Polychromify
Deep Convolutional Autoencoder for Landscape Image Colorization

Davide Ghiotto † 1236660

github: https://github.com/davide97g/polychromify

I. 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, for example applied to historical photos. In this project I focused on nature landscapes, an image category which is very diverse and colorful, hence very challenging. Following the recent trends in the literature I wanted to experiment with a 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. Starting from the LAB color space of an image, the goal of this project is to build a model that is capable to reconstruct the AB channels (primary colors) from an input L channel (grayscale).

II. Related Work

A. 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)* [1] algorithm was based on the concepts of luminance-weighted chrominance blending and fast intrinsic distance computations.

B. 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.

C. 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* [2]). *Zhang et al.* [3] proposed a fully automatic approach 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 in the result. 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.* [4] proposed an hybrid solution where they still relied on a CNN framework but they also integrate user inputs in the form of color *hints*. However, *Su et al. (2020)* [5] observed that learning semantics at either image-level or pixel-level cannot sufficiently model the appearance variations of objects. They proposed an *instance-aware* colorization method where they first isolate the different subjects in the image (with a specific pre-trained model) and then apply some localized colorization with some context.

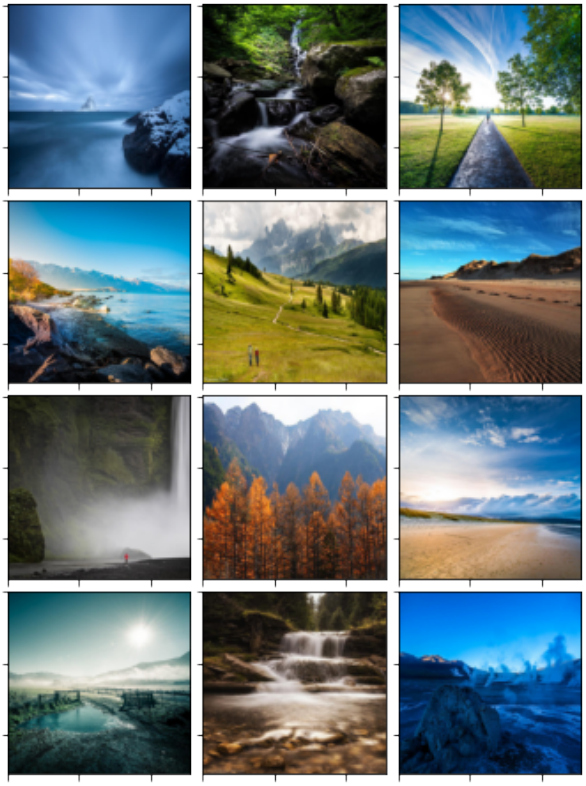

Fig. 1: Sample of the landscapes images dataset

†Department of Mathematics, University of Padova,
email: davide.ghiotto.2@studenti.unipd.it

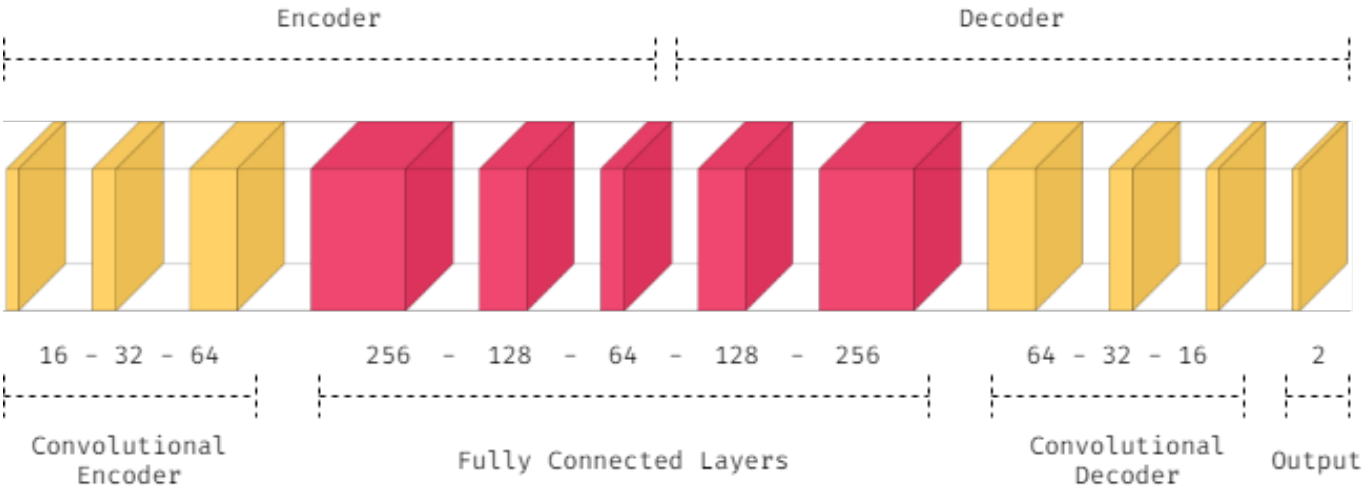

Fig. 2: *PolyChromify* Architecture

III. Dataset

For this project I used a specific dataset composed of 4300 landscapes images published on Kaggle[1]. This dataset is composed as follows: landscapes (900 pictures), landscapes montain (900 pictures), landscapes desert (100 pictures), landscapes sea (500 pictures), landscapes beach (500 pictures), landscapes island (500 pictures), landscapes japan (900 pictures). In Fig.1 I reported a (resized) sample of the dataset.
In order to correctly feed the data to the algorithm I needed to resize all the images to a fixed size of 128x128. Then I applied a conversion 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. 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.
My setup is defined by a dataset split of:

- training set (64%): 2752 images
- validation set (16%): 688 images
- test set (20%): 860 images

IV. Method

PolyChromify architecture is composed of different layers and the diagram in Fig.2 provides a visual reference of the model, that can be summarized as follows:

- **Input layer**: the input is a 3D tensor of shape $128 \times 128 \times 1$: it is simply the L channel of the image, resized to 128×128 pixels.
- **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 a series of spatial (2D) convolutions with increasing number of filters and with a fixed kernel size of 3. This operation performs a sliding window over the image, enabling the model to extract visual local features. There are 3 layers with 16-32-64 filters respectively.
  - **Dense Layers**: I applied a series of *fully-connected* layers with decreasing number of neurons and with *relu* activation function. There are 3 layers with 256-128-64 neurons respectively.
- **Decoder**: it is just a mirrored version of the *encoder* with first *dense layers* and then *convolutional layers*.
  - **Dense Layers**: I applied a series of *fully-connected* layers with increasing number of neurons and with *relu* activation function. There are 2 layers with 128-256 neurons respectively.
  - **Convolutional Layers**: I applied a series of spatial (2D) convolutions with decreasing number of filters and with a fixed kernel size of 3. There are 3 layers with 64-32-16 filters respectively.
- **Output layer**: the final layer is just a *dense* layer, with *tanh* activation function, composed of two neurons that map to the desired output of shape $128 \times 128 \times 2$. 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* loss to express as better as I could the quality of a reconstructed colorization of an image.

I trained *PolyChromify* for 25 epochs, with *Adam* optimizer, fixed *learning rate* and a *batch size* of 32, and kept track of the validation loss and accuracy.

[1]Kaggle dataset: https://www.kaggle.com/arnaud58/landscape-pictures
[2]CIELAB: https://en.wikipedia.org/wiki/CIELAB_color_space

V. Experiments

A. Model selection

- Vanilla AE (v0): vanilla autoencoder with just 5 *dense* (fully connected) layers: 3 for the encoder (128-64-32 neurons), 2 for the decoder (64-128 neurons).
- Convolutional AE (v1): to increase the power of the hidden state representation I used 2D convolutional layers instead of dense layers. This autoencoder has almost the same number of filters compared to number of neurons of the previous simple dense autoencoder (AE v0).
- Convolutional Dense AE (v2): to boost even more the hidden state representation power I added a dense architecture in the middle of the autoencoder.
- Very Deep Convolutional AE (v3): the previous models were not able to increase the training accuracy enough, and not even generalize of course. So I decided to add more convolutional layers and add strides of size 2 on the encoder side and upsampling on the decoder. In this way I hoped to encapsulate a local invariant strategy to produce color consistency.
- CNN: a basic simple CNN composed of 4 layers with 16-32-64-128 filters respectively and *batch normalization* layers between every convolutional layer.

Every *convolutional* or *dense* layer was implemented with *relu* activation function. Moreover, the output layer was always the same: a *dense* layer with 2 neurons and *tanh* activation function. The plot in Fig.3 shows us the history of the

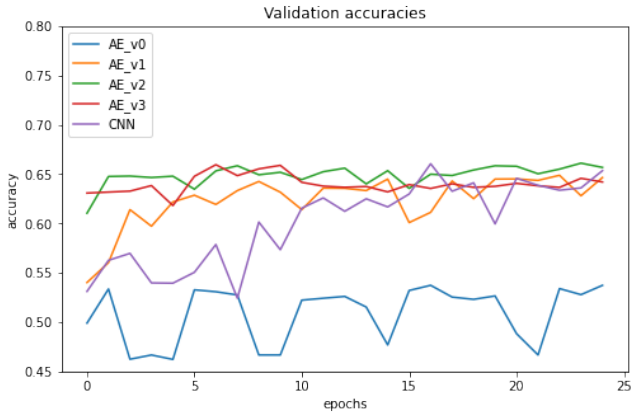

Fig. 3: Model selection

validation accuracies during training of the compared models. We can clearly see that the Vanilla autoencoder (AE v0) has a net gap from all the other models. The main reason is that a huge boost is given by the convolutional part of the other models. In fact, to demonstrate this behavior, I also trained a basic CNN (purple line) and even a simple model could perform quite as good as the other after some epochs.
The best architecture seems to be the version 2 (AE v2) where I added some extra dense layers to the convolutional baseline.

B. Hyper-parameters tuning

In order to select the best configuration for my model I validated some hyper-parameters and tested different options and values:

- Number of neurons:
  - $Low : [64 - 32 - 16 - 32 - 64](L)$
  - $Medium : [128 - 64 - 32 - 64 - 128](M)$
  - $High : [256 - 128 - 64 - 128 - 256](H)$
- Number of filters:
  - $Low : [16 - 32 - 64 - 64 - 32 - 16](L)$
  - $High : [32 - 64 - 128 - 128 - 64 - 32](H)$
- Strides & UpSampling (S/U): $\{Yes|No\}$
- Batch Normalization (BN): $\{Yes|No\}$

In the Tab. 1 are reported a compact subset of the most representative results of the validation phase. I run 5 different test

	Neurons	Filters	S/U	BN	Accuracy	Time
v1	L	L	N	N	0.6814	35min
v2	M	L	N	N	0.6792	52min
v3	H	L	N	N	0.6851	1h 5min
v4	H	H	Y	N	0.6727	1h 45min
v5	H	H	Y	Y	0.6527	1h 52min

TABLE 1: *PolyChromify* hyper-parameter tuning

with different configurations of hyper-parameters. Focusing only on raw accuracy the best model is version 3: high number of neurons, low number of filters, no strides, no upsampling e no batch normalization. Surprisingly, increasing the number of filters did not boost the performance. Even the configuration with strides and upsampling did not perform well: maybe the strides reduced too much the image resolution.

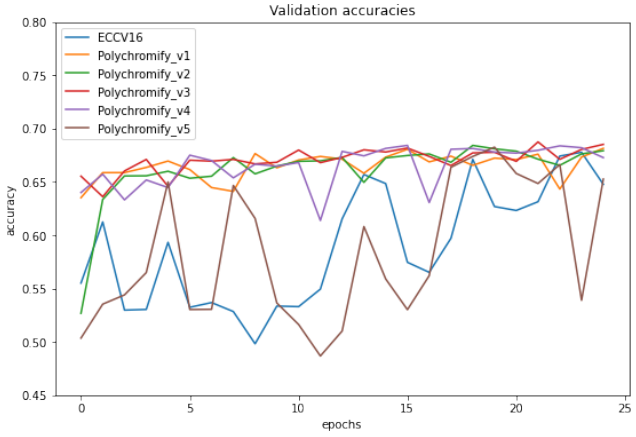

Fig. 4: History validation accuracy

In order to better visualize the training history the Fig.4 displays the five versions experimented in tuning phase. For comparison I also reported the training of *ECCV16* (*Zhang et al. (2016) [3]*) model. Visually we notice that model v1, v2 and v3 are slowly, but steadily, increasing, while v5 and ECCV16 are way more jumpy from one epoch to another:

the latter may required a larger batch size or more epochs to stabilize. So I tried to increase the number of epochs but it was not useful given the fact that the validation accuracy stabilizes after 30/35 epochs and we would risk to just overfit the more powerful models.
I also reported the training time[3] of every test because I found interesting to compare the different requirements: the first combination (v1) reaches a very good accuracy and only requires half the time of the best (v3).

C. Results

In the Tab 2 are reported the accuracy scores for the final implementation of *PolyChromify*, ECCV16 (trained from scratch on this project dataset) and the basic CNN.

Model	# Parameters	Time	Accuracy
CNN	98×10^3	2h 10min	0.6534
ECCV16	32×10^6	6h 40min	0.6400
PolyChromify	293×10^3	1h 40min	**0.6845**

TABLE 2: Accuracy results on test set

The best model in terms of accuracy seems to be *PolyChromify* but it was fine tuned on this specific dataset. ECCV16 was originally trained with a very large dataset and for more epochs: eventually, in the long term, it will almost surely outperform my model. However *PolyChromify* still performs quite good in relation to the training time needed and the number of parameters required.
Finally, the Fig.5 reports some examples of the colorization results. Using their relative *API*[4], both *ECCV16 (Zhang et al. 2016)* [3] and *SIGGRAPH17 (Zhang et al. 2017)* [4] produce more vibrant colors compared to *PolyChromify*.

VI. Conclusion

In this project I proposed *PolyChromify*: a fairly simple deep convolutional autoencoder that is capable of automatically colorize landscapes images. The final results are not extraordinary compared to the state-of-the-art methods: the colorized images present light gradient of colors, mostly ranging from blue to yellow. However, in the future, it could be interesting to train *PolyChromify* the same way as other state-of-the-art methods. With more epochs and a larger dataset (*e.g., ImageNet*) the generalization ability could increase while preserving its simple structure.

A. Limits of Autoencoders

This project allowed me to test several vanilla *autoencoders* variations. They are excellent models when it come to specific application like *denoising*, *data compression*, *data reconstruction*, but they are not well fitted for a problem like colorization. In fact, the bottleneck structure of an autoencoder might be good for compression purposes but do not reflect the aim of the colorization task: state-of-the-art methods start from a grayscale images and infer colors from learned hidden representation along with a prior image classification/segmentation. This is one of the reasons why my simple autoencoders, without any inception from other models, could not colorize complex, context-based images.

[3]Computation done on Leonovo Thinkpad T590 using CPU only
[4]https://github.com/richzhang/colorization

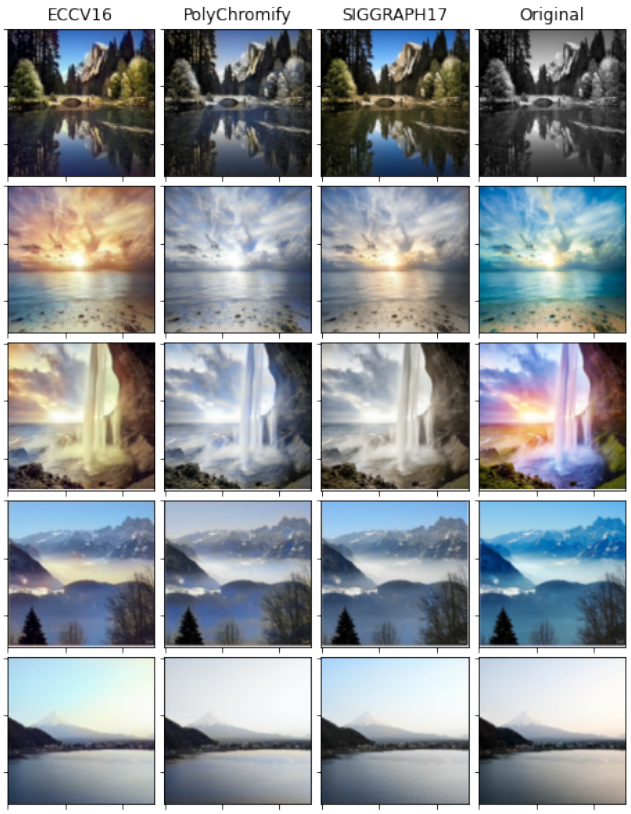

Fig. 5: Colorization results

B. Working with limited resources

Being an exam project I had several constraints that impacted my methodology. First of all I had limited time available: this was a problem because I had to carefully chose how to select one model architecture over the others in order to run more sophisticated tuning on it. Second, I had limited computational power in the sense that my personal machine could not perform neural network training exploiting GPUs: this made me realise how heavy every extra layer in the network could be, in terms of computational needs. I was then forced to experiment with smaller architectures, but, on the other hand, I could focus on optimizing training whenever possible without decreasing the power of the model.

C. Dataset is key

Concluding, during this project, I could appreciate how valuable is a good quality dataset. It is fundamental to the success of the entire development of the learning framework. I tried creating a dataset from scratch using a *webscraper*, but it was very difficult not to introduce redundancy, repetitions or mistakes (*e.g. images not related to a specific category)*.

REFERENCES

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

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," 2009.

[3] R. Zhang, P. Isola, and A. A. Efros, "Colorful Image Colorization," *ACM Transactions on Graphics*, 2016.

[4] R. Zhang, J.-Y. Zhu, P. Isola, and X. Geng, "Real-Time User-Guided Image Colorization with Learned Deep Priors," *ACM Transactions on Graphics*, 2017.

[5] J.-W. Su, H.-K. Chu, and J.-B. Huang, "Instance-aware image colorization," 2020.