# Image Colorization using CNN and Autoencoder

Lucia Depaoli
lucia.depaoli.1@studenti.unipd.it

Simone Mistrali
simone.mistrali@studenti.unipd.it

## Abstract

*Image colorization is one of the most challenging task in computer vision of the last years, due to its multimodal nature. Since most of the objects can assume a wide spectrum of different colors, the smarter way to deal with the problem is not by looking at the ground truth color distribution of the image, but to find a plausible one. In this project we will tackle this task via CNN and via an autoencoder network architecture, inspired by previous works. We will try to give an explanation of the problems we will face, and we will perform several experiments in order to understand the complexity of the task.*

## 1. Introduction

In the last years, many researchers and coding amateurs have investigated the task of automatic black and white images colorization. Before machine learning popularity explosion, the job was performed manually (using photo-editing software or by hand), usually applied to old photos or old movies. Having this last task as target, automatic image colorization can be considered as its starting point.
The main problem of this task is its multimodal nature. For example, an apple can be green or red, the sky can assume different color during the day, a dress or a t-shirt can be painted by approximately every color of the visible spectrum. This should lead us to consider a *classification* task instead of a regression one. This is the main distinction between all the different approaches. But, even if see it as a classification task is more appropriate, it is more complicated to deal with because we have to define a probability distribution over the possible color, instead of just looking at the ground truth.
One of the main challenge of automatic image colorization is the desaturation problem. When a regression loss is used, the network learn to minimize the difference between the ground truth color distribution and its output. Due to the multimodal nature, it will understand that the colors that guarantees less difference are the desaturated ones. For example, if we are training our dataset only on pictures of red apple, the network will learn to produce the appropriate color. But what if our dataset include also green apple? The output of the network will have grayish or sepia color, because this will be an half way between red and green. The approach proposed by Zhang et al. in their work [2] partially solves this problem.
The task is usually approached in the CIE *lab* color space. The reason is simple: if we have to hallucinate the color of a RGB image, we have to identify 3 channels. Whereas, the LAB color space have 1 channel used to define the luminosity of the image (it is a B&W channel that goes from 0 to 100) and other 2 channels (*a* and *b*) defining the *RGB* colors (both go from $-128$ to 127). So, given in input the *L* channel of an image, the network just need to guess 2 channels and put everything together.

## 2. Related works

The main important work related to automatic image colorization is the one made by Zhang, Isola, Efros [2]. It approaches the problem as a multimodal classification task, while previous attempts consider it as a regression task. The paper proposes to map the output of the network into a probability distribution over the set of possible colors (quantized *ab* values), and then compare it to the ground truth probability distribution using a multinomial cross-entropy loss, appropriately rebalanced by a weight function in order to avoid desaturation. The network has been trained on 1.3M images from ImageNet training set. Our CNN architecture taken is based on the one proposed in this work, except for the loss and rebalancing part which we have avoided in order to reduce the complexity of the task.
For the autoencoder part we were inspired by an article in the online data science magazine Towards Data Science[1].

## 3. Dataset

### 3.1. Fruit dataset

Since our task is to reproduce results achieved previously by other researchers, we are not interest in create a reusable network trained over million of image. That said, for our task we have selected a small dataset (around 16K

[1] https://towardsdatascience.com/image-colorization-using-convolutional-autoencoders-fdabc1cb1dbe

images for training and 5K for test) composed by fruit images. Some of them are shown in figures below. Notice that the absence of the background make the task simpler respect to real-life images. Such dataset allows us to train our networks and achieve impressive results with small computation resources and very few iterations. Also, we seems not to encounter the desaturation problem. This is probably due to the small number of different fruits and to the fact that different images from the same fruit are very similar each other (data augmentation seems useless in this case). We download this dataset from Kaggle[2].

### 3.2. Animals dataset

We have selected real-life images of dogs, chickens, horses and cats from this dataset in Kaggle[3]. We have selected 11K images for training and 1K images for test set.
The animals dataset is more complex than the one we have used before. Images are larger, more detailed, the background is present and objects have a multitude of different color variation. Here we perform a resize of the image and then a `CenterCrop` to 100x100 in order to have small images without distortion.
Automatic colorization of real-life images is very much harder than the one concerned the fruit dataset. For example, the presence of the background can lead towards a mostly-green colorization, if we are considering naturalistic photos. Also, subjects are not-static and they have different dimension and different positions along the images of the dataset.

## 4. Method

There are a lot of CNN architecture proposed to perform the automatic colorization task. Most of them are based on Zhang's network, some have an Encoder-Decoder structure, some on GAN. We have explored the first two categories, in order to compute the differences between the two methods.

### 4.1. CNN inspired by Zhang's work

We have used the Zhang's network without rebalancing has a starter point for our network. Since this network has been trained on million of complex images, we have removed few layer inside in order to reduce the time complexity. The network we have achieved at the end is reported in Fig. 1
We have `Conv2d`, `norm_layer` and `Upsample` layers, with in between `ReLu` activation functions and one `SoftMax` at the end (because we have normalized our value between 0 and 1). We have also changed the last layers, that used to have 313 channels as output. This is due to the fact that we have not performed the class rebalancing, so it is useless to us to map our network into 313 layers (that were the number of discretized bins present inside the probability color distribution of the LAB color space). It is not useful to use `pool` layer because it increases the information density (useful in classification tasks), but distorts the image. We need to pass through the network layers that maintain the original spatial dimension, or we will lose the ability to having an output corresponding to the ground truth. Without `pool` layers, we are able to restore original input dimension using `Upsample` layers.
We have explored different loss functions. The first one is the `MSELoss`, used in the majority of the paper we have seen as alternative to Zhang's loss. The second one is `SmoothL1Loss`, it is a compromise between *L2* and *L1* loss function. We have the *L2* loss function when we have not too many difference between the ground truth, and the *L1* loss function when we are away from it. It penalize outliers less, which is better for multiple color objects, but not for uni-color one.

### 4.2. Autoencoder

Autoencoders are a particular type of Neural Network architecture. They consist of two main parts:

- The Encoder: given an input, it transforms it in a lower dimensional representation. In order to do this, the encoder must learn only the most important features of the input.
- The Decoder: given a lower dimensional representation of the data, it recovers the input.

Autoencoders may be thought as a special case of feedforward networks and they can be trained with all the same techniques, typically minibatch gradient descent following gradients computed by back-propagation.
Our network consists of 13 convolutional layers, six for the encoder and seven for the decoder part. They are composed by `Conv2D` and `UpSampling` layers, with in between `ReLu` activation function and at the end the `tanh` activation function, because in this case the $a$ and $b$ channels are bounded to $[-1; 1]$.
In the autoencoder part we studied the two losses applied to the animal dataset.

## 5. Experiments

Our experiments focus mainly on the different results between Encoder-Decoder architecture and the CNN one. Also, we explore different loss function and different datasets.

### 5.1. CNN

- *Fruits dataset*: At first, we train the original network by Zhang on our fruit dataset. Even if we are able to

[2]https://www.kaggle.com/sshikamaru/fruit-recognition
[3]https://www.kaggle.com/alessiocorrado99/animals10.

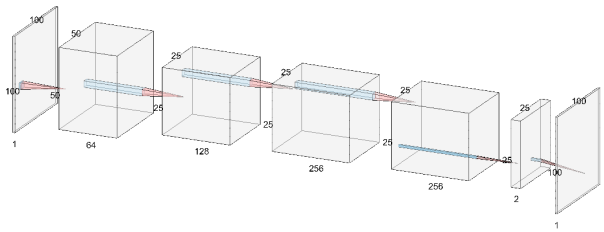


Figure 1. Architecture of the network used for the colorization of the fruit dataset.

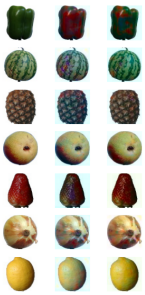

Figure 2. Fruit colorization with CNN architecture. First column is the true images. Next columns are the output of the network at different epochs.

colorize test images correctly, the computational time is extremely slow. We are satisfied by the performance, but given the fact that our train set is very simple, we think we can reach similar results with few parameters (this network has around 32M parameters).
So we start removing inside layers, resulting in the network reported in Fig. 1, which has around 1M parameters. In Fig. 2. are reported our results for different images of the test set, for different number of iteration. We can see that, even with small number of iteration, we are able to colorize the images in a meaningful way.

We have then modify the loss function from `MSE` to `SmoothL1`. Results are reported in Fig. 3, compared to the one obtained before. Outputs are similar, `SmoothL1Loss` seems to make less confusion when we have an objects that can assume different color, whereas `MSE` colors seem to be more saturated.

Lastly, since our images are all similar and pretty easy to identify, we want to show the performance of the network in a different kind of image. Result is shown in Fig. 4. As expected, results are not as good as the ones obtained using test images. This is mostly because the objects in the train set are centered and the background is absent. Also, probably the network is unable to colorize multiple different fruits in the same image. Anyway, the network is able to recognize which fruit it is and make an attempt to colorize it.

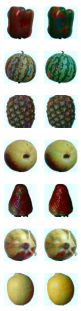

Figure 3. Fruit colorization with 2 different loss function. First column is using L1Smooth. Second column using L2.

Figure 4. Real-world photos fruit colorization with CNN.

- *Animals dataset*: having a very different kind of dataset, the architecture of the previous network is expected to be too simple to recognize pattern in this case. We can not achieve any relevant results in the colorization task. Output images are desaturated and any pattern recognition is absent. This could be due to the small amount of parameters present in the network. We re-implement some layers we have deleted before from the Zhang's network, changing a few parameters in order to fit the dataset and the batch size. Resulting network has around 3.5M parameters. At the beginning, the network can't distinguish different objects, so it applies to every images the same *ab* values. After 50 / 60 iterations it starts to produce relevant results, especially on images belonging to the training/test set. If we try to apply our network to an image it has never seen, we achieve worst results.

## 5.2. Autoencoder

- *Fruits dataset*: at first, we try a much simpler architecture (that network had around 1M parameters), but even if the dataset is really simple, after 600 epochs it

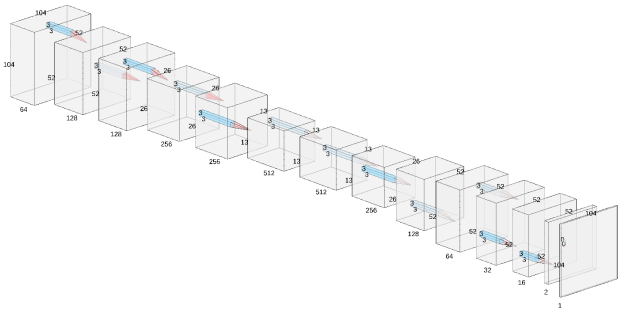

Figure 5. Architecture of the autoencoder used for the colorization.

Figure 6. Fruit colorization with autoencoder architecture. First columns the true images. Next columns are the output of the network at different epochs.

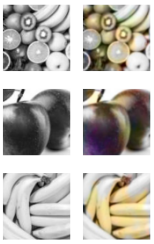

Figure 7. Real-world photos fruit colorization with autoencoder.

does not color anything. The autoencoder we use for the rest of the project has $\approx$ 6M parameters and already after 10 epochs it colorize perfectly the images. In Fig. 6 we can see that the images from the test set have a good colorization. In the other cases, the neural network does not reproduce meaningful results, because the autoencoder learn the low dimensional features of the dataset given in input, and our train dataset deviates from the "real-world" images used in Fig. 7.

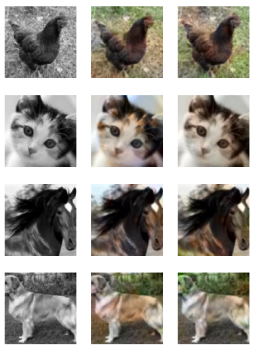

Figure 8. Comparison between the L1smooth (second column) and the MSE (third column) losses of the autoencoder architecture after 150 epochs.

- *Animals dataset*: in this case we do not change the architecture of the autoencoder, even if the dataset is more complex. We can see that using the `L1Smooth` loss, the color is less desaturated, but there are some gray stains. On the other hand the `MSE` tends to desaturate the image, but the stain are almost absent, as we can see from Fig. 8. Also, in Fig. 10 we can see that the autoencoder tends to desaturate the color of images it has never seen.

In Fig. 9 and Fig. 10 are shown the comparisons between the outputs of the two different networks. We can see that the autoencoder network seems to learn faster, but the output of the CNN is less desaturated and more accurate. The background color is the main problem for both methods. It seems that the images are green-ish, this could be due to the fact that the majority of our dataset contains pictures with grass or nature in it.

## 6. Conclusion

We have seen the results of three different architectures (two for the CNN part and one for the autoencoder). Both methods lead to satisfying results, given our limited computing resources. The fruits dataset is pretty simple so the two different networks work both very well. On the other hand, even if the animals dataset is more complex, our simple architectures succeed to colorize pictures in a meaningful (but not in "realistic") way. Let's notice that the

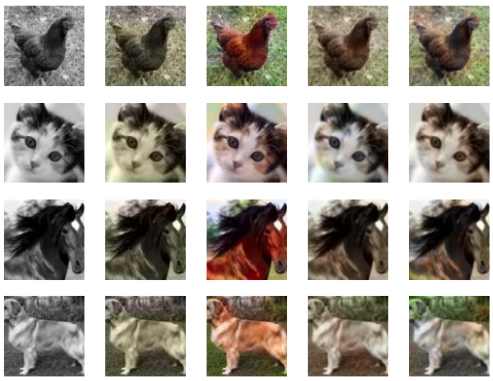

Figure 9. Comparison between outputs of the CNN and of the autoencoder (MSE loss), using images from the train/test set. The first column is the black and white picture, the next two are the outputs of the CNN (the first after few iterations, the second after about 150 iterations), the last two are the outputs of the autoencoder (the first after few iterations, the second after about 150 iterations).

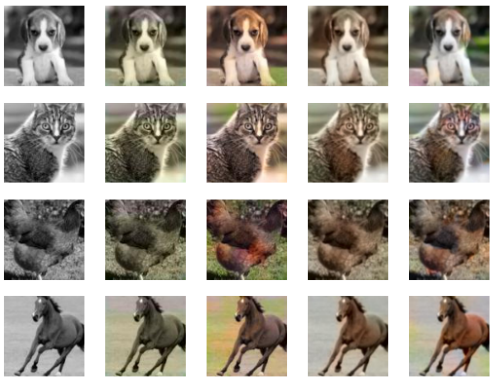

Figure 10. Comparison between outputs of the CNN and of the autoencoder (MSE loss), using images the networks have never seen. The first column is the black and white picture, the next two are the outputs of the CNN (the first after few iterations, the second after about 150 iterations), the last two are the outputs of the autoencoder (the first after few iterations, the second after about 150 iterations).

datasets are really small and we use few number of epochs.

### 6.1. Further studies

Given the outputs of our three different architectures, the performances seem similar. To improve the results, a good idea may be to take in example larger datasets and to perform more iterations.
Another good architecture candidate to study in further work is the GAN (**G**enerative **A**dversarial **N**etwork) [1], which seems to tackle this problem in a smarter way. Since the target is produce realistic images, the presence of the discriminator (one of the two main parts of the GAN) should improve the performances, given the fact that our networks learn in comparison to the ground truth images, while the discriminator have to select the ”realistic” picture between a real one and a generated one.

## References


[1] Mehran Ebrahimi Kamyar Nazeri, Eric Ng. Image colorization with generative adversarial networks. 2018.

[2] Alexei A. Efros Richard Zhang, Phillip Isola. Colorful image colorization. 2016.