Hochschule Esslingen
University of Applied Sciences

Studienarbeit

Colorization of black and white images with Deep Learning

Daniel Ketterer

Datum: 17.08.2019

Betreuender Professor: Prof. Dr. Thao Dang

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den August 17, 2019 _____

Unterschrift

Contents

List of Figures

List of Tables

Code Listings

Acronyms

GPU Graphics processing unit.

ILSVRC ImageNet Large Scale Visual Recognition Challenge.

OpenCV OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.

Tensorflow TensorFlow is an end-to-end open source platform for machine learning.

1 Introduction

Since the rise of deep-learning in the recent years it helped different research fields from audio recognition, natural language processing to image recognition to make large advances. Driven through the development of neural networks for semantic segmentation all kind of image-to-image translation problems were tackled with deep neural networks. Colorization is the prediction of a plausible color map to a given black-and-white image.

From the invention of photography to the 1940s photographers took mainly black-and-white images. Also, early television until the late 1960s was broadcasted in black-and-white. It is a very important task to colorize of those historical documents. For example, director Peter Jackson colorized about 100 hours of video footage from the first World War to produce the movie “They Shall Not Grow Old”.

Colorization of black-and-white images was long an artistic and slow process. The use of neural networks jointly with fast GPUs can make it faster and easier accessible for amateurs. Deeper understanding of image colorization can also be useful for other image-to-image translation problems like the restoration of spatial information from images.

This work gives an overview over the roots of colorization techniques and its recent advances through deep neural networks.

2 Fundamentals

This section gives an overview over different methods to colorize black-and-white images. First the underlying problem is elaborated and a description of what colorization should achieve is given. Next different color spaces are presented, because they are ubiquitous in this field. Lastly descriptions of different methods of colorization from scribble-based over example-based to automatic colorization are given.

The subdivision into different colorization methods is taken from [35].

2.1 Colorization

Merriam-Webster defines colorize as "to add color to (a black-and-white film) by means of a computer" [36].

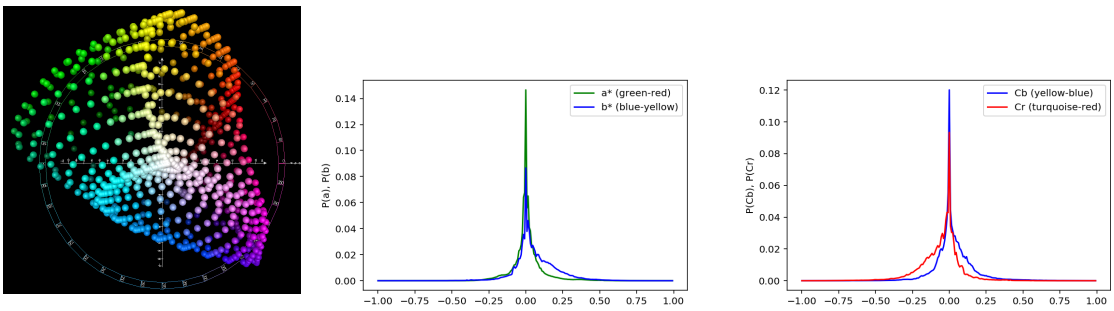
Historically colorization is a mainly manually performed computer aided task. People used image editors to paint colors on black-and-white images. They picked the right colors with the context and maybe additional information about objects in the picture in mind. Then they drew the right colors on objects while paying attention to their borders and preventing colors running into each other. Finding an artificial-intelligence-based solution for this problem would increase speed and automation. The prerequisite is that the results are visually appealing and accepted by humans.

The problem of colorization can be described as:

$$y = f(x),$$

where x is a black-and-white image, f is a function that creates y, the plausible colors for the image.

It is a important part that there is not one true solution y to a given x. Removing color from an image is a surjective operation. Colorization is then a one-to-many problem. Most artificial objects can have different colors. A car can be red, blue, yellow. Grass is usually green the sky is blue and zebra is black-and-white. So, a good function f must either give some kind of plausible color for objects which are ambiguous or take colors from a condition. Objects like grass or trees should be given the usual colors like they appear in nature. Models that take a hint or an example color image z can be denoted



(a) The CIE Lab space from the top into the ab dimension [37]

(b) Left: Probabilities of channels a and b of Lab images. Right: Probabilities of channels Cb and Cr of YCbCr images.

Figure 1

as:

$$y = f(x||z).$$

Colorization can also be described as a multimodal problem [10]. This work focuses on colorization of images and not movies with the use of deep-learning techniques. While earliest studies work with scribbles, the recent publications use example images or are model-centered.

A basic consideration for colorization can be that "the chrominance of an image (especially as perceived by humans) is of much lower spatial frequency than the luminance. In fact, some image storage formats, such as JPEG, exploit this intuition and store the color channels at lower resolution than the intensity channel" [30]. This can for example be used to reduce computing by predicting color in lower spatial size then the original image and then upscaling.

2.2 Colorspace

Different approaches to colorization use different colors paces to operate within. A color space builds the target space of a colorization function or model. It is rational to pick a color space that fits the to the special problem.

A lot of methods operate with the CIE Lab or L*a*b* color space. L stands for the lightness from black (0) to white (255), a depicts from green (0) to red (255) and b

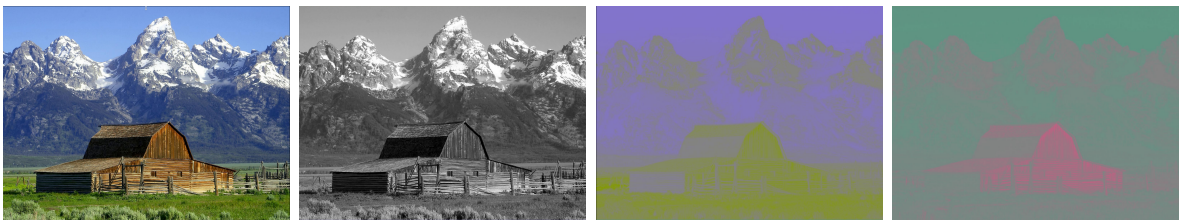


Figure 3: Example of the channel separation in YCbCr [38]

from blue (0) to yellow (255) within an 8-bit color space. Figure 2a shows the color space from the top into to ab-dimension, which is widely used as target space. One of the most important characteristics of the CIE Lab color model is that it is perception related. Colors are defined such that the space between two values is equivalent white as a human perceives it.

The digital YCbCr color space and its analog pendant YUV were developed for PAL and NTSC video encoding. YCbCr is the digital variant which is for example used by JPEG and MPEG. It consists of a luminance channel Y and two chrominance channels Cb and Cr. Cb describes color from yellow to blue and Cr from turquoise to red. It is used since most of the entropy is encoded in the luminance channel and the Cb and Cr channels have lower frequency. Figure 3 exemplifies the YCbCr colors pace.

In Figure 2b the distributions of the different color channels in CIE Lab and YCbCr. The distributions were calculated over the first 1000 images of the ImageNet [15] ILSVRC-CLS-LOC 2017 validation dataset.

2.3 Scribble-based Colorization

Scribble based methods are the oldest and focus on propagating sparse user scribbles to the entire gray-scale image [35]. Those scribbles can be color strokes or points.

The method of Levin et al. [3] is based the premise that neighboring pixels that have similar intensities should have similar colors. They then build a quadratic cost function and optimize the resulting equations with standard techniques. Their method can also be used on videos. To identify neighboring pixels over time they use optical flow.

Qu et al. [7] and Luan et al. [8] use the textures besides the neighboring feature to propagate colors.

The scribble-based methods suffer from color bleeding. Huang et al. [4] are using an

Figure 4: **Scribble-based Colorization:** The Results of Levin et al. [3], Left: gray-scale image, Middle: Result of their algorithm, Right: Ground-truth

adaptive edge detection algorithm to get clear edges.

Since the creation of the scribbles is most often a thing of experience, their usage is error prone and can't be used in a highly automated environment.

Lately Zhang et al. [34] showed a user guided deep learning approach. They are mapping color points and and a gray-scale image to a colorized image. They train on large-scale data and simulate the color hints while training. To make colorization easier they also propose a system that helps users to create color hints.

2.4 Example-based Colorization

Exemplar based colorization models generate a colored image from a gray-scale image and a colored reference. The gray-scale image provides the semantic, the edges and textures and the reference is used to decide the multimodal coloring problem. It aims to manually determine the colors of specific objects. Some approaches require that the reference image has the same semantic as the target image.

Welsh et al. [2] and Reinhard et al. [1] transfer colors between images by using simple statistical analysis to impose one image's color characteristic to another. They use the CIE Lab color space, keep the lightness channel of the target and add the a and b channel with colors from the reference. Welsh et al. enhance this method with user provided matching areas. These approaches were dissatisfying since they ignored areas on the target which did not match the semantic of the reference.

Since then different techniques were tested to improve the matching between reference and target which are based on regions after segmentation [5] [6] [9], super-pixels and pixels. Newer works use features gained from deep learning and are closer to the idea of style transfer [17]. The work of Liao et al. [33] is not only focused on colors but

also textures and style. They generate colors but may also alter the semantics of the target. He et al.[31] instead focus on only transferring the color. The latest work of He et al. [35] combines model-based colorization with exemplar-based colorization. They show an architecture that can select and propagate colors from the reference where it matches the content of the target and otherwise predict colors learned from large-scale data. To achieve this, they first measure the semantic relationship between the reference and the target. Their input to the model is the lightness channel of the target, similarity maps to the reference and the transformed ab channels of the reference. They train with two weighted loss functions: Perceptual loss [25] and smooth L1 between target ab and predicted ab channels. Additionally, they make it easier to find color references and present an algorithm to retrieve related images from a database.

2.5 Automatic Colorization

Modern approaches to colorization often aim to provide fully-automatic colorization frameworks. This seems to be a direct result of the breakthrough of deep-learning. Deeper models allow to lay more abstraction into a neural network. Model-based colorization means that a deep neural network can predict the color to a monochrome input without any additional information. The model can infer all context from the gray-scale input and hallucinate plausible colors.

Colorization using neural networks came up with the advent of deep convolutional neural networks and the availability of large-scale image databases through the internet. A deep convolutional neural network (DCNN) learns feature representations from large amounts of images. Those features range from low level textures to higher abstract visual concepts. Colorization uses the possibility to infer global or regional color context from the features. The global context of image sub-region is a good predictor for the color of the sub-region.

Special attention needs to be payed to the problem of predicting more than one plausible color map per image. The character of models is that during their training they get to learn from pairs of input and ground truth. Since the model fits to those explicit pairs, this makes it difficult to allow the model to predict plausible colors. Different attempts were made to counter that. [29], [30] and [26] are predicting distributions over the color space. Table 1 gives an overview over the different approaches. The following subsections describe what they did.

Name/Ref.	Model	Color	Loss	Multi	Dataset
LTBC [23]	CNN	Lab	L2 + class CE	N	MIT places
LRAC [26]	CNN	Lab + LCh	KL divergence	N	ImageNet
CIC [29]	CNN	Lab	CE	N	ImageNet
cGAN [24]	GAN	YUV	Adv + L1 + PatchGAN	Y	LSUN
PixColor [30]	PixelCNN + CNN	YCbCr	CE + L1	Y	ImageNet

Table 1: Automatic Colorization overview

2.5.1 Colorization as Classification

To tackle the inherent problem of desaturated color predictions Zhang et al. [29] re-frame the colorization problem from a regression to a classification task with rebalancing. They use the Lab color space and divide the ab-2D-space into 313 bins. Their model is an image-to-image CNN and it predicts a multinomial distribution for each pixel. During training they rebalance the color space by weighting the loss of each pixel according to the rarity of the ground truth color bin. The model is trained using cross-entropy loss. The model is not trained in an end-to-end fashion since the final color prediction is not the mode of the distribution but a value found by a method they call *annealed-mean*.

The rebalancing shows promising results. The color tone is far less dominated by sepia tones then in previous works of color predictions with DCNNs. The model predicts vivid colors, though they sometimes do not fill whole objects. Despite the re-frame of colorization as classification, they do not solve the one-to-many color prediction problem. Figure 5 shows the best images created with the rebalanced classification model.

2.5.2 Colorization with Regression

In 2016 Iizuka et al. [23] improve colorization by combining global and local features. The basic concept is having a network that convolves and reduces the spatial dimension down to 1/8th of the input and then splitting into local features and 2 more blocks of convolution and fully-connected layers down to a fixed length high-level representation. This high level 256-dimension feature vector is then concatenated to the mid-level feature block. To get the spatial size the global feature vector is replicated HxW times, to each spatial pixel. Basically creating 256 new filters. Each filter has the same values in the spatial dimension. They then deconvolve up in several blocks. The architecture can take inputs of any size, since all but the global feature extractor is only convolutional. The

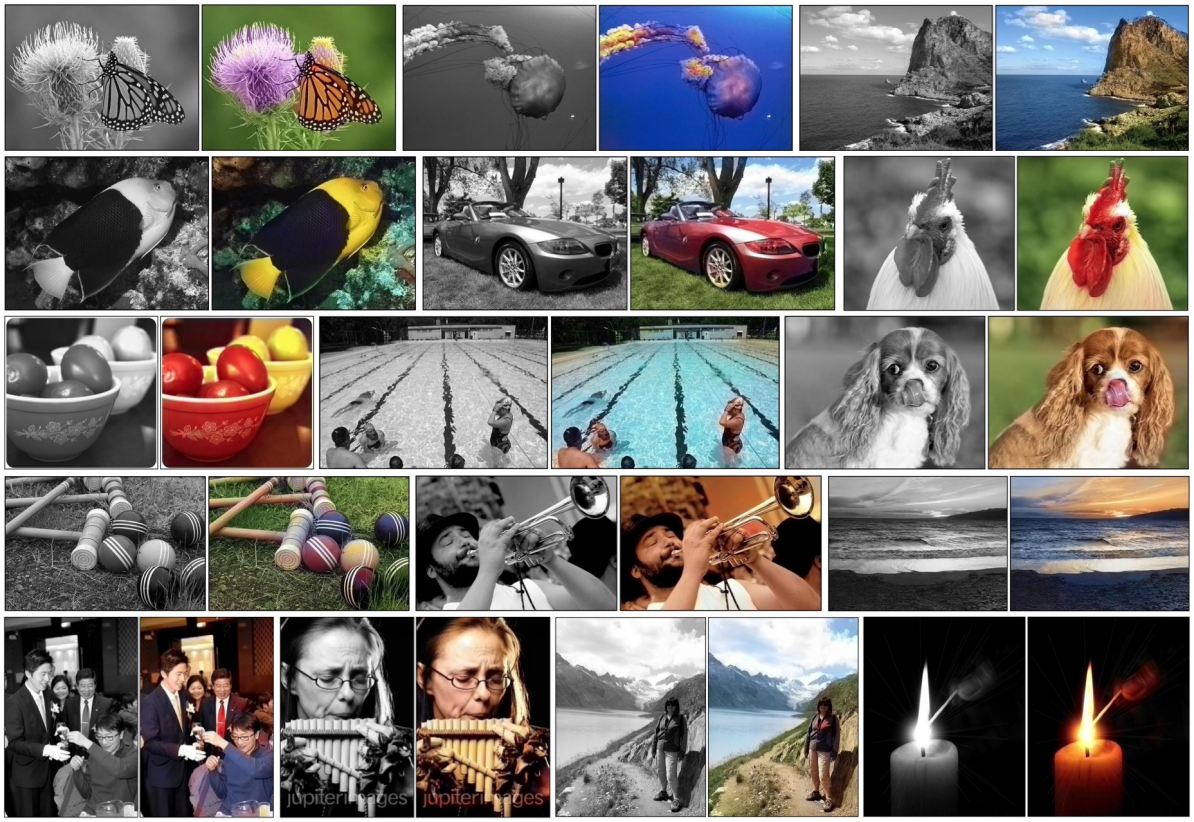


Figure 5: The best results of colorization as classification of [29]

fully-connected layer requires an input shape of 224x224. If the image has another size, the image is fed through the first part in two different sizes: first with fixed-resolution of 224x224 to extract the global features. Then with the original size and the global feature are concatenated to the mid-level ones. For performance reasons they only train with 224x224 pixel images.

The study shows that they can train joint on colorization and classification. A multinomial distribution for the classes is predicted from the global feature vector. This part of the network is trained with cross-entropy loss. The color part uses L2 loss. The model predicts the a and b channels in the CIE Lab space. The model is fed with the lightness channel during training.

The classification part of the model can keep up with a VGG-16 model. It reaches 50.6% Top-1 Accuracy instead of 52.8%. They show that taking the global features into account improves the understanding of the image context. In a version without global features they get incoherent results with blurry edges.

2.5.3 Colorization with Conditional Adversarial Networks

Isola et al. [24] use conditional Generative Adversarial Networks (cGAN) [14] as they study different image-to-image problems and also make progress on the colorization task. GANs do not need a special loss function, but rather learn to produce outputs that are indistinguishable from real data. A cGAN differs from GANs [12] by giving the generator an input label or image on which its output must be conditioned. The discriminator input also gets that condition. Its judgment must take that information into account. Practically for colorization this means: The generator gets the monochrome image and a random noise vector and generates the color information. The discriminator now takes the same monochrome image, the true color channels and the generated ones and must decide which one is real and which is fake. The combined objective for both subnetworks is for the generator to minimize the discriminators loss and for the discriminator to maximize it.

The objective function of the cGAN can be denoted as:

$$\mathcal{L}_{cGAN}(G, D) = E_{x,y}[logD(x, y)] + E_{x,z}[log(1 - D(x, G(x, z)))],$$

where G is the generator, D is the discriminator, x is the condition, y the output of the

generator and z is random noise prior.

Loss functions of colorization models are a critical part. L2 and L1 regression loss tend to produce desaturated images, since it is minimized by averaging all plausible outputs. The loss of a per-pixel regression or classification is unstructured which means the loss of a pixel does not depend on its neighbors. Inopposition the loss of a cGAN is structured. However Isola et al. find that the models achieve better results if they accompany the cGAN loss by a L1 loss over the output of the generator. This way the generator yields smoother results that do not differ so far from the ground truth. The resulting loss function is then:

$$G^* = \arg \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

As network architecture they use the same as [20]. For the generator they add skip connections between layers of same level on the encoder and decoder side, so that it is similar to the U-Net [21]. The discriminator is a simple CNN with four blocks which ends in a single sigmoid output. Neither generator nor discriminator use pooling layers. They achieve better results when they use a novel idea, they call PatchGAN for the discriminator. The discriminator does not judge the whole generated image at once, but in a sliding window fashion where it only compares NxN patches of the generated image and the ground truth. They find the patches of 70x70 yield the sharpest results while not leading to artifacts.

The final visual results for the colorization task are mixed. The cGANs can produce good results with vivid colors but have a failure modes where they give no color at all or very desaturated predictions.

2.5.4 Learning Representations for Automatic Colorization

2017 Larsson et al. present a new method [26] for automatic colorization. They acknowledge the one-to-many problem and train a "model to predict per-pixel color histograms" [26]. The model they use is a pre-trained VGG-16 and to link low- and mid-level features to predictions they use a *hypercolumn*. They draw the inspiration for their method from similar architectures used for semantic segmentation and edge detection.

For the input and output of their model they experiment with two color spaces: CIE-LCh and CIE-Lab. The input during training is the lightness they gain through $L =$

$\frac{R+G+B}{3}$ and a black and white image for inference. To predict distributions, they need to divide the output spaces into bins. They split the Lab axes into "evenly spaced Gaussian quantiles ($\mu = 0, \sigma = 25$)" [26] and encode them into two separate marginal distribution and one joint distribution. As loss function for Lab they use Kullback-Leibler-Divergence:

$$L_{hist}(x, y) = D_{KL}(y||f(x))$$

where x is the input the the model, f the model and y the the ground truth distribution. They extract the ground truth distribution by looking at the adjacent pixels of the center pixel.

For the CIE-LCh color space they "only consider marginal distributions and bin axes uniformly in [0, 1]". To mitigate the problem of unstable hue when chroma is near zero they weigh down that extremes. That results in the loss function for the hue/chroma experiments:

$$L_{\text{hue/chroma}}(\mathbf{x}, \mathbf{y}) = D_{KL}(\mathbf{y}_C||f_C(\mathbf{x})) + \lambda_H \mathbf{y}_C D_{KL}(\mathbf{y}_H||f_H(\mathbf{x}))$$

To evaluate and predict they post process the given histograms. For simple binned Lab they achieve the best results by using the expected value of the distribution. For the hue/chroma experiments they have to take another approach. For the chroma they take median of the distribution. For hue they calculate the expectation, but since the hue is a circular histogram it gets quite complex. The result is mapped to the [0, 1] interval. Again, they weigh the hue since it gets instable at extreme values when chroma is near zero. They train the neural network on resized images with at most 256 pixels in the smaller dimension, with stochastic gradient descent. As dataset they use ImageNet [15].

To compare results, they propose two metrics: the root mean squared error (RMSE) in the Lab color space and the peak-signal-to-noise ratio (PSNR) in RGB, like [22]. They achieve the best results with a model trained with the hue/chroma output and a post processing where they use a technique, they call chromatic fading.

They compare their results to the work of [29] and state that "the two systems are competitive in terms of quantitative measures of colorization accuracy". But that "their system, set to produce more vibrant colors, has an advantage in terms of human-measured preferences".

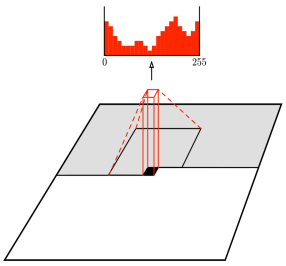


Figure 6: **PixelCNN:** predict a pixel by the histogram of the previous pixels [28]

2.5.5 PixColor: Pixel Recursive Colorization

The color information of a picture must not be as fine grained as the black and white structural information. On this hypothesis build Guadarrama et al. [30] a novel approach designing a complex method to predict colors. They train a conditional PixelCNN [28] to create low resolution color channels (28x28) and then resize them and refine them in a fully convolutional network, like a U-Net.

PixelCNNs are recursive generative models that predict one pixel after another. Figure 6 shows the principle of a PixelCNN. PixelCNNs are especially good for this purpose, since their output is multimodal and conditioned on a random prior during inference.

They use the YCbCr color space, which is also used by JPEG. The trained PixelCNN is conditioned on the brightness channel of the image. The extracted features from a ResNet-101 of the input image are fed through an adaption network and are then the prior to the PixelCNN.

The output of the PixelCNN network is a multinomial distribution over 32 color bins for Cb and Cr respectively and the loss is cross-entropy.

The refinement network is not trained on the output of the PixelCNN, but on down-sampled examples. This ensures that it does learn from good colors, since the quality of the PixelCNN varies. They take the architecture of [23] but add more layers. They train the refinement network with L1 loss.

The architecture allows to sample different versions of colors for a gray input image. This directly addresses the one-to-many nature of the problem. Sampling from a PixelCNN is a rather slow process, since it is a recursive algorithm which predicts one pixel after another. The idea of predicting 28x28 chrominance channels, then resizing and refining with a fast neural network, makes the concept attractive.

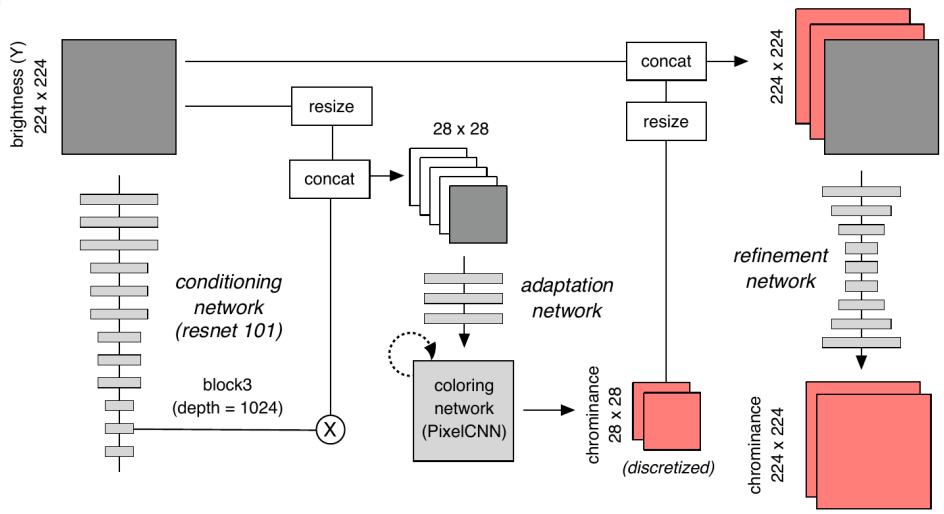


Figure 7: **PixColor:** Architectural overview [30]

After converting the results to the Lab color space, they compare the histogram intersection of the a and b channel over the ImageNet test set. They find that the average image created with their method has a higher intersection with the histograms of the test set then [23] [29] [26]. They conduct a Visual Turing Test (VTT) by showing colorized examples to test persons in the same way as [29]. A random result by showing twice the same image would be 50%. They reach values between 33.2% and 35.4% for different PixelCNN seeds. A version where a human "Oracle" picks the best result per image from different samples convinced in 38.3% of the cases.

3 Practical Work

To carry out the practical work Keras with the Tensorflow backend was used. Keras is a high-level neural networks framework that functions as an abstraction for several implementations of neural networks: Tensorflow, Theano and CNTK. It builds on the four guiding principles *user friendliness*, *modularity*, *easy extensibility* and it completely *works with Python* [16].

This section reports how a CNN was trained for the colorization task. It reports the results and also tries to compare these results to other methods.

The code for this project is available at: https://github.com/dketterer/colorization.

3.1 Datasets and Dataloader

A good first step to train a neural network is having a look at the data. Several datasets were used while working on this project.

The first prototype used the CIFAR10 set [11] which consists of 60 000 images representing 10 classes. Each image has a resolution of 32x32 pixels, which means the whole dataset fits easily in RAM and no data loader must be written. The CIFAR10 dataset is integrated into the Keras subpackage datasets from where it can directly be imported, and files are downloaded silently in the background. The objects in CIFAR all fill the complete image and the object context plays almost no role.

The Common Objects in COntext (COCO) dataset [13] was first released in 2014. It is a dataset used for a challenge around object detection, segmentation and image captioning. Objects are from 80 categories and the image sizes are not consistent. It comes pre-split into 118287 training, 5000 validation and 40670 test images. The advantage of COCO over CIFAR10 is, that a network trained on it can be easier visually evaluated than one from CIFAR10. Images are so large, that one can plainly recognize all the objects and assess the quality of the colors.

The ImageNet [15] ILSVRC 2012 dataset for the object localization challenge was used to train models extensively. The dataset was intended for classification and localization tasks and images show objects from 1000 categories. It has around 1.2M images in the training subset and 50000 in the validation subset.

Images in the COCO and ImageNet dataset are in their natural context, which is good for the colorization task. The neural network should learn to colorize objects according to the context. Validation examples in which an object is shown out of context may produce contradictory results.

To train a model, Keras offers the *fit_generator* function of the model class. It takes a Python-generator function as a parameter amongst other things. The generator must yield pairs of training examples and their ground truth. To efficiently load and preprocess the images a custom multiprocessing generator is written. The speed of training neural networks is often constrained by the IO operations and preparation of the data. Python multithreading is only suitable if high latency IO and low processing power is required. To gain real parallel processing, the multiprocessing library of Python can be utilized. Inter-process-communication and data transfer is established through queues.

The custom ImageDataGenerator class is composed of a coordinator process which takes a list of all image paths and makes batches of them which it puts into a queue to a worker process. A generator has around 6-12 workers; a good value is the number of available threads on the system. A worker waits until the blocking queue is not empty and dequeues a batch of image paths and prepares them for training. The worker puts the pairs of input and groundtruth into a final queue. The ImageDataGenerator in the main process has a function that can dequeue tuples from this queue and yield them for training.

3.2 Architecture

As architecture of the neural network a fairly popular design was chosen. The so-called U-Net [21] was first used for semantic segmentation of biomedical images. To the original architecture batch normalization [19] layers were added, which the original did not use. Batch normalization makes the training more robust and regularizes the network to reduce overfitting. They are added after each convolution and before dropout. Dropout was used in each of the encoder levels between the two convolutions with a drop rate of 0.2.

Upsampling was not done with transposed convolution (deconvolution), but with upsampling layers. Odena et al. [27] show that deconvolution layers produce checkerboard artifacts. This is due to uneven overlaps of the convolution kernels. The results of this network do not show such checkerboard artifacts.

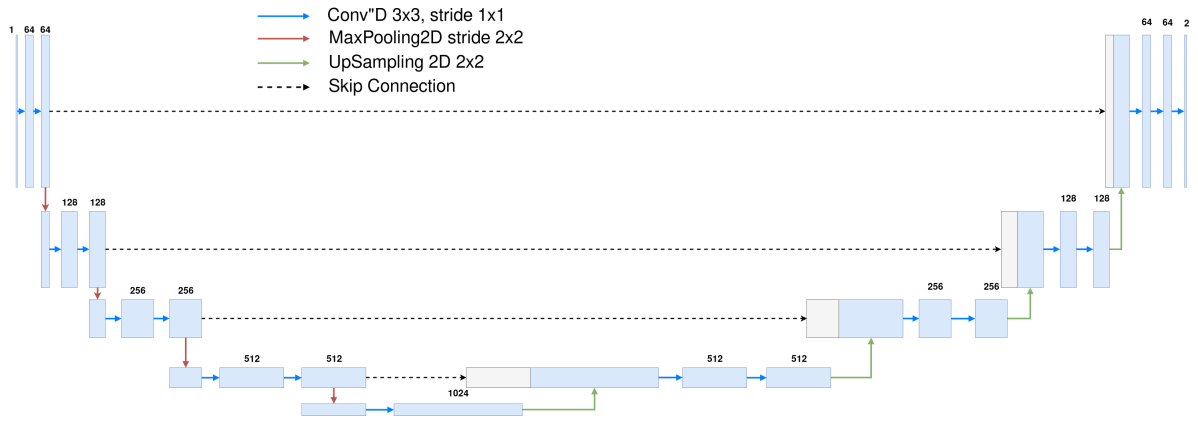


Figure 8: **U-Net Architecture:** The blue boxes correspond with the convolution filters, the number of filters doubles when going down a layer and is quartered in spatial resolution through max pooling. On top of the convolution layers the number of filters is written. The grey boxes stand for the "copied" filters that are reused during the upsampling phase. Operations are shown with arrows.

The output convolution layer was changed to two filters, which apply *tanh* activation. The whole network was initialized with a He uniform initializer [18]. Figure 8 shows the complete architecture, which has 31,389,634 trainable parameters. The network can take images of all sizes as long as width and height are multiples of 32.

3.3 Loss Functions

As loss functions L1-norm and L2-norm regression loss were used. Though they are known to deliver washed out and desaturated results, it is very easy to implement them. Both functions are applied to the output channels a and b separately and then added and divided by 2. The model is described through $Y_{\mathrm{HxWx2}} = f(X)_{\mathrm{HxW}}$ and learns the mapping $\hat{Y} = f(X)$. The loss functions are then mean absolute error:

$$\mathcal{L}_1(\hat{Y}, Y) = \frac{1}{4 * h * w} \left\| Y_{h,w,2} - \hat{Y}_{h,w,2} \right\|_1,$$

and mean squared error:

$$\mathcal{L}_2(\hat{Y}, Y) = \frac{1}{4 * h * w} \left\| Y_{h,w,2} - \hat{Y}_{h,w,2} \right\|_2.$$

3.4 Training

Finding the right training process and hyper-parameters was difficult since good metrics besides visual appearance are missing. It took a long time to find a bug which caused training always with batch size one. The loss falls anyway, but the results are only sepia colored images. Thats why the visualizations were one of the first things developed.

Codelisting 1: **Image Preprocessing**

```
import cv2.cv2 as cv
import numpy as np

def split_channels(image):
    lab = cv.cvtColor(image, cv.COLOR_RGB2LAB)
    l = lab[:, :, 0].copy()
    ab = lab[:, :, 1:].copy()
    l = np.expand_dims(l, -1)
    return l, ab

def preprocessing(image):
    image = image.astype('float32')
    image /= 128.
    image -= 1.
    return image

def worker(images, input_size):
    for img_path in images:
        img = cv.imread(img_path, cv.IMREAD_COLOR)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

        if len(img.shape) == 2:
            continue

        img = resize_keep_aspect_ratio(img, input_size)

        grey, ab = split_channels(img)

        grey = preprocessing(grey)
        grey = zero_padd(grey, input_size)

        ab = preprocessing(ab)
        ab = zero_padd(ab, input_size)

        grey_batch.append(grey)
        ab_batch.append(ab)
    if len(grey_batch) != len(images):
        continue
    grey_batch = np.array(grey_batch)
    ab_batch = np.array(ab_batch)
    return grey_batch, ab_batch
```

For image reading and preprocessing OpenCV and Numpy were used. Codelisting 1 shows in a compact way how preprocessing is executed. Some of the functions are not shown. All images are read with channels in BGR order, since it is OpenCV standard. Black-and-white images are filtered, and all images are then resized while keeping the aspect ratio. The images are coverted to the CIE Lab colorspace and then split into L and

ab channels before values are squashed to [-1,1]. After that images are are zeropadded to the joint input size. All images in a batch must have the same dimensions for efficient training.

The models were trained on an input size of 256x256 after some experiments. It was evaluated if larger or smaller size has an impact on the quality of results when images with higher resolution are used during inference. After training with 128x128 larger images showed a bit more artifacts. Since images with lower visual frequency, say fewer objects and more homogeneous qualitatively have better results, a even larger training input size could be advantageous. An alternative could be to train with batches of different sizes between 64x64 and 512x512. Training would have the same speed and the neural network could learn to combine features in different sizes.

The training of the neural network was carried out on a Nvidia GTX 1080 Ti. A training step with batch size 8 took 340ms, which means a whole epoch takes around 15 hours. The neural networks were trained for 4 epochs with a constant learning rate of 0.0003.

3.5 Inference

For the inference images need to be prepared as for the training. The input gray channel is squashed to the interval [-1, 1] and the whole image is resized. Firstly, for performance reasons, which means images larger then 1000 pixels on any side are resized while keeping the aspect ratio. And secondly to match the networks requirements of width and length. The new values $w*$ and $h*$ are calculated from the original sizes w, h as follows: $w* = w - (w \bmod 32)$ and $h* = h - (h \bmod 32)$.

The inference speed varies relative to the image size. On a laptop GPU Nvidia GTX 1050 Ti a small image with with 320x240 pixels take 90ms and larger images 640x480 pixels take 330ms. Performance gains can be made through increasing the batch size, but this is only possible if images have the same resolution, for example after scanning old photographs from slides.

3.6 Results

The colored results are as expected desaturated and washed out. Only a few images may have chance in a Visual Turing Test. Some of the best images are shown in Figure 9.

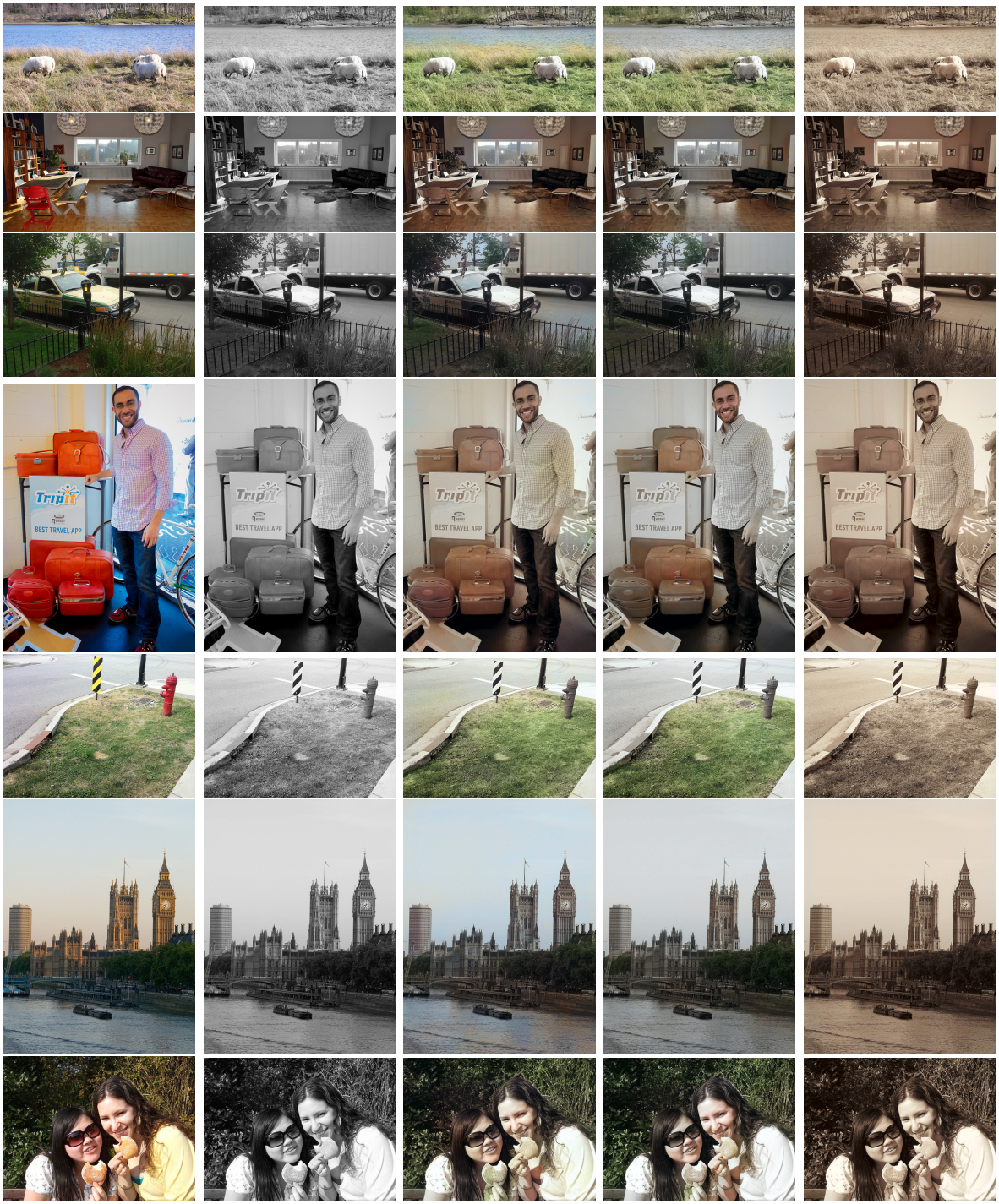


Figure 9: Example Result images. Selection was made by hand. All images are from the ImageNet validation set. The models used are U-Nets trained on 256x256 pixel input size, with L1 or L2 and Batch size (BS) 1 or 8.

Too Desaturated **Inconsistent Colors** **Edge Pollution** **Color Bleeding**

Figure 10: **Common Failure Modes:** *Too desaturated:* Colors are only close around the mean. *Inconsistent Colors* Colors are stained, one object does not have a consistent color. *Edge Pollution:* Artifacts at the image edges. *Color Bleeding.* Colors do not stay in the bounds of an object but bleed to neighboring objects.

	Training set		Validation set	
	B1	B8	B1	B8
L1	0.1195	0.0522	0.2826	0.056
L2	0.01	0.0089	0.0877	0.0128

Table 2: **Loss values after 4 epochs:** Training on the ImageNet training set. Validation on the ImageNet validation set.

Most images show common failure modes, that others already have reported. Figure 10 shows the typical failure modes “Desaturated”, “Inconsistent Colors”, “Edge Pollution” and “Color Bleeding”.

Table 2 shows the results after the 4 epochs of training. Larssson et al. [26] report a RMSE value of 0.318 for their L2-Lab model. The best value RMSE value for a L2 model achieved here, was with batch size 8: 0.1131. Since they do not show any visual results it is impossible compare the results qualitative.

In many fields of deep-learning it is possible to use ensembles of models. This is not possible for colorization tasks, because different models could give different colors for an object. Taking the mean for single pixels can produce all kind of problems and artifacts. Multiple models could be used to provide a selection of colored images to a user.

4 Future Work

One of the major problems of this method is that training starts from scratch. Future work should determine if pre-training the encoder part with ImageNet classification task, would improve the results. Training for colorization would start with a frozen encoder and a moderate learning rate and train only the decoder for 2 epochs or so. Then unfreeze all layers and train until the network converges.

Training could also be improved by using more data like the 9M images from the Open Images dataset [32]. Also, standard augmentation methods like random cropping, resizing and horizontal flipping could be used to increase the variety of training data.

For the colorization of video material, it is important that objects in subsequent images are colorized with the same color. Colorization is a multi-modal problem where possibly multiple results seem correct. It seems possible that a model predicts different color for an object when the background of this object changes. Future work should experiment with videos and models recurrent models.

5 Conclusion

It has been shown how a simple neural network for colorization can be trained. Colorization is a complex task where a simple model that is trained with L1- or L2-loss can achieve first results. To produce good color-images more advanced methods must be used. Recent publications show better results, but none of them is really outstanding yet. They made bigger steps through the formulation of better objective functions, that take the one-to-many problem of predicting plausible colors into account. Colorization with deep-learning benefited from the large amounts of data that is available for training.

Bibliography

[1] Erik Reinhard et al. “Color Transfer between Images”. en. In: *IEEE Computer Graphics and Applications* (2001), p. 8.

[2] Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. “Transferring Color to Greyscale Images”. In: *ACM Trans. Graph.* 21 (July 2002), pp. 277–280. DOI: 10.1145/566570.566576.

[3] Anat Levin, Dani Lischinski, and Yair Weiss. “Colorization Using Optimization”. In: *ACM SIGGRAPH 2004 Papers*. SIGGRAPH ’04. New York, NY, USA: ACM, 2004, pp. 689–694. DOI: 10.1145/1186562.1015780.

[4] Yi-Chin Huang et al. “An Adaptive Edge Detection Based Colorization Algorithm and Its Applications”. In: Jan. 2005, pp. 351–354. DOI: 10.1145/1101149.1101223.

[5] Revital Irony, Daniel Cohen-Or, and Dani Lischinski. “Colorization by Example”. In: *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*. EGSR ’05. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2005, pp. 201–210. ISBN: 978-3-905673-23-4. DOI: 10.2312/EGWR/EGSR05/201-210.

[6] Yu-Wing Tai, Jiaya Jia, and Chi-Keung Tang. “Local Color Transfer via Probabilistic Segmentation by Expectation-Maximization”. en. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. San Diego, CA, USA: IEEE, 2005, pp. 747–754. ISBN: 978-0-7695-2372-9. DOI: 10.1109/CVPR.2005.215.

[7] Yingge Qu, Tien-Tsin Wong, and Pheng-Ann Heng. “Manga Colorization”. In: *ACM SIGGRAPH 2006 Papers*. SIGGRAPH ’06. New York, NY, USA: ACM, 2006, pp. 1214–1220. ISBN: 978-1-59593-364-5. DOI: 10.1145/1179352.1142017.

[8] Qing Luan et al. “Natural Image Colorization”. In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR’07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 309–320. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/309-320.

[9] Guillaume Charpiat, Matthias Hofmann, and Bernhard Schölkopf. “Automatic Image Colorization Via Multimodal Predictions”. en. In: *Computer Vision – ECCV 2008*. Ed. by David Forsyth, Philip Torr, and Andrew Zisserman. Vol. 5304. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 126–139. ISBN: 978-3-540-88689-

1 978-3-540-88690-7. DOI: 10.1007/978-3-540-88690-7_10. URL: http://link.springer.com/10.1007/978-3-540-88690-7_10 (visited on 07/24/2019).

[10] Jiquan Ngiam et al. "Multimodal Deep Learning". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. USA: Omnipress, 2011, pp. 689–696. ISBN: 978-1-4503-0619-5. URL: http://dl.acm.org/citation.cfm?id=3104482.3104569 (visited on 08/16/2019).

[11] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto* (May 2012).

[12] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *arXiv:1406.2661 [cs, stat]* (June 2014). arXiv: 1406.2661 [cs, stat]. URL: http://arxiv.org/abs/1406.2661 (visited on 08/08/2019).

[13] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *arXiv:1405.0312 [cs]* (May 2014). arXiv: 1405.0312 [cs]. URL: http://arxiv.org/abs/1405.0312 (visited on 08/03/2019).

[14] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *arXiv:1411.1784 [cs, stat]* (Nov. 2014). arXiv: 1411.1784 [cs, stat]. URL: http://arxiv.org/abs/1411.1784 (visited on 08/08/2019).

[15] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *arXiv:1409.0575 [cs]* (Sept. 2014). arXiv: 1409.0575 [cs]. URL: http://arxiv.org/abs/1409.0575 (visited on 07/24/2019).

[16] Francois Follet et al. *Keras*. 2015. URL: https://github.com/keras-team/keras/.

[17] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style". In: *arXiv:1508.06576 [cs, q-bio]* (Aug. 2015). arXiv: 1508.06576 [cs, q-bio]. URL: http://arxiv.org/abs/1508.06576 (visited on 08/01/2019).

[18] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". en. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE, Dec. 2015, pp. 1026–1034. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.123.

[19] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]*

(Feb. 2015). arXiv: 1502.03167 [cs]. URL: http://arxiv.org/abs/1502.03167 (visited on 08/15/2019).

[20] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". en. In: *arXiv:1511.06434 [cs]* (Nov. 2015). arXiv: 1511.06434 [cs]. URL: http://arxiv.org/abs/1511.06434 (visited on 08/08/2019).

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *arXiv:1505.04597 [cs]* (May 2015). arXiv: 1505.04597 [cs]. URL: http://arxiv.org/abs/1505.04597 (visited on 07/24/2019).

[22] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. "Deep Colorization". In: *arXiv:1605.00075 [cs]* (Apr. 2016). arXiv: 1605.00075 [cs]. URL: http://arxiv.org/abs/1605.00075 (visited on 07/24/2019).

[23] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. "Let There Be Color!: Joint End-to-End Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification". en. In: *ACM Transactions on Graphics* 35.4 (July 2016), pp. 1–11. ISSN: 07300301. DOI: 10.1145/2897824.2925974.

[24] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: *arXiv:1611.07004 [cs]* (Nov. 2016). arXiv: 1611.07004 [cs]. URL: http://arxiv.org/abs/1611.07004 (visited on 07/24/2019).

[25] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". en. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Vol. 9906. Cham: Springer International Publishing, 2016, pp. 694–711. ISBN: 978-3-319-46474-9 978-3-319-46475-6. DOI: 10.1007/978-3-319-46475-6_43. URL: http://link.springer.com/10.1007/978-3-319-46475-6_43 (visited on 07/25/2019).

[26] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. "Learning Representations for Automatic Colorization". In: *arXiv:1603.06668 [cs]* (Mar. 2016). arXiv: 1603.06668 [cs]. URL: http://arxiv.org/abs/1603.06668 (visited on 04/26/2019).

[27] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and Checkerboard Artifacts". en. In: *Distill* 1.10 (Oct. 2016), e3. ISSN: 2476-0757. DOI: 10.23915/distill.00003.

[28] Aaron van den Oord et al. "Conditional Image Generation with PixelCNN Decoders". In: *arXiv:1606.05328 [cs]* (June 2016). arXiv: 1606.05328 [cs]. URL: http://arxiv.org/abs/1606.05328 (visited on 08/07/2019).

[29] Richard Zhang, Phillip Isola, and Alexei A. Efros. "Colorful Image Colorization". In: *arXiv:1603.08511 [cs]* (Mar. 2016). arXiv: 1603.08511 [cs]. URL: http://arxiv.org/abs/1603.08511 (visited on 04/23/2019).

[30] Sergio Guadarrama et al. "PixColor: Pixel Recursive Colorization". In: *arXiv:1705.07208 [cs]* (May 2017). arXiv: 1705.07208 [cs]. URL: http://arxiv.org/abs/1705.07208 (visited on 08/07/2019).

[31] Mingming He et al. "Progressive Color Transfer with Dense Semantic Correspondences". en. In: *arXiv:1710.00756 [cs]* (Oct. 2017). arXiv: 1710.00756 [cs]. URL: http://arxiv.org/abs/1710.00756 (visited on 08/01/2019).

[32] Ivan Krasin et al. *OpenImages: A public dataset for large-scale multi-label and multi-class image classification.* 2017. URL: https://storage.googleapis.com/openimages/web/index.html.

[33] Jing Liao et al. "Visual Attribute Transfer through Deep Image Analogy". In: *arXiv:1705.01088 [cs]* (May 2017). arXiv: 1705.01088 [cs]. URL: http://arxiv.org/abs/1705.01088 (visited on 07/25/2019).

[34] Richard Zhang et al. "Real-Time User-Guided Image Colorization with Learned Deep Priors". In: *arXiv:1705.02999 [cs]* (May 2017). arXiv: 1705.02999 [cs]. URL: http://arxiv.org/abs/1705.02999 (visited on 07/24/2019).

[35] Mingming He et al. "Deep Exemplar-Based Colorization". In: *arXiv:1807.06587 [cs]* (July 2018). arXiv: 1807.06587 [cs]. URL: http://arxiv.org/abs/1807.06587 (visited on 04/26/2019).

[36] *Definition of COLORIZE*. en. URL: https://www.merriam-webster.com/dictionary/colorize (visited on 08/13/2019).

[37] Holger Everding. *Der Lab-Farbraum von Oben (Blick Entlang Der L-Achse). Die Spektralfarben, d. h. Farben Maximaler Intensität in Einem Oder Mehreren Wellen-*

längenbereichen, Bilden Seine Äußere Hülle. URL: https://upload.wikimedia.org/wikipedia/commons/0/06/CIELAB_color_space_top_view.png.

[38] Mike1024 and Jon Sullivan. *YCbCr*. URL: https://upload.wikimedia.org/wikipedia/commons/d/d9/Barns_grand_tetons_YCbCr_separation.jpg.