Image Colorisation using Deep Convolutional Networks
Final Project Report

Paul Jacob
paul.jacob@polytechnique.edu

Timothée Darcet
timothee.darcet@gmail.com

Figure 1. Some of our colorization results

Abstract

The paper studied in this project presents an image colorization method through deep convolutional neural networks, outperforming state-of-the-art methods in this task when measuring using a user study. This result is achieved using a specifically designed architecture that computes features both at a global and local level and combines them, while being trained simultaneously for colorization and scene classification. For this project, we aimed to implement this architecture from scratch, and train it following the protocol of the authors to try and get results as close as possible to theirs. Even though the training process was complicated to engineer because we did not have access to as much resources as the authors, we managed to get very acceptable results, and consider our project to be a success in partially recreating the work of the authors.

1. Introduction

1.1. Let there be Color!

Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification (Iizuka et al. 2016 [3]) is an image colorization article published in 2016 in ACM Transactions on Graphics. The general idea of this paper is to use a CNN architecture (Fukushima 1988 [2], LeCun et al. 1998 [5]) to improve state-of-the-art methods on image colorization. While this idea was used before, namely by [1], their approach was less general, used less training data and had a narrower scope. On the contrary, [3] used a very general end-to-end training on a very large dataset, allowing them to get very good results on a variety of images.

1.2. Applications

While the authors do not dwell too much on applications of their work, we can note that it can find its use in historical works, for example to colorized old black and white photographs. Nowadays, it is less common to have only grayscale images as color cameras are commonplace, but derivatives of this model could be used for other applications where we only have one color channel, such as on infrared cameras or other low light devices. On the other hand, this model could be adapted to take as input the full color image, but to predict other channels, such as a depth channel, or a normal orientation channel.

2. Let there be color: the model

2.1. Architecture

The architecture (figure 2) was designed specifically for this task and can be summed up as

- The model computes a global feature vector using a

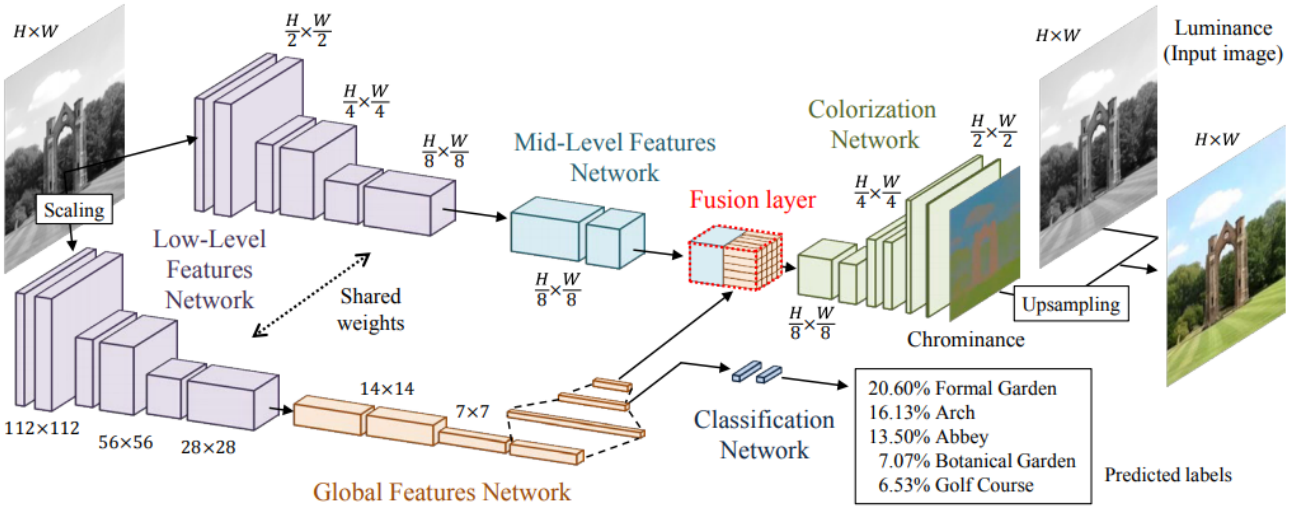

Figure 2. The architecture of the model

CNN + fully connected layers

- in parallel, local features are computed by feeding the image through a CNN, giving a smaller image of features.
- These two results are concatenated at each pixel.
- The image is then upsampled to original size via another CNN, outputting the color image.

This architecture allows the model to both get global information, in order to know what kind of scene it is dealing with (outdoors, snowy, inside a home...), and local info computed by convolutions (i.e. uniform light patch, this could be sky, leaf texture, this could be a plant...).
Finally, a small fully connected classification network is added next to the end of the global feature network, helping the training process by optimizing for classification of the scene.
We can note that the architecture is very regular, using the same patterns in each network. The main building block consist of a convolution with 3×3 kernel, padding of 1 to keep the same size, either a stride of 1 or 2, to reduce image size, and either the same number of output channels as the input, or the double. It is followed by a batch normalisation, and a ReLU nonlinearity.
Let us then go over each part of the model separately.

2.1.1 Low-level features

The low level feature network is the first part of the model, taking as input a grayscale image with only one channel. It is fully convolutional, thus allowing any size of image through it. It consists of alternatively putting stride 1 and stride 2 convolutional layers, in order to reduce the width and height and simultaneously increase the number of channels.
Two parallel pipelines are used to compute the mid-level features an the global features. Both pipelines use this low-level feature network, and share its weight. However, these two parallel networks do not always output the same image size. The mid-level feature pipeline is fully convolutional, thus enforcing no fixed size, so in this pipeline the low-level feature network outputs an image of size $\frac{H}{8} \times \frac{W}{8}$, with H, W the original dimensions of the image. The global feature pipeline, on the other hand, has to produce a single vector, so the low level feature network has to output an image of fixed size 28×28. In order for this pipeline to work correctly, the input image is then resized to a standard size of 224×224 before being fed into this pipeline.
This also means that when using an image already of size 224×224, the low-level feature network only has to run once and its output can be reused for both pipelines. This saves a lot of computing power, and thus it is important to try to use 224×224 images at least for training.

2.1.2 Global features

The global feature network takes as input a 24×24 with 512 channels, and uses convolutional layers then fully connected layers to get a single global feature vectors. This single vector is supposed to sum up the information contained in the image, and it is for that purpose that the global feature network is connected to the classifier, which helps it extract info about the nature of the scene.

2.1.3 Mid-level features

The mid-level feature network is very small, consisting of two layers that do not affect the width and height of the im-

age passing through it. Its purpose is to avoid connecting directly the low-level feature network to the colorization network: since the low-level network is trained to output features that serve two purposes (mid-level and global level), it is possible these features are not yet optimal to be fed into the colorizer. This small networks adds a few step where these features can be optimized before the reconstruction.
By adding a few more convolutional layers, this also enlarges the receptive field of the features outputted, allowing to compute more complex features.

2.1.4 Fusion layer

The fusion layer is conceptually very simple, yet it is what allows the model to combine the global and local information, arguably one of the reasons this model works well. The process is simple: at each pixel location, concatenate the vector of the pixel channels with the global feature vector, giving the new pixel channels. Then, feed this through a 1×1 convolutional layer. This last step is effectively the same as applying a single fully connected layer to the vector of channels at each pixel.

2.1.5 Colorizer

The colorizer network then reconstructs the color of the image through a serie of convolution and upsamplings. The upsamplings used are simply processes of doubling the resolution by using a nearest neighbor interpolation, while the convolutions progressively reduce the number of channels. Since the input is of size $\frac{H}{8} \times \frac{W}{8}$, with H, W the original dimensions of the image, a serie of three upsampling is sufficient to recover the original image size.
The goal being to reconstruct the color image, we could imagine outputting the full RGB image. However, we would have no garantee that the image would even resemble the original grayscale image. Since we already have the grayscale information, it is more efficient to only predict the chrominance information, and combine that with the original luminance information from the grayscale image. This implies choosing an appropriate color space separating chrominance from luminance. The paper proposes a comparison of using the YUV, RGB, and L*a*b* color space. They conclude that L*a*b* is the most suited to this task. The colorizer network then predicts the two channels a* and b* of the image.

2.1.6 Classifier

In order to help the training, a small classifier network is added next to the global feature network. It takes as input the activation of the second-to-last layer of the network, feeds it into a few fully connected layers, and outputs a classification score for the type of scene. Since the network is trained on a dataset featuring classifications of pictures into different scene types, we can use this info to compute a loss, and backpropagate this to the global feature network so that it learns to recognize scene, thus extracting information that is interesting for the colorization.

2.2. Optimisation

The loss used for the training is a combination of an MSE loss on the predicted a*b* channels, and cross entropy on the classification. The classification loss is scaled by a factor $\alpha = \frac{1}{300}$, chosen so that the two losses can be of similar magnitude. Finally, the loss is :

$$L(I_{pred}, I_{gt}, y_{pred}, y_{gt}) = MSE(I_{pred}, I_{gt}) + \alpha \cdot CrossEntropy(y_{pred}, y_{gt})$$

The training process used is a stochastic gradient descent with an Adadelta optimizer (Zeiler et al. 2012 [6]).

3. Our implementation

3.1. Differences

We made a few choices that were different from those in the original paper in our implementation.

First, we used Adam (Kingma et al. 2017 [4]) as our optimizer. we did a few tests to compare it with Adadelta and found better performance, so we stuck with it.

Secondly, we did not use batch normalization. Even though the original paper uses some, the description of where to put them and what parameters to use were unclear, and our model performed poorly when we included batch normalization.

We slightly modified the classifier network. The original architecture featured one hidden layer with 256 units, and an output layer with 205 classes. However, the version of the dataset we had featured 365 classes, so we thought it did not make sense to keep 256 hidden units (less than the number of classes). We therefore used 400 hidden units.

3.2. Details

We implemented the model in Python, using the Pytorch library and the wrapper Pytorch-Lightning. The code is structured as follows:

`main.py` Contains the training script, currently configured for multi-node training on a cluster, and an `argparse` CLI.

`model.py` contains the architecture defintion. The `ConvModule` and `FCModule` classes are the building blocks of the networks while `LTBC` is the full model. Following `Pytorch-Lightning` convention, the optimizer is defined there too in the `configure_optimizers` method.

`data.py` defines the dataset loading and preparation.

`show_examples.py` is a utility script to compute a few colorization results and display them.

`utils.py` is a bank of useful functions.

`submit.sh`, `launch_worker.sh` and `kill.sh` are driver scripts used to launch a multi node training.

3.3. Training

3.3.1 Training hardware

We first tried training using our PC and Google Colab, however the limited power available did not allow us to get good results besides easy outdoor images. At most, the network managed to learn that a uniform patch at the top of an outdoors image corresponded to the sky and was blue, but not much more. We then tried two other approaches to get more computing power: one using a Google Cloud Virtual Machine (VM), which featured better GPUs and a possibility of longer trainings, the other using distributed training over about a 100 less powerful GPUs. Even though the distributed training approach was promising, and even allowed us to train on the whole dataset in reasonable time, it wound up giving poor results for unknown reason, possibly related to the fact that distributing the training implies a much larger effective batch size. Since the Google Cloud VM training gave good results, we settled on this final model for the results that we present in this report.

3.3.2 Training details

To make the training possible in a reasonable time on a Google Cloud VM, we used a fraction of approximately 20% of the *Places365* dataset for our final training. That is, we chose 1000 images from each category, for a total of 365.000 images, and used a $80\%/20\%$ split for the training and validation. The training took 30h on a Tesla P100 GPU. We stopped the training at 14 epochs, at which the validation loss was best. The parameters used were $\alpha = 0.005$, Adam optimizer with a learning rate of 0.0001 and batch size of 32.

4. Results

4.1. Qualitative study

Here we evaluate our final model on test images that have never been seen during the training process (neither in our training dataset and validation dataset). Some of our results are reported in figures 3 and 4, with also the ground truth color image as a comparison. The 32 images presented here come from a completely random batch selection in the test data (we did this random selection so that we don't purposely choose images where the network did succeed best).

As one can see, in most cases, the model seems to capture general information such as the sky, the land, and the vegetation in outdoor scenes, and the clothes, faces and decoration in indoor portraits. See how in most cases, the vegetation is colored in green, the sky is colored in blue, or the sand is colored in light yellow. Also, local elements are generally speaking well retrieved. See for instance how the buildings, clothes and people are colored accurately.

Generally speaking, both indoor and outdoor scenes seem to be understood in a reasonable scope, and the model succeeds in injecting plausible colors in those regions. As a comparison with ground truth images, in most cases, the real images often contain more subtleties in the colors. However, when one does not have access to the real ground truth image, the artificially colored images look plausible by themselves.

4.2. Failure cases

In some cases, however, the network struggles to input plausible colors. We have noticed two general types of visual artifacts.

The first one, which is the most important, is what we refer to as the "sepia" effect. Sometimes, when the networks sees a difficult and specific image which does not looks like its training corpus, the output color is just a uniform "sepia-like" filter. See, for instance, the top-right corner image of the figure 3, or the second image of the first column of 4. During the first epochs of the training, this sepia effect is very present, and the only images that do not contain it significantly are easy outdoor images with a green grass foreground and a blue sky background. As the training process goes on, the sepia effect progressively disappears, and the images that still lead to a sepia effect are difficult image with a complicated structure.

The second artifact which happens sometimes is when the network fails in understanding a region of the image. Sometimes, the network colors what is obviously water or concrete ground in green, probably because it thinks that is corresponds to grass. As an example heren see in the bottom-left corner of figure 4, that the network does not succeeds in accurately detecting the pool.

5. Conclusion

As a conclusion, our work for this project has been to study, implement, train and test the deep learning method presented in the article by Iizuka et al. (2016) [3]. Although our computational power was limited, we have managed to reach plausible colorisation results for a various range of outdoor and indoor scenes.

For further enhancement, one could try to perform more finetuning, which is always time-consuming on deep learning pipelines. Moreover, additional ideas that we had include adding residual connections between convolutional

layers of the same dimension to retrieve more details, and adding adversarial losses in some way to ensure the plausibility of the resulting images, since the actual goal is not necessarily to produce the true colorization, but a plausible one.

References

[1] Z. Cheng, Q. Yang, and B. Sheng. Deep colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 415–423, 2015. 1

[2] K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988. 1

[3] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Let there be color! joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Transactions on Graphics (ToG)*, 35(4):1–11, 2016. 1, 4

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017. 3

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[6] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 3

Figure 3. Visual results on test images: Input Grayscale Image, Output Image, Ground Truth RGB Image

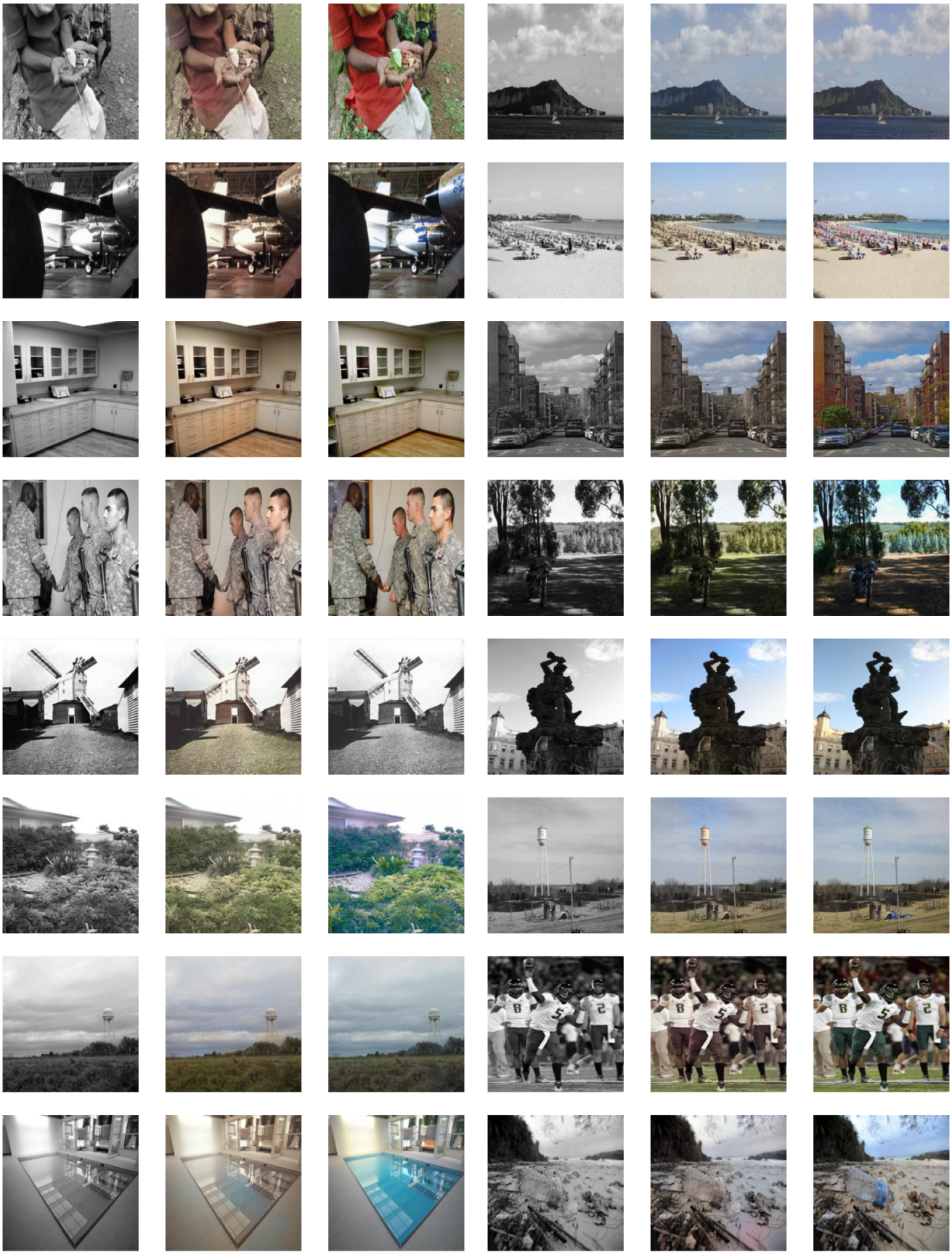

Figure 4. Visual results on test images: Input Grayscale Image, Output Image, Ground Truth RGB Image