Colorizing grayscale images using neural networks

Denis Sinanaj
Computer Science Department
Universiteti “Ismail Qemali”,
Vlore Albania
denissinanaj@gmail.com

Dezdemona Gjylapi
Computer Science Department
Universiteti “Ismail Qemali”,
Vlore Albania
dezdemona.gjyapi@univlora.edu.al

Abstract

Colorization of grayscale images has become a new research area in the recent years, thanks to the advent of deep convolutional neural networks. The automatic image coloring consists in adding colors to a new grayscale image without any user intervention. The automatic image coloring suggests that we cannot determine the colors that are needed in an image, so it will be done without any prior knowledge.
In this paper, the convolutional neural network is used as a method for colorizing grayscale natural images. The model architecture is a combination of a convex networking architecture with Inception-ResNet-v2, which assists the overall coloring process by extracting high-level features. The approach is tested on machines using CPU and GPU. In both cases the output image is very close to reality, except in machines using CPU it takes a lot of time for the network to be trained.

1. Introduction

Neural networks are constantly on trend and are considered as key points in many research areas.
Artificial neural networks have generated a lot of optimism with respect to research in the Machine Learning industry, thanks to very important results in language recognition, computer vision and text processing.
The purpose of this paper is to introduce neural networks as a highly efficient technique in coloring grayscale images.
There are many other methods that represent color images, but they require the user's attention.
The method presented in this paper, is intended by the user to make no action on the image.
Expectations from the use of neural networks in the coloring grayscale images are:

- The method will be fast, giving the result in a few minutes.
- The result will be very close to reality.

2. ANN

A Neural Artificial Network (ANN) is a computable model inspired by the way biological neural networks process information in the human brain. An ANN consists of a number of simple and interconnected processors, also called neurons, analogous to biological neurons in the brain.
Each neuron receives a number of input signals through its connections; however, it gives no more than one output signal. The output signal is transmitted through the neuron’s exit line [Neg05].
Developing an ANN comprises the definition of [Gjy16]:

1 - the network architecture, which is defined by the basic processing elements (i.e. neurons) and by the way in which they are interconnected (i.e. layers);
2 - the NN Learning, which implies that a processing unit is capable of changing its input or output behavior as a result of changes in the environment,

i.e. to adjust the weights based on input vector values;

3 - the data used for training, testing and validating the neural network.

The basic unit of calculation in a neural network is the neuron, often called a node or unit. It receives input from some other nodes, either from an external source, and calculates an output. Each input has a related weight (*w*), which is determined based on its relative importance to other inputs.

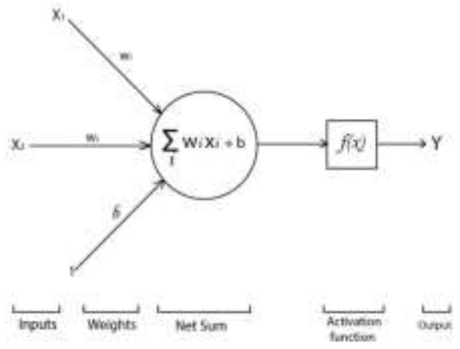

Figure 1: Perceptron

The output Y from the neuron is calculated by the function *f(x)*. The function *f(x)* is nonlinear and is called the Activation Function. The purpose of the activation function is to derive a nonlinear result in the output of a neuron. This is important because most of the real data in the world are not linear and we want neurons to learn this non-linear data.

There are some activation functions that we may encounter in practice:

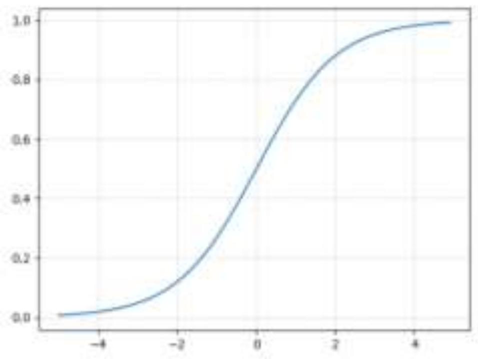

Figure 2: Sigmoid function graph

The sigmoid function takes a real value as input and transforms it between values 0 and 1 as shown in figure 2. It is very suitable for probability calculations.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

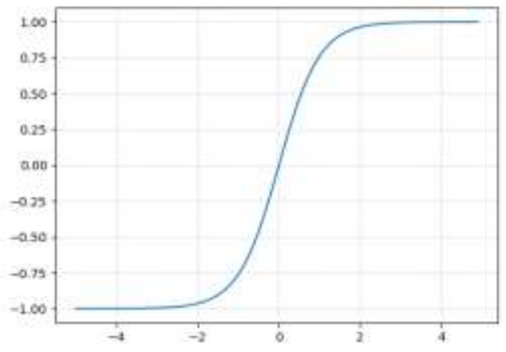

Figure 3: Hyperbolic Tangent Function Graph

The hyperbolic tangent function, *g(x)=tanh(x,* which takes a real input value and transforms it between values -1 and , as shown in figure 3.

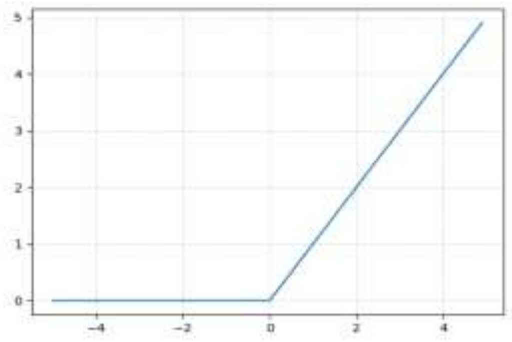

Figure 4: ReLU function graph

Rectified linear units function (ReLU), figure 4:

$$f(x) = \begin{cases} x, per\ x > 0 \\ 0, \qquad\qquad \neq \end{cases}$$

The advantage of ReLU is the speed of calculation. The ReLU is extremely fast and is now widely used in most layers and activation units.

3. Convolutional Neural networks

Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks that have been very effective in areas such as image recognition and classification [Mor17]. They have also been successful in identifying faces, objects, and traffic signs, but also in the field of robots and autopilot machines.

3.1 Convolution layer

Convolution is a mathematical operation that is used in the signal filtration process, to find patterns in signals, etc. [Mor17]. In the convolutional layer, all neurons apply the convolution operation to the inputs, so they are called convolutional neurons. The most important parameter in a convolutional neuron is the size of the filter. Let's say we have a layer with a filter size of 5 x 5 x 3. Also assume that the entry given to the convolutional neuron is an image 32 x 32 with 3 channels.

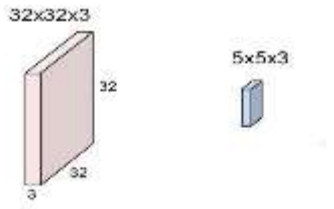

Figure 5: Image and Selected Filter

We used one of the 5x5x3 parts out of the image and calculated the convolution (point product) with our filter (w). This operation gives as a result a single number as output. We will also add bias (b) to this output.
In order to calculate the product of the points, it is mandatory for the filter third size to be the same as the number of channels in the input. I.e. when we do the calculation, the point product is a multiplication of the 5 x 5 x 3 matrix with filter size of 5 x 5 x 3.

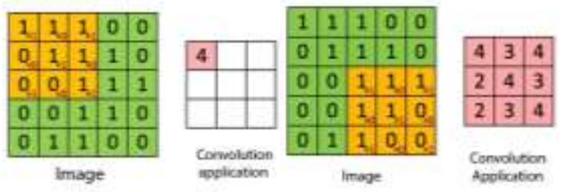

Figure 6: Convolution application scheme

In this case, we moved to our image with 1 pixel per step. In some cases, it can be exceeded by more than 1 pixel. This number is called a big step (stride). The stride controls how the filter convolves around the input volume.
From the figure above, after each convolution, the output decreases in size (as in this case we are going from 32 x 32 to 28 x 28). In a multi layered deep neural network, output will become too small in this way, which does not work very well.
A standard practice is to add zeros to the input layer boundary so that the output has the same size as the input layer.
Let's say we have an input of size N * N, the filter size is F, we are using S as the stride and the input has no padding, then the output size will be:

$$(N-F + 2P) / S + 1$$

3.2 Pooling Layer

Spatial pooling (also called downsampling) reduces the dimension of each map of features, but saves the most important information.
Spatial pooling can be of different types: Maximum (Max), (Average) Average, (Sum) Sum etc.
In the case of Max Pooling, we define a space (for example, a window 2 × 2) and we get the largest element from the corrected feature map within that window.
Instead of getting the biggest element, we can also get the average (Average Pooling) or the sum (Sum Pooling) of all elements in that window.
In practice, Max Pooling has been shown to work better.

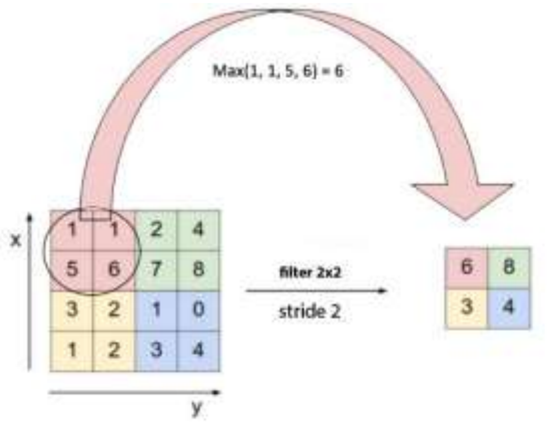

Figure 7: Pooling using 2x2 Filter and a stride of 2.

Source: [Kar18]

The figure 7 shows the Max Pooling operation on a mapping obtained by applying the ReLU + convolution operation, using a 2x2 filter.

ReLU is an elementwise operation (applied to the pixel) and replaces all negative pixel values in the feature map with zero. The purpose of the ReLU is to introduce non-linearity in our convolutional network, as most of the real-world data we would like to be learned by our network should be non-linear (Convolution is a linear operation – multiplication and matrix addition element for element, so by introducing a non-linear function such as ReLU we included nonlinearity).

The ReLU operation can be clearly understood from the below figure.

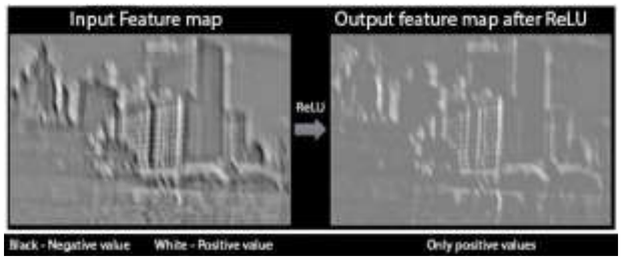

Figure 8: ReLU Operation. Source: [Kar16]

Other non-linear functions such as *tanh* or *sigmoid* can also be used instead of ReLU, but ReLU performs better in most situations [Sze16].

If the input is of size w1 * h1 * d1 and the filter size is f * f with step S. Then output sizes w2 * h2 * d2 will be:

w2 = (w1-f) / S +1

h2 = (h1-f) / S +1

d2 = d1

The most common union is done with a size filter of 2 * 2 with a step of 2. As can be calculated with the formula above, pooling essentially reduces the input size by 50%, and makes the number of parameters and network calculations more manageable.

It makes the network unchanged by small transformations, distortions and translations in the input image (a small distortion in data will not change pooling output – since we get the maximum / average value in a local space).

It also helps us reach an almost unchanged view of our image. This is very important because we can discover objects in an image no matter where they are.

3.3 The Convolutionary Network Elementary Blocks

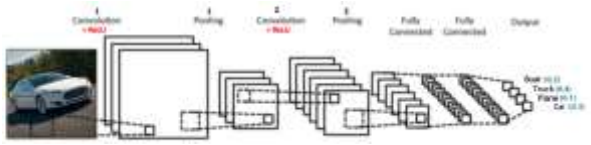

Figure 9: The convolutional neural network blocks

Source: [Kar16]

Above we have seen the Convolution, ReLU and Pooling. It is important to understand that these layers are the basic blocks of each convolutional neural network. As shown in Figure 9, we have two sets of Convolution, ReLU, and Pooling layers – the second Convolution layer converts to First Pooling Layer output using six filters to produce six feature maps. Then the ReLU is individually implemented in all these six featured mappings that are also followed by the Max Pooling operation.

Together, these layers extract useful features from images and place non-linearity on our network.

3.4 Fully connected layer

If each neuron on one layer receives input from all neurons from the previous layer, then this layer is called a fully-connected layer. The production of this layer is calculated by multiplying the matrix followed by the bias offset.

Output from the convolutional layers and the pooling layer presents high-quality image input features. The purpose of the fully-connected layer is to use these attributes for classifying the image of entry into different classes based on the data set of training.

In addition to the classification, the addition of a fully-connected layer is also an easy way to learn non-linear combinations of these characteristics. Most of the features from convolutional and pooling layers can be good for the task of classification, but combinations of these characteristics may be even better.

4. Architecture

Following all the above aspects of convolutional networks a method proposed for colorization is represented by using this network and his classification capabilities.

The model architecture is a combination of a convex networking architecture with Inception-ResNet-v2, which assists the overall coloring process by extracting high-level features.

The first part is an encoder. The last part is decoder. In the central part lies the fusion layer. The fusion layer gets the output from the encoder and the embedding generated by the Inception-Resnet-V2 model and connects the two results before proceeding. Model Inception-ResNet-V2 is trained in the data set from the Unsplash site.

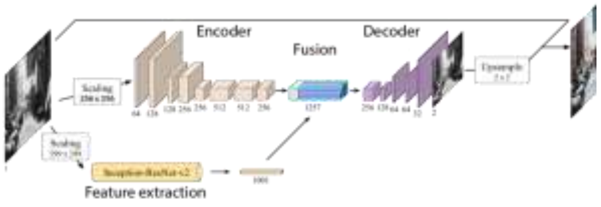

Figure 10: Network architecture

It is a very good architecture to understand the dynamics of the coloring problem.

5. The Image

An Image is a matrix of pixel values. All the above architecture is used taking as input an image. All we need to know in regards of this problem is to understand the image color spaces and components used by them. The two colors spaces that were used were RGB and Lab.

5.1 RBG color space

Channel is a conventional term used to refer to a particular component of an image. An image from a standard digital camera has three channels – red, green and blue – that we can imagine as three 2d matrices placed on each other (one for each color), each with pixel values ranging from 0 to 255 [Skr17].

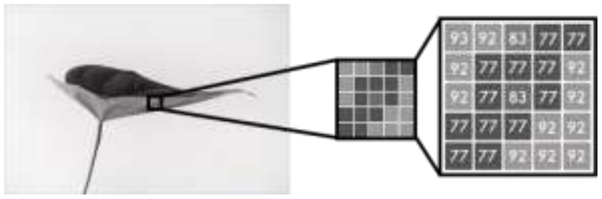

Figure 11: Grayscale Image. Source: [Wal17]

A grayscale image, on the other hand, has only one channel. So we will only consider grayscale images, so we will have a single 2d matrix representing an image. The value of each pixel in the matrix ranges from 0 to 255 – zero that indicates the black and 255 indicates the white.

Color image in RGB consist of three layers: a red layer, a green layer and a blue layer. This may be counter-intuitive for us. Imagine dividing a green leaf with a white background in three channels. Intuitively, we can think that the plant is only present in the green layer but this is not true.

But, as seen below, the sheet is present in all three channels as an image in RGB has three different layers(Red, Green, Blue). Layers not only define color but also lightening.

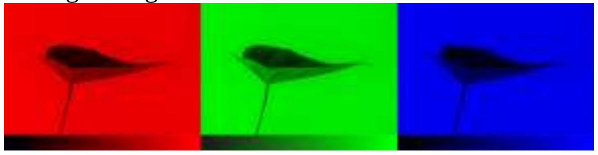

Figure 12: Three layers that define the color. Source: [Wal17]

Like grayscale images, each layer in a colored image has a value of 0-255. Value 0 means that there is no color in this layer. If the value is 0 for all color channels, then the image pixel is black. As is well known, a neural network creates a connection between an input value and output value. To be more precise with our coloring task, the network needs to find the features that connect grayscale images to colored images.

So, the features needed to be found are those connecting the values of the grayscale image pixel matrix to those of three color matrices.

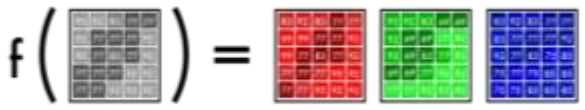

Figure 13: Illustration of the connection to be found. Source: [Wal17]

We started by making a simple version of our neural network to color an image. This way, we saw the core syntax of our model while adding features to it.
By means of the prototype code, we were able to make the transition below. The picture in the middle is made with our neural network and the picture to the right is the original color photo. The network is trained and tested on the same image.

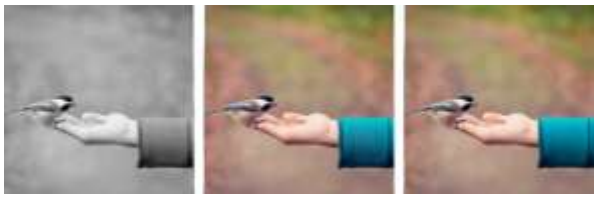

Figure 14: Input image (left), output (middle), Original (right)
On the next step, we changed color channels, from RGB to Lab.

5.2 Lab color space

L represents the degree of lightness of the image, while the a and b spectrum of colors: green-red and yellow-blue. In the Figure 15, an image presented in Lab mode has a grayscale layer, and is packed in two, three color layers that could have been in RGB mode. This means that we can use the original pixel image in our final prediction.

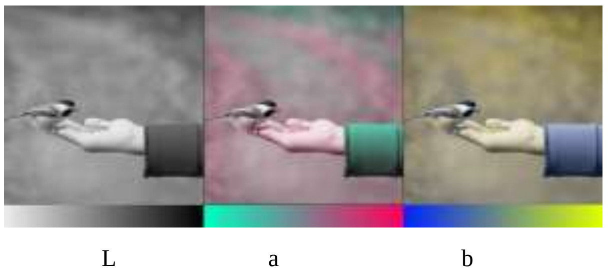

Figure 15: Image in Lab color space

Lab packing makes it possible for us to have only two channels to predict.

Our final prediction will be:

We have a grayscale layer as the input, and we want to predict two colored layers, a and b in the Lab color space. To create the ultimate color image we will include the L / grayscale image we used for the input. The result will be creating a Lab image.

6. Processing

Our neural network finds features that connect grayscale images with their color versions [Wal14].

We had to color the grayscale images – but as a restriction, we set to see only nine pixels at a time. Each image can be scanned from left to right from start to finish and it is possible to predict what color each pixel should be.

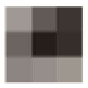

Figure 16: nine pixels

For example, these nine pixels may be the bird's tail corner in the photo we used in the picture below. It is very difficult to make a good coloring, so it is necessary to divide the coloring into steps.

First, it searches for simple models: lines, circles, simple shapes, all black pixels, and so on. Looks for the same exact pattern in any space and removes pixels that do not match. So 64 new images from 64 mini filters are generated.

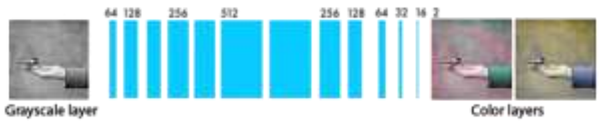

Figure 17: Image generated filters

If the image is scanned again, the same small models that are already detected will be viewed. To get a clearer picture of the image, the image shrinks in half.

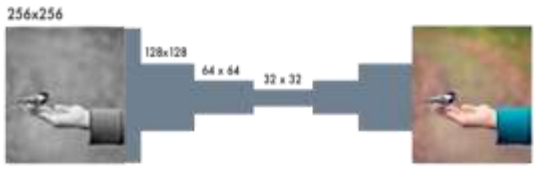

Figure 18: Image filtering

Still there is only a 3x3 filter scanning any image. But combining nine new pixels with lower-level filters can reveal more complex models. A combination of pixels can form a half circle, a small dot, or a row. Again, the same model from the image is repeatedly extracted. This time, 128 new filtered images are generated.
After a few steps, the filtered images that are produced will look like in figure 19.
As mentioned, it starts with small features, such as a corner or very simple shapes. Layers close to Output are combined in models. Then, they are combined in detail, and eventually transformed into a view of the image.

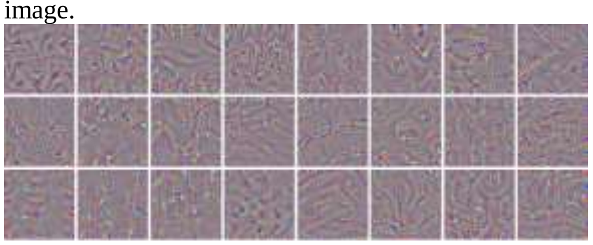

Figure 19: Filtering result

The process is similar to that of most neural networks that deal with vision. Therefore and here the network is defined as a neural convolutional network. In these networks, we combine some filtered images to understand the context of the image. With the context meaning the network can determine the exact color with which the coloring has to be realized.

The implementation of the network is made using python as programming language and tensorflow [Aba16], an open source library for machine learning. Also Keras [Gul17] was used due to his high support for modularity.

We used a cloud platform called floydhub for running the project in a 2 hour free GPU usage to have a better result due to his reliable structure for handling such projects.

7. Training

The neural network acts in such a way that makes tests and makes mistakes. First, it makes a random prediction for each pixel. Based on the error for each pixel, it works in reverse through the network to improve the characteristics extraction [Dah16].
It begins to adapt to situations that create the biggest mistakes. In this case, the settings are: whether to color or not, and how to locate different objects.
The network begins by coloring all the objects in the image by brown color. It is the color that is most similar to all other colors, thus producing the smallest error.
The main difference that the network has from other neural visual networks is the importance of pixel locations. In coloring networks, the size or aspect ratio stays the same throughout the network. In other types of network, the image becomes distorted when approaching as many final layers as possible.
Max pooling layers in classification networks increase the amount of information, but also distort the image. It only provides information, but not the presentation of an image. In colorizing networks, a stride of 2 (two) is used to halve the width and height. This also increases the amount of information, but does not distort the image.

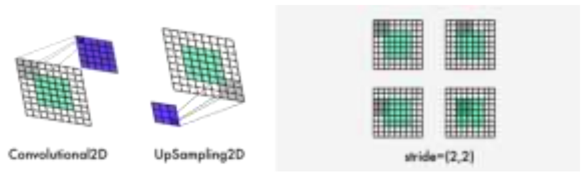

Figure 20: Image Upsampling and padding

Two further distinctions are: increasing the number of pixels by upsampling and maintaining the proportions of the image. Classification networks give as result a final classification of the objects in an image. Therefore, they continue to reduce the size and quality of the image while passing through the network.
Coloring networks maintain continuously the same image ratio. This is done by adding white padding as the above image.

8. Results

The outputs below are obtained by training the network only through the CPU.
During training using CPU and GPU the image is very close to reality.
CPU training only takes a lot of time, so for this problem the training definitely requires a very good GPU.
The training is done through the CPU to give the idea that the network learns on the right track.
Due to the inability of a powerful GPU, a 2-hour floydhub time is used using their cloud graphics card.
The result archived is presented in figure 21.
The main limitation of the method lies in the fact that it is able to color images that are in common with images for which the network is trained.
The training is done in open and closed environments to mitigate this.
Coloring will only apply to real-world images rather than self-generated images.
To get good quality transfer results, it's important for both images have a semantic level of similarity between them, though the image itself may change drastically.
The best results are achieved when images belong to a context, though they may be very different from each other.

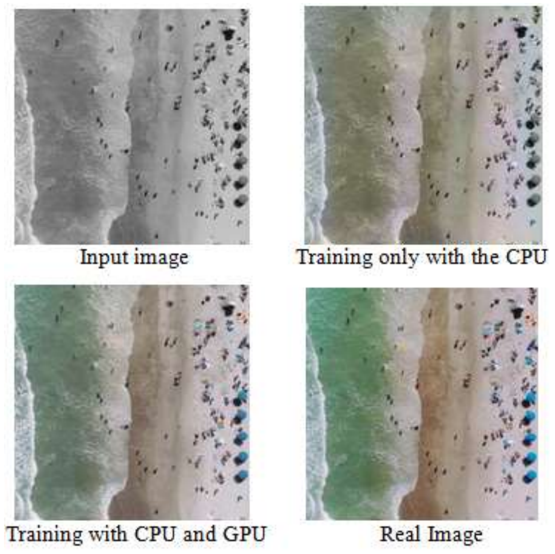

Figure 21: Results from colorization

Coloring is also an unclear problem: e.g. what color can have a plastic object? This has no unique solution. By learning from the data, our model will mainly use the predominant colors it has learned.

9. Conclusions

We merged global and local information architecture for coloring grayscale images.
The time to reach a satisfactory result is enormous because the network is trained only through CPU capabilities.
Using GPU in a 2 hour time span was used in few images but yielded a good result in a few minutes.
The implementation containing the convolutional neural network was able to perform coloring without any user intervention. Our network is trained in a very large set of data that determines what is in the image and stains the images from the context of the image.
The implementation allows us to process images in a 256x256 resolution leaving as color matching work for images of any resolution.
As the network is trained with a data set, it enables the coloring of an image to be based on the context of another.

10. Further work

The network is executed in a small set of data due to limited resources. In the future, will be trained in a much larger set of data and compare the results.
We aim to do this by using the entire Unsplash dataset from which we used only some training images in the above case. Achieving this and enabling training on a computer with very good parameters such as a powerful GPU would accomplish the process in a shorter time, a few seconds.

References

[Neg05] Michael Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems.*: Pearson Education, 2005.

[Gjy16] Dezdemona Gjylapi, Eljona Proko, and Alketa Hyso, “Genetic Algorithm Neural Network model vs Backpropagation Neural Network model for GDP Forecasting,” in *2nd International Conference on Recent Trends and Applications in Computer Science and Information Technology*, vol. CEUR Workshop Proceedings 1746, Tirane, 2016, pp. 23-29.

[Mor17] Diego González Morín, Lucas Rodés-Guirao Federico Baldassarre. (2017, Dec.) Deep Koalarization: Image Colorization using CNNs and Inception-Resnet-v2. arXiv:1712.03400. [Online]. HYPERLINK “https://arxiv.org/pdf/1712.03400.pdf”

[Kar18] Andrej Karpathy. (2018) Convolutional Neural Networks for Visual Recognition. [Online]. HYPERLINK “http://cs231n.github.io/convolutional-networks/”

[Kar16] Ujjwal Karn. (2016) An Intuitive Explanation of Convolutional Neural Networks. Kaggle Forum. [Online]. HYPERLINK “http://ens.ewi.tudelft.nl/Education/courses/et4351/intuitive_cnn.pdf”

[Sze16] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. (2016, August) Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. arXiv:1602.07261v2. [Online]. HYPERLINK "https://arxiv.org/pdf/1602.07261.pdf"

[Skr17] Ole-Johan Skrede. (2017, Mar.) color images, color spaces and color image processing. [Online]. HYPERLINK "https://www.uio.no/studier/emner/matnat/ifi/INF2310/v17/undervisningsmateriale/slides_inf2310_s17_week08.pdf"

[Wal17] Emil Wallner. (2017, OCTOBER) FloydHub Blog - Colorizing B&W Photos with Neural Networks. [Online]. HYPERLINK "https://blog.floydhub.com/colorizing-b-w-photos-with-neural-networks/"

[Wal14] Stefan van der Walt et al., "scikit-image: image processing in Python," *PeerJ*, June 2014.

[Aba16] Martín Abadi' et al., "TensorFlow: A System for Large-Scale Machine Learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16)*, Savannah, GA, USA, 2016, pp. 265-283.

[Gul17] Antonio Gulli and Sujit Pal, *Deep Learning with Keras*. Birmingham, UK: Packt Publishing, 2017.

[Dah16] Ryan Dahl. (2016, January) Tiny Clouds. [Online]. HYPERLINK "http://tinyclouds.org/colorize/"