AI Lab project: Colorizing images using PyTorch and common AIML techniques

Alessandro Candelori, Davide di Trocchio

July 11, 2022

Contents

1 Introduction

We aim to develop a model in pytorch for image coloring. Its main goal is to colour a never seen before, black and white image, ranging from old photographs, decolourized images, sketches and black and white media.
The task at hand is a regression model built with inspiration from models like the Image-to-Image Transaltion and Colorful Image Colorization. In particular, we took inspiration from the latter and got really similiar results to their regression models. As we continued to study further, we managed to find many other methods of coloring images, but we choose the simplest one, as the other needed too much control and very large datasets, that we could not include due to the nature of this project.
We stressed a classical software development approach as we wanted to study how we could deploy AI apps, as opposed to publish our research to a simple Google Colab page or a Jupyter Notebook. We like to think that we succeeded in creating a cool, user-level application that could be understood and used by everyone.

2 Method

As said in the introduction, we employ a ResNet regression structure to learn related information from a converted grayscale image. Each image is transposed to a LAB format before learning, extracting the features from the grayscale image (using the L channel).

2.1 LAB format and image processing

Before processing any image, we aim to bring them to a given resolution (256×256) and then convert them from the RGB format to the LAB one. This is done because the RGB format makes it more difficult to differentiate colors from the grayscale, whereas in the LAB format we have immediately, without computation, the Lightness channel and the color channels (AB) divided from the start.

Figure 1: The LAB color space. L for including lightness and AB for every other space

We will apply regression on the L channel and use the AB channels as the ground truth.

2.2 The net

We make use of the ResNet-18, which is an image classification network with 18 layers, of which we will only use 6 modifying the first one to accept grayscale images. The other layers are simple Convolutional, Batchnorm and upsample layers in the order presented inside the code.

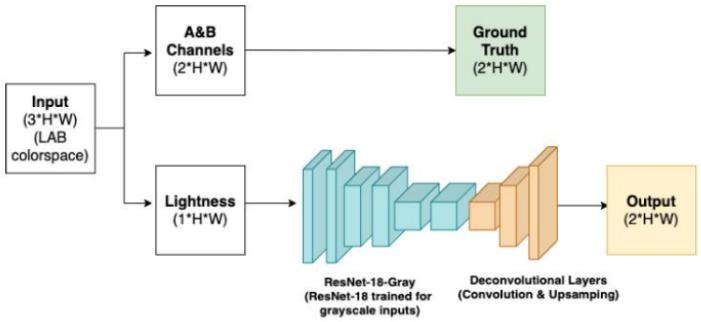

Figure 2: ResNet Model including image processing steps

2.3 Loss function

As the loss function, we work with the simple Mean Squared error. We try to minimize the squared distance between the color value we try to predict and

the actual ground truth color value.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

2.4 Activation Function

Since we're using ResNet18, our model is bound to use ReLu. We tried using newer techniques such as Mish, but after observing few results we can safely say that Mish performs too drastically on this model.

Figure 3: On the left the model using mish, on the right ReLU

This is because the Mish activates earlier than the ReLU, leaving us with more rash decisions.

2.5 Optimizer

We use the Adam optimizer simply because it is one of the most officient optimizer available to us.

2.6 The dataset

The dataset we used is made of 40,000 colored images in 256×256 format. Nearly every dataset made of images in whatever format works fine, as we then apply our transformations on it and derive the grayscale image. The ones we used are a combination of MIT Places, various kaggle datasets ranging from people to cars to forests and such. As said before, nearly every dataset provides useful in this situation.

2.7 Basic Code documentation

The scope of the project was to create an AI app through standard software development techniques. Inside of the project you'll find different files that are here described.

2.7.1 net.py

net.py Holds the core to train the entire application. It is made of two big functions, train and validate, which are then used when the model is created to train it. Each time we finish training an epoch, we validate it and store our loss. If the loss results lower than some other model, we create a "checkpoint" inside the same folder, in order to start working with that model in our frontend.py Running this python file will start training with the settings found in it.

2.7.2 grayscale.py

grayscale.py is a utility class which supers torchvision dataset imageloader. This is done because each image before being loaded needs to be split into L and AB, providing the ground truth and the black and white images we use to train our model.

2.7.3 frontend.py

frontend.py holds the core of the application. Using the streamlit python module, we create a simple web app which lets the end user upload a file and have it colored through a saved model. This is all done in real time speed and it also holds a simple cache which deletes the validated images once uploaded. It is runnable with:

```
streamlit run frontend.py
```

This brings the whole application to life as it is deployable to the internet to show a simple demo. Important! This file contains absolute paths and creates problems when working with relatives, as such, if you wish to run it, please consider updating all paths to include your preferred method.

2.7.4 colorization.py

Colorization.py holds the model used. It is a simple class which holds the 6 ResNet18 layers.

2.7.5 averagemeter.py

In averagemeter we have the AverageMeter class. This is a utility class often found in Ai projects for taking averages. In the program it is used to update the batch time and the loss time when training and validating the model.

2.7.6 dataset.py

dataset.py holds an utility class aimed towards helping organize an unorganized image dataset, which is often the case when working with datasets deriving from kaggle. Running it will simply create two folders following a 70/30 splitting scheme.

3 Results and ending

Given the nature of training the model on our own devices, our results were capped at about 25 epochs. It takes us, on average, about 10 hours to get this far, but we plan nonetheless to bring a presentation to the exam training up to the maximum.
What follows are some cherry picked results of some images we thought our model best colored.

Figure 4: On the left the ground truth, on the right the derived image

Figure 5: On the left the ground truth, on the right the derived image

Figure 6: On the left the ground truth, on the right the derived image

As we can see, the model does a pretty good job considering it trained so little. It has some problems determining where to put blues, which could be seen as an overfit, but we produced some good results nonetheless.

During the final presentation we will present a more trained model on even more cases, such as photographs, sketches and black and white media.

4 Credits and links

https://colab.research.google.com/github/moein-shariatnia/Deep-Learning/blob/main/Image%20Colorization%20Tutorial/Image%20Colorization%20with%20U-Net%20and%20GAN%20Tutorial.ipynb#scrollTo=mGqbGWUQhzmk https://haoyiq.com/project/eecs442/
https://arxiv.org/pdf/1611.07004.pdf
https://arxiv.org/pdf/1603.08511.pdf