Colorful Image Colorization with Tensorflow

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

April 17, 2020

Contents

1 Introduction/Motivation

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 Important background knowledge (including reference to most relevant publications)

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.

- what is CNN for image colorization?

In image classification, usually convolutional neural networks (CNNs) are used, which are inspired by the visual cortex of the brain. The idea is that highly specialized components or, in the case of CNNs, filters learn a very specific task, which is similar to the receptive fields of neurons in the visual cortex (QUELLE https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/). These components can then be combined to high-level features, which can then in turn be combined to objects that can be used for classification. In CNNs this concept is implemented by several

successive convolutional layers in which one or more filters are slid over the input generating so-called feature maps. Each unit in one feature map looks for the same feature but in different locations of the input. In recent years, CNNs became so good that they outperform humans in many classification tasks (QUELLE http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/ , QUELLE https://arxiv.org/abs/1409.0575). Although machine learning exhibits very promising results and a lot of research and literature is available on the topic, many branches of industry still rely on traditional computer vision techniques in their implementation of image classification.

CNN apply trainable filtering kernels to extract features from an input image and project these features onto feature maps (?) (?) (?). Based on these maps, the label is predicted by a standard fully-connected layer. Though using a CNN on the raw data images seemed promising, the results were only slightly better than chance level for the training accuracy.

- wie ist der status quo: – related work
- was sind die schwierigkeiten von den colorization sache (sepia ding)
- was hat Zhang gemacht - was hat er besser gemacht

```
#packages needed
import numpy as np
import tensorflow as tf
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
  for loop
'ich bin ein string'
if
return
MyClass,__init__
```

- We started looking at this paper:
https://arxiv.org/abs/1603.08511 ,
- at their best solution

3 The model and the experiment (MAIN PART). This part should feature code.

As previously mentioned we re-implemented the main ideas from the paper by Zhang et al. 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.
- hier nur erklären was wir machen
- Image colorization in general
- the model of the original paper
- theoretical basis:

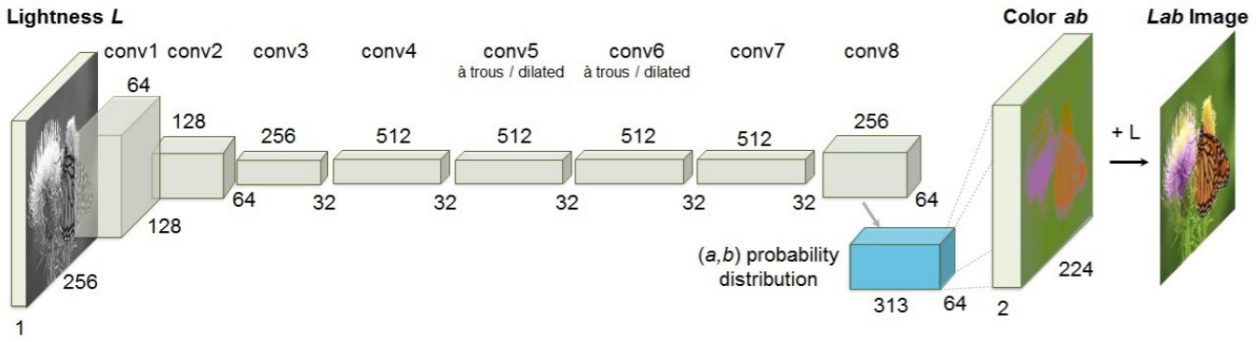

Figure 1: The network architecture of Zhang et al. (2016).

3.1 Dataset

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 Preprocessing

- image preprocessing documentation
https://keras.io/preprocessing/image/#imagedatagenerator-class
- preprocessing via ImageDataGenerator() from keras.prepeocessing.image
- takes in traindata, validation data and test data
- featurewise-center and featurewise std
- classmode: none -> is for predictions

3.3 Modelstructure

- besser hier layer und loss-function

3.4 Layer

- built the same layers aas Zhang et al, but no probability distribution
-networkstructure and design

3.5 Loss-Function

- the original loss-function
- we tried what loss function?

4 Visualization and discussion of your results.

oder lieber bilder als screenshot einfügen?

```
1  import numpy as np
2  import tensorflow as tf
3  #from skimage import color
4  import cv2
5
6  from tensorflow.keras.preprocessing.image import ImageDataGenerator
7
8  from tensorflow.keras.models import Sequential
9
10 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Activation, BatchNormalization
11 #from keras.layers import Dense
```

Figure 2: Hier kann man dann auch noch etwas dazu schreiben

4.1 Training

4.2 Testing

4.3 Conclusion and Future Work

- vergleich ziehen zu Zhang

5 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/.

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*.

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