Image Colorization using CNNs

Simone Toso
simone.toso.2@studenti.unipd.it

Francesco Zane
francesco.zane.1@studenti.unipd.it

Abstract

The goal of this report is to implement two approaches to image colorization. We follow two ifferent approaches, based on the minimization of two different loss functions, and compare the results. The first approach is based on MSE minimization. The second approach, proposed by Zhang et. al. [5], views colorization as a classification task in which each pixel is classified to its corresponding color. We implemented both methods using the `Pytorch` *framework and compared the results. The code is fully available at this link.*

1. Introduction

The task of image colorization consists in reconstructing a colorized version of an image, given its grayscale representation. This problem is often modelled as predicting the values of the a and b channels in CIE-LAB representation, starting from the L channel, which encodes the pixel's lightness. This problem is affected by what is known as multimodal uncertainty: an object (say, for example, a dress) could be of various colors, all different to distinguish given the grayscale representation.

A standard approach is to evaluate the goodness of the prediction using the MSE loss, trying to minimize the euclidean distance between the predicted ab channels and those of the original image. In this sense, the colorization task is formulated as a regression problem [1]. This approach, being effectively a regression task, heavily penalizes large deviations from the ground truth.

Another approach, proposed by Zhang et. al., is specifically designed to obtain a more intense and saturated colorization [5]. The idea of this approach is to turn the colorization task into a classification problem: the range of possible ab values is discretized in bins of fixed width and the goal is to predict, for each pixel, a probability distribution across the bins. The loss function is specifically designed to penalize the misclassification of rare colors.

We implemented both methods in `Pytorch`, two different datasets, and compared the results.

2. Datasets

During our experiment we trained the model on two different datasets, comparing the results.

- `CIFAR10`: We used $50\,000$ images out of `CIFAR10`, a dataset typicly used for image classification. We allocated 80% of the data for training and the remaining 20% for validation. The low resolution of this dataset (32×32) allows to store a large number of images in memory. In addition to that, the fact that the dataset was originally used for classification in 10 classes implies that many images (all the pictures of planes, all the pictures of trucks...) are quite similar. This should make it easier for our model to pick up on common motifs which should suggest the correct colorization.
- `MIRFLICKR`: The second dataset we used is a subset of `MIRFLICKR25K`, an image collection originally used for image retrieval benchmarks. This subset was published on `Kaggle` to be used for image colorization [4]. The images have a resolution of 224×224. We used $10\,000$ images to train the model with the MSE loss and $3\,000$ images for the model with the weighted loss, as the latter is more computationally demanding. The images of this dataset are quite heterogeneous in nature. They depict very different scenes in terms both of object content and of lighting conditions. This dataset was also split in training and validation set following a 80% vs. 20% ratio.

During training, we implemented a random horizontal flip with probability $p_{flip} = 0.5$ for data augmentation. As suggested in [5], we didn't apply a vertical flip because some spatial information (e.g. the sky is usually in the top half of the image) would be lost.

3. Method

We followed two different approaches, based on two different loss functions.

3.1. MSE loss

Let $\mathcal{X} \in \mathbb{R}^{1\times h\times w}$ be the L channel of the image and let $\mathcal{Y} \in \mathbb{R}^{2\times h\times w}$ be the a and b channels. Given a prediction

$\tilde{\mathcal{Y}}$, we can evaluate the loss as

$$l(\tilde{\mathcal{Y}}, \mathcal{Y}) = \frac{1}{2} \sum_{\substack{i \in \{a,b\} \\ 1 \le j \le h, 1 \le k \le w}} (\tilde{\mathcal{Y}}_{ijk} - \mathcal{Y}_{ijk})^2 \qquad (1)$$

The loss can then be minimized with an optimizer (e.g. Adam, see 4.1). This method, while straightforward to implement, is known to lead to unsaturated, washed out colors. This is due to the fact that the MSE loss penalizes large deviations from the true color. Indeed, when in doubt on the color of an object, the MSE minimization will result in averaging across the possible colors, ending up with ab values near zero.[5].

3.2. Weighted loss

A different approach proposes to treat the task of colorization as a classification problem[5].

The goal is to encourage the CNN to use bolder, more intense coloring. Indeed, Zhang et. al. argue that a bold and believable colorization is preferrable to the desaturated colors obtained through MSE minimization. The network will be trained to predict a probability distribution for the color of each pixel, from which we will reconstruct a plausible coloring. This approach, which doesn't simply assign one color to each pixel but rather gives each color a probability, reflects the multimodal uncertainty of the colorization task.

The first step is to discretize the possible (a, b) values. To do so, one can plot a histogram of the (a, b) values in the whole dataset, using bins of width 10. Let Q denote the number of non-empty bins.
The task will now be to predict, for pixel i, j a probability distribution $\hat{\mathcal{Z}}_{i,j} \in [0, 1]^Q$ over the Q bins. The ground truth $\mathcal{Z}$ is obtained from the original image via one-hot encoding: if k^* is the index of the bin containing the (a, b) value of pixel i, j, we have

$$\mathcal{Z}_{i,j,k} = \delta_{k,k^*}. \qquad (2)$$

The loss function, as typical in classification tasks, is the cross entropy. There is, however, a small variation: Zhang et. al. [5] assign a different weight to the loss of each pixel, according to its true (a, b) value. The weights and the loss are computed as follows.

Let $\vec{p} \in \mathbb{R}^Q$ be the vector containing the frequency of each of the Q bins. We define a normalized weight vector $\vec{w} \in \mathbb{R}^Q$. Its entries are defined as

$$\vec{w_i} \propto \big((1-\lambda)p_i + \frac{\lambda}{Q}\big)^{-1}, \ \sum_j p_j w_j = 1, \qquad (3)$$

where λ is a parameter that we fixed at $\lambda = 0.5$. The loss is then computed as

$$L(\hat{\mathcal{Z}}, \mathcal{Z}) = -\sum_{h,w} v(\mathcal{Z}_{h,w}) \sum_q Z_{h,w,q} \log(\hat{\mathcal{Z}}_{h,w,q}) \qquad (4)$$

where the weight $v(\mathcal{Z}_{h,w})$ is given by

$$v(\mathcal{Z}_{h,w}) = \vec{w}_{q^*}, \ q^* = \text{arg} \max_q \mathcal{Z}_{h,w,q}. \qquad (5)$$

In other words, we are evualating a cross entropy loss, in which the terms corresponding to a rare color are given more weight than those corresponding to common colors.

3.2.1 From class probabilities to colorization

Given a predicted probability distribution $\hat{\mathcal{Z}}_{h,w}$ for the class (i.e. color bin) of pixel h, w, we have to assign it an (a, b) value to print the colorized image. We followed Zhang's approach and implemented an annealed mean. The corresponding equations for the output color (a_{out}, b_{out}) are

$$a_{out} = \frac{\sum_q a_q \exp\big(\log(z_q)/T\big)}{\sum_q \exp\big(\log(z_q)/T\big)} \qquad (6)$$

$$b_{out} = \frac{\sum_q b_q \exp\big(\log(z_q)/T\big)}{\sum_q \exp\big(\log(z_q)/T\big)} \qquad (7)$$

The temperature T determines how peaked the softmax distribution will be. If we set $T = 1$ we are making a simple arithmetic mean, while setting $T \to 0$ leads to picking the bin of maximum probability. We set $T = 0.38$, as this value is the one proposed by Zhang et. al.

4. Experiment

We now present the architectures used for colorization and the results we obtained.

4.1. Network architecture

Given the limited hardware resources, we followed an approach often used to reduce the number of parameters and implemented a CNN on top of `ResNet18` [3]. In the first layers of `ResNet-18` the feature map size is repeatedly halved by convolutional layers of stride 2. This is done to preserve the time complexity at each layer [2][3]. In all the networks we trained, the first layers were given by the first three Convolution-Identity blocks on `ResNet18`. Since at each `ResNet` block the feature map size is halved, the output of these blocks will be given by 128 channels with $1/8$ of the input size (e.g. $224 \times 224 \to 28 \times 28$). The subsequent layers, which are the ones we actually trained, are shaped accordingly to the corresponding loss function. In table 4.1 we report the structures used for training. Each convolutional layer is followed by a batch normalization

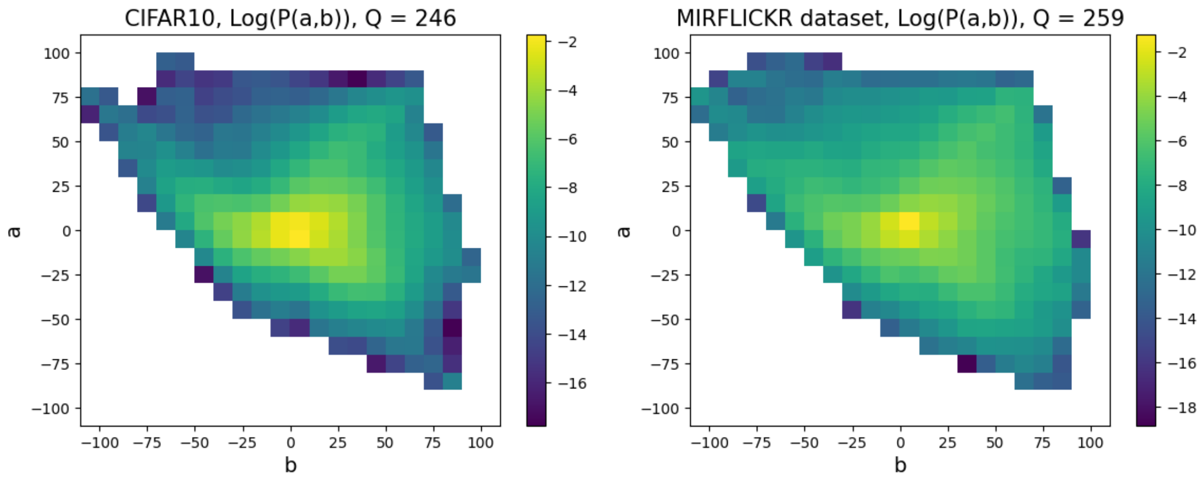

Figure 1: Histogram of the (a, b) values in the two datasets. As one can see, most of the pixels have (a, b) values near zero, corresponding to milder colors. The MIRFLIKR dataset, having higher resolution and more diverse images, has a higher Q and a less peaked distribution.

and a ReLU layer with the corresponding number of channels. When training the weighted loss network on the MIRFLICKR dataset, the upsampling layers were placed differently than with the CIFAR10 dataset to reduce the training time.

MSE	Weighted loss
ResNet (first 3 blocks)	ResNet (first 3 blocks)
Conv(128,128)	Conv(128, 128)
Conv(128, 128)	Upsample(2)
Upsample(2)	Conv(128,128)
Conv(128, 64)	Conv(128,64)
Upsample(2)	Upsample(2) (only in CIFAR)
Conv(64, 32)	Conv(64, 64)
Conv(32, 32)	Conv(64, Q)
Conv(32, 32)	Upsample(2) (only in CIFAR)
Upsample(2)	Upsample(4) (only in MIRFLICKR)

Table 1: Network architectures (always used kernel size 3, stride 1)

4.2. Data preprocessing

Before training the models, we had to preprocess the data to suit our task.

The first step was to discard the labels from CIFAR10, convert the images to CIE-LAB representation, and separate the L channel (the input features) from the a and b channels (the ground truth). This step was not needed in the MIRFLICKR dataset, as it had already been done on the version we found on Kaggle.

We also had to prepare the corresponding datasets to be used with the weighted loss. In this case, as explained in section 3.2, the ground truth is the probability distribution among the Q bins for each pixel. To do so, we assigned each pixel of every image to its corresponding bin in the (a, b) histogram and saved the bin indeces in memory. To avoid having to preprocess the data at the beginning of each session, we stored the datasets in .pkl format using the pickle module.

4.3. Color distribution and evaluation of $\vec{w}$

When applying the weighted loss, we had to evaluate the histogram of (a, b) as described in section 3.2. In figure 1 we show the histograms obtained in the two datasets. In both datasets we observe that the distribution is peaked around the origin $(0, 0)$, in the area of the less saturated colors. The observed values of Q were $Q_{CIF} = 246$ and $Q_{MIR} = 259$ for the CIFAR10 and MIRFLICKR datasets respectively. From the histograms, one can see that the distribution is less peaked in the MIRFLICKR dataset. Our interpretation of the fact is that the higher resolution and more diverse set of images results in a higher variety of colors. After plotting the histograms, we computed the weight vector $\vec{w}$ for both datasets using equation 3.

4.4. Training and colorization of test images

We then trained the architectures described in section 4.1 on the datasets. The training was done for a maximum of 100 epochs. An early stopper checked if a new minimum validation loss had been reached in the past 10 epochs and stored the network parameters corresponding to the last minimum. In all cases the early stopper interrupted the training before the 50th epoch. Parameter update was implemented using `Pytorch`'s Adam optimizer with learning rate 10^{-3}. The batch sizes used on the `CIFAR10` dataset were 200 and 100 for the MSE and weighted loss models respectively. The batch sizes used on the `MIRFLICKR` models were 100 and 10 respectively. All models were trained on `Google Colab`, leveraging the platform's GPUs.

In figures 2 and 3 we display some images from the validation set of the `CIFAR10` and `MIRFLICKR` dataset respectively.

On the `CIFAR10` images, the network train with the MSE loss results in less saturated colors. This is consistent with what we expected from previous works. The weighted loss, on the other hand, results in more vivid colors (see, for example, the first row of figure 2). Both models are affected by a bias given by the small variety of images: in the last row of figure 2, for example, we can see that both models incorrectly assumed that the horse was running on a field of grass. Similarly, in the first row of figure 2, both models color the car red. This is an example of the multimodal uncertainty of image colorization.

When trained the models on the `MIRFLICKR` dataset, the network trained with the MSE loss behaves similarly to the one trained on `CIFAR`, producing colorizations that are unsaturated (as expected) but without notable problems. The model trained on the weighted loss shows a less consistent behaviour: some images (e.g. the first two rows of figure 3) are colorized in a very believable way, while other images show patchy, uneven colorizations (e.g. last two rows of figure 3). One reason for this behaviour could be that, as said in section 2, we trained the `MIRFLICKR` weighted loss network on 3 000 images, whereas we used 10 000 for the MSE network. Another possible cause is the upsampling used in the weighted loss network (see section 4.1); a more accurate model could be obtained by performing the upsampling in between the last convolutional layers, resulting in larger activation maps and a more precise colorization.

5. Conclusions

The goal of this project was to implement two implement two different approaches to the task of image colorization using `Pytorch`. We trained both models on two datasets, with low (32 × 32) and higher (224 × 224) resolution respectively. The first method, based on MSE minimization, performed similarly in both datasets, resulting in reasonable but unsaturated colorizations. The second method, proposed by Zhang et. al. [5], tackles colorization as a classification problem. When trained on the small resolution dataset, it produces more lively and believable colors. In higher resolution model, this method shows some imperfections in the higher resolution model. A possible improvement could be obtained by working on a larger dataset and by performing the upsampling in between the convolutional layers rather than at the end of the network.

References

[1] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. Deep colorization, 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[3] Luke Melas. Image colorization with convolutional neural networks, 2018. `https://lukemelas.github.io/image-colorization.html`.

[4] Shravankumar Shetty. Image colorization. `https://www.kaggle.com/datasets/shravankumar9892/image-colorization/data`.

[5] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. *CoRR*, abs/1603.08511, 2016.

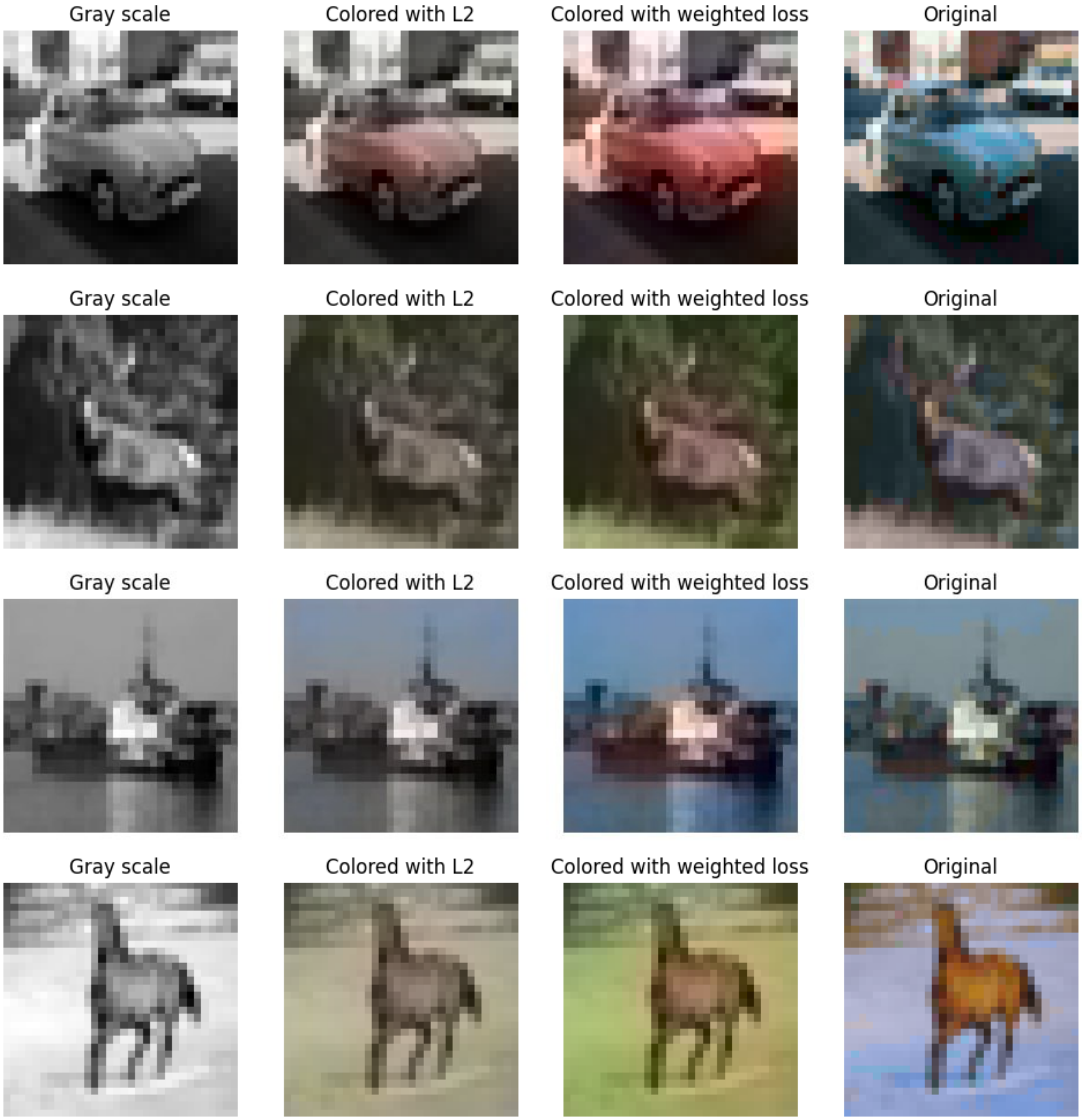

Figure 2: Some images from the validation set of dataset `CIFAR10`. The weighted loss model typically produces a more believable colorization, while the MSE model prefers grayish tones.

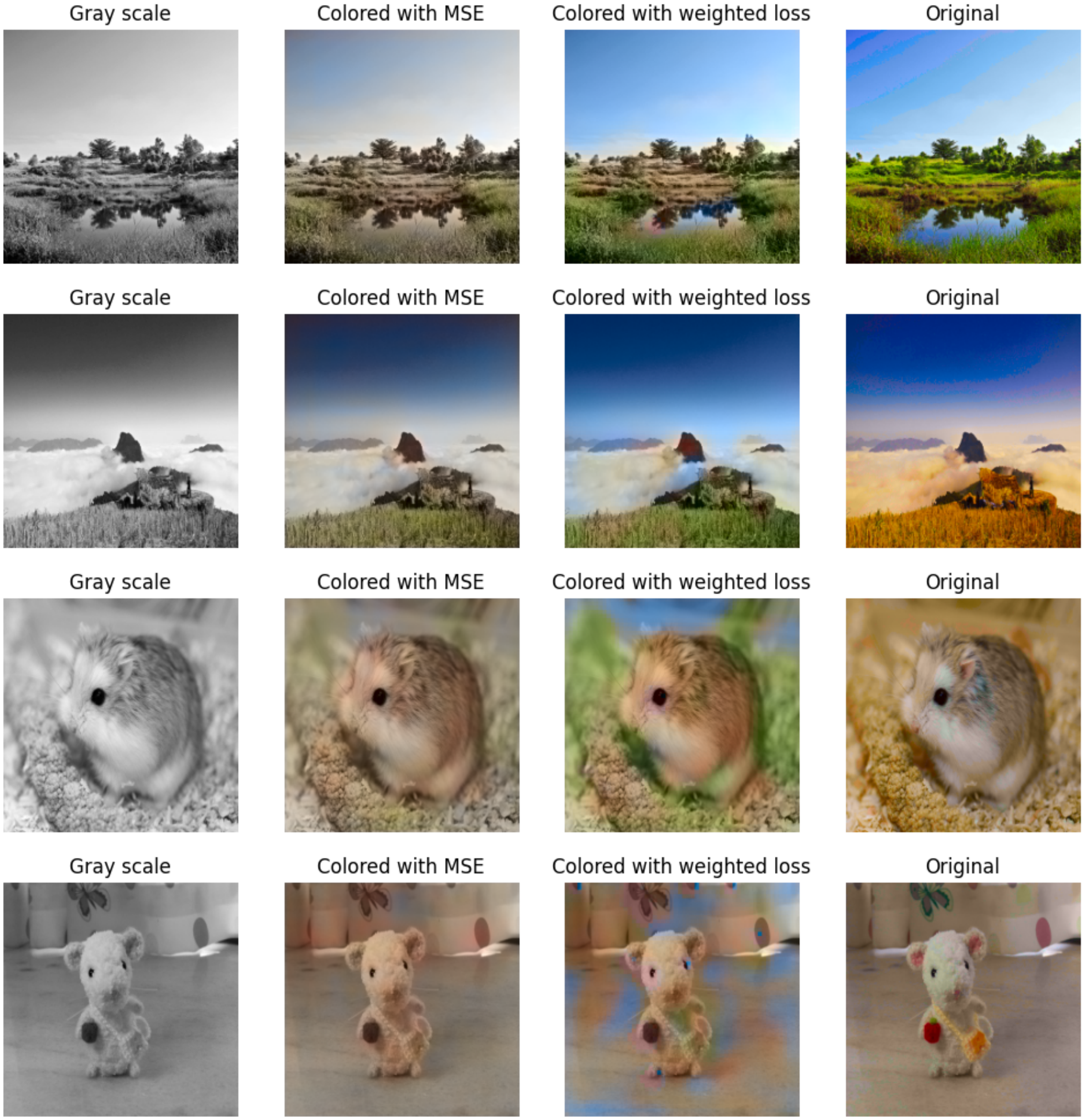

Figure 3: Some images from the validation set of dataset `MIRFLICKR`. We can see that the model trained on the MSE loss prefers desaturated colors, while the weighted loss results in more vibrant colors. On the other hand, the MSE model behaves consistently across all images, while the weighted loss model often produces unpleasant patches of color (e.g. last two rows).