

Convolutional Neural Networks for Image Colorization

Jaime Cernuda Garcia CWID: 20432547

Rodrigo Aranguren Carmona CWID: 20432501

Abstract

In the following document we will present the problem, implementation and results of the final project for CS 512 Computer Vision at the Illinois Institute of Technology. The project attempts to obtain the results of [1] by modelling, training and using a convolutional neural network capable of colorizing an image from an input in the form of a grayscale image.

Problem Statement

The aim of this project is to build a convolutional neural network capable of producing a plausible image in color, by replicating the model presented by [1].

While colorization has existed for a long time, it has usually been seen as an artistic method, where a professional tries to apply missing colors from legacy grayscale images. Usually they make use of the context of the image (image subjects, photograph era, etc.) to produce a plausible color image. This “plausible” is what makes this problem difficult to be solved in an unsupervised manner. However, with advancements in the deep learning area, and with convolutional neural networks being able to understand and recognize more from an image, it is becoming more frequent to find unsupervised networks trying to solve this exact problem.

In the training process, color image are transform into the Lab color space and the L channel is fed into the input of the convolutional network. Depending of the specific implementation, the output of the model will be either the ab channels or codified version of them, our paper implements and compares one option for both methods. By comparing the output of the model with the real ab channels of our image we can train the neural network to approach an increasing level of realism.

During the project we will be using Keras [2] with TensorFlow [3] as a backend. Other packages used for support will be reference during the implementation section.

As such, we propose a set of objectives for our project:

- Find, download and prepare the dataset for the training and testing of the model.
- Understand and implement the preprocessing of the images required for both models.
- Understand and implement the convolutional neural network model.
- Perform the training of the model under the dataset.
- Process the model outputs and inputs to generate RGB images.

Method and implementation

As we have previously explained, the idea is to generate the missing a and b channels from a grayscale (only L channel) image using a convolutional neural network. In this section we will address all parts that make up our problem solution.

The CIE Lab colorspace

The reasoning behind why colorization is a problem usually worked in this particular colorspace instead of others is twofold:

- Firstly, it is known that euclidean distances in this colorspace are related to human perception of colorization. This means that colors that we perceived as different by humans are apart in this colorspace.
- In second place, it is not restricted to visible color gamut.

The Lab colorspace is made up of three channels:

- Luminosity (L), which contains most of the details and what truly makes the image. In an image it usually takes values from 0 (pitch black) to 100 (full white).
- a is the green-red axis. It usually takes values from -128 to 127. A value in this channel signifies the weight of either green (negative values) or red (positive values).
- b, same as a but in the blue-yellow axis.



Image 1: Decomposing a color image into its L, a, b channels

We can see above a decomposition of a sample image in its respective Lab channels. While the luminosity L is an accurate representation of the grayscale image, the a and b channels (using their -128 to 127 range) actually only carry color information. Additionally, it can be seen how the a, b channels don't bring definition to the image, which will work in the CNN's favor, because enough features can be detected on the grayscale image.

With this known we can use a loss which can be used in the training process of a neural network:

$$L_2(\hat{Y}, Y) = 1/2 \sum_{h,w} \|Y_{h,w} - \hat{Y}_{h,w}\|_2^2$$

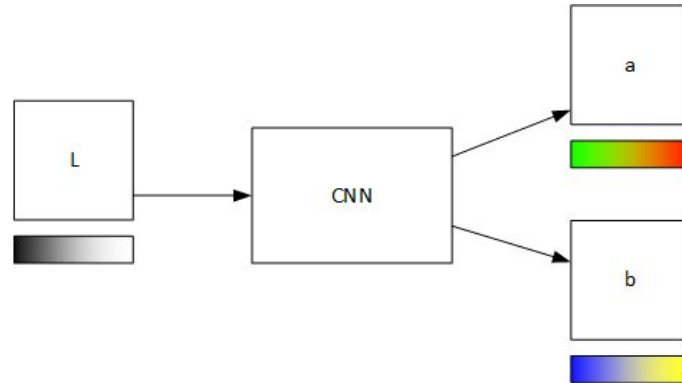


Image 2: A simple CNN that uses regression to generate a,b channels

Using this of loss makes sense for this colorspace, as distances have a significant meaning. Also, it's easy to implement and understand. However, It doesn't really address the ambiguity problem of colorization: we have to know that image colorization isn't to produce the real image colors, but actually producing a plausible colorization for a grayscale image. For example, if we build a model to colorize apples and train it with a sufficiently large image dataset, we should expect our colorized apples to have plausible colors. However, because an apple can take several colors (red, yellow, green, but never blue), with all of them are plausible the optimal solution to the euclidean loss will be the mean of the set. This can tend to be grayish and desaturated tones, or even implausible tones if the set is non-convex



Image 3: an apple can take on different colors, but it will always be an apple.

In order to overcome this problem, [1] proposes treating the problem as a multinomial classification instead of a regression. The idea is to quantize the ab space into 313 values that are in gamut and using a modified multinomial cross entropy loss:

$$L_{cl}(\hat{Y}, Y) = - \sum_{h,w} v(Z_{h,w}) \sum_q Z_{h,w,q} \log(\hat{Z}_{h,w,q})$$

Where:

- $\nu(\cdot)$ is a weighting term, used to rebalance the loss to boost rare colors.
- $Z(\cdot)$ is a function that encodes ground truth color into a soft-encoding scheme. This is the opposite of the function defined to map the predicted distribution to a point in the ab space.

However, even treating the problem as a multinomial classification, the distribution of ab values is highly dependant on the training set. On average, it will be seen that the model will favor low ab values resulting on colorized images with a sepia tone. To overcome this, [1] proposes a method to rebalance the obtained probability distribution, interpolating by adjusting the temperature T of the softmax distribution by hand in the evaluation phase.

Summing up, we can see that [1] does not truly achieve unsupervised training, but it gets rather close: the system is basically end-to-end trainable except for the single parameter T .

CNN architecture

As previously noted, in this project we implement two different architectures based on their complexity.

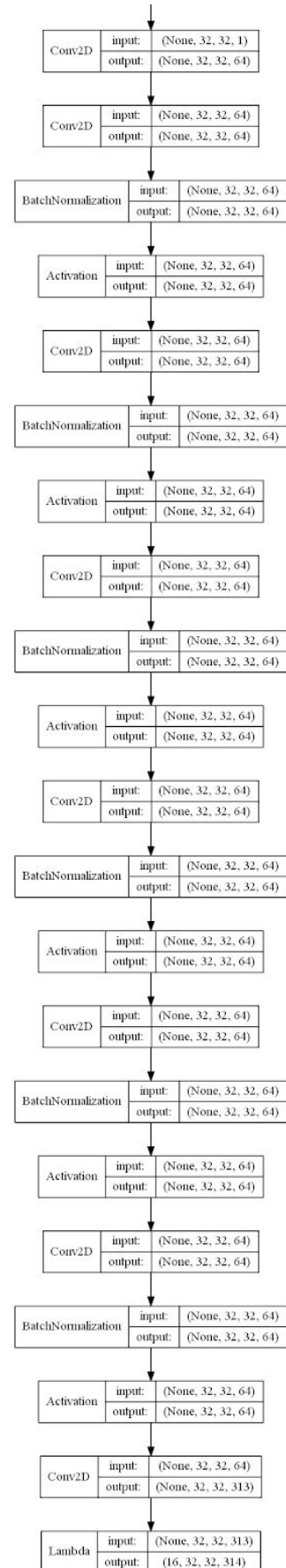
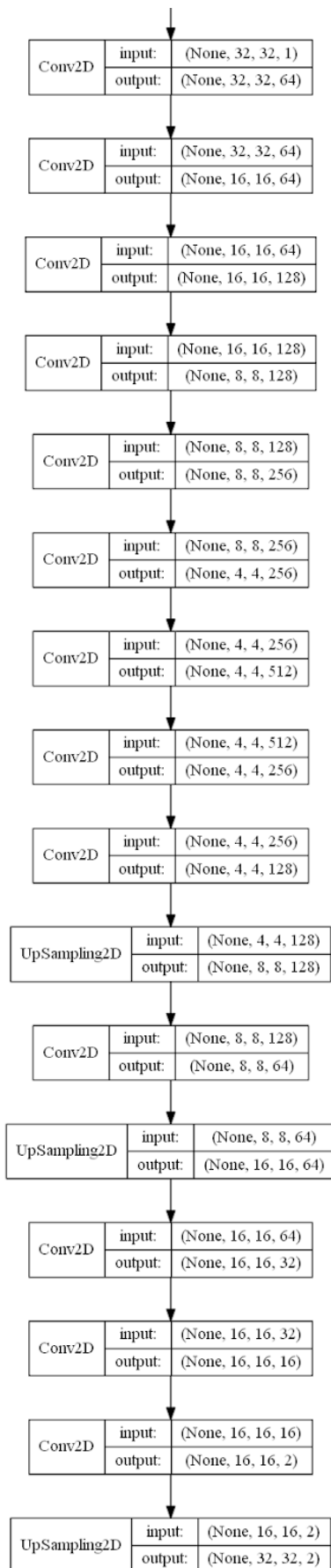
Model 1 (left): with 3 896 530 trainable parameters, it uses the L2 loss. It makes use of standard convolutional layers with ReLu activation and 3x3 kernel dimensions. No max pooling layers, as spatial downsampling is done by the convolutional layers only. Before the output we mirror the previous convolutions using upsampling to produce coherently dimensioned images.

This model is trained on square 32x32 images using variable batch sizes (note the first dimension of the tensors at each layer)

Model 2 (right): a Keras interpretation of [1]. Even though it has less trainable parameters (243 321), it is more complex. It treats the problem as a multinomial classification with class rebalancing, so it uses the multinomial cross entropy loss. It is made up of blocks of 2 or 3 convolutional layers each, which can be considered as a modification of the first layers of the widely used VGG architecture.

The final layer yields a final tensor containing probability estimations for each of the quantized nodes of the Lab colorspace. These will be concatenated with the original L channel to form a Lab image (which can be converted back to RGB trivially). Unfortunately, the implementation of this model in Keras requires to know the batch size before compiling, which makes the model impossible to test on a single image without loading a different model architecture.

Additionally, it includes an additional optimizer as recommended by [1] to address weight decay (from [5,7]).



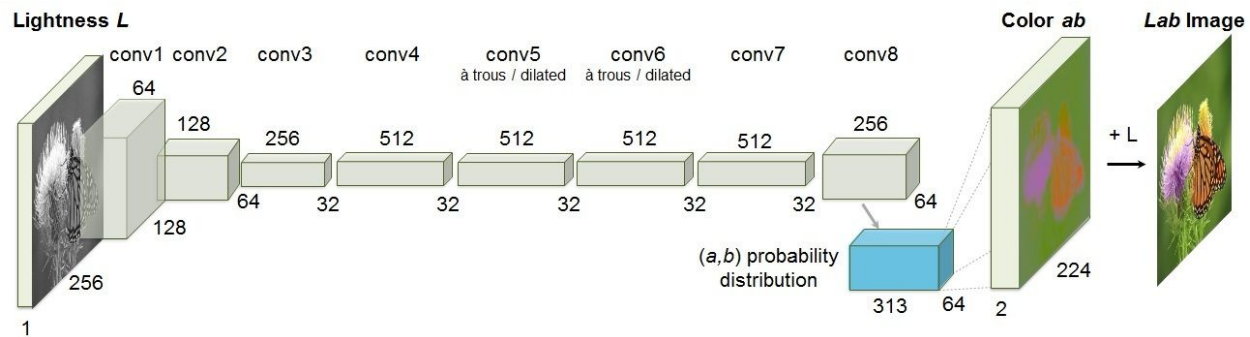


Image 4: Model 2 is a Keras interpretation of the architecture proposed by [1]

Datasets used

Most testing has been done on the CIFAR 10 [10] dataset of 50 000 small images. This is because we are able to import it easily on every working environment (including Google Colab, which has been our primary working environment), as it is one of the default datasets included in Keras.

A smaller, more manageable dataset was formed from the repository at [12]. It consists of a small dataset of faces. This was used because all images in the dataset presented similar colors, which would prevent the model 1 from shifting towards gray colors.

ColorNet, a colored dataset by [6] with varied “artistic” square images.

Additionally, we planned to use a subsample of the OpenImages dataset [11] because it features around 100 000 images of varied sizes, up to 1024 pixels on the largest dimension, but due to time and storage constraints, we weren’t able to use it fully.

Guidelines to use the program

There are 4 programs in the src folder:

- `Simpler_model.ipynb` and `ComplexModel.ipynb`: these two notebooks are used to train our models. While we don’t expect them to be run, in case they are they require to be connected to a Google Drive because they were developed in Google Colab. Once connected, in the first cell, it is important to change some of the paths to folders since is impossible for us to know how the tester folders are going to be organize.
- `Colorize_Simple.py` and `Colorize_Complex.py`: these two programs are used to evaluate our model. Running them with the ‘-h’ or ‘--help’ will provide a help menu.

Without arguments, they will run on the default parameters that can be overridden by the user following the instructions of the help menu.

Results and discussion

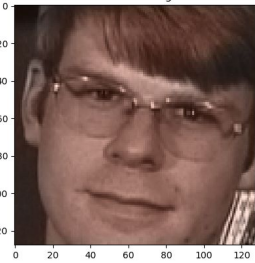
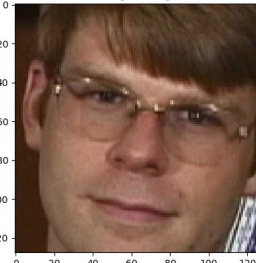
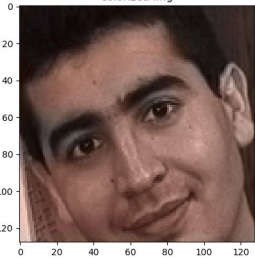
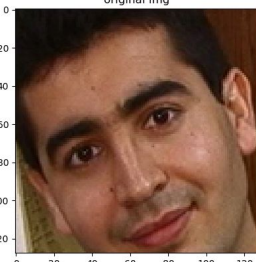
As previously discussed, we have implemented two models. In this section we will analyze how the models perform with the datasets, a visualization of the outputs from the models and the issues found with the results and the analysis of their possible causes.

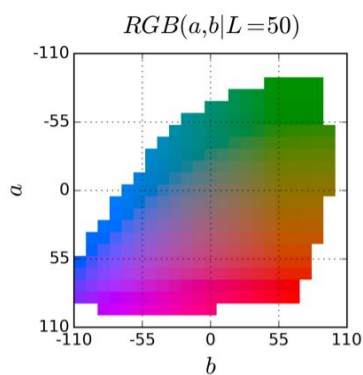
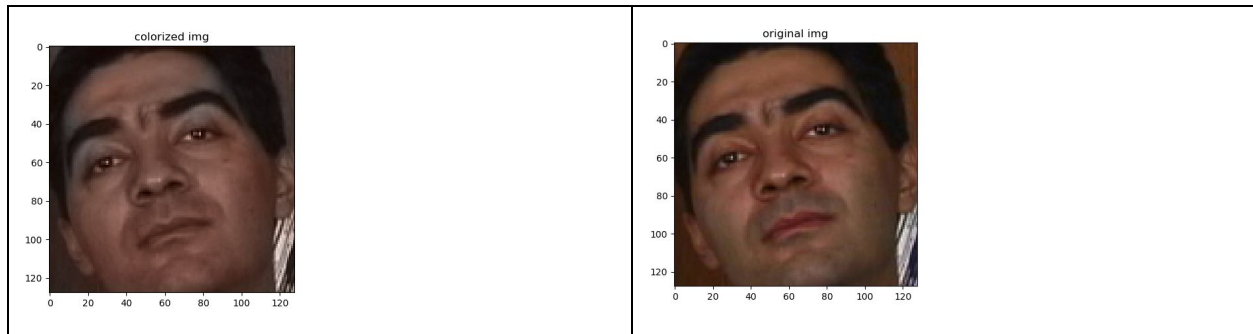
Model 1

The first model is implemented with a L2 loss function and outputs the full ab channels. It concatenates the output of the model with the input, the L channel, and the result is the colorized image.

This model has been trained using faces dataset by [12].

Some of the results of the model can be seen in the following table.

Deep Colorization	Ground Truth
 <p>A colorized image of a man with brown hair and glasses, looking slightly to the right. The image is displayed on a grid with x and y axes ranging from 0 to 120.</p>	 <p>An original image of a man with brown hair and glasses, looking slightly to the right. The image is displayed on a grid with x and y axes ranging from 0 to 120.</p>
 <p>A colorized image of a man with dark hair, looking directly at the camera. The image is displayed on a grid with x and y axes ranging from 0 to 120.</p>	 <p>An original image of a man with dark hair, looking directly at the camera. The image is displayed on a grid with x and y axes ranging from 0 to 120.</p>



As we can see in all of the images, the model has trained itself to generate images that tend to be very grayish and desaturated.

This set of colors can be seen on the center of the image on the left and correspond to the center of the Lab color space.

The reason why this set of colors is so prevalent in the output of the model is the tendency of the Euclidean loss function used in this model to choose the mean as the optimal solution.

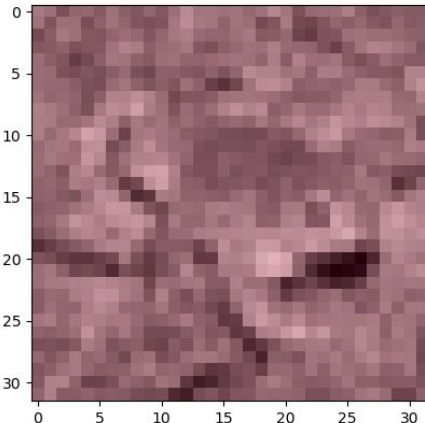
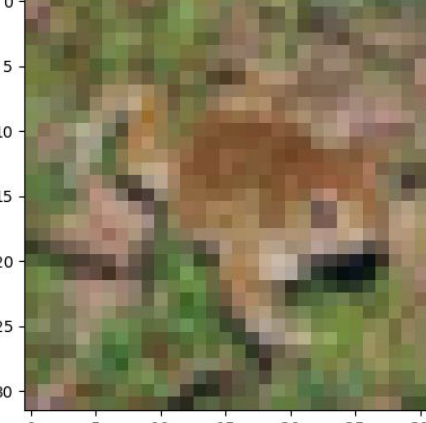
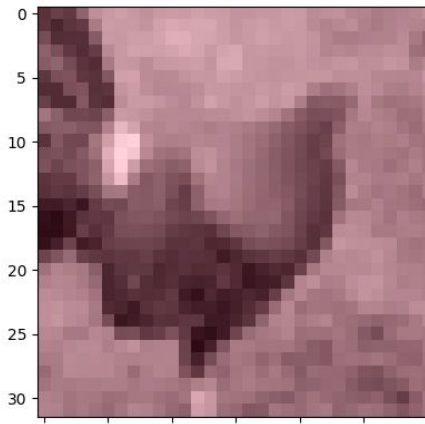
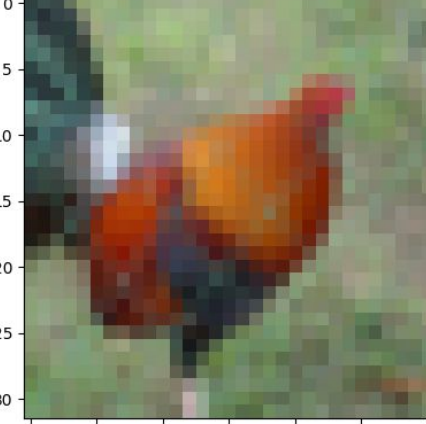
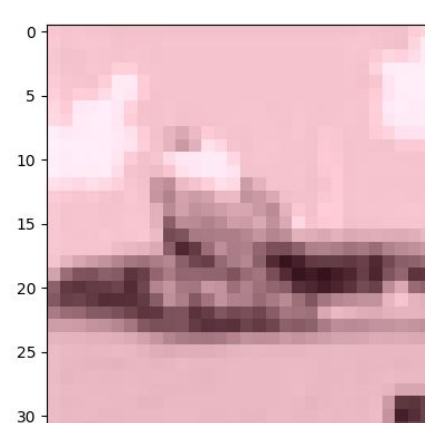
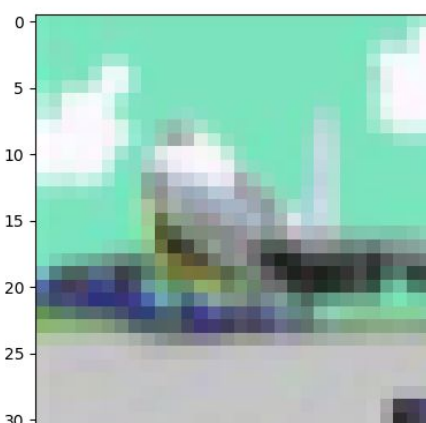
On the other hand, the pale and desaturated colors are very prevalent in faces due to the similarity to the tone of human skin. As such, we can see that the outputs of the model when trained on images containing mostly faces are relatively accurate to the ground truth (and thus, plausible). It is important to note that this is a consequence of leveraging the inherent issues of the model and not a consequence of the model obtaining better results. It can be argued that this dataset fits the model and not the opposite.

Model 2

The second model is implemented with a multinomial cross entropy loss function and works as a classification model outputting the probabilities of the pixel belonging to one of the 313 pairs of the quantized ab color space. The output is the concatenation of the L channel and the weighted sum of the output of the network as the ab channels.

This model has been trained using the CIFAR-10 [4] dataset.

Some of the results of the model can be seen in the following table.

"Colorized" CIFAR-10	Ground Truth
 <p>A 32x32 pixel image of a dog, heavily desaturated to shades of purple and pink. The image is displayed on a coordinate grid with axes ranging from 0 to 30.</p>	 <p>A 32x32 pixel image of a dog, showing natural colors. The image is displayed on a coordinate grid with axes ranging from 0 to 30.</p>
 <p>A 32x32 pixel image of a dog, heavily desaturated to shades of purple and pink. The image is displayed on a coordinate grid with axes ranging from 0 to 30.</p>	 <p>A 32x32 pixel image of a dog, showing natural colors. The image is displayed on a coordinate grid with axes ranging from 0 to 30.</p>
 <p>A 32x32 pixel image of a dog, heavily desaturated to shades of purple and pink. The image is displayed on a coordinate grid with axes ranging from 0 to 30.</p>	 <p>A 32x32 pixel image of a dog, showing natural colors. The image is displayed on a coordinate grid with axes ranging from 0 to 30.</p>

As we can see the results are not what we expected. While in the previous model we knew that the model will be inclined to pale colors, our expectations were that this model could fix that.

When analyzing the issues, it appears that the model has trained itself to output almost the same probability to every single pair of the ab color space. Being so conservative the model will still tend towards the colors on the center, sometimes with a very uniform colors across the whole image.

While we performed training with other datasets, like the ColorNet dataset by [6] with 256x256 images and with the face dataset used in the first model the results have persisted.

We propose some of the possible issues that could have derived this:

- The simpler option, the network is too deep, we don't have the dataset or the computing power to train the network to its full success. While it could be an option, especially considering that [1] trains the model for 450k iterations over a dataset of over a million images, we would expect to see random colors not a such a deviation into outputting always the same color.
- Dying ReLU, there is a chance that since we have increased the learning rate from the value used in [1], to compensate for our lack of images and computing power, that our ReLU activation could have died, "a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again" [8]. If this has happened it could explain the results since the death of most ReLU units could cause the input to the softmax layer to be mostly zero.

While we have attempted solutions to the problem, like varying the learning rate to reduce the effect of dying ReLU, changing optimizers, training with different model for longer periods or even consider using the ELU [9] activation units, a lack of time and computational resources have prevented us from moving forward with some of the solutions.

Bibliography

- [1] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. ECCV, 2016. Available at <https://arxiv.org/abs/1603.08511>. Accessed 19 Nov. 2018.
- [2] "Keras Documentation." <https://keras.io/>. Accessed 19 Nov. 2018.
- [3] "TensorFlow." <https://www.tensorflow.org/>. Accessed 19 Nov. 2018.
- [4] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Available at <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>. Accessed November 19, 2018.
- [5] I. Loshchilov, F. Hutter. Fixing Weight Decay Regularization in Adam. CoRR 2017. Available at <https://arxiv.org/abs/1711.05101>. Accessed 19 Nov. 2018.
- [6] E. Wallner. ColorNet dataset. Available at <https://www.floydhub.com/emilwallner/datasets/colornet>. Accessed Nov 19, 2018.
- [7] G. Lambard. Fixing Weight Decay Regularization in Adam - For Keras 😊. Available at https://github.com/GLambard/AdamW_Keras. Accessed November 19, 2018.
- [8] A. Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. Available at <http://cs231n.github.io/neural-networks-1/#actfun>. Accessed November 19, 2018.
- [9] M. Heusel et al. Fast and Accurate CNN Learning on ImageNet. Available at http://image-net.org/challenges/posters/JKU_EN_RGB_Schwarz_poster.pdf. Accessed November 19, 2018
- [10] A. Krizhevsky. CIFAR-10: [Learning Multiple Layers of Features from Tiny Images](#), Accessed November 19, 2018.
- [11] OpenImages dataset: <https://storage.googleapis.com/openimages/web/index.html> Accessed November 19, 2018
- [12] M Chimbadian. Deep Colorization, A basic convolutional neural network (CNN) approach to image colorization. Available at <https://github.com/2014mchidamb/DeepColorization>. Accessed 19 November 2018.