Image Colorization using CNNs || Team iGAN

Bhumi D. Bhanushali[1]
bbhanush@andrew.cmu.edu

Kathan N.Mehta[1]
kathannm@andrew.cmu.edu

Avinash H.Raju[1]
ahemaesh@andrew.cmu.edu

Atulya Ravishankar[1]
aravisha@andrew.cmu.edu

December 7, 2019

Abstract

We propose a CNN-based deep model that uses a combination of low-level local image features and high-level global features to colorize grayscale images. Our approach is based on the model developed by Baldassaree et al. from KTH Royal Institute of Technology in 2017. They used an encoder-decoder (CNN) architecture and a feature extractor to combine local and global image features for improved colorization. We demonstrate the performance improvements gained over the baseline model by modifying activation functions, adding skip connections and modifying the feature extractor. Conclusions and avenues for future exploration are also discussed.

1 Problem Statement

Being able to take a black-and-white image and convert it to a realistic full-color image is a task that has a wide variety of applications. For instance, historians can use such technology to modernize old images and security systems can use this technique to convert grayscale feeds from CCTV cameras into color feeds for easier monitoring. The information contained in single-channel images (e.g. grayscale) is relatively limited. Thus, by being able to convert a grayscale image to a color image, we can enrich the image by adding more information in the form of additional channels. Not only will this improve the viewing experience for humans, but it should also enable automated systems to extract more insights from the image (e.g. richer image features, better semantic information about the scene etc.). In fact, the performance of current deep learning models like ResNet and VGG is better on color images compared to grayscale images [1].

This project builds on a model proposed in 2017 by researchers at KTH Royal Institute of Technology [1]. In their approach, they combine both low-level local features and high-level global image features to improve colorization performance. They use an encoder for low-level feature extraction, a pretrained

Inception-ResNet-v2 network for global feature extraction and a decoder to combine both of these to produce the output. In this paper, we demonstrate and discuss the results of our proposed improvements. We used Pytorch to implement our pipeline from the ground up and we trained our model using both the ImageNet and COCO datasets.

2 Literature Review

A lot of interesting work [3,5,6,8] has been done to solve the problem of colorization of grayscale images and it is still a very open-ended problem being pursued quite actively even today. After going through multiple papers and blogs covering this topic, we decided to implement and modify the technique proposed by Baldassarre et al. [1]. The authors use a deep Convolutional Neural Network to process the luminance component of an image. In order to obtain high-level features, they have used the Inception-Resnet-v2 pre-trained model and an encoder to extract low-level features. These two sets of features are combined in a fusion layer and then up-sampled and decoded using another Convolutional Neural Network.

Another technique suggested by Mingming He et al. [2] uses an exemplar based colorization network. This method expects the user to provide a reference image which is semantically related to the target image. The authors have used Deep Image Analogy for matching the two semantically-related images. They have further used the VGG-19 model to train on the luminance channel of an image. The loss criteria used in this research is noteworthy. The authors have calculated two losses - Chrominance loss and Perceptual loss and tried to minimize both the losses separately. But, this method assumes that we have a semantically related image for every image we train, which may not be the case in a real-world system.

We also found a couple of blog posts to be very useful in improving our understanding of the architecture and methodology. The blog by Emil Wallner on FloydHub [4] is a fairly comprehensive overview of the different methods and architectures that can be applied along with the advantages and drawbacks of each. The author suggests three architectures that incrementally build in complexity. One of the three architectures mentions the use of a Fusion layer to merge the feature embeddings with the encoder output, closely mimicking what we have implemented.

The Colorful Image Colorization paper by Richard Zhang et al.[9] treats the problem as that of optimizing a multinomial classification problem. This is another unique way of calculating the loss that can be potentially used for this problem. Along with that, the authors also come up with a 'Colorization Turing Test', wherein the true measure of a good model is the percentage of the output of the model which can be used to fool the human into believing that it is the ground truth. There is a lot of work in this domain that uses classical Computer Vision techniques like Segmentation, key frame generation and color transfer illustrated by Vivek George Jacob and Sumana Gupta in [10].

Table 1: Dataset Information

Category	COCO Images	ImageNet Images
Train	70,000	65,400
Validation	5,000	5,000
Test	11,000	7,000

3 Data sets

The problem of Image Colorization has been studied in great detail previously and a lot of different data sets have been used depending on the implementations of the task. Also, because of the nature of the problem, it is possible to use basically any diverse data set that is readily available. This is because the input requirement of the network is just a grayscale image, which can be extracted from any color image. Owing to this, we had a vast choice of data sets to choose from.

To begin with, we considered training our model on the ImageNet dataset as used by Baldassarre et al. in [1]. But, due to availability issues we decided to look elsewhere for other viable options. Also, the overhead of training our model on a data set as large as ImageNet has its own challenges. For our initial development, we finally settled on the COCO (Common Objects in Context) dataset [11] for a few different reasons. The dataset being readily available and smaller in size compared to ImageNet meant that it was more practically feasible to use. COCO is a large scale dataset and is one of the primary datasets used in Image Segmentation as and Recognition tasks. We trained our model on 70,000 images. We take the grayscale version of these images and compare the output color image of our model to the ground truth to calculate the loss. The dataset consists of 12 common object supercategories, each of which have a considerable number of images. Examples of these supercategories include indoor, outdoor, kitchen, animal and vehicle. Apart form the train images, we used a validation data set of 5,000 images and a test data set of 11,000 images. We were eventually also able to procure the ImageNet dataset, however, given its immense size, we chose to only use a portion of it. We trained and tested our model on both datasets independently and found that there was no significant difference in the quality of the outputs between the two. Consequently, we chose to stick with the COCO dataset for all further development. A summary of the datasets is shown in Table 1.

4 Baseline Model

We have chosen to adopt an approach strongly based on that proposed by Baldassarre et al. [1]. The input to the network is a single-channel image representing a grayscale input. In our case, this is the L-channel of an image

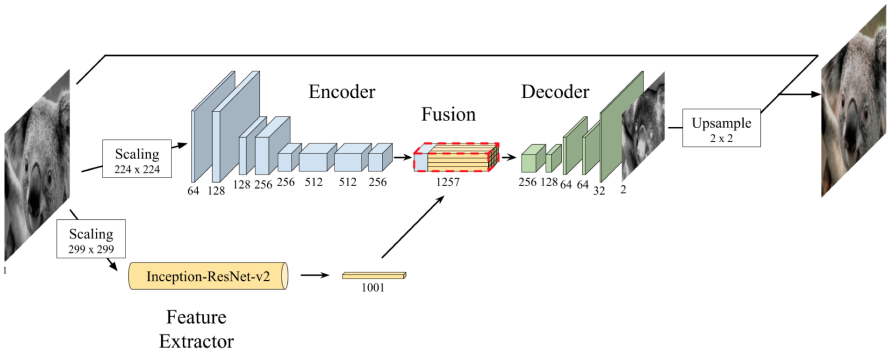

Figure 1: Baseline architecture.

that has been converted to the CIE *Lab* color space [7]. The pixel values of the input are normalized to the range [-1 1]. The output of the network is a 2-channel image that represents the predicted a- and b-channels of the colored image. The original L-channel is combined with the a- and b-channels output by the network to form the final color image.

At a high level, the network can be broken down into four distinct parts. Firstly, a scaled version of the grayscale input is passed through a pretrained model, which acts as a Feature Extractor. In the original paper, the model used was Inception-ResNet-v2, however in our baseline implementation we used Inception-v3 due to its availability in Pytorch. Simultaneously, a scaled version of the grayscale input is also passed through an Encoder network, which is just a CNN. The first, third and fifth layers of the Encoder downsample the image by a factor of 2, which means the output of the encoder is a $\frac{H}{8} \times \frac{W}{8} \times 256$ tensor. The third component of the network is the Fusion layer, which simply concatenates the output of the Encoder with the output of the Feature Extractor (element-wise). This is then passed through a single Convolution layer that outputs 256 channels. Finally, this tensor is passed through a Decoder, which is also a CNN that upsamples and produces a 2-channel output that matches the size of the input. Each Convolution layer is followed by a ReLU layer.

The Feature Extractor finds high-level global features about the image, such as scene characteristics (indoor, outdoor, season etc.). This information can be critical in determining appropriate colors for regions of the scene. For instance, there will be a higher likelihood of green in natural scenes during the summer as compared to winter, where brownish hues would be more prevalent. The Encoder's job is to find a low-level condensed feature representation of the input image. This conveys local and spatial information about the image that is helpful during the decoding. The Fusion layer exists solely to combine the high-level and low-level features computed by the Feature Extractor and Encoder, respectively. This ensures that the context information about the image is

uniformly distributed across all spatial regions of the encoder representation. And finally, the Decoder is used to upsample (using basic nearest neighbor) and generate two channels that contain the color information for the image.

5 Improvements to Baseline Model

5.1 Activation Functions

One subtle change we made to the baseline model was to swap the BatchNorm and ReLU layers in the encoder and decoder. This is because ReLU clips all values below 0 and the negative values were being clipped initially before normalizing over the batch. In order to incorporate those negative values obtained from convolution layer, we made sure to insert the BatchNorm layer before the ReLU layer, thus taking the entire range of values into account when computing the BatchNorm parameters.
Additionally, the baseline model's final convolution layer had a tanh activation followed by a final upsampling layer. We reversed the order of these operations (i.e. first upsample then apply tanh activation). Since the tanh activation clamps the outputs in the range -1 to 1, applying it before upsampling results in a loss of information. By reversing the order, we ensure that all the information is used for upsampling before clamping the outputs to the range -1 to 1. The upsampling layer takes an average of the neighboring pixel values and there is no learning involved while doing this interpolation. Upsampling toward the end of the network makes the output blurry and for this reason, the upsampling layer was shifted to earlier in the decoder's architecture and is now followed by convolution layers to let the model learn the filters best suited for the upsampling layer.

Additionally, we added an additional convolution layer at the end of the decoder, which is equivalent to a 1×1 convolution. This is similar to cross-channel parametric pooling and it makes the colors more acute.[14]

5.2 Skip Connections

The downsampling operation in the Encoder meant that there is a loss of information in the process. This resulted in slightly blurred outputs from the model after upsampling in the decoder. After going through some literature, we came accross the UNET architecture that is also being used in image colorizing problems. The architecture also uses an encoder and decoder network but it adds 'Skip connections' across the encoder and decoder as and when the size of the encoder output matches the decoder output. The architecture that we followed is as described in [13], and as seen in Figure 2, only with a slight modification with regards to the layers at which we add the connections.

By adding these skip connections, the information loss is minimized as some of the information is preserved while upsampling in the decoder. The encoder output at a particular layer is concatenated with the decoder output

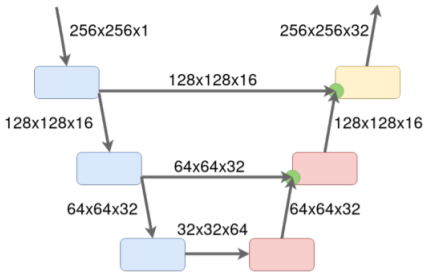

Figure 2: UNET architecture.

at its corresponding layer and then we add another convolutional layer in order to combine the information and maintain the number of channels. In our implementation we add three skip connections when the number of channels are equal to 64, 128 and 256. The output of the model after adding the skip connections seemed sharper and the color representation was also good but not as good as the other modified models we tried. A couple of reasons as to why we could not get better colors with the model might be because in [13], Vincent Billaut et al. considered the colorization problem to be a classification problem and used the Cross Entropy loss instead of the Mean Squared Error loss that we used. They also operated in the YAB color space instead of the LAB space.

5.3 Feature Extractor

While implementing our network, we decided to test whether Resnet50 would have better performance than Inception-v3 as our feature extractor. When comparing the results, we found that there was little difference in the output, prompting us to question whether the feature extractor was actually having any effect at all. To explore this further, we ran a version of our model without the feature extractor. In essence, our model now had an encoder-decoder style architecture. We found that the results generated by this simpler model were better than those generated with the model that included the feature extractor.

In the baseline model, the encoder contributed 256 dimensions to the fusion layer while the feature extractor contributed 1000 dimensions. This meant that the output of the fusion layer was being dominated by the feature extractor (global image context feature). Consequently, the encoder's local image features were not given as much importance. This meant that the network as a whole was learning how to colorize general image categories rather than specific images themselves. By removing the feature extractor from the network, we forced the encoder to generate better local image features, which made it easier for the decoder to assign colors to individual objects in the scene. This can be seen in the results (see Experimental Results section), where the outputs of this simpler model result in richer color assignments (especially for the bus) as compared to the baseline model.

6 Evaluation

6.1 Implementation

The code written by the authors of the paper that proposed the baseline model used the TensorFlow framework. However, we chose to use Pytorch for our implementation as we were already familiar with the library and believed it would be easier for us to implement changes in Pytorch as compared to TensorFlow. Consequently, this meant we had to implement everything ourselves from the ground up and we could not leverage existing code bases. We used the OpenCV, Numpy and Matplotlib libraries for image processing and we used the Pytorch library to implement the model itself. Because Pytorch did not have Inception-Resnet-v2 as part of its pretrained module set, we decided to use Inception-v3 instead as it would still serve as a suitable feature extractor and did not differ significantly from Inception-Resnet-v2 in its output dimensions.

One of the major learning experiences we gained by having to do this was understanding the importance of data manipulation. We ran into several issues because the built-in functions in OpenCV, Numpy and Matplotlib were making different assumptions about the image (e.g. BGR vs. RGB convention, pixel value ranges etc.). By having to debug each of these issues painstakingly, we now understand how important it is to properly understand the data being passed into the network. You can find our implementation on our GitHub repository: `https://github.com/ahemaesh/Deep-Image-Colorization`.

6.2 Training

The loss function applied during training was Mean Square Error (MSE) Loss between the estimated values in the 2-channel output of the network and the ground truth values in the training image's a- and b-channels. For a given recolored image X and the ground truth image $\hat{X}$, the MSE loss is computed using the following equation:

$$MSELoss(X, \hat{X}) = \sqrt{\sum_{c \in channels} \sum_{p \in pixels} (X_c(p) - \hat{X}_c(p))^2}$$

During training, the Adam Optimizer was used with an initial learning rate of 0.001. Our network was trained for 20 epochs with a batch size of 32. Figure 3 shows how the training and validation losses decay during the training process.

6.3 Experimental Results

From the results shown in Figure 4, it is evident that our model clearly differentiates between objects scenes, i.e it recognizes trees, buses, buildings, people etc. The model assigns color to the objects in the scenes accordingly. We started off with a baseline model as described in the original paper [1]. We then made modifications in the architecture as described in 5.1, 5.2 and 5.3. Column 2 is

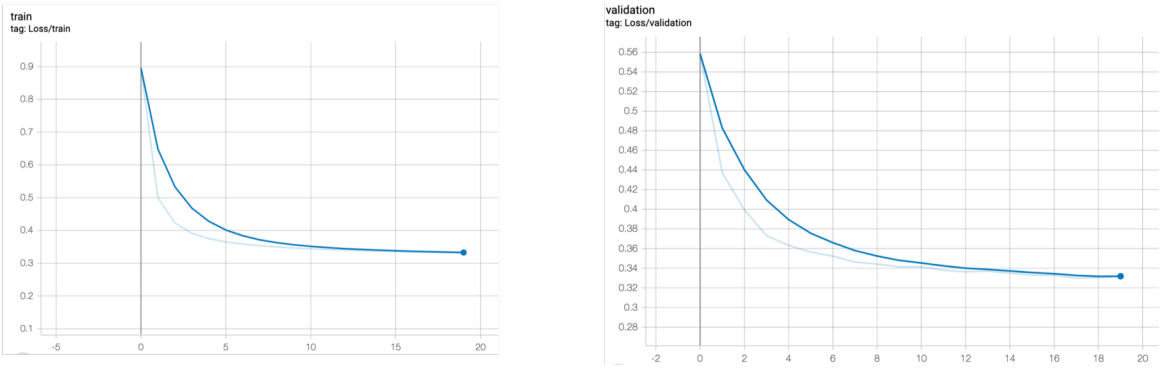

Figure 3: Training and Validation Loss Progression.

Table 2: Quantitative Analysis

PSNR	Baseline	Final upsampling activation	Skip connection	No feature extractor
1	26.91	26.62	25.46	24.72
2	17.93	18.81	18.08	18.87
3	19.67	20.46	18.85	19.14

the baseline model output and columns 3, 4 and 5 in Figure 4 are the outputs from modified architectures 5.1, 5.2 and 5.3 respectively.

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

$$PSNR = 10log_{10}(\frac{MAX_I^2}{MSE})$$

where,
I is the original image and K is the reconstructed image
MSE is the mean square error
MAX_I is the maximum possible pixel value of the image

Table 2 shows the peak-signal-to-noise-ratio between our architecture outputs and the baseline model. For the old building with water image, PSNR is highest for the baseline model. For the bus image, the PSNR for the no-feature-extractor image is the highest. For the baseball image, the PSNR is highest for the final-upsampling-activation image. Though the PSNR values vary with images, we found that this value is not necessarily a definitive metric for determining how realistic our output images appear to be.

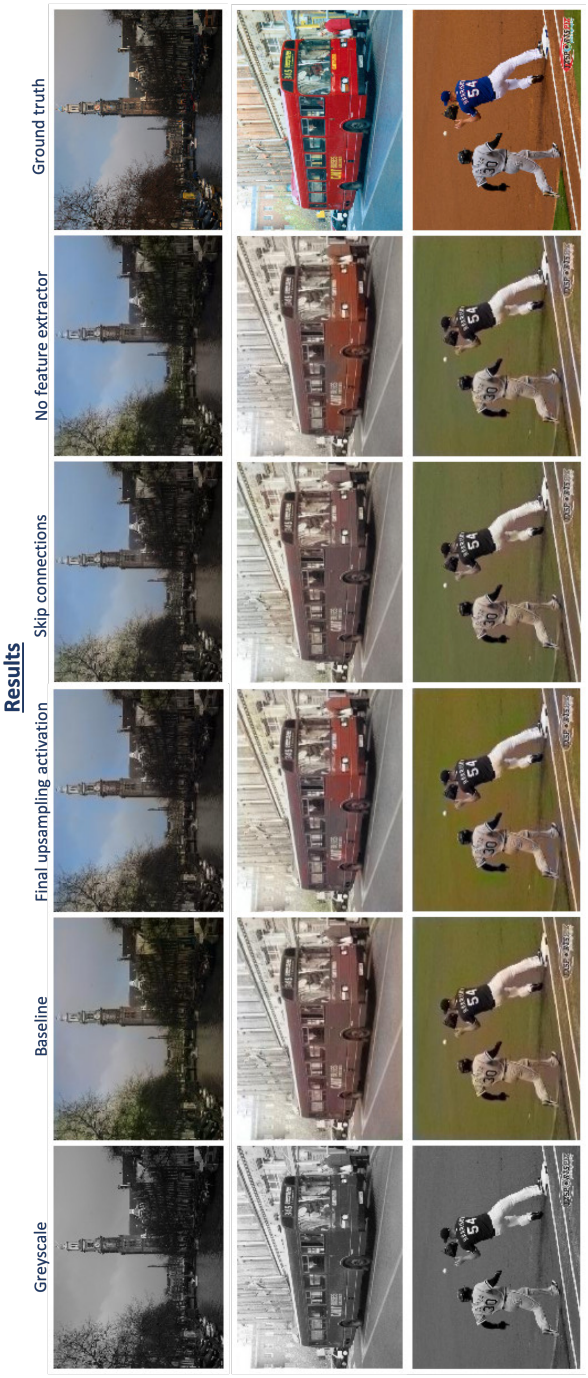

Figure 4: Comparison of ground truth, grayscale and recolored images generated by our models.

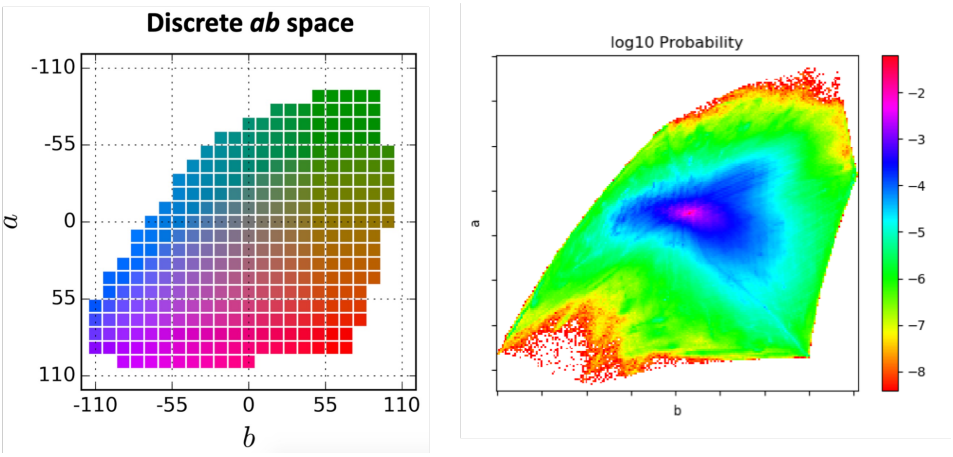

Figure 5: *ab* color spectrum and probabilities.

Also if you see the baseball images, although the ground truth image has an brown-ish portion field color, our model has consistently hallucinated a it to be green. This is to do with the dataset. Majority of the playing field images have large green regions (grass) and thus our model has chosen to color the field in sport images to green accordingly.

We found that the outputs of our no-feature-extractor model are better than the others because, when humans were asked to compare them to the baseline, 80 percent found the former to be more realistic. We conducted a user study to determine which models' outputs humans found most realistic. Of the 10 participants included in the study, 8 chose column 5 (no-feature-extractor) and 2 chose column 3 (final upsampling activation). None of the participants chose column 2 or 4.

We noticed that some of the output images had a subtly discernible brownish tint. This can be explained by studying the color distribution of our training set. Figure 5 (left portion) shows the *ab* colors as they appear to the human eye while Figure 5 (right portion) shows the log probability of those colors appearing in any given pixel based on sampling our training dataset. As you can see, the color region with the highest log probability (and hence, probability of appearing in general) is light-brown. Therefore, when the network in trained on MSELoss and is unsure of which color to assign to a given pixel, it simply chooses the most probably color from the dataset, which is brown. This is because this assignment reduces the expected error, which it was trained to minimize. Thus, some of our output images have a brownish tint.

7 Conclusion

The modified model outlined above demonstrates that it is possible to improve the baseline deep colorization model proposed by [1]. After implementing the baseline model, we proposed three modifications and generated results for comparison.

First, by moving the final upsampling layer to be an intermediate layer, the earlier interpolated images are now passed through further convolution and ReLU layers which result in a sharpened output. Upsampling in general averages over the neighboring pixels and performs interpolation, thus, blurring the image. With the aforementioned change, we have overcome the blurring effect. Second, skip connections are added to regain the information lost during downsampling (bottlenecking) in the encoder. The lost information is preserved using these connections and added back into the output step-by-step in the decoder. And finally, removing the feature extractor and using only the encoder-decoder architecture has forced the model to learn a better representation of the features as it is no longer dominated by the outputs of the feature extractor, which is visible in the relatively richer colors of outputs.

Overall, we found that removing the feature extractor produced the best results, followed closely by modifying the upsampling layer. The skip connections only produced marginal improvements over the baseline model.

8 Future Work

The final layer of the feature extractor is an exhaustive representation of the 1000 ImageNet classes. An intermediate layer of the feature extractor of lower dimension (for example, a 1×80 vector) could be used when the dataset used for training has fewer classes. This is because deeper representations carry less information about the specific contents of the image and more information related to the general context of the image.

On a separate thread, MSE loss has its drawbacks, the most obvious of which is producing a grayish-brown tint when it is unsure which color to assign. This can be solved by classifying the images in the dataset and emphasizing the rare colors in those particular classes. A classification network could be added to the feature extractor which calculates the classification loss (or CrossEntropy Loss). This final loss of the network could be a weighted sum of classification loss and MSE loss. This would penalize the model less for selecting the rare incorrect colors pertaining to those specific classes.

And as a final suggestion, a discriminator network such as the one used in GANs could be used to determine whether the colored images generated by the network look plausibly realistic. This could be used as a substitute for the loss function on which the model is being evaluated.

References

[1] Federico Baldassarre, Diego Gonzalez Morin and Lucas Rodes-Guirao. *Deep Koalarization: Image Colorization using CNNs and Inception-Resnet-v2.* In: CVPR(2017)

[2] Mingming He, Dongdong Chen, Jing Liao, Pedro V. Sander and Lu Yuan. *Deep Exemplar-Based Colorization.* In: ACM Transactions on Graphics (TOG). Volume 37., ACM (2018)

[3] Welsh T., Ashikhmin M. and Mueller K. *Transferring color to greyscale images* In: ACM Transactions on Graphics (TOG). Volume 21., ACM (2002)

[4] FloydHub : `https://blog.floydhub.com/colorizing-b-w-photos-with-neural-networks/`

[5] Larsson G., Maire M. and Shakhnarovich G. *Learning Representations for Automatic Colorization.* In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. Lecture Notes in Computer Science, vol 9908. Springer, Cham

[6] Charpiat G., Hofmann M. and Scholkopf B. *Automatic image colorization via multimodal predictions.* In: Computer Vision-ECCV 2008 (2008)

[7] CIE LAB Color Space : `http://docs-hoffmann.de/cielab03022003.pdf`

[8] Iizuka, S., Simo-Serra, E. and Ishikawa H. *Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification.* In: ACM Transactions on Graphics (TOG) 35(4) (2016)

[9] Richard Zhang, Phillip Isola, Alexei A. Efros. *Colorful Image Colorization.* In: Computer Vision-ECCV 2016 (2016)

[10] Vivek George Jacob and Sumana Gupta. *Colorizing of Grayscale Images and Videos using a Semi-Automatic Approach.* In: 16th IEEE International Conference on Image Processing (ICIP) (2009)

[11] Lin TY. et al. *Microsoft COCO: Common Objects in Context.* In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition* In: arXiv 2015

[13] Vincent Billaut, Matthieu de Rochemonteix, Marc Thibault. *ColorUNet: A convolutional classification approach to colorization* In: arXiv 2018

[14] Kamyar Nazeri, Eric Ng, Mehran Ebrahimi. *Image Colorization using Generative Adversarial Networks* In: arXiv 2018