# Colorful Image Colorization with Tensorflow

Sophia Schulze-Weddige Malin Spaniol
Maren Born
Implementing Artificial Neural Networks with Tensorflow
Universität Osnabrück

April 18, 2020

# Contents

# 1 Introduction

Based on the paper *Colorful Image Colorization* (Zhang et al., 2016) this project aims to reimplement a similar artificial neuronal net that transforms grayscale images into colorful pictures.
This involves first creating a dataset based on pictures that are converted into the CIELAB colorspace (Lab), such that the first channel "L" can be considered as input as it is grayscale whereas the "a" and "b" channel form the target labels to be predicted. Thus, the problem can be handled as classification task. In the second step, the aim was to closely rebuilt the layers of the original model (which used "caffe" (Jia et al., 2014)) using tensorflow 2.0 (richtige version?).
Other project have trained convolutional neural networks (CNNs) on the color prediction problem before (e.g. Cheng et al. (2015), Dahl (2016)). The training data is easily available which enables training on large datasets. Problem about previous approaches is that they try to predict the ground truth rather than a possible truth. A conservative loss function tries to minimize Euclidean error between estimate and ground truth. As objects can have various plausible colors, these predictions are multimodal. Thus, the approach of Zhang et al. (2016) innovates a loss function that predicts plausible colors for pixels, rather than the original color (Zhang et al., 2016).

# 2 Theoretical Background

Image colorization is for example made with photoshop. By colorizing old black and white pictures (https://www.reddit.com/r/Colorization/), images become more vivid and modern. The TV-series *Greatest Events of WWII in Colour* (https://www.imdb.com/title/tt9103932/) used colorized footage in order to make the events more contemporary.
Automating the colorization problem with deep learning seems to be achievable, as training data is easy to get.

## 2.1 Related work

Automatic approaches solving the colorization problem mostly differ in acquisition and handling of the data in order to model the accurate correspondence (Zhang et al., 2016). Non-parametric methods predict colors based on one or more reference images. These source data is provided by the user (e.g. Scribble-based colorization by Levin et al. (2004), example-based colorization e.g. by Welsh et al. (2002)) or automatically. From the provided data, more precise from a analog reference image, the color is transfered to the target grayscale image. In these methods,

the outcome depends heavily on the provided data Cheng et al. (2015).
Parametric methods on the other hands, learn prediction functions from large datasets of color images. The problem can be posed as regression or classification of quantized color values (Zhang et al., 2016).

## 2.2 Convolutional neural networks for image colorization

In image classification mostly convolutional neural networks (CNNs) are used. They are inspired by the visual cortex of the brain. The idea is that highly specialized components learn a very specific task, which is similar to the receptive fields of neurons in the visual cortex (Hubel and Wiesel, 1962). These components can be combined to high-level features, which again can be merged to classes. In CNNs this concept is implemented by several successive convolutional layers: a weight kernel moves over the two-dimensional input image. For each pixel, the kernel also "scans" the neighboring pixel. After all being multiplied by the weights and summed they form the new pixelvalue. Different kernels can generate different so-called feature maps. One feature map targets the same feature (e.g. edges) in different imagesections. In this manner, CNN can store spatial information about pixel and features. In a subsequent pooling layer dimensions are reduces by summarizing over the imagesection (e.g. max pooling takes the highest value of a certain imagesection - that can be 1x1 pixel). This facilitates the computation and drops unnecessary information (Effenberg, 2019). In recent years, CNNs improved such that they outperform humans in many classification tasks (Russakovsky et al., 2014).
- in our case we dont need the spatial information that much, but we benefit from the 2D input? or warum?

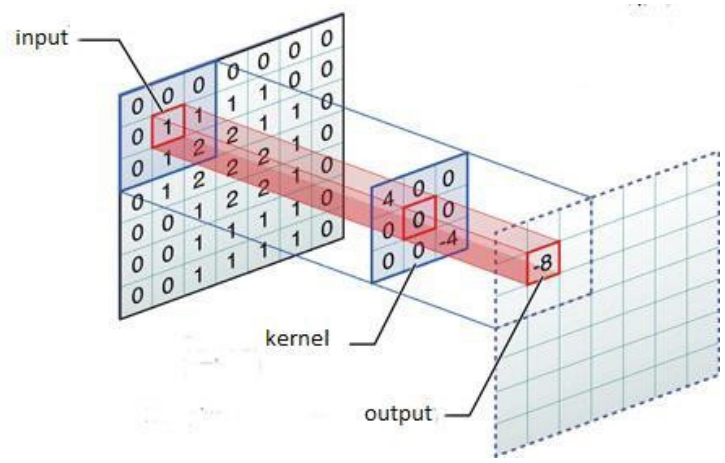


Figure 1: internet bild https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1

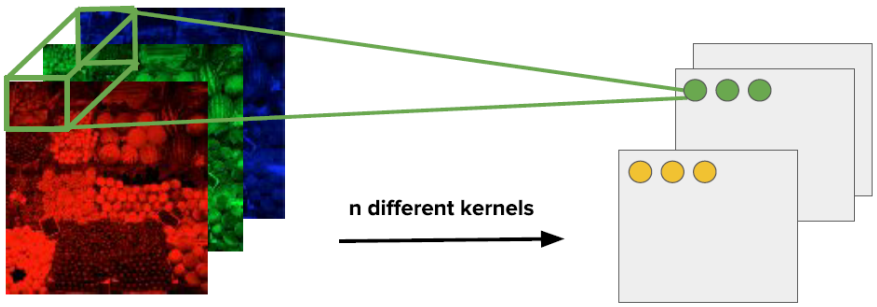


Figure 2: lukes bild

## 2.3 Guiding Paper

- was sind die schwierigkeiten von den colorization sache (sepia ding)
- was hat Zhang gemacht - was hat er besser gemacht

Zhang and colleagues (2016) propose a fully automatic approach to colorize grayscale images. They choose to solve this task with a feed-forward CNN as a classification task with a classification loss and class-rebalancing at training time in order to increase the diversity of the colours in the finals result. This is implemented as a feed-forward pass in a CNN at test-time and trained over a million color images. Concerning the CNN architecture, they use a single-stream, VGG-styled network with added depth and dilated convolutions. The network consists of eight convolution blocks, of which each consists of two or three repeated convolutions and ReLU layers, followed by a BatchNorm layer. The network has no pooling layers. Changes in resolution are achieved only by spatial up- and downsampling between the convolutional blocks. The network is trained on ImageNet.

The mapping, which Zhang and colleagues aim to learn, results out of the previously mentioned information of the Lab colourspace: The input is the L lightness channel, the target channels are the a and b color channels. First, Zhang and colleagues used the Euclidean loss. As the Euclidean loss favors the mean, this leads to overall grayish colors. Therefore, Zhang and colleagues choise to treat the problem as multinomial classification task. The ab output space is therefore divided into bins of grid size 10, and the 313 color values in-gamut span the 313 possible color combinations. Following, for each input, a mapping to a probability distribution over all 313 possible colors is learned. Further, the ground truth color is converted to a vector, using a soft-encoding scheme and a multinomial cross entropy loss, which is responsible for the class-rebalancing. Thereafter, they map the probability distribution to the color values. The class-rebalancing operates pixel-wise. The loss of each pixel is re-weighted at training time, based on how often the color occurs. Moreover, the network used the ADAM optimization algorithm. Zhang and colleagues compared their approach to different intermediate steps.

# 3 The Model and Implementation

```python
import numpy as np
import tensorflow as tf
#from skimage import color
import cv2

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Activation, BatchNormalization
#from keras.layers import Dense
```

Figure 3: Hier kann man dann auch noch etwas dazu schreiben

As previously mentioned we re-implemented the main ideas from the paper by Zhang et al. (2016). This was done with two approaches, which are going to be explained in more detail in this chapter. In both approaches the ImageNet2012 dataset was used. Firstly, we implemented the model structure as described in the paper (see figure 1) and trained the model with the color layers of the input images as the target. Secondly, we translated the colorization task to a classification problem and trained the same model structure with the altered problem representation.

## 3.1 The Data

To train our model we used images from the ImageNet2012 challenge, which were also used in the paper by Zhang et al. and which were available to us through the university server. As we were working with a large amount of data in each batch, it would have been impossible to load the whole dataset at once, hence we used a data generator to load the input and target images successively for each batch. Although there are inbuilt data generators available from keras that allow for some means of data augmentation, we built our costum generator to ensure the functionality we were aiming at. The generator takes the batch size and a list which contains the paths to the images that should be used. Besides the functionalities that one would expect, like shuffling after the whole training set has been seen, the generator further creates the input and target arrays as described in the following. First, the images are loaded and resized to a uniform shape of (224, 224, 3), which corresponds to the height, the width and the number of the color channels, respectively. Then the images are transformed from BGR to LAB color space. The luminance layer (L layer) which displays most of the structure, is separated from the other two and used as the input for our model. The remaining two layers (ab layers) encode the color information of the images and are used as the target of the model in the first approach.

In the second approach, these first steps remain the same, but additionally the target arrays are transformed to match a classification task.
This is done in three main steps. The first step is to discretize the continuous color space and reduce the number of possible colors by quantizing the a and b color ranges into 11 bins. This yields a total of 121 possible colors by combining the a and b layers. In the second step, the a and b layers are combined into a single layer, which keeps the same height and width dimensions as before. This means the color information of each pixel is now encoded in a single number rather than two. Cantor pairing is used to generate a unique and deterministic number from the two a and b values of each pixel. As cantor pairing is reversible, one can easily translate the pairing result back to the original color values with no loss of information. Now that there is a single value for each pixel that encodes its color, the third and last step is to translate this value into a one-hot encoding. With the help of a dictionary, the cantor values are translated to the numbers from 0 to 120. These numbers then serve as the index in the one hot encoding. Hence, color values that were previously represented in two values (a and b color channels) are now encoded by the index within the one-hot encoding. The target array has a shape of (224, 224, 121) and at each pixel position lies a one hot vector.
- loading large amount of data
https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-
das ist (Brownlee, 2019)

## 3.2 Modelstructure

The model structure is strongly inspired by the original implementation by Zhang et al. Only minor changes occur, mostly due to the translation from caffe to keras. The model consists of eight blocks of two or three repeated convolution and relu activation layers followed by a batch normalization. There are no pooling layers in the model, as changes in resolution are achieved through spatial downsampling or upsampling between the convolution blocks. A transpose convolutional layer is used to inverse the convolution and upsample the output back to the correct size.
In the first approach, a stochastic gradient descent optimizer (SGD optimizer) with a learning rate of 0.001 and a momentum of 0.9 is used, which is inspired by the original paper. Further, this approach uses mean squared error as the loss function. In the second approach, the softmax activation function is used in the last layer to prepare the model's output for the categorical crossentropy loss that is used in this approach. Besides, both SGD and Adam optimizer are tested. In both approaches, kernel weights are initialized with the glorot uniform initializer which draws samples from a uniform distribution depending on the number of input and output units of the corresponding weight tensor. Biases are initialized as zeros.

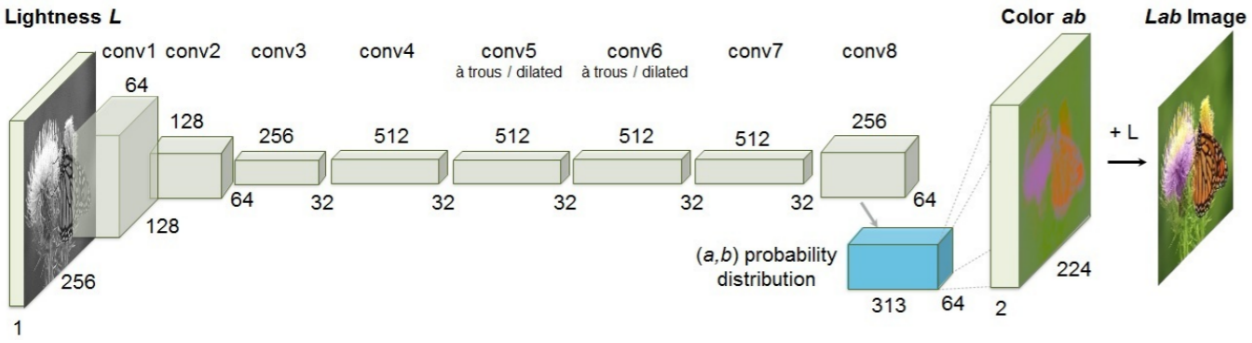


Figure 4: The network architecture of Zhang et al. (2016).

## 3.3 Training

The models were trained on the grid of the institute of cognitive science at Osnabrueck University. A helper script was written that distributed several grid jobs on different computers in order to experiment with the hyperparameters such as the learning rate and the batch size. Through this script, one can easily change these parameters as well as select the data set and the environment and switch between the training mode and the prediction mode. The models were trained with 2000 images and batch sizes of 10 or 20 images. The learning rate varied from 0.1 to 0.001.

## 3.4 Testing

To evaluate the goodness of our two approaches, colors for unseen test images are predicted and evaluated via visual inspection. When the script is started in prediction mode, the model is not trained, but the weights from the corresponding training process are loaded. The model then predicts the most plausible colors for the L layers of the input images. In the first approach, the model's output can be interpreted as the ab layers. That means the output can be combined with the input (L layer) and displayed as an image straight away. For the second approach, three decoding steps are necessary before yielding human-readable images. Firstly, the output of the softmax layer is decoded to find the index of the most likely color value. Secondly, this index is translated to the cantor pairing value it represents with the help of a dictionary. And thirdly, the cantor pairing value is transformed back to the a and b values it is constituted of. These a and b layers can finally be combined with the L layer to display the predicted image in LAB color space.

# 4 Result

# 5 Discussion

## 5.1 Conclusion and Future Work

- vergleich ziehen zu Zhang

# 6 Literature


Brownlee, J. (2019). How to load large datasets from directories for deep learning in keras. https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/.

Cheng, Z., Yang, Q., and Sheng, B. (2015). Deep colorization. *2015 IEEE International Conference on Computer Vision (ICCV)*.

Dahl, R. (2016). Automatic colorization. https://tinyclouds.org/colorize/.

Effenberg, L. (2019). Implementing anns with tensorflow.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

Levin, A., Lischinski, D., and Weiss, Y. (2004). Colorization using optimization. *ACM Trans. Graph.*, 23(3):689–694.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. (2014). Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575.

Welsh, T., Ashikhmin, M., and Mueller, K. (2002). Transferring color to greyscale images. *ACM Trans. Graph.*, 21:277–280.

Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization. *CoRR*, abs/1603.08511.