Image Colorization

Markéta Minářová

December 31, 2022

1 Assignement

The goal is to create and implement models that take grayscale images as input and predict colourized images as output. The colourization will focus on art images - paintings, portraits, etc. The framework used for this semestral work is Pytorch.

Some subgoals need to be completed, specifically:

- create an appropriate art dataset,
- desing and implement ML models,
- evaluate the results on testing data.

2 Introduction

Image colorization is one of the many current challenges in image processing. Given a grayscale image, the goal is to estimate colours for all of the image's pixels. Deep learning techniques have been broadly used throughout the time and some of them resulted in pretty solid solutions. However, it is an ambiguous task sometimes as not all objects tend to possess one colour shade – for example, the colorization is quite straightforward when we are given a tooth - it will most probably be white (or some shade of white to yellow), but when given a car, it could be any color.

The reason for colouring images can be for example restoring old black and white images and enhancing them by giving them colors. Most of the works that I reviewed focused on coloring houses, landscapes, people - sometimes pretty straightforward colouring. However, I was interested in what the models can do when faced with paintings, portraits, and various art throughout the time periods. Therefore, I downloaded a representative art dataset containing paintings of landscapes, portraits and more. In the next part, I will review some of the methods that have been utilized by others.

3 Related literature

The first paper I read was [1]. The authors use CNN to colorize black and white images from the ImageNet dataset [2]. The images were represented in the Lab color space, which expresses color as three values: Lightness, a and b, which expresses four colors – red, green, yellow and blue. The lightness defines black at 0 and white at 100 and is the one channel that we want when representing a grayscale image. The combinations of a and b axis make the whole colour. Negative values at axis a tilts more toward green and positive ones tilt to red. Negative values at axis b tilt toward blue and the positive ones towards yellow.

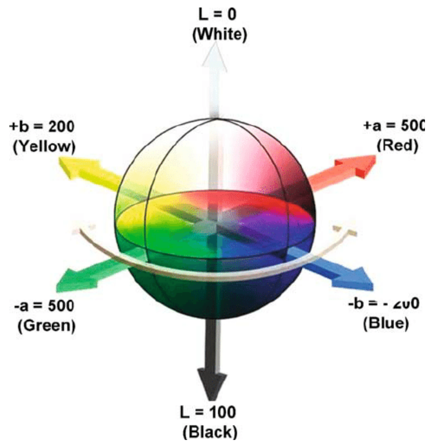

Figure 1: The LAB color space. Source: `https://www.researchgate.net/figure/The-cubical-CIE-Lab-color-space_fig3_23789543`

Their network architecture consists of 8 convolutional layers, while each of them is repeated twice of thrice. There are also ReLU layers and a BatchNorm layer, which speeds up the training.

As the colorization task is multimodal – meaning that there can be several 'right' colors for an object, it is not ideal to utilize standard loss functions, where the goal is to minimize the error between a prediction and the real value. Thus, the authors decided to predict a distribution of possible colors for each pixel. They also re-weight the loss while training to make sure some non-generic colours have a chance. The final colorization process takes a mean of the distribution for each pixel.

The next paper I read was one that uses uses the Split-Brain Autoencoders for the colorization task [3]. The split in this case means that the network is divided into two sub-networks, where each one is trained to estimate values from the other one. This is called the cross-channel prediction.

They compare two approaches regarding the loss function. Firstly, they simply use the L_2 distance and the Sum of Squared Errors, letting the layers predict color pixels of one another. In the second approach, they let the layers

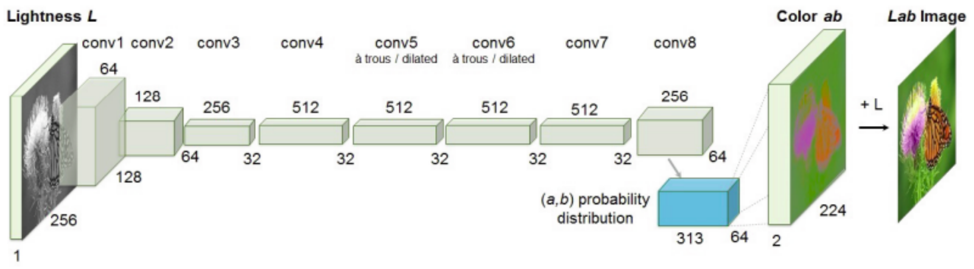

Figure 2: The Architecture. Source: [1]

predict the distribution of possible colours. To measure the errors, they use the standard cross-entropy loss between the real and predicted distributions.

They tested their model on Lab images and RGB-D Imaged (combination of an RGB image and their depth image). They conclude that their approach is comparable to the state-of-the-art approaches, if not a little better.

4 Dataset

I downloaded several separate art datasets and then concatenated them into one of 6883 images. The first dataset I downloaded is from `https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time`. The author downloaded paintings from the top 50 most influentual artists from artchallenge.ru website. The second dataset I downloaded was from `https://data.mendeley.com/datasets/289kxpnp57/1`, which is a collection of of portraits.

As for the data preprocessing, I converted the images to grayscale, thus creating a parallel folder for the coloured ones. I also only used half of the data, as the training with all data was too slow and the kernel was constantly out of memory.

5 Models

I implemented three models in total - CNN, Autoencoder and Variational Autoencoder.

The CNN model consists of 6 convolutional layers and 3 Batch Normalization layers. All the layers except the last one were also put into the ReLU activation function.

The Autoencoder model consists of 8 convolutional layers in the Encoder part and 5 convolutional layers in the Decoder. Four of the 5 layers in Decoder were implemented using the ConvTransposed2D module from Pytorch, a deconvolution layer. The last layer is a standard convolution layer. All the layers in

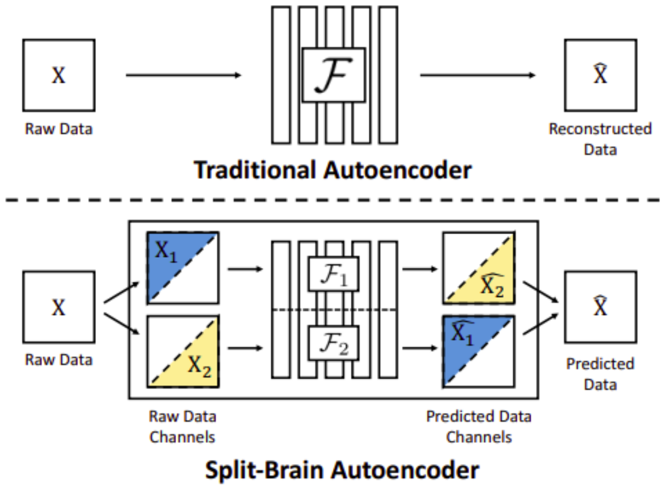

Figure 3: Difference between autoencoder and split-brain autoencoder. Source: [3]

the Encoder and Decoder are put through the ReLU activation function except for the last layer in the Decoder - which is put through the Sigmoid function.

The last model - Variational Autoencoder - consists of 5 convolutional layers in the Encoder, and 5 deconvolutional layers in the Decoder. After the Encoder part, there is a flattening operation so the vector can be put into a Linear layer. The mean and variance are obtained from the flattened vector, and then the latent vector is sampled from the distribution given by the mean and variance. Then, the vector is put into Linear Layer and after being unflattened, it passes onto the Decoder part. This model was inspired by this GitHub implementation [1].

The images were represented in the RGB color space.

6 Results

The results can be found in table below.

Model	Loss Function	Optimizer	Epochs	Batch size	Image size	Test Loss
CNN	MSE	Adam	100	32	200x200	0.0074
AE	MSE	Adam	300	128	200x200	0.0093
VAE	MSE + KLD	Adam	12	16	128x128	0.05756

[1]`https://github.com/sksq96/pytorch-vae`

All the models used the Adam optimizer. CNN and AE used MSE loss functions, but VAE loss function consist of two parts - the reconstruction loss and the loss of the distribution (the regularizer). The reconstruction loss is the same as with other models - MSE - but it is summed with the distribution loss, which is the KL divergence. It calculates the distance between our distribuion and the Gaussian distribution, which is eventually what we aim to [2].

Below are the image predictions for individual models for comparison.

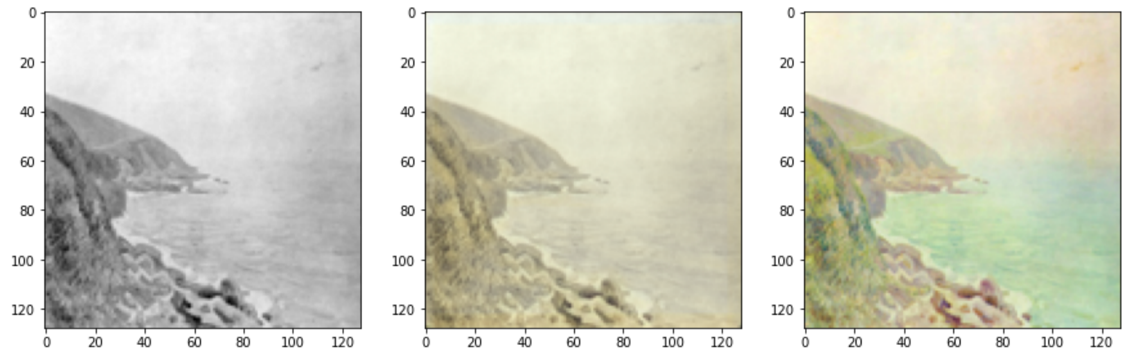

Figure 4: CNN - input, predicted, output

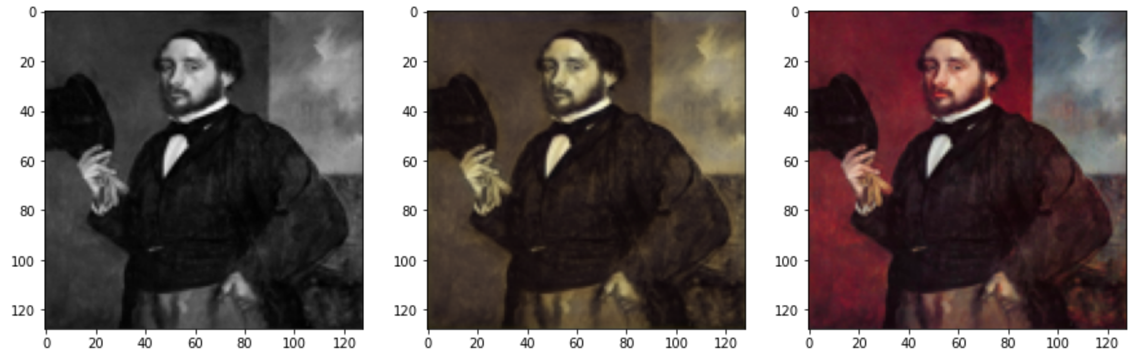

Figure 5: CNN - input, predicted, output

[2]`http://adamlineberry.ai/vae-series/vae-code-experiments`

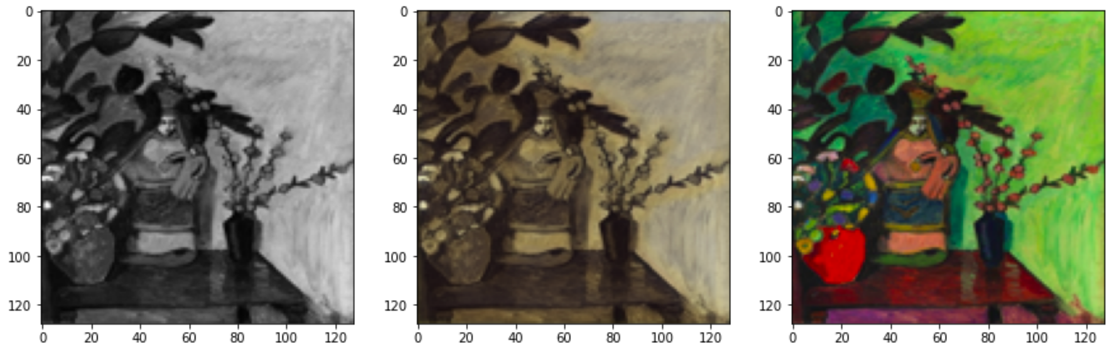

Figure 6: CNN - input, predicted, output

As we can see, although CNN has the smallest test loss, its predictions are dull, only predicting neutral colours.

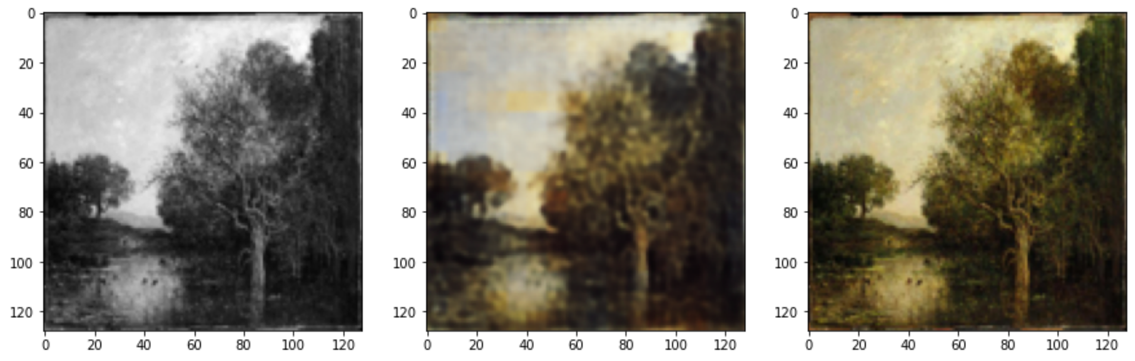

Figure 7: AE - input, predicted, output

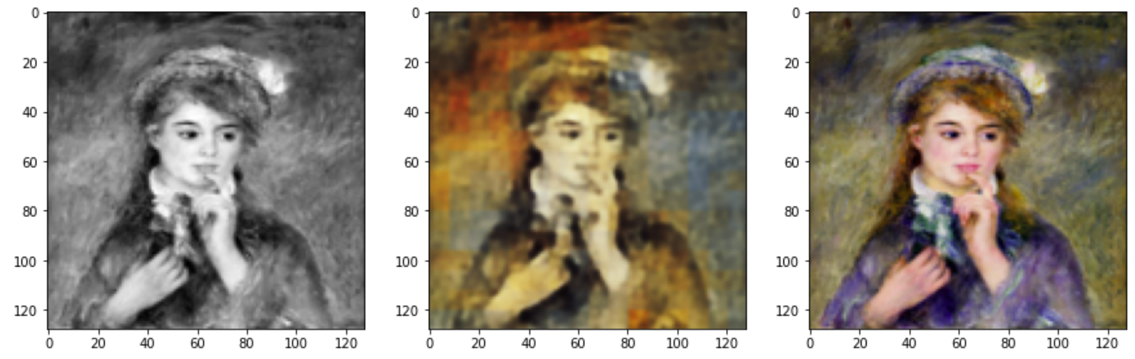
Figure 8: AE - input, predicted, output

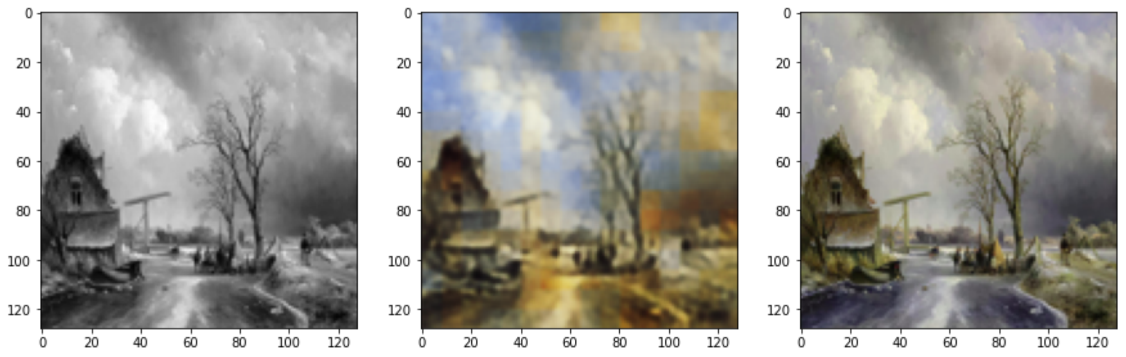
Figure 9: AE - input, predicted, output

Autoencoders are much better with the prediction - the colours are more diverse and bright. One setback is that some images are being predicted with only blue and orange-yellowish colours, which is only ideal for some situations. Also, the image resolution is a bit worse after the prediction.

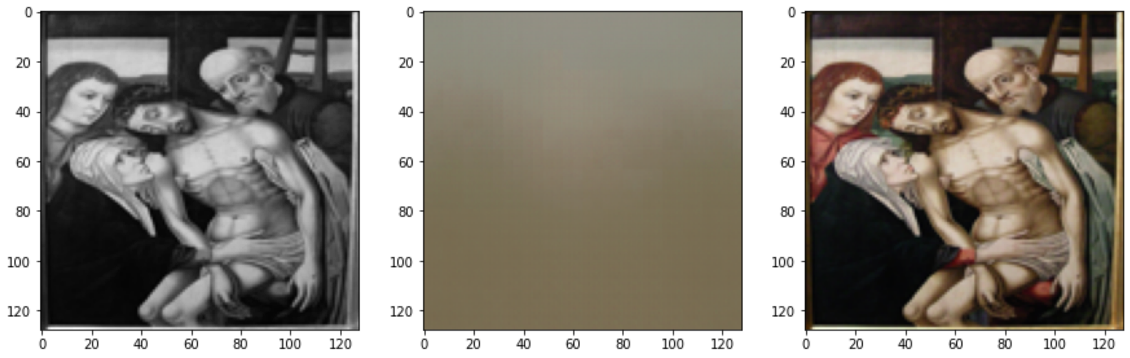

Figure 10: VAE - input, predicted, output

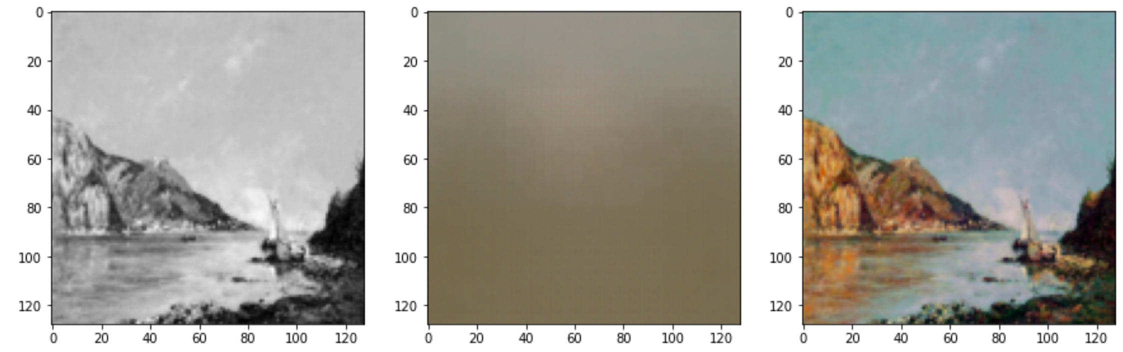

Figure 11: VAE - input, predicted, output

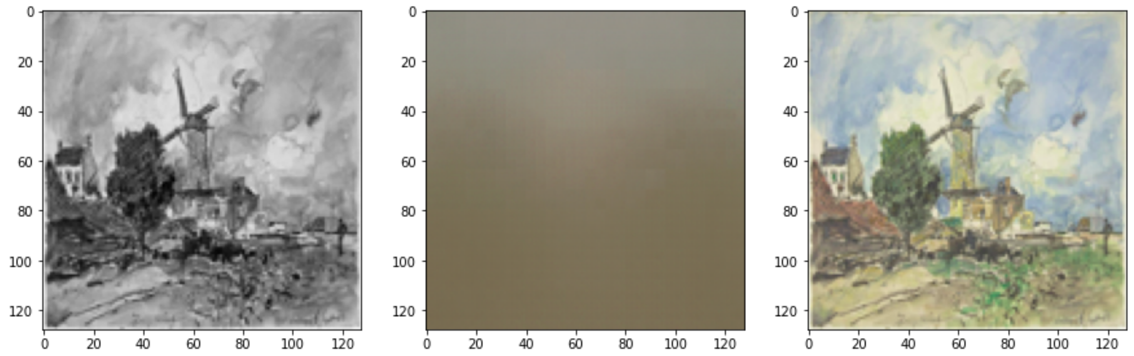

Figure 12: VAE - input, predicted, output

VAEs perform very poorly and honestly, I have no idea why :D I tried everything I could to see where the trouble was, but nothing led to success. Also, implementing the model a few days before the deadline is probably not the best approach...

7 Conclusion

Overall, the best model for image colourization is Convolutional Autoencoder. The colours were the most diverse, on-spot, creating engaging, colourful images.

The CNN model gave us dull, same vintage colours, and the model might need more layers and kernels for better colourization.

The VAE model predicts very blurry images, which might be because of many factors - latent dimension, kernel size in convolutional layers, number of layers, or poor dataset or there must be something fundamentally wrong about the model.

The dataset might be one reason the models do not perform that well - it is imbalanced, there is a large number of landscapes and portraits, or other modern art could be represented better.

The work could be developed more by trying the Lab space instead of RGB, for example. Also, predicting the distribution of colours for each pixel is more efficient than predicting the exact colour of the pixels. The dataset could be regathered to be more representative - or it could contain only specific content, landscapes, for example, instead of everything mixed.

References

1. ZHANG, Richard; ISOLA, Phillip; EFROS, Alexei A. Colorful image colorization. In: *European conference on computer vision.* Springer, 2016, pp. 649–666.

2. RUSSAKOVSKY, Olga; DENG, Jia; SU, Hao; KRAUSE, Jonathan; SATHEESH, Sanjeev; MA, Sean; HUANG, Zhiheng; KARPATHY, Andrej; KHOSLA, Aditya; BERNSTEIN, Michael, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*. 2015, vol. 115, no. 3, pp. 211–252.
3. ZHANG, Richard; ISOLA, Phillip; EFROS, Alexei A. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1058–1067.