sample of 50 generated and ground truth images from the validation set, as well as the progression in generated image quality over the 50 training epochs. The general consensus on our team is that generated image quality did improve over training. This speaks to the inaccuracy of PSNR as a metric for image colorization, as it is possible for a model to generate a realistic colorization that doesn't match the original image. Our training loss further supports this. As we train further, our generator loss continues to decrease, where as our discriminator loss follows a parabola shape. Logically, this could indicates that our generator eventually becomes good enough to fool our discriminator.

5 Conclusion

To summarize the results, we see similar behaviors between the regression and adversarial models, while the classification model has distinct strengths and weaknesses in comparison to these two.

The regression and adversarial models generally always output the most common colors found in the training dataset: warm colors, generally brown and grey. This results in their outputs being smoothly-colored and correct when the image contains the most common training-set colors, meaning they almost always look mostly realistic as a result. However, they struggle to predict less common colors when they're present.

In contrast, the classification model outputs a wide variety of colors. This results in its outputs having a higher chance of correctly predicting varying/less common colors when they're present, but also when they are NOT present, which often makes the output look somewhat randomly-colored and unrealistic. In addition, this model's inability to account for similarity between colors causes its outputs to contain bizarrely-colored artifacts.

To conclude, the regression and adversarial models are better for outputting realistic-looking, smoothly-colored images (with the downside of most of them being monochrome), while the classifier model is better for outputting diverse-color images (with the downside of most of them being less realistic and containing oddly-colored artifacts).

It's important to consider how these models could have been improved if we had more time to work on them. Regarding the classification model, we could first better tune the hyperparameters to prevent overfitting. We could also rerun the training process to avoid whatever caused the Epoch 32 loss anomaly discussed earlier; we were unable to do this for this report due to Colab GPU usage limitations. Finally, we could also modify the model so that when it doesn't have high confidence in any color class for a pixel, it switches to some other algorithm that DOES understand "color similarity," and uses that to color the pixel instead. This could reduce the oddly-colored artifacts we see in the output images.

The regression model was mostly limited due to training time. The model took almost 2 and a half hours to train, making debugging difficult under time contraints. We did experiment with adding more layers after resnet, which did improve colorization with blues and greens (these results can be found in the

appendix), but made a lot of the other images worse. Even so, it is reasonable to think that more layers and longer training time would improve our results. Ideally, there is a lot more our team would like to do to improve the GAN pipeline and make it more unique. As stated in section 3.4, much of our architecture and code, outside of the generative model, is taken verbatim from other research on the subject of image colorization. Due to time and compute constraints, as well as the instability of generative adversarial networks, we weren't able to vary our architecture or hyper parameters drastically from previous work. Furthermore, we were forced to reduce the scale of our architecture and input image size when compared to the state of the art. As a next step for this effort, we would like to expand the size of our U-Net architecture, and use a larger input image size, to extract more information from the image during compression. We believe that this will allow us to achieve more realistic colorization.

6 References

Baldassarre, Federico, Diego González Morín, and Lucas Rodés-Guirao. "Deep koalarization: Image colorization using cnns and inception-resnet-v2." arXiv preprint arXiv:1712.03400 (2017).

Francis, William. *CNN-Image-Colorization-Pytorch.*https://github.com/williamcfrancis/CNN-Image-Colorization-Pytorch/blob/main/train.py.

He, Keiming et al., "Deep Residual Learning for Image Recognition." https://arxiv.org/abs/1512.03385 (2015)

Lin, Tsung-Yi et al., "Microsoft COCO: Common Objects in Context." https://arxiv.org/abs/1405.0312 (2015)

Mayalı, Berkay. *Image-Colorization.* https://github.com/mberkay0/image-colorization?tab=readme-ov-file.

P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

Shariatnia, Moein. *Colorizing Black & White Images with U-Net and Conditional Gan - A Tutorial.* Medium, Towards Data Science, 18 Nov. 2020, towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8.

Wallner, Emil. *How to colorize black and white photos with just 100 lines of neural network code.* https://emilwallner.medium.com/colorize-b-w-photos-with-a-100-line-neural-network-53d9b4449f8d.

Zhang, Rich. *Colorization.* https://github.com/richzhang/colorization/tree/master.

Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14. Springer International Publishing, 2016.

7 Appendix

You can find all code used in this project below. **Any code that is borrowed is noted in the comments.**
https://github.com/B-N-Brown/EECS442_final_project