Deep Neural Networks Project 2021-2022

Image Colorization

Foteini Maria Kostopoulou 7115152100006

Kostas Panoutsakos 7115152100012

Konstantina Stoikou 7115152100015

Introduction

Image colorization is the process of converting grayscale (black and white) images into visually acceptable colorized images.

Today, turning a black-and-white image into a vibrant, full-color image is usually done by a photo editing expert using different softwares, like Photoshop. It demands a large amount of human effort, time, skill, and extensive research. For instance, a human face alone might need up to 20 layers of pink, green and blue shades to colorize it just right. That means, the colorization of a picture can take up to one month.

Images by Jordan Lloyd

Colorizing images can have a big impact on many domains, for instance, old photograph colorization, image harmonization, color image enhancement, coloring scientific illustrations or re-master of historical images etc. Automatic image colorization is a new challenging field that aspires to solve grayscale image colorization by employing special type of deep learning architecture, like Convolutional Neural Networks (CNNs).

It is important to comment that automatic image colorization is mainly an ill-posed problem, since it has not a unique solution and requires the mapping

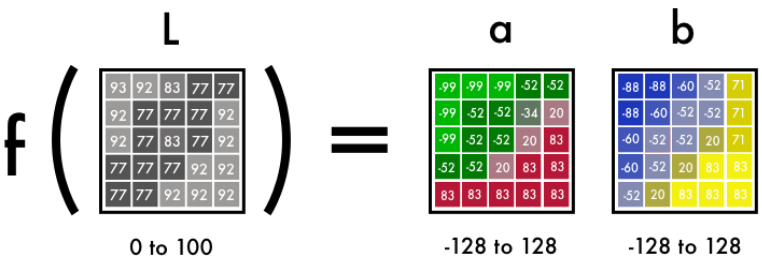
of one-dimensional grayscale image to three-dimensional colorized image. That implies that grayscale images, unlike colorized images that are mapped in RGB space, consist only of intensity values, where white color represents the strongest intensities and black the weakest. That way, the main goal of CNN-based automatic image colorization is to output a realistic colorization, where each color has a semantic meaning, for example sea is expected to be blue and grass green. However, the semantic representation is expected to become more challenging in cases of features that are difficult to determine its color like shirts, cars, flowers etc.

Dataset

The dataset consists of 2000 RGB images that depict landscapes and urban scenes and was downloaded from Kaggle. The dataset is determined to be consistent, which means that no image exists that is entirely black or white, because it provides no information as a grayscale image to the CNN. The dataset contains a variety of scenes (mountains, sea, forests, skyscrapers, towns, humans, etc.) instead of being narrowed down to images with similar concepts.

LAB space

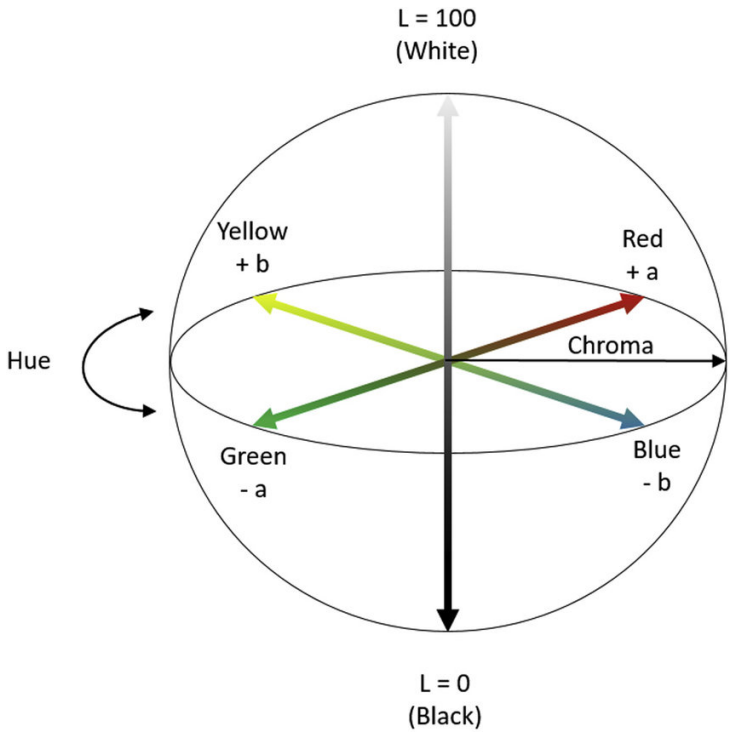
As mentioned before, in this project we are using a dataset of 2000 RGB images. However, the autoencoder needs as input a grayscale image in order to generate the colorized one. For this reason, the CIELAB color space, also referred to as LAB space, is a more suitable choice.



Particularly, the Lab color space expresses colors as three values:

- L : the lightness, which is used to represent the grayscale image - values range from 0 (black) to 100 (white),
- a : green-red color spectrum - values range from -128 (green) to 127 (red)
- b : blue-yellow color spectrum - values range from -128 (blue) to 127 (yellow)

The CIELAB color space diagram is given below:



Preprocessing Steps

Resize

The images are resized before entering each model. For the baseline model, the images are resized to dimensions 256 x 256. Additionally, for the VGG-16-based model images are resized to 224 x 224 pixels.

Normalization

Before converting the images from RGB space to LAB space, we have to normalize them by dividing them by the maximum value that RGB pixel can reach. Specifically, for each color channel the RGB channel ranges from 0-255. This normalization provides faster convergence and better training because it's easier to compute the error from the predicted colorizations.

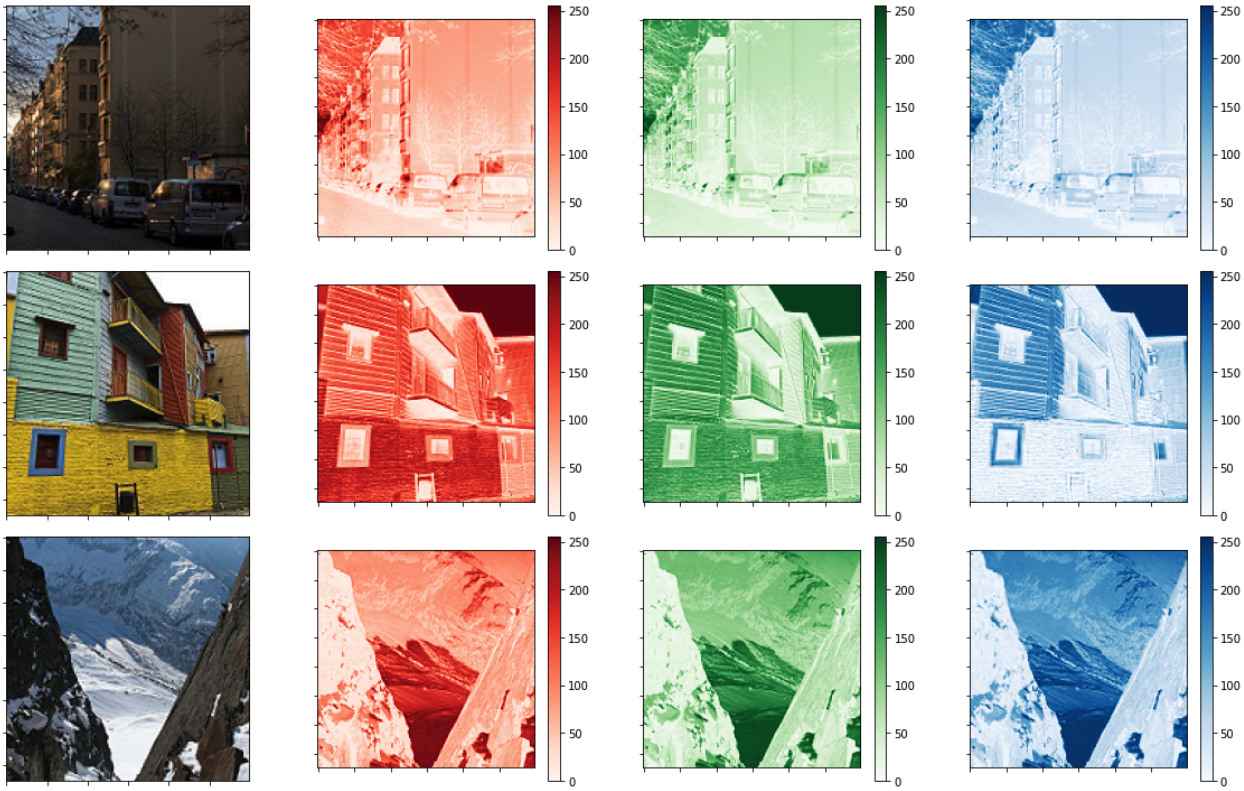
Normalization is applied again to the a and b features after converting the images to LAB space. The a and b values range from -127 to 128, so we divide them by 128 in order to restrict values to a range between -1 and 1. We choose this normalization because in the autoencoder model the mapping of the predicted values is done using the tanh activation function. As known, the tanh function ranges between -1 and 1.

In the end, after each model completes the prediction of the colorized images the output values range from -1 to 1, so we multiply the values by 128 to restore the values of ab channels.

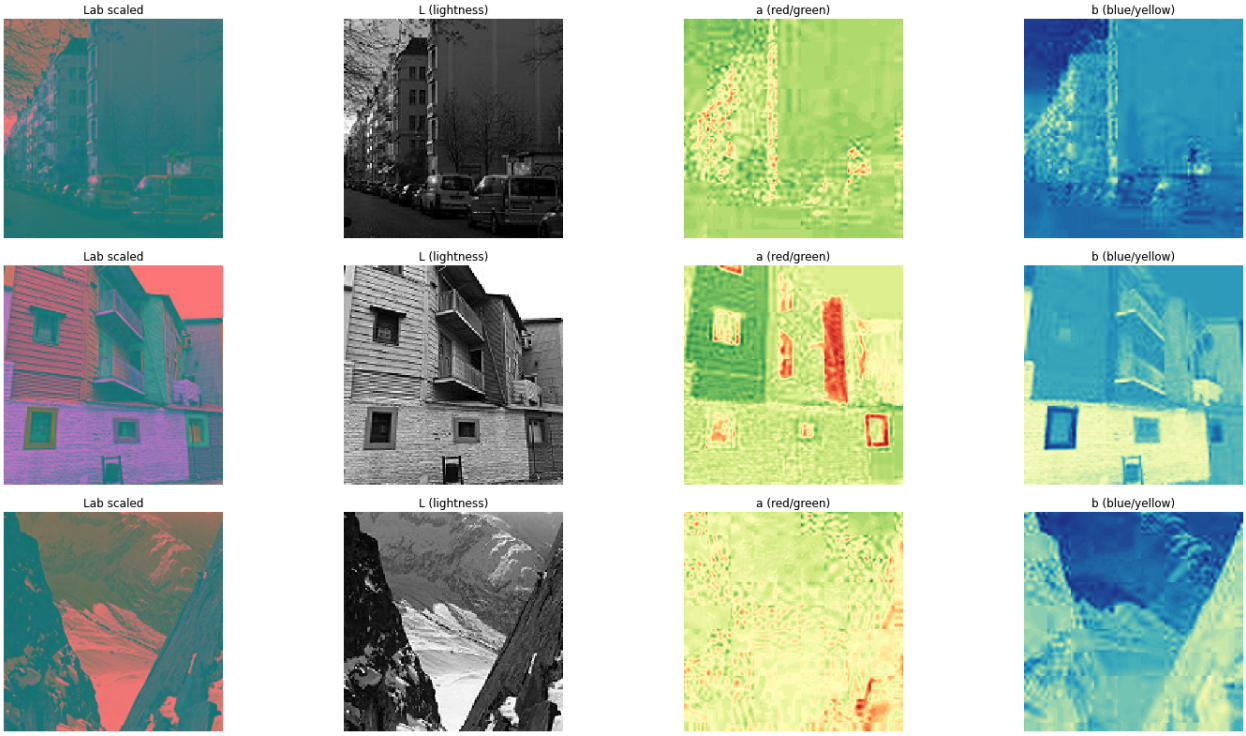
Lab space Transformation

The training set is converted from the RGB space to LAB space. As described in the previous section, the L values referring to grayscale images are used as

input to models. Below, a sample of the images can be seen along with their corresponding RGB layers:



In the next figure, the transformation to the LAB color space is given followed by the its layers:

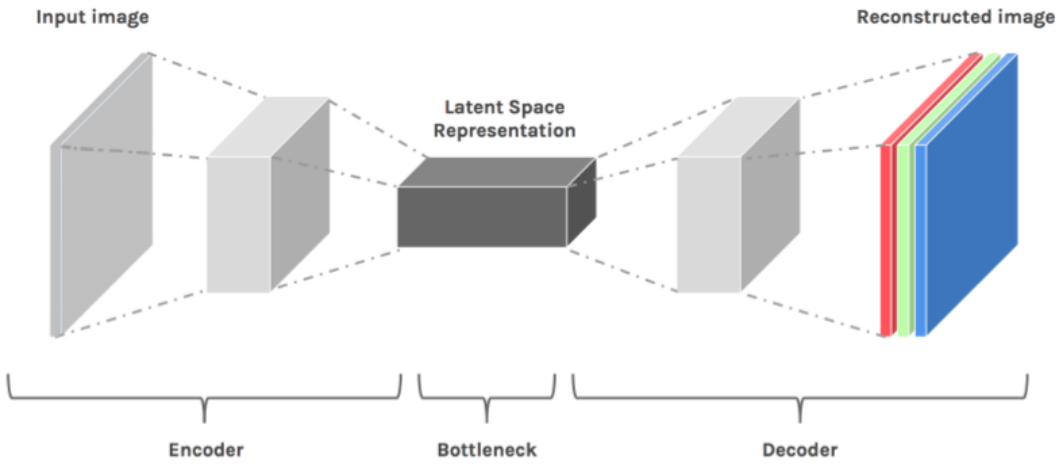

Lab scaled
L (lightness)
a (red/green)
b (blue/yellow)
Lab scaled
L (lightness)
a (red/green)
b (blue/yellow)
Lab scaled
L (lightness)
a (red/green)
b (blue/yellow)

Autoenconders

In this project we employ Autoenconders in order to solve the automatic image colorization. An Autoencoder is a type of artificial neural network used to learn data encodings in an unsupervised manner. In general, it aims to learn a lower-dimensional representation (encoding) for a higher-dimensional data, such as the input, and then reconstruct the output from this representation.

Autoencoders consist of 3 parts:

1. ***Encoder***: maps the input into the code , meaning an encoded , "compressed", representation.
2. ***Bottleneck***: contains the compressed knowledge representations.
3. ***Decoder***: maps the code to a reconstruction of the input back from its encoded form, meaning it "decompresses" the knowledge representations The decoded output is then compared with the input using a loss function.



Convolutional Neural Networks

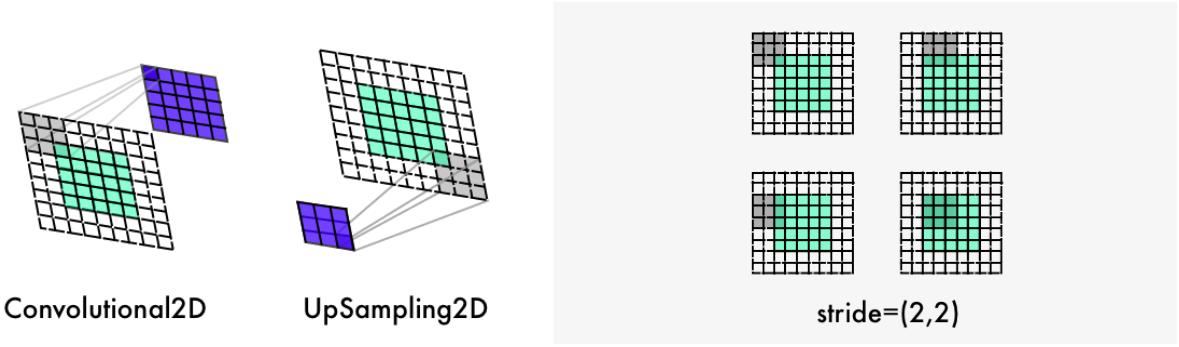
In this project, convolutional neural networks (CNN) will be employed through the encoder part of the autoencoder, for feature extraction, and similarly in the decoder part, followed by upsampling layers, for the reconstruction to the original image dimensions.

CNNs are a type of deep learning neural networks that enable extracting features from the images. The CNN architecture is summarized in :

- The input layer.
- Hidden layers consisting of convolution layers, ReLU (rectified linear unit) layers, the pooling layers, and a fully connected Neural Network.
- The output layer.

As already mentioned, the goal of the convolutional layers is to extract potential features by applying convolution filtering. The convolutional layers read the input (such as a 2D image) and drag a kernel (of a specified shape) over the image. How many times the kernel will be shifted across the image is called “Stride”. The kernel represents the features we want to locate in the image and for each convolutional layer the input is multiplied by the values of the kernel. The addition of convolutional layers depends on the complexity of the image and usually in order to maintain the original dimension, the image is padded with values on both ends.

After applying a convolutional layer, a non-linear activation function is applied to the feature vector. This way, the non-linearity of the network gets increased without affecting the receptive fields of convolution layers.



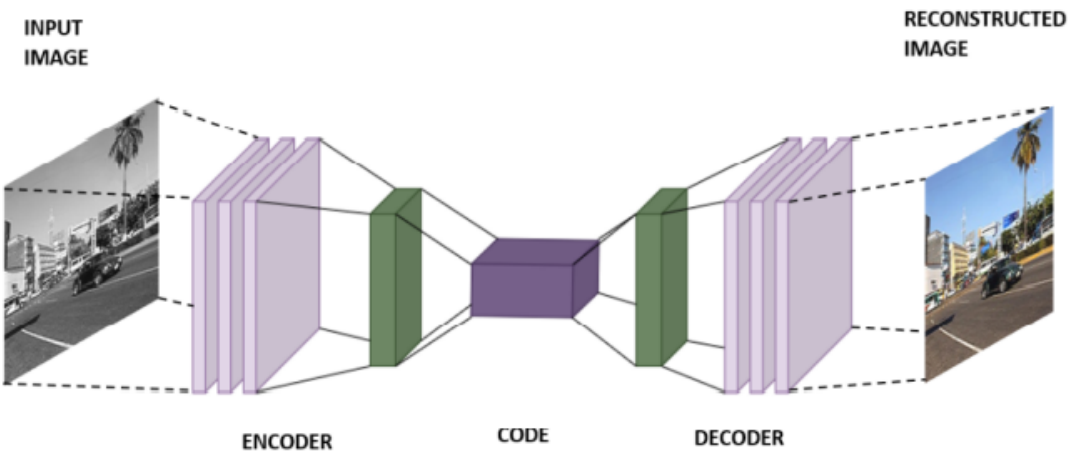
In the next step, it is optional to use pooling layer, which applies a non-linear down-sampling on the convolved feature so as to reduce the computational complexity.Usually, there are two types of pooling, Max Pooling, that returns the maximum value from the portion of the image covered by the Pooling Kernel and the Average Pooling that averages the values covered by a Pooling Kernel.
The opposite of the pooling layers are the upsampling layers which in their simplest form only resize the image (or copy the pixel as many times as needed).

Convolutional and pooling (aggregating) layers can be stacked on top of each other to provide multiple layers of abstraction. Many popular image classification architectures are built in a similar way, such as AlexNet, VGG-16, or ResNet.

Architecture of proposed Autoencoders

The architecture of autoencoders used to solve the automatic image colorization mainly consists of four components :

1. ***Encoder*** **:** processes H x W grayscale images, which are given as input and are square sized, and outputs H/8 x W/8 x 512 feature representation.
2. ***Feature Extractor*** **:** extracts high-level features of the images and passes on the image information aimed for colorization
3. ***Fusion*** **:** takes as input the output of the encoder and the feature vector from the feature extractor and produces fused learned features by concatenating them into a single layer and performing 2D convolution
4. ***Decoder*** **:** applies a series of convolutional and up-sampling layers in order to predict the coloring of the input grayscale image. Up-sampling is performed using the nearest neighbor interpolation algorithm, so that the output's height and width are twice the input's.



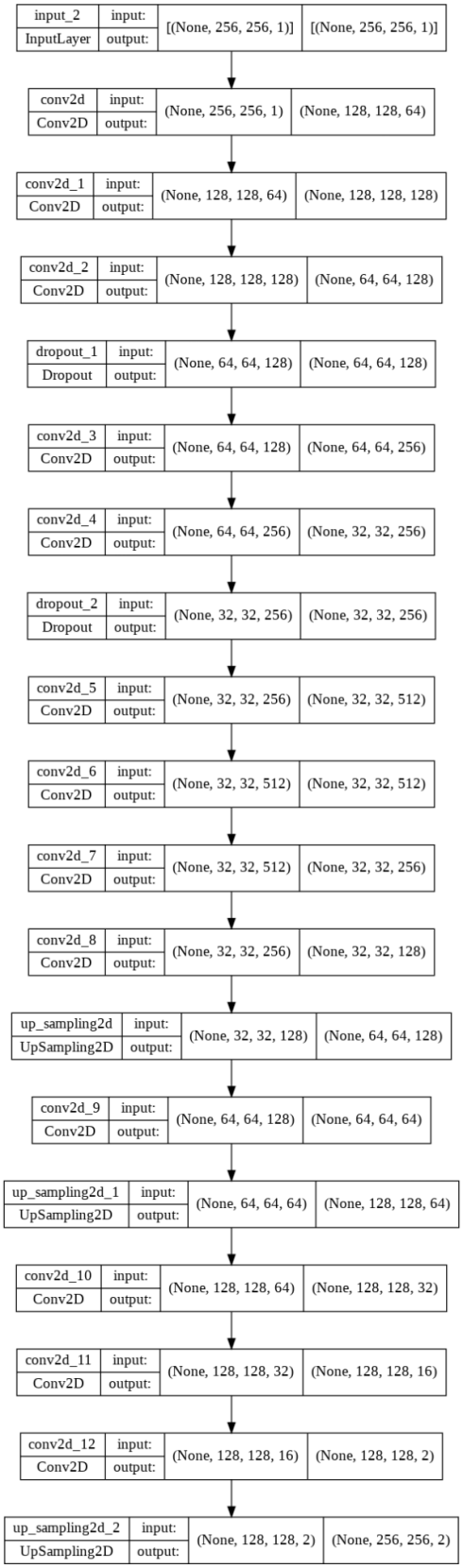
Baseline Model

After applying preprocessing steps in the training set , we give as input to the baseline autoencoder the grayscale images of size 256 x256 x 1 that contain only the lightness values.

The architecture of the baseline model is described as follows:

- The encoder part consists of 8 convolutional layers with 3 x 3 kernels. Each convolutional layer uses the RELU activation function. Also, in the first, third and fifth layers is applied a stride of 2 for decreasing the width and height of the image vector.
- The fusion network consists of one convolution layer with 1 x 1 kernel and stride of 1, which contribute to the uniform distribution of the semantic information conveyed by the feature vector. The RELU activation function is used again.
- The decoder part has 5 convolutional layers with up-sampling layers in order to “decompress” the fusion feature vector to the dimensions of the input image (256 x 256 pixels). Upsampling is done by a factor of 2 in order to effectively double the size of the image in each step. Also, in the final layer the decoder reconstructs the image with 2 filters that each one represents the ab channels. All layers are using the RELU activation function except for the last one that uses the tahn. The reason for choosing this activation function is because the normalized ab values have a range between (-1,1) and tanh is used for squashing the values between (-1,1).

The architecture of the baseline model is summarized below:


input_2 | InputLayer | input: | output: | [(None, 256, 256, 1)] | [(None, 256, 256, 1)]
conv2d | Conv2D | input: | output: | (None, 256, 256, 1) | (None, 128, 128, 64)
conv2d_1 | Conv2D | input: | output: | (None, 128, 128, 64) | (None, 128, 128, 128)
conv2d_2 | Conv2D | input: | output: | (None, 128, 128, 128) | (None, 64, 64, 128)
dropout_1 | Dropout | input: | output: | (None, 64, 64, 128) | (None, 64, 64, 128)
conv2d_3 | Conv2D | input: | output: | (None, 64, 64, 128) | (None, 64, 64, 256)
conv2d_4 | Conv2D | input: | output: | (None, 64, 64, 256) | (None, 32, 32, 256)
dropout_2 | Dropout | input: | output: | (None, 32, 32, 256) | (None, 32, 32, 256)
conv2d_5 | Conv2D | input: | output: | (None, 32, 32, 256) | (None, 32, 32, 512)
conv2d_6 | Conv2D | input: | output: | (None, 32, 32, 512) | (None, 32, 32, 512)
conv2d_7 | Conv2D | input: | output: | (None, 32, 32, 512) | (None, 32, 32, 256)
conv2d_8 | Conv2D | input: | output: | (None, 32, 32, 256) | (None, 32, 32, 128)
up_sampling2d | UpSampling2D | input: | output: | (None, 32, 32, 128) | (None, 64, 64, 128)
conv2d_9 | Conv2D | input: | output: | (None, 64, 64, 128) | (None, 64, 64, 64)
up_sampling2d_1 | UpSampling2D | input: | output: | (None, 64, 64, 64) | (None, 128, 128, 64)
conv2d_10 | Conv2D | input: | output: | (None, 128, 128, 64) | (None, 128, 128, 32)
conv2d_11 | Conv2D | input: | output: | (None, 128, 128, 32) | (None, 128, 128, 16)
conv2d_12 | Conv2D | input: | output: | (None, 128, 128, 16) | (None, 128, 128, 2)
up_sampling2d_2 | UpSampling2D | input: | output: | (None, 128, 128, 2) | (None, 256, 256, 2)

It is important to mention that in the model no pooling layers were used for reducing the dimensionality of the input grayscale image. That is because in the case of image colorization it is crucial that the image doesn't get distorted. Pooling helps the neural network to increase the information density but at the cost of decreasing the image's size and quality. This is not a problem for other visual neural networks aimed for the task of image classification, because the main goal is the identification of certain features. Hence, in the baseline model the dimensionality reduction was done using the stride of 2 in some convolutional layers so as to decrease the width and height of the image vector by half. This way information density is increased but without distorting the image.

Furthermore, in order to keep the ratio of the image the same as the original image, padding was used in every convolutional layer. Without using the padding each convolutional layer would cut the image.

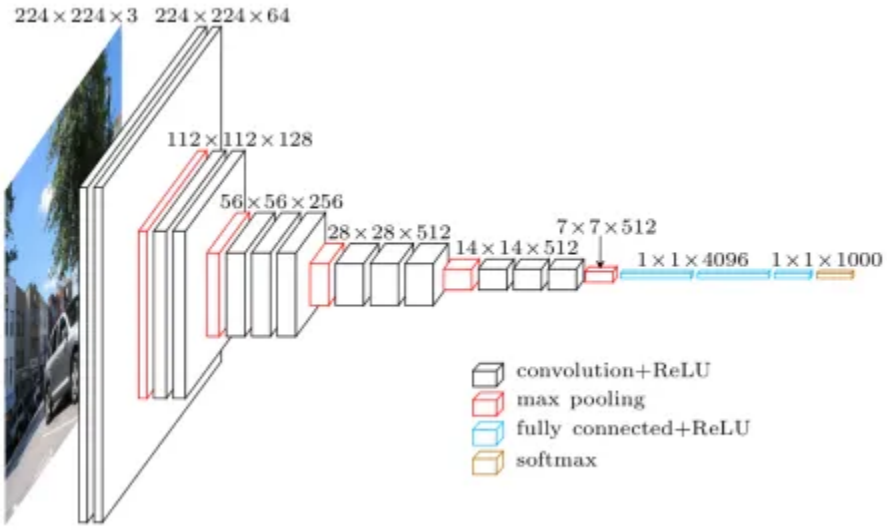
Transfer Learning

Transfer learning is a popular approach in deep learning where pre-trained models can be exploited as the starting point in a second task. By pre-trained, we define models that are saved and were previously trained on a large dataset. This means that we expect them to serve satisfiably well as generic models of the visual world.

For this project, we aim to extract high-level features from the VGG-16 pre-trained model.

VGG-16

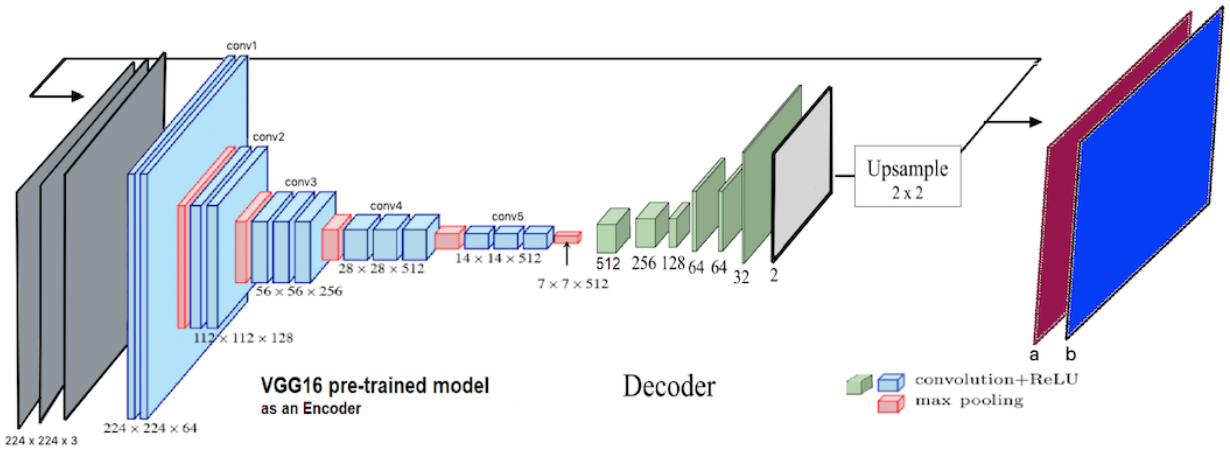
The VGG16 is a standard deep Convolutional Neural Network (CNN) architecture that took second place in the ImageNet Large Scale Visual Recognition Challenge in 2014 (ILSVRC 2014) and is considered to perform very efficiently till date. 'VGG' is the abbreviation for Visual Geometry Group, which is a group of researchers at the University of Oxford who developed this architecture, and '16' implies that this architecture has 16 layers.



The figure above illustrates the architecture of the VGG16 model. The input layer takes an image of fixed size 224 x 224 with 3 channels, and the output layer is a softmax prediction on 1000 classes, because it was designed for the image classification task. The input layer is alternately followed by convolution layers of 3x3 filter with stride 1 and same padding and then maxpool layers of 2x2 filter and with stride 2. In the final two fully connected layers softmax is applied for output. The VGG-16 is a large network that has about 138 million parameters.

For this project, our goal is to use the VGG-16 model for feature extraction. This means that we want to replace the encoder part that we used in the baseline model with a part of VGG-16.

In particular, the feature extraction part of VGG-16 is considered to be from the input layer to the last max-pooling layer (labeled by 7 x 7 x 512 in the figure), while the rest part of the network is considered to be the classification part, which we are going to remove. Therefore we will employ the first 19 layers of the model, use the corresponding feature vector of the VGG-16 to feed the decoder and finally generate the ab channels of the input image.
In order to take full advantage of the pre-trained model, we will not change the original weights of the VGG-16 model by applying further training and use them as they're provided by the keras library.



As mentioned above, the VGG-16 model requires input images of size 224 x 224 x 3. Since we are not using RGB space as it is expected for the VGG-16

model, we assign to each channel the lightness values that correspond to each grayscale image.

Post-processing Steps

All preprocessing steps applied to the training set are repeated to the test set. After predicting the ab channel for each test image, we concatenate the L channel that we used as the input to the model, together with the predicted channels to reconstruct the image in the LAB space and then we transform it back to RGB to be able to display it.

Loss Function

The optimal model parameters for the models are found by minimizing a loss function defined over the predicted output and the target output. So, for the training we are using the Mean Square Error as the loss function between the estimated pixel colors in a*b* space and their real value.

Also the RMSPROP optimizer is employed and the number of epochs is a "Trial and error" choice.

Evaluation method

Evaluating the quality of the predicted colorized images is quite a challenging task. In comparison with classification problems, image colorization deals with generating images instead of labels. This implies that the main challenge in the image colorization task is that there is no unique solution. For many objects various colors could be acceptable. For this reason, it is proposed that the results are evaluated both quantitatively and qualitatively.

For the quantitative approach of the evaluation, we employ Mean Squared Error (MSE) as a loss function, like the aforementioned. However, the problem of image colorization is multimodal, so it is possible that the model prefers to colorize features with more desaturated colors than bright colors, because desaturated colors would result in a penalty of lower MSE. In cases where the colorization of features is fixed by natural laws, the quantitative evaluation can be helpful. For example, the ocean should have a shade of blue, grass green etc.

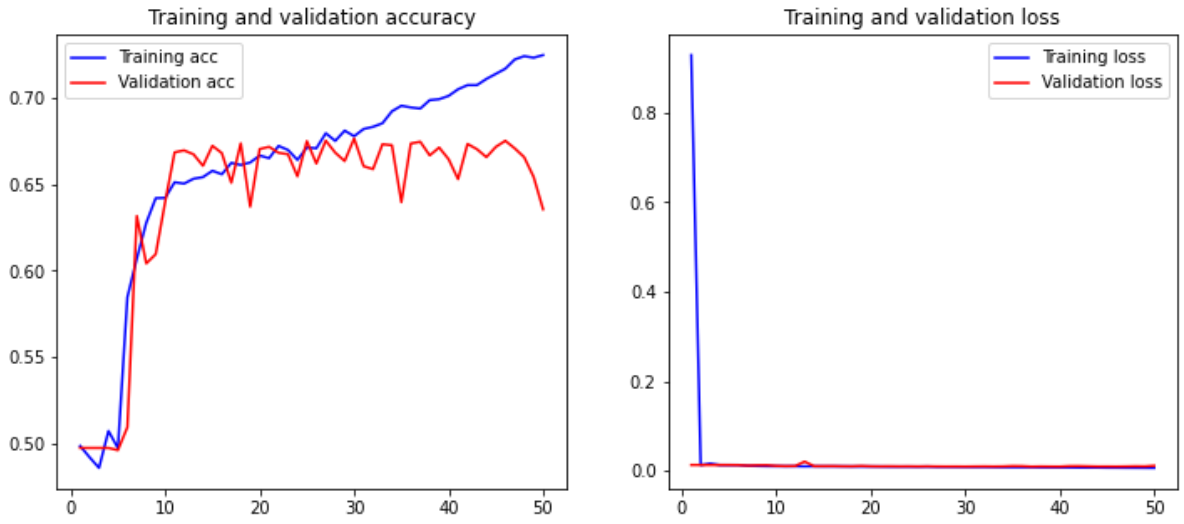
Additionally to MSE, we also employed the accuracy measure. Except for the MSE loss function, unfortunately there is no available metric which clearly specifies the quality of a colorization.

For the qualitative approach of evaluation, the predicted images were shown to humans for visual inspection so as to judge how realistic each feature coloring is in the predicted images.

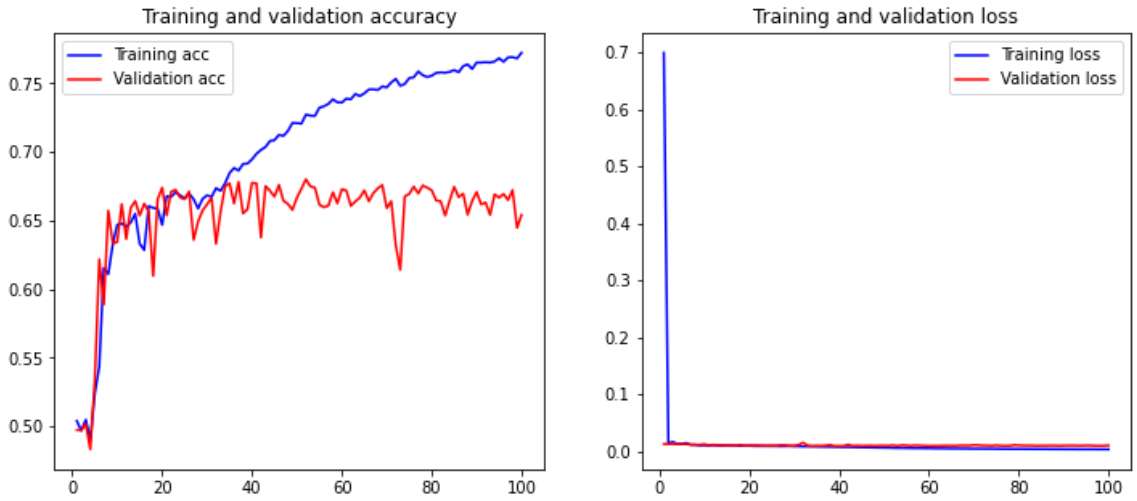
Experimental Results and Discussion

After the training process we printed the loss curves together with the accuracy curves for the training and validation sets for both the baseline and transfer learning model.

- For the baseline model:
  Firstly, for 50 epochs of training the mean accuracy score was 67.48 % and bellow are presented the training curves:



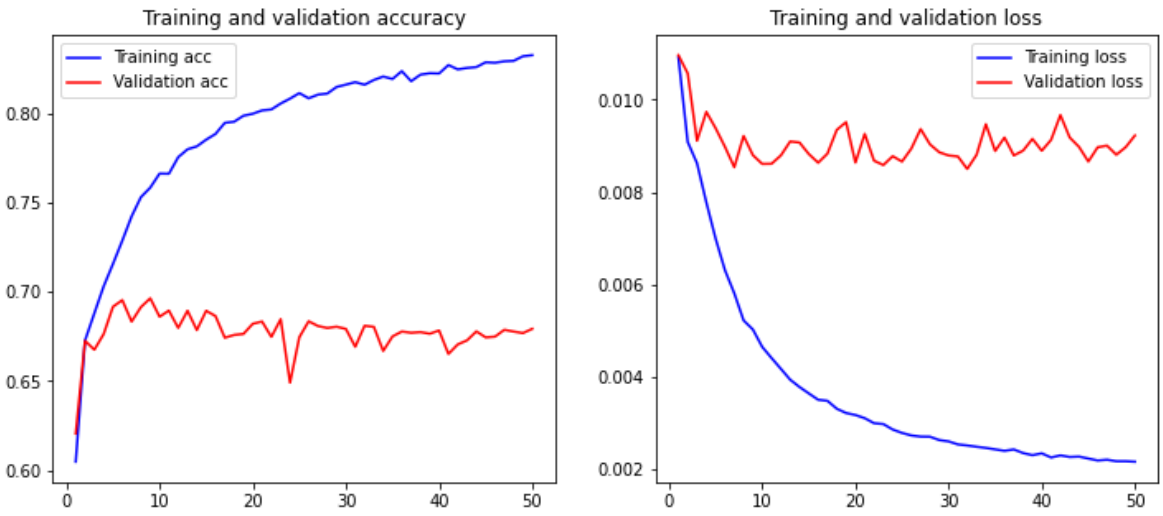
For 100 epochs the mean accuracy score was 67.62 % and we retrieved the following training curves:



For both number of epochs the training and validation loss curves are close to each other and show a long plateau. The curve of accuracy though, for 100 epochs is more stabilized for the validation set. Also by taking a visual inspection of the predicted images we concluded that the results for 100 epochs are more satisfactory.

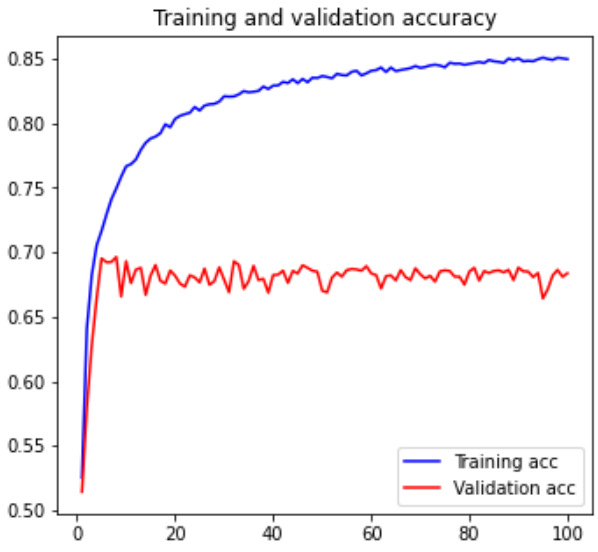
For the model using the VGG-16, the corresponding training curves are :

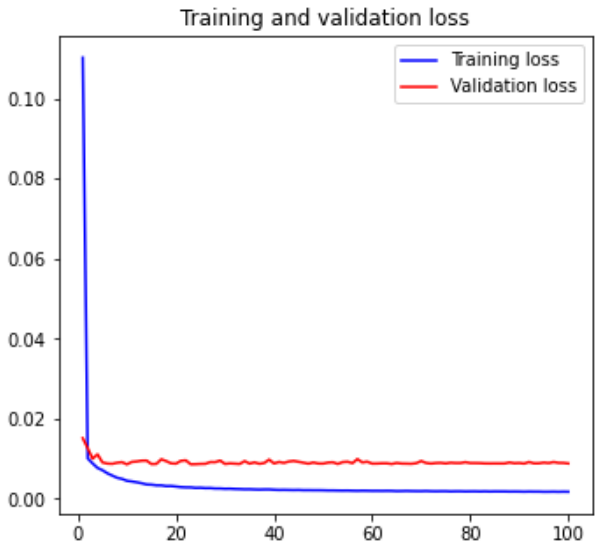
For 50 epochs the mean accuracy score was 67.77 % and the accuracy and loss curves we got during training are presented below:



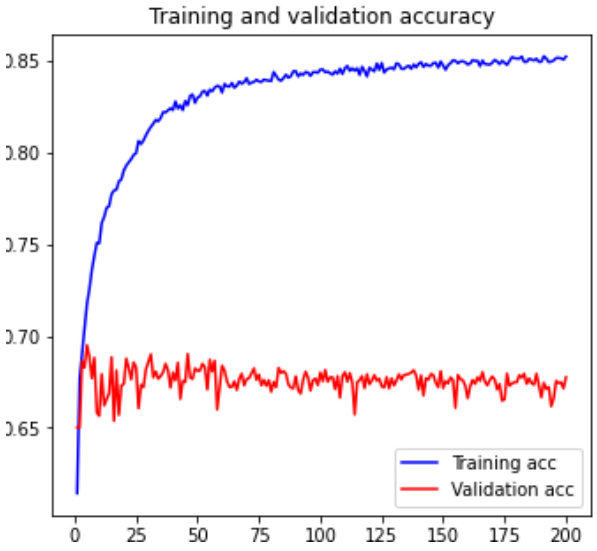
The loss curve of the training set seems like it hasn't yet reached a minimum value and has no plateau.

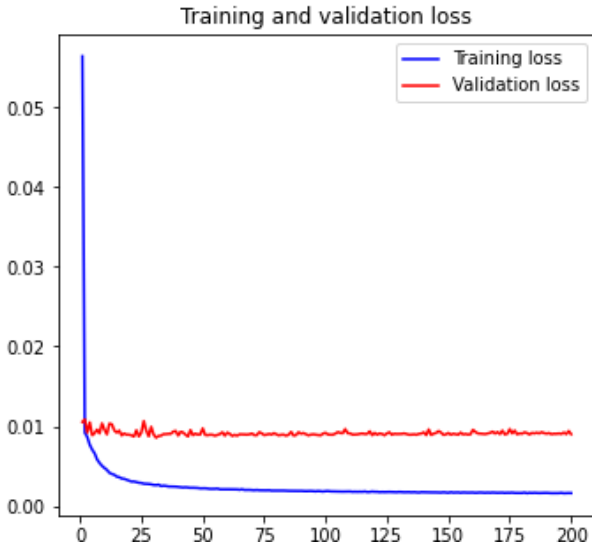
For 100 epochs the mean accuracy was 67.91% and during training we got the following curves:





For 200 epochs the mean accuracy was 67.43% and bellow are the corresponding accuracy and loss curves during training:





Both loss curves for the training set show a long plateau and have reached a minimum value. Nonetheless, the accuracy curve of validation set for the 200 epochs presents more fluctuations. Additionally, we compared the predicted images for 100 and 200 epochs and concluded that for 100 epochs the performance was a bit better.

After completing the quantitative evaluation of the results, we additionally visually inspect the results.
In the left side is depicted the actual image of the dataset belonging to the test set, in the middle the predicted colorized image from the baseline model and in the right side the predicted image from the transfer learning model.

❖ **Image 1**

Both models were able to clearly colorise the cloudy sky and the mountaintop. However in the image of the baseline model, there are some "specks" of green-brown color in the snow area, probably because the model was expecting to place some grass or soil. In the model where VGG-16 was applied, the snow is more bright white.

❖ Image 2

It seems like the model that uses the VGG-16 model can identify the features of the building more precisely but it uses lighter colors than the original photo. The baseline model uses a little more bright blue for the sky but the building is colorised with some green areas that are not present in the original photo.

❖ Image 3

Both images of the models can be semantically acceptable, meaning that the mountain land could be green if it contained grass. However, in the original photo the land is drought and has light brown color. The sky has correctly in both cases a light blue color but more desaturated than the original photo.

❖ Image 4

This image is a complex photo of an urban environment containing many features like roads, cars, signs, skyscrapers etc. For this reason, both models faced difficulty in colorising all of them. In the right image, the sky is light blue, the road is correctly colored in gray and all buildings are correctly colorized but with more desaturated colors than the original image. Only the pavement is wrongly colorized the same color as the road and not red as expected. In the middle image, the colorization of the buildings contains some small light green blur areas.

❖ **Image 5**

In this image the model using the VGG-16 performed poorly and the intensity of all colors is very low. The road is light gray, the grass in the right corner is light green and above trees there is some portion of light green. In the baseline model, the selection of colors is more suitable and closer to the original picture but the colorising areas have surpassed the right ones.

❖ **Image 6**

This is also a very complex image that contains a lot of information. The VGG-16-based model basically managed to colorize with light green the ivy in the roof and the bellow plant. The baseline model, placed a more vibrant green color additionally in the plants placed in the right side of the picture and tried to colorise with yellow the stripe in the road.

❖ **Image 7**

Again this image contains a lot of information. The right picture contains in the sky a very light blue color and light green in the trees. The baseline model uses more bright colors but is not so consistent in placing them correctly.

❖ **Image 8**

This image contains an interesting blue gradient colorization in the sky. The VGG-16-based model has achieved this gradient colorization using a little darker shade of blue. In the baseline model, the transition to the darker blue area in the top of the image is more abrupt. Neither of the models managed to colorize with blue the small area between the clouds and the mountain.

Overall we can conclude that both models managed to make some plausible colorizations. We noticed that the model using the VGG-16 was on the one hand more accurate in identifying the edges of the objects in the image, but on the other hand more conservative in using more vibrant colors. On the other hand, the baseline model was able to use more bright and vibrant colors that were closer to the original images and it also chose to color bigger areas of the image, resulting usually in the colorizations exceeding the limits of the objects.

Future Thoughts

As we concluded from the predicted results, the image colorization task is indeed a very challenging and complex problem that needs a lot of experimentation. Some ideas that we suggest to increase the effectiveness of our models are the following:

- Run the same models on a larger dataset of the same topic

We noticed that the consistency of the dataset's content plays a game role in the effectiveness of the results. In our project, the volume of the dataset was not that big and its content was diverse, containing scenes from complex urban environments and nature landscapes at the same time. So, for instance in order to achieve better colorization of photos depicting cities we should focus on this content.

- Experiment on different loss functions except for MSE
- Overcome the limitations of computational resources (Google Colab)

Another challenge that was very crucial on this project was the limited access to RAM and GPU provided by the Google Colab. We tried running on a bigger dataset of 5000 images and on the existing dataset for more epochs but the session had crashed.

- Experiment on other pre-trained models like InceptionResNetV2

Bibliography shows that there are available pre-trained models that can be employed for feature extraction which are more complex than the VGG-16.

- Apply it to video

It would be very interesting to see the results of applying colorization to black-and-white video. Of course the applications of video colorization could exceed the ones of image colorization.

References

How to colorize black & white photos with just 100 lines of neural network code | by Emil Wallner | Medium

Image Colorization using Convolutional Autoencoders | by Eryk Lewinson | Towards Data Science

Automatic Colorization (tinyclouds.org)

Auto Colorization of Black and White Images using Machine Learning "Auto-encoders" technique | by Mahmoud El-Jiddawi | Becoming Human: Artificial Intelligence Magazine

Recolorizing Black and White Landscape Images Using Convolutional Neural Networks, Lauren Zhu, Jacqueline Ennis, Jacob Reiter, Stanford University

Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2, Federico Baldassarre, Diego Gonzalez Morin, Lucas Rodés -Guirao

Fast Colorization of Grayscale Images by Convolutional Neural Network, Swathy Titu, Jency Rena N.M

Fully automatic image colorization based on Convolutional Neural Network, Domonkos Varga*†, Tamas Szir ´ anyi ´ *‡ *MTA SZTAKI, Institute for Computer Science and Control