Polychromify: Deep Convolutional Autoencoder for Fully-automatic Image Colorization

Davide Ghiotto 1236660
davide.ghiotto.2@studenti.unipd.it

1. Introduction

Image colorization is a interesting and highly debated problem. Hallucinating colors from a black and white image is a difficult task even for human and takes hours of expert and delicate work. It would be desirable to automate the colorization process in order to avoid any human intervention. Following the recent trends in the literature, I wanted to experiment with a particular deep learning model: the **autoencoder**. A special kind of neural network architecture that is able to compress the input data and decode it back while learning hidden features representation.
In this report I present *Polychromify*, a model that starting from the LAB color space of an image **reconstructs the** AB **channels** (colors) from an L input channel (grayscale). I measured the quality of the colorization process with three metrics: mean squared error (**MSE**, used as loss function), peak signal-to-noise ratio (**PSNR**) and structural similarity index (**SSIM**). However, colorization is an ill-posed problem with multiple acceptable answers: it is difficult to exactly quantify the quality of the colorization. It is still under investigation which metric is the most appropriate that can reflect human point of view.
I also experimented with another architectural version, called *PolychromifyAB*, where I split the learning task between two identical models: each model learned the mapping from the L channel to just one color channel (A and B). After training, I recombined the two separate predictions in order to obtain back the colorized image.

2. Related Work

Color points colorization

Initial attempts to the image colorization task relied on user defined color points. In this way the colorization process was guided by these color *hints* and then the algorithm tried to propagate the color and optimize it based on specific metrics. In particular, *Yatziv et al.(2006)*[2] algorithm was based on the concepts of luminance-weighted chrominance blending and fast intrinsic distance computations.

Example-based colorization

In order to reduce the user intervention, some works focused on colors statistics from a set of similar images to infer the colorization of an input image. This set can be obtained by a database search or through the internet. The performance of these methods is highly dependent on how similar the reference images are to the grayscale input. However, finding a suitable similar image is a non-trivial task and can be computationally expensive.

Learning-based colorization

In recent years, due to the resurgence of deep learning, deep convolutional neural networks have become the main approach to learn color prediction from a large dataset (*e.g., ImageNet*[1]). *Zhang et al.(2016)*[3] proposed a fully automatic approach, called **ECCV16**, that produces vibrant and realistic colorization. They addressed colorization as a classification task and used *class-rebalancing* at training time to increase the diversity of colors for the predicted results. The system is implemented as a feed-forward pass in a CNN at test time and is trained on over a million color images. Later on, again *Zhang et al.(2017)*[4] proposed an hybrid solution, called **SIGGRAPH17**, where they still relied on a CNN framework but they also integrate user inputs in the form of color *hints*.

My approach

The approach I took is similar to the learning-based colorization in the sense that I trained a deep learning algorithm with a dataset in a fully automatic way. However, it differs from *Zhang et al.(2016)*[3] because I setup the colorization problem as a regression task: instead of predicting only one of a finite number of color classes, *Polychromify* predicts one continuous value of the full range of color spectrum. It also differs from *Zhang et al.(2017)*[4] because my algorithm is not based on any user input, only the image channels.

3. Dataset

For this project I used a limited version of the original *ImageNet* dataset, called **Tiny ImageNet 200**[1]. This dataset is a subset of the original one and it contains roughly 120 thousands images, of size 64 by 64 pixels, divided into 200 classes. In Fig.1 I reported a sample of the dataset.

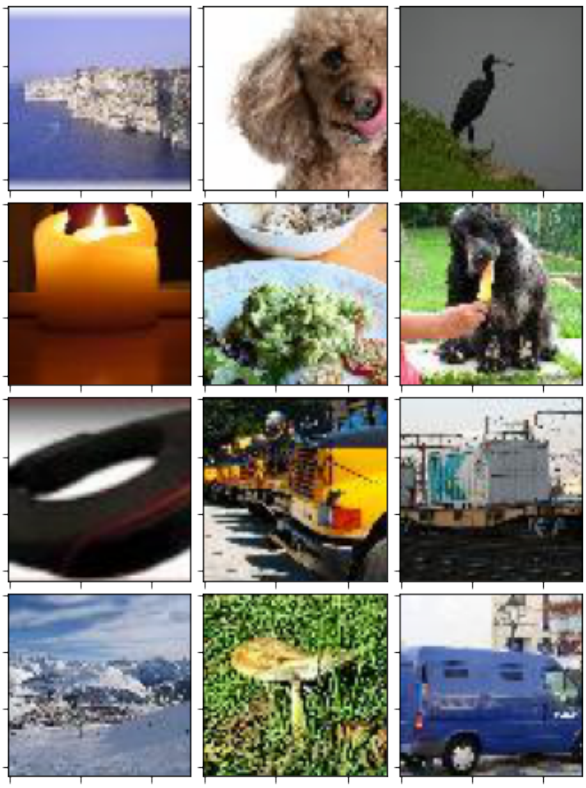

Figure 1. Sample images from Tiny ImageNet

I chose this dataset because of its quality: all the images are of the same fixed size and already split into training, validation and test set. Moreover, it has a high number of training samples and it is used for many other image-related tasks in deep learning. This fact allowed me to make direct comparisons with previous related works (e.g. with *Zhang et al. 2016*[3] and *Zhang et al. 2017*[4]).
As a preprocessing step, I converted each image from RGB to CIELAB[2] (also called *LAB*) color space. In this new space, designed to approximate human vision, the L represents the *perceptual lightness* (*e.i. grayscale*), while A and B describe the two other spectrum of color, *red-green* and *blue-yellow* respectively. With this clear division I could separate input (L) from output (AB) data.
Finally I scaled the AB features to the range $[-1, 1]$: this normalization step was done to boost the performance of network training because the output now aligns with the activation function (*hyperbolic tangent*) range of the output layer, avoiding gradient saturation.
I did not apply any data augmentation technique because the dataset was already too big for the available resources. I could solve this problem implementing online data augmentation but the dataset size ensured I had enough training data and redundancy for my models to generalize with.
My setup follows the same dataset split as proposed for the Tiny ImageNet dataset:

- training set (83%): 100k images
- validation set (8.3%): 10k images
- test set (8.3%): 10k images

4. Method

As already mentioned in the section 1, in this project I wanted to experiment with **deep autoencoders**. In particular, I developed a main model called *Polychromify* that given a input L channel (grayscale) maps to an output corresponding to the AB channels (colors). However, this model starts from one channel and produce two channels an output. In order to simplify the learning task I then tried to divide the training into two separate models: *PolychromifyA* learns to map from L to A; *PolychromifyB* learns to map from L to B. I finally merged the outputs from the two models and I called this ensemble *PolychromifyAB*.

Polychromify

Polychromify is the main model of my project and its architecture is composed of different layers. The diagram in Fig.2 provides a visual reference of the model, that can be summarized, in order, as follows:

- **Input layer**: the input is a 3D tensor of shape $64 \times 64 \times 1$: it is simply the L channel of the image.
- **Encoder**: it is composed of two parts: the *convolutional layers* dedicated to feature extractions and the *dense layers* specialized on increasing the hidden representation power of the network.
  - **Convolutional layers**: I applied 3 spatial (2D) convolutions with 64-128-256 filters respectively, fixed kernel size of 3 and *strides* of 2 each. This operation performs a sliding window over the image, enabling the model to extract visual local features. The *strides* progressive decrease the size of the image.

[1] `http://cs231n.stanford.edu/tiny-imagenet-200.zip`
[2] CIELAB: `https://en.wikipedia.org/wiki/CIELAB_color_space`

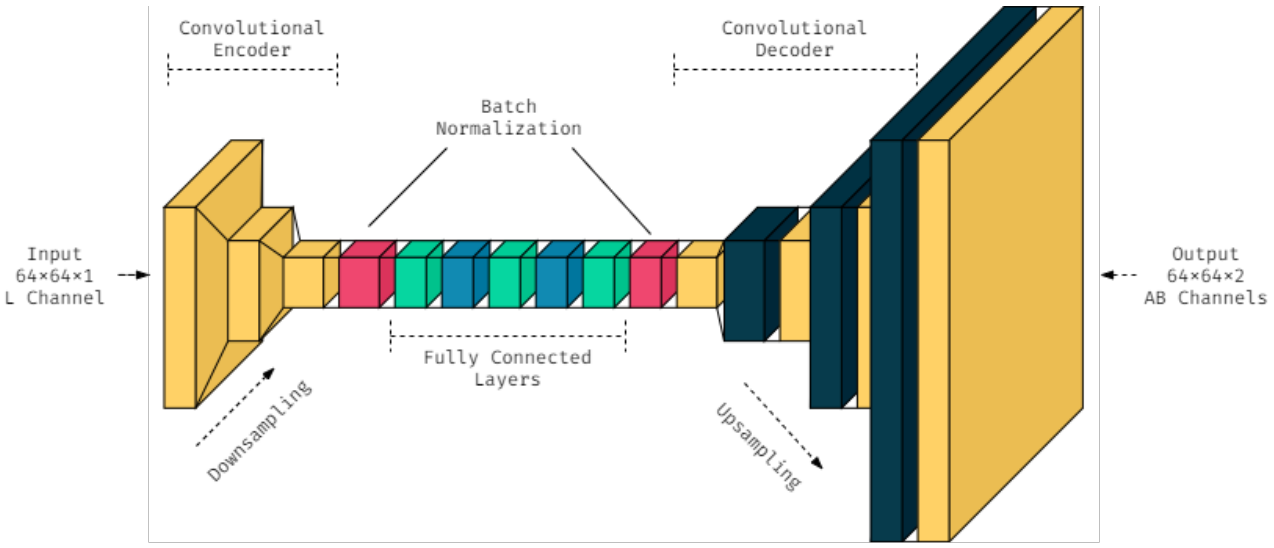

Figure 2. *Polychromify* Architecture

  - **Batch Normalization layer**: a normalization layer before the series of convolutional layers. Useful to speed up training and avoiding saturation of the gradient.
  - **Dense layers**: I applied 2 *fully-connected* layers with 128 and 64 neurons respectively and with *relu* activation function.
  - **Dropout layers**: I applied 2 *dropout* layers, each with probability 0.5, between the *dense* layers in order to reduce overfitting.

- **Decoder**: it is just a mirrored version of the *encoder* with first *dense layers* and then *convolutional layers*.
  - **Dense layer**: one *fully-connected* layer with *relu* activation function and 128 neurons.
  - **Batch Normalization layer**: a normalization layer before the series of convolutional layers. Useful to speed up training and avoiding saturation of the gradient.
  - **Convolutional layers**: I applied 3 spatial (2D) convolutions with 256-128-64 filters respectively, fixed kernel size of 3.
  - **Upsampling layers**: I applied 3 *upsampling* layers between the *convolutional* layers in order to revert the output back to the correct resolution of 64×64 that was decreased by the *strides* on the *encoder convolutional* layers.

- **Output layer**: the final layer is just a *convolutional* layer, with *tanh* activation function, composed of two filters that map to the desired output of shape $64 \times 64 \times 2$. More precisely, the output represent the two channels A and B that incorporate the color spectrum to be reconstructed.

- **Loss function**: I used the *mean squared error* (MSE) as loss function to express as better as I could the error of the colorization process. More precisely, the MSE is calculated comparing the groundtruth AB channels to the predicted ones: we are measuring the error on a $64 \times 64 \times 2$ continuous set of values.

I trained *Polychromify* for 25 epochs, with *Adam* optimizer, fixed *learning rate* and a *batch size* of 64, and kept track of the validation loss (MSE) and validation PSNR. Choosing these metrics, along with SSIM, allowed me to make direct comparison with other related work (*e.i. Zhang et al.(2017)[4]* that also implemented them.

PolychromifyAB

PolychromifyAB is an ensemble of two distinct models: *PolychromifyA, PolychromifyB*. Each model focuses only on a single channel and their architecture is identical to the one of *Polychromify* except for the *output* layer that, in this case, has just one filter, correctly mapping to the desired shape of $64 \times 64 \times 1$. The model diagram is summarized by Fig.3. I trained both models of *PolychromifyAB* with the same hyper-parameters as for the unified model, except that I trained for 30 epochs instead of 25 because the separate models were not overfitting as fast as *Polychromify*. During training I kept track of the PSNR and MSE loss, but for the final results I computed them after merging together the two predicted channels.

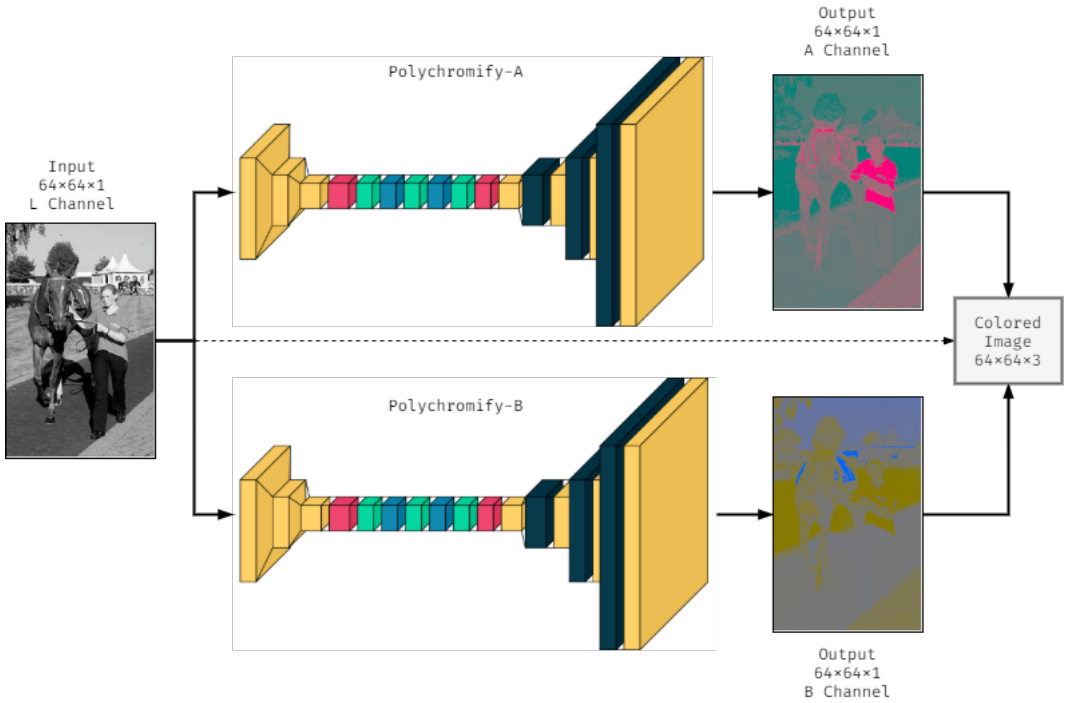

Figure 3. *PolychromifyAB* Architecture

5. Experiments

Model selection

In order to select the best configuration for my deep autoencoder model I validated some hyper-parameters and tested different options and values. Here I report the most significant ones:

- Size of network (S):
  - S1:
    - convolutional: $[32-64-128-128-64-32]$
    - dense: $3 = [128-64-128]$
  - S2:
    - convolutional: $[32-64-128-256-256-128-64-32]$
    - dense: $5 = [128-64-32-64-128]$
- Strides & UpSampling (S/U): $\{Yes|No\}$
- Epochs (E): $25-50$
- Batch Normalization (BN): $\{Yes|No\}$
- Dropout (D): $\{Yes|No\}$

In the Tab. 1 are reported a compact subset of the most representative results of the validation phase. The logic of this model selection is quite simple. I started with a simple model with just a basic structure (v1). It did not perform bad, but it took a lot of time to train. Then I tried to increase the network power (v2) adding more convolutions or more dense layers. However, the performance were worse and the training time was becoming prohibitive (in a project context). I revert back to a much simpler structure, adding strides and upsampling (v3): the image it is now resized and upscaled, saving a lot of computation in between. This model performed just like the first, but was trained in a fraction of time. Moving on, I tried to speed up even more the training adding batch normalization layers (v4): I recorded a minor reduction with slightly improvements on PSNR. Next, I trained for 50 epochs, but it clearly overfitted after 30/35 epochs. Lastly, I test the configuration a regularized version, with dropout layers (v6): improved a bit the generalization ability of the model. In order to better visualize the effects of the regularization, the Fig.4 displays the training and validation PSNR of two versions: the blue and

	S	S/U	BN	E	D	PSNR	Time
v1	S1	N	N	25	N	20.56	5h 50min
v2	S2	N	N	25	N	20.18	6h 15min
v3	S1	Y	N	25	N	20.55	45min
v4	S1	Y	Y	25	N	20.62	42min
v5	S1	Y	Y	50	N	19.54	85min
v6	S1	Y	Y	25	Y	**20.66**	42min

Table 1. *Polychromify* hyper-parameter tuning

orange lines refers to the overfitted version (v5), whereas the green and red describe the trend of the final version of *Polychromify* (v6).

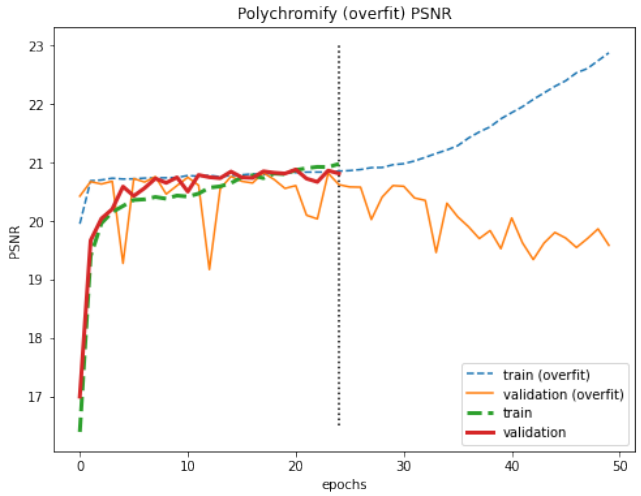

Figure 4. Polychromify History

On the other hand, *PolychromifyAB* did not undergo a model selection step, but was just another experimental architecture I wanted to try out. In fact, its architecture is completely identical to *Polychromify*, just adapting the output layer.

Results

In the Tab 2 are reported the PSNR and SSIM values for the final implementation of *Polychromify*, *PolychromifyAB*, ECCV16 and SIGGRAPH17 (both pre-trained). I also listed a simple CNN I used as a reference baseline.

Model	Params	PSNR	SSIM
CNN	10^5	21.5735	0.9061
Polychromify	10^6	21.9590	0.9056
PolychromifyAB	10^6	22.4440	0.9231
ECCV16	32×10^6	22.1403	0.9230
SIGGRAPH17	35×10^6	**23.9000**	**0.9334**

Table 2. Metrics results on test set

One important consideration is that these values of the PSNR and SSIM were calculated on the full images of the test set considering all 3 channels in LAB color space (also the L input channel). This is the reason why we have higher values compared to the ones during model selection.
Finally, the Fig.5 (last page) reports some examples of colorization results. The first column is the grayscale input, the last is the original image. In between, all the predicted colorization from different algorithms: (from left to right) *ECCV16 (Zhang et al. 2016)*[3], *SIGGRAPH17 (Zhang et al. 2017)*[4], *Polychromify*, *PolychromifyAB*. The colorization for ECCV16 and SIGGRAPH17 pre-trained models was possibile thanks to their public *API*[3].
Looking at the different outcomes we can see that ECCV16 and SIGGRAPH17 usually colorize better than my models. However, these results are not in line with the quantitive results for the PSNR (*PolychromifyAB* has higher PSNR than *Polychromify* and ECCV16). In my opinion, this is a metric problem: the computed PSNR for a given pair of images is not directly a measure of good colorization, but instead, a reconstruction error. One algorithm that predicts all sepia tones can actually (on average) be more correct than one who tries to guess more sophisticated colors. Considering the MSE loss is even clearer: we could reach a local optimum of validation MSE when we predict the average color for each pixel. This is one of the reasons why I think there is a gap between metrics and human evaluation.

6. Conclusion

In this project I proposed *Polychromify*: a simple deep convolutional autoencoder that is capable of automatically colorize images belonging to a wide variety of categories. This model is easy to implement and quick to train. However, the final results are not extraordinary compared to the state-of-the-art methods.

Metrics vs quality

As already mentioned in the results, I notice that the quantitative evaluation of this colorization task is not trivial: metrics and human point of views differ.

Working with limited resources

Being an exam project I had several constraints that impacted my methodology. First, I had limited time available: I had to carefully choose how to select one model architecture over the others to run more sophisticated tuning on it. Second, I had limited computational power. I used my personal machine and google colaboratory: this made me realise how heavy every extra layer in the network could be, in terms of computational needs. I then focused on optimizing training whenever possible (e.g. using *Batch Normalization layers*) without decreasing the power of the model.

Future extensions

In the future, it could be interesting to train *Polychromify* with more epochs and a larger dataset (*e.g., full ImageNet*): the generalization ability could increase while preserving its simple structure. Moreover, we could further investigate the reconstruction of an image as two separate learning tasks (one channel at the time), as *PolychromifyAB* does.

[3] https://github.com/richzhang/colorization

Figure 5. Colorization results

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. 2009.

[2] L. Yatziv and G. Sapiro. Fast image and video colorization using chrominance blending. 2006.

[3] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful Image Colorization. *ACM Transactions on Graphics*, 2016.

[4] Richard Zhang, Jun-Yan Zhu, Phillip Isola, and Xinyang Geng. Real-Time User-Guided Image Colorization with Learned Deep Priors. *ACM Transactions on Graphics*, 2017.