Intelligent Systems Project Report
Image Colorization using CNNs

Lorenzo Catoni Leonardo Giovannoni

Contents

1 Introduction

The goal of this project is to colorize grayscale images using Convolutional Neural Networks. The model is trained on the CelebA dataset, which contains over 200,000 celebrity portraits images. Figure 1 shows an extract of the dataset. For our purpuses a subset of 10,000 images will be sufficient.

In Chapter 2 we will train a regression model to predict the ab channels of the Lab color space from the L channel. The LAB color space has the property of separating the luminance (brightness) channel from the chrominance channels (color). The L channel encodes the brightness of the image, while the ab channels encode the color information. By doing so, the model will have to predict only two channels (AB) instead of three (RGB), which will make the training process faster.

In Chapter 3 we will change the problem from a regression to a classification problem. We will follow the approach of the paper Colorful Image Colorization by Zhang et al. In this section we will try to get a more colorful output by weighting the loss function according to the rarity of the color. In realizing this part great help was given by this repository: nn-photo-colorization.

Figure 1: Extract of the CelebA dataset

The current standard in image colorization, showcased in Chia et al.'s paper Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2, involves complex training with a pre-trained Inception-ResNet-v2 model and the integration of a GAN for improved output quality. However, due to the extensive computational demands and complex architecture, we opt to pursue a simpler replication of Zhang et al.'s results, which employs a more manageable architecture based on a VGG-styled network. Other influential papers considered include Larsson et al.'s Learning Representations for Automatic Colorization in which they use hypercolumns on a VGG network and Iizuka et al.'s Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification where they implement a two-stream architecture in which they fuse global and local features. Our focus for this project will be on simplicity and efficiency in implementation.

2 Regression Model

In this first part, we will train a regression model to predict the ab channels of the Lab color space from the L channel. We will use Mean Squared Error (MSE) as the loss function, it makes sense since the color of each pixel is a continuous value and we want to minimize the difference between the predicted color and the ground truth color.

2.1 Data preparation

The images are loaded as RGB; we had to convert them to Lab and extract the L channel (brightness) and the ab channels (color). We used the scikit-image library to convert the images to Lab and extract the channels.

The L channel is normalized to the range [0, 1], and the ab channels are normalized to the range [-1, 1] (the values of the L channel are in the range [0, 100], and the values of the ab channels are in the range [-128, 127]).

Since we couldn't load the entire dataset into memory, we used a generator to load the images in batches. The generator also performed the preprocessing steps described above. As per documentation, the generator must return a tuple $(\text{inputs}, \text{targets})$ where inputs is a list of input data arrays and targets is a list of target data arrays. Both lists must have the same length, i.e., the batch size.

2.2 Model

The model is a simple CNN Autoencoder. The encoder part is a series of `Conv2D` layers with `ReLU` activation, downsampling is performed using a stride of 2. The decoder part is a series of `Conv2D` layers with `ReLU` activation, upsampling is performed using `UpSampling2D`. The choices made for the model architecture are based on what we observed was a common practice for this type of problem.

The activation function of the last layer is `tanh`, which works well for the ab channels since the values are in the range [-1, 1].

The loss function is the mean squared error between the predicted ab channels and the ground truth ab channels, it makes sense because each pixel is a continuous value and we want to minimize the difference between the predicted color and the ground truth color.

Figure 2 illustrates graphically the model architecture, while Table 1 shows all the layers of the model and the dimensions of the hidden layers.

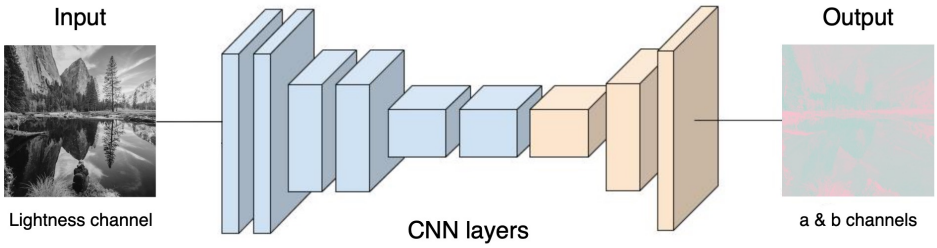


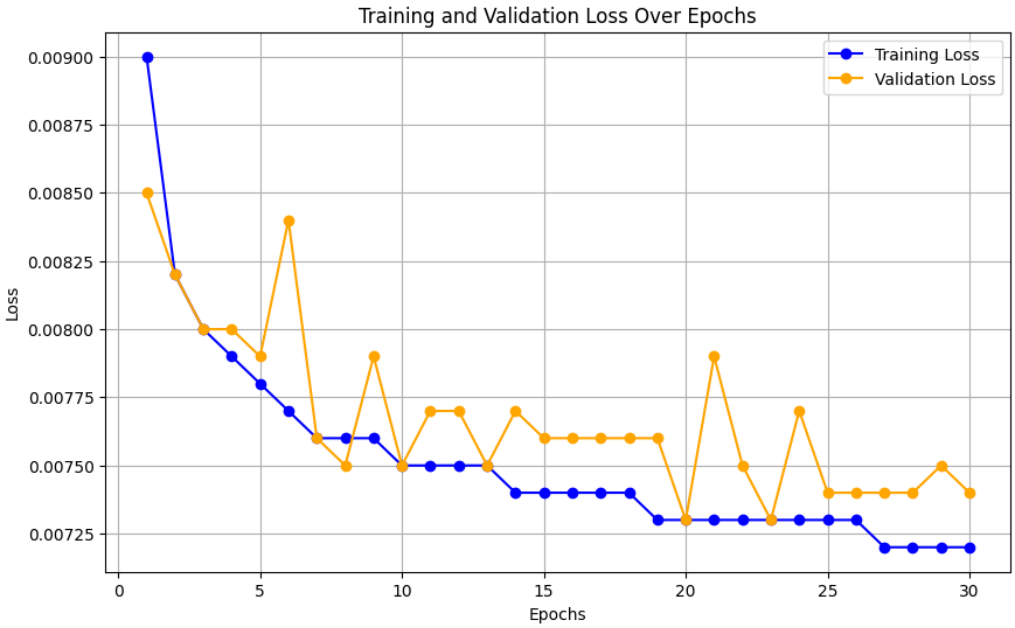
Figure 2: Representation of our CNN

Table 1: Model Summary

Layer (type)	**Output Shape**	**Param #**	
conv2d	(None, 216, 176, 32)	320	
conv2d_1	(None, 108, 88, 32)	9248	
conv2d_2	(None, 108, 88, 64)	18496	
conv2d_3	(None, 54, 44, 64)	36928	
conv2d_4	(None, 54, 44, 128)	73856	
conv2d_5	(None, 27, 22, 128)	147584	
conv2d_6	(None, 27, 22, 256)	295168	
conv2d_7	(None, 27, 22, 128)	295040	
conv2d_8	(None, 27, 22, 64)	73792	
up_sampling2d	(None, 54, 44, 64)	0	
conv2d_9	(None, 54, 44, 32)	18464	
up_sampling2d_1	(None, 108, 88, 32)	0	
conv2d_10	(None, 108, 88, 16)	4624	
up_sampling2d_2	(None, 216, 176, 16)	0	
conv2d_11	(None, 216, 176, 2)	290	
Total params: 973810 (3.71 MB)			
Trainable params: 973810 (3.71 MB)			
Non-trainable params: 0 (0.00 Byte)			

2.3 Training

The model was trained for 30 epochs using the RMSprop optimizer with default parameters. We observed that the validation loss plateaus after 30 epochs.



2.4 Results

Figure 3 and 4 show some sample images from the test set. The images are displayed in groups of three: the first image is the grayscale input, the second image is the predicted image, and the third image is the ground truth.

Figure 3: Colorization of the model on the test set (triplets are: grayscale input, predicted image, ground truth)

Figure 4: Colorization of the model on the test set (triplets are: grayscale input, predicted image, ground truth)

2.5 Evaluation

Unfortunately, it is hard to find a metric to evaluate the performance of the model since the problem of colorization is, by nature, ambiguous. In many instances, the model will predict a color that is different from the ground truth but still plausible. There are even some cases where the model predicts a colorization that looks more natural than the ground truth (in cases where the white balance of the image is off, for example), see Figure 4 second row first column and sixth row second column.

That being said, we can see that the model tends to predict desaturated colors and never predicts bright colors. We can see this clearly in the scatter plot depicted in Figure 5, where the predicted ab channels are plotted against the ground truth ab channels. The plot shows that the model tends to predict colors that are close to the origin, meaning that the colors are desaturated.

There are a few reasons why the model tends to predict desaturated colors, which we will address in the next part of the project.

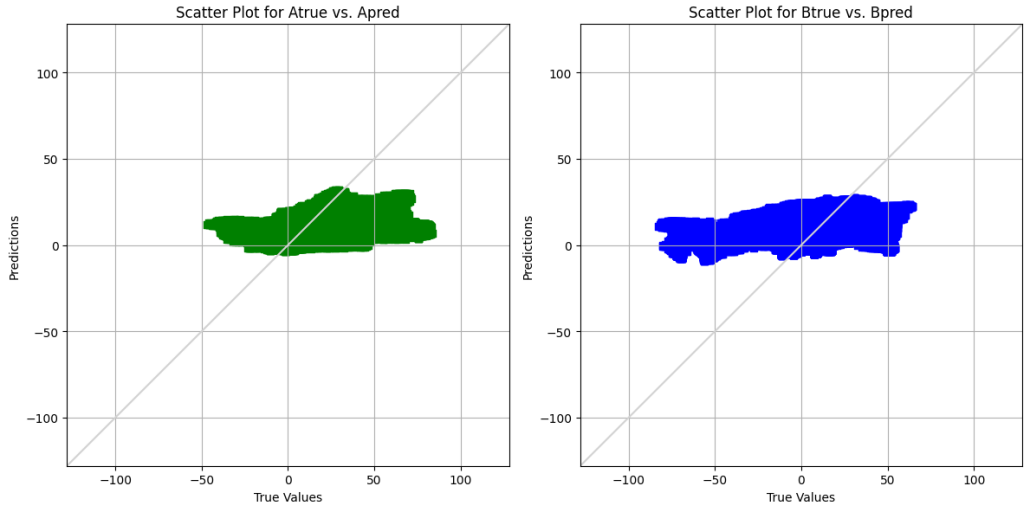


Figure 5: Scatter plot for True AB values vs Predicted AB

3 Classification Model

In this second part, we tried to improve the results of the previous model by approaching the problem as a classification problem. The reasons why the regression model generated desaturated images are the following:

- *Mean squared error favors low intensity colors*: If an object can take a set of distinct ab values, the optimal solution with respect to MSE is the mean of this set. In color prediction, this averaging effect favors grayish, desaturated results.
- *Bright colors are rare*: The distribution of colors in the ab space is not uniform, and bright colors are greatly underrepresented. This bias is reflected in the model's predictions.

To solve these problems, we used a classification model with class rebalancing by reweighting[1] the loss of each color class.

3.1 Data preparation

3.1.1 Distribution of colors

In this section we show that the distribution of colors is indeed strongly biased towards desaturated colors (i.e., colors with ab values close to 0). We took a meaningful subset of the training data and plotted the distribution of colors in the ab space as a 2D histogram, the resulting plot is shown in Figure 6, note that the scale of the histogram is logarithmic, as shown in the legend. Figure 7 shows the colors present in the training data. We can indeed see that the colors close to 0 are grayish, while the bright colors are close to the edges of the ab space.

[1] class_weight parameter of the fit function

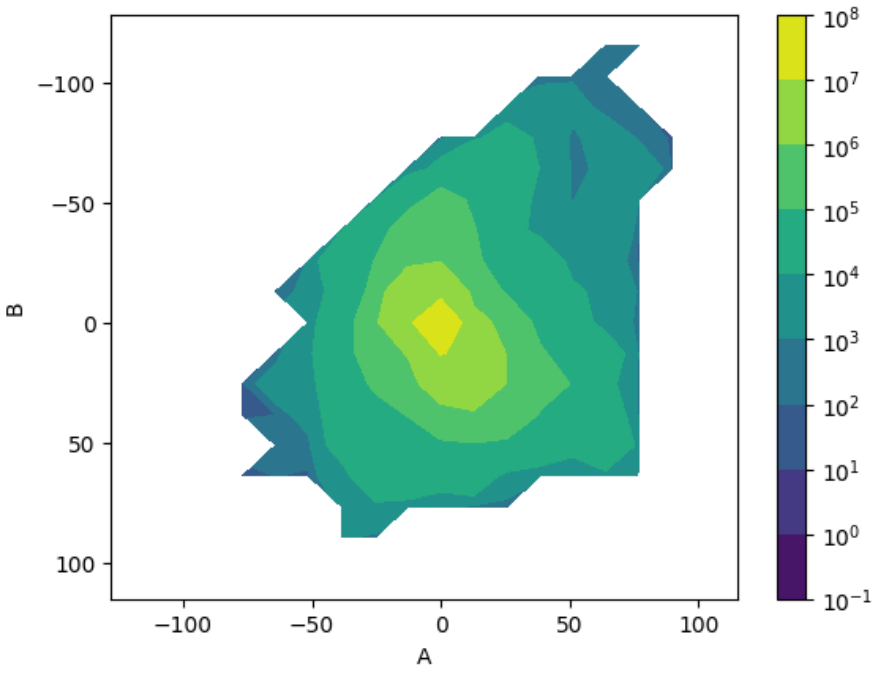


Figure 6: Color distribution

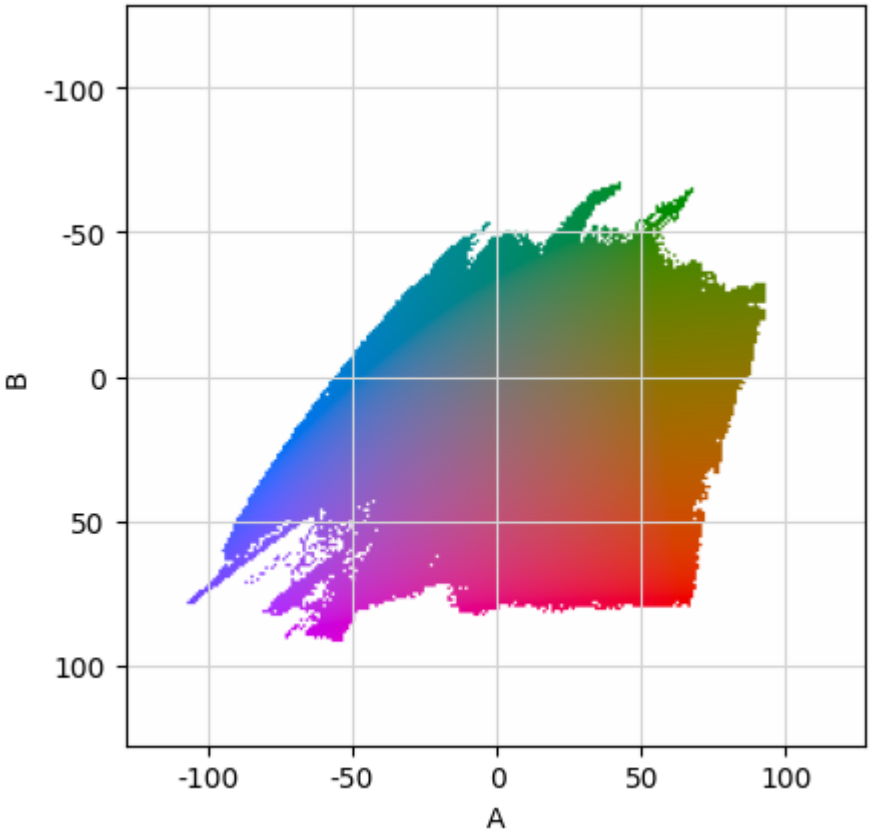


Figure 7: Color gamut of the dataset

3.1.2 Class definition

Obviously, we can't have one class for each possible (a,b) pair; that would be too many classes and most likely impossible for the network to learn. Instead we quantize the ab space into bins of size 10; each color belongs to

the class of the bin it falls into.

Essentially, we round each color to the nearest multiple of 10; this lowers the number of classes to an acceptable number. It obviously makes the color space much smaller, which is good for training but bad for the visual quality of the results. Actually, the image quality doesn't suffer from this color space reduction; we will show this soon.

Figure 8 shows the quantized ab color space; Figure 9 shows that this color reduction doesn't affect the visual quality of the results; the pair of images are the original image (left) and the image with the quantized colors (right). Little to no visible difference can be perceived.

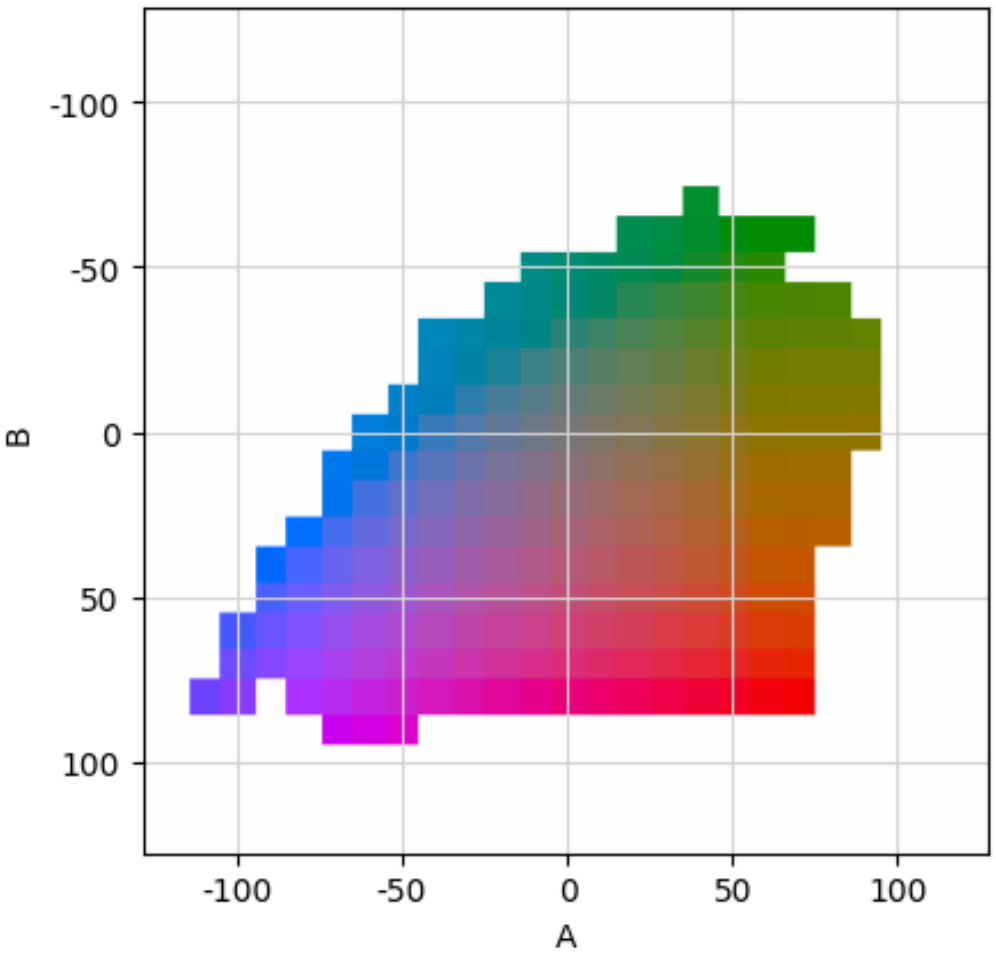


Figure 8: Quantized ab space with bins of size 10

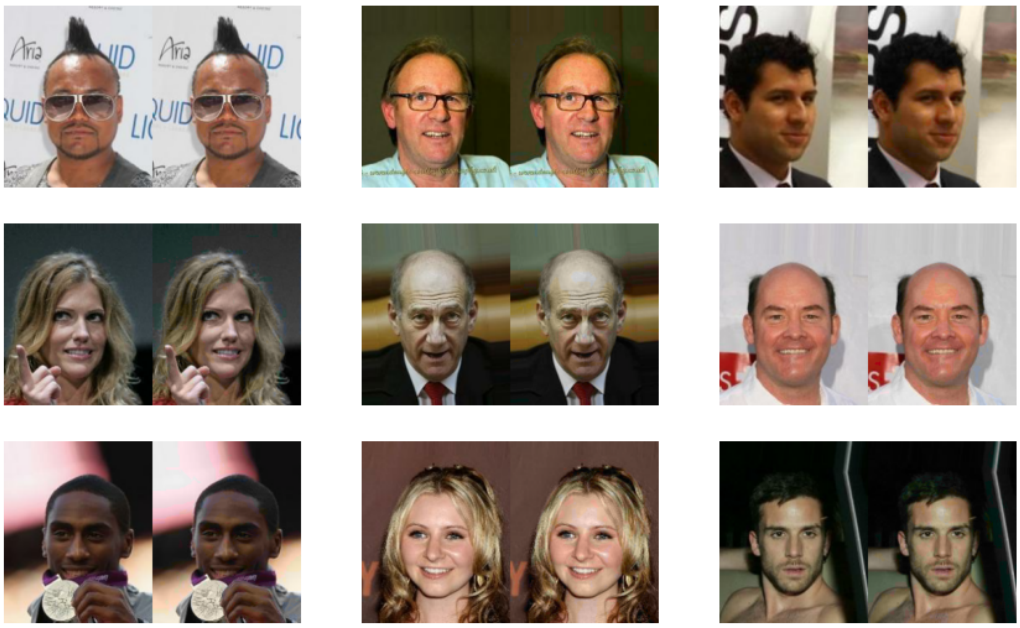


Figure 9: Original image (left) vs quantized image (right)

We proceeded to calculate the classes needed to cover the whole ab space of our training data. We did so by iterating over the whole training data and keeping record of each quantized (a,b) pair we find, as well as the number of times it appears in the training data. The distribution is necessary in a later moment to calculate the class weights for the loss function. From this calculation, we found that there are 243^2 classes in total.

3.1.3 Input and Label split

This time we needed to do a bit more processing to prepare the input and label data for the training phase. The input data is still the L channel like in the regression model, on the other hand the label is a 3D array where the last dimension is a one-hot vector of size 243, encoding the color class for that pixel.

To put it more clearly: it helps to think of the label as a 2D array of shape (height, width), each pixel is a vector of size 243, where the i-th element is 1 if the pixel belongs to the class i, and 0 otherwise. (Technically it has one more dimension for the batch size)

3.2 Class weights

For the class weights, we used the formula from `sklearn.utils.class_weight.compute_class_weight`, which is the following:

$$w_i = \frac{n\_samples}{n\_classes \cdot n\_samples_i}$$

Basically the weight is proportional to the inverse of the frequency of each class, so that rare classes have a higher weight.

3.3 Model

The model is similar to the previous one; the difference is that the output layer has 243 channels instead of 2. The activation function of the output layer is `softmax`, and the loss function is `categorical_crossentropy` since we are doing classification. Table 2 shows all the layers of the model and the dimensions of the hidden layers, it's the same as the regression model except for the last two layers.

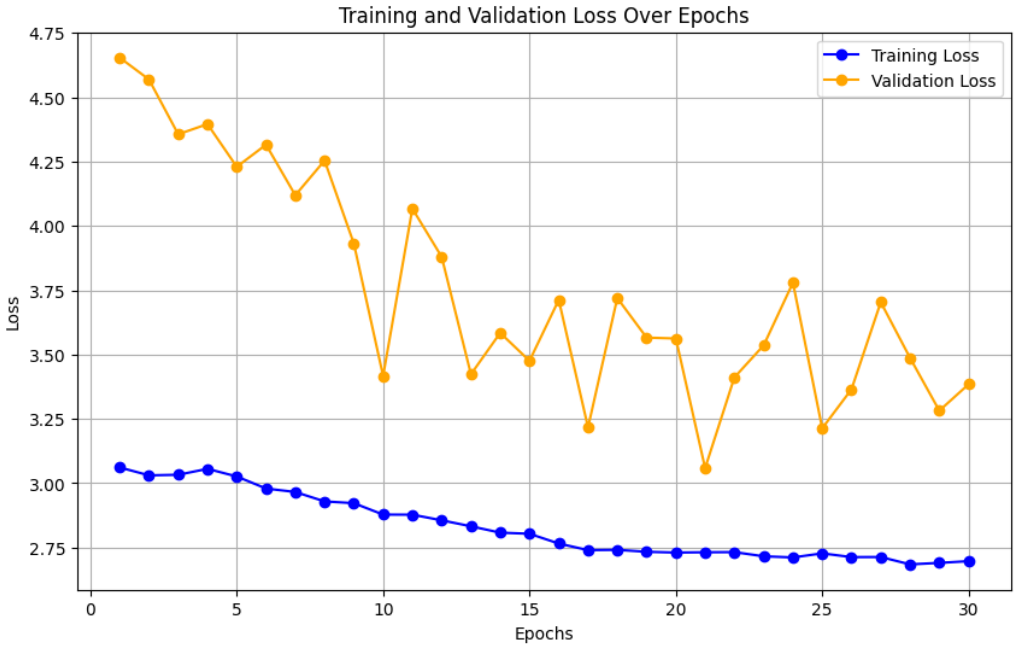
[2]The authors of the paper Colorful Image Colorization found 313, that's because the exact number depends on the dataset

Table 2: Model Summary

Layer (type)	Output Shape	Param #	
conv2d	(None, 216, 176, 32)	320	
conv2d_1	(None, 108, 88, 32)	9248	
conv2d_2	(None, 108, 88, 64)	18496	
conv2d_3	(None, 54, 44, 64)	36928	
conv2d_4	(None, 54, 44, 128)	73856	
conv2d_5	(None, 27, 22, 128)	147584	
conv2d_6	(None, 27, 22, 256)	295168	
conv2d_7	(None, 27, 22, 128)	295040	
conv2d_8	(None, 27, 22, 64)	73792	
up_sampling2d	(None, 54, 44, 64)	0	
conv2d_9	(None, 54, 44, 32)	18464	
up_sampling2d_1	(None, 108, 88, 32)	0	
conv2d_10	(None, 108, 88, 16)	4624	
up_sampling2d_2	(None, 216, 176, 16)	0	
conv2d_11	(None, 216, 176, 243)	4131	
activation	(None, 216, 176, 243)	0	
Total params: 977651 (3.73 MB)			
Trainable params: 977651 (3.73 MB)			
Non-trainable params: 0 (0.00 Byte)			

3.4 Training

The model was trained for 30 epochs using the RMSprop optimizer with default parameters. We observed that the validation loss stops improving after 20 epochs, so we could have stopped the training earlier. The only difference with the training of the previous model is that we pass the class weights parameter to the `fit` method for the loss function.



3.5 Results

Figure 10 shows some sample images from the test set. The images are displayed in groups of four: the first image is the grayscale input, the second image is the predicted image choosing the mode (argmax) of the predicted class for each pixel, the third image is the predicted image taking the expected value of the predicted class for each pixel, and the fourth image is the ground truth.

Taking the expected value of the predicted class for each pixel helps reduce patchiness. The idea comes

from the paper cited at the beginning of this document. But their method is more sophisticated as they use an operation called *annealed mean*, which basically gives a result between the mode and the expected value.

Figure 11 shown the colorization given by this model when it's trained without class rebalcing (i.e. no class weight parameter).

We tried different formulas for the class weight and different optimizers for the training, but none of the

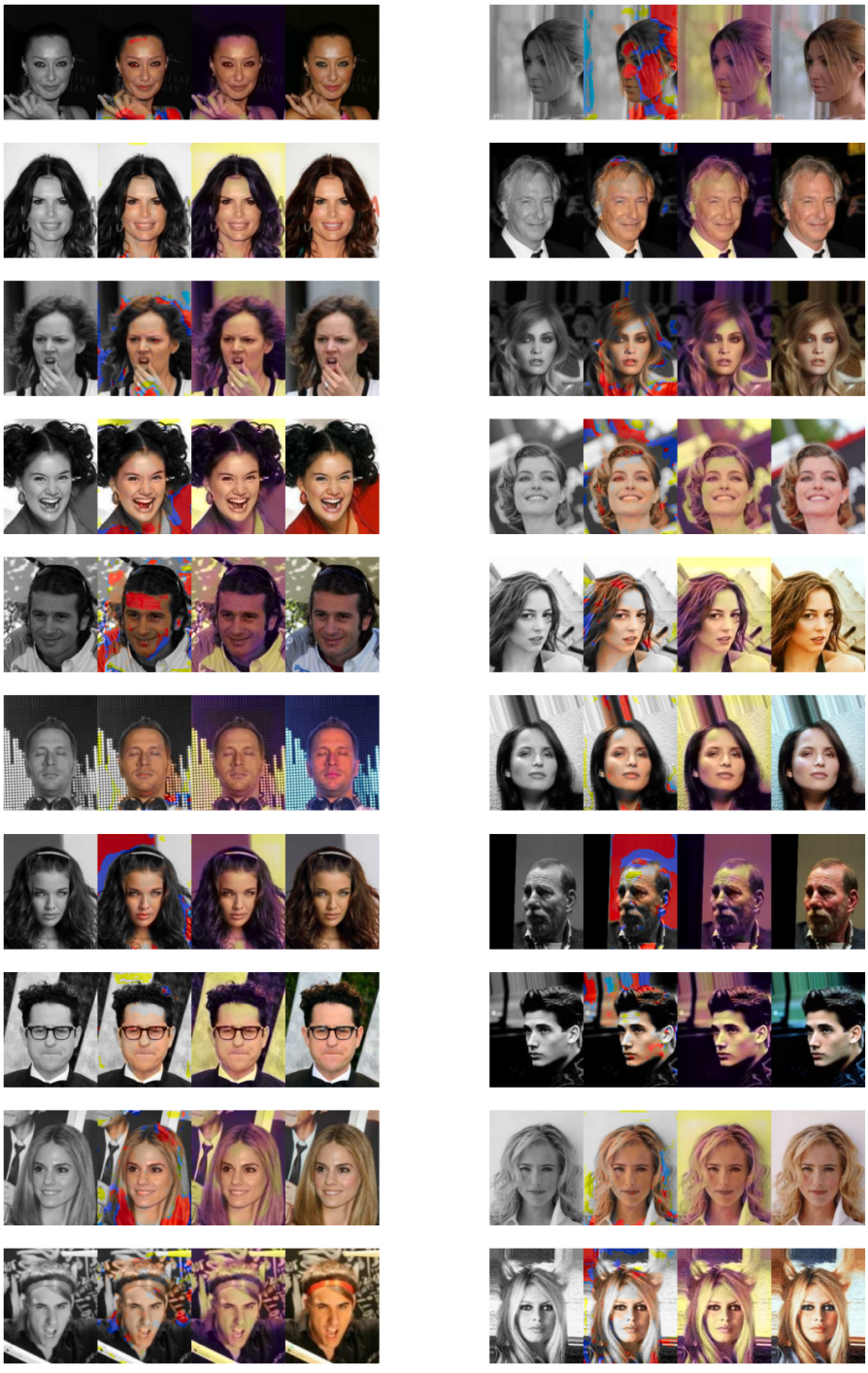


Figure 10: Colorization of the model on the test set (four images are: grayscale input, mode, expected, ground truth)

Figure 11: Colorization of the model on the test set trained without class rebalancing (triplets are: grayscale input, predicted, ground truth)

3.6 Evaluation

Again, it's hard to find a good metric to evaluate the results, in the paper Colorful Image Colorization they rely on how compelling the colors look to a human observer to evaluate the result. Anyways, visually we can see that the results are not great; bright colors do appear but are not spatially coherent and in the wrong places. On the other hand, when taking the expected value, the patchiness is gone but the colors are basically yellow and purple in all the pictures, which is far from the ground truth and far from a compelling result.

The difference from our implementation and the paper's implementation is the use of soft encoding for the labels (theirs), and they use a bigger model with more layers and more filters per layer and also a bigger training set.

It's also possible that our model didn't properly learn because there isn't enough information about bright colors; we must remember that reweighting the loss function doesn't add information, it just changes the loss function to favor rare classes, but if the model doesn't have enough information about the rare classes, it won't learn anything useful.

4 Conclusions

In conclusion we can say that the regression model is able to colorize the images and produce acceptable results, even if the images are bit desaturated and dull. The classification model instead didn't provide the expected results. We can see more bright colors in the output imgaes, but the colors are often in the wrong places. To improve the results we could try a different network architecture, for instance having a fully connected layer in the middle of the network, or we could follow more closely the approach of the paper, which uses soft encoding instead of one-hot encoding.

Figure 12: Colorization of black and white photos with the regression, classification and classification + class rebalancing models