CSI 4106 – Introduction to Artificial Intelligence

Winter 2022

École de Génie Électrique et Science Informatique University of Ottawa

Project Report: Image Colorization

Professor: Dr. Prasadith Kirinde Gamaarachchige

Student: Paquette, Simon - 300044038

Submission Date: 2022/04/14

Abstract

Identified problem

Developing an AI model about colorization of black and white images seems an interesting project where it could be applied to our old photos to bring back the original color. Automated image colorization is a topic gaining a lot of notice recently in the exploration of computer vision and deep learning.

The dataset will consist of 7129 landscape images (150x150) in black and white and their corresponding colored images [ref.1]. Here is a summary: “This dataset consists of street, buildings, mountains, glaciers, trees, etc. and their corresponding grayscale image in two different folders. The main objective of creating this dataset is to create autoencoder network that can colorize grayscale landscape images” [ref.1]. However, because the way we use the images and convert them to the LAB color space, it is easy to create our own grayscale images with the L channel. I could easily add a new dataset of images to extend the project with people, animals, or objects.

Proposed solution

The input of the model will be a grayscale image (in this project, it will be a landscape image) and the output will be the same image but with an LAB value for each pixel. During the last few years, many different solutions have been proposed to colorize images by using deep learning.

Colorful Image Colorization paper [ref.3] approached the problem as a classification task and they also considered the uncertainty of this problem (e.x. a car in the image can take on many different and valid colors and we cannot be sure about any color for it)

However, another paper approached the problem as a regression task (with different technic like GAN) [ref.6].

There are pros and cons to each approach. The problem with regression task is a more conservative result, so more grayish images. Personally, I dive into the regression approach. Approaching the problem as a classification of ab color in 313 different bins is a bit more complicated. We need to apply a specialized loss function and download some prebuilt model of detection to classify each pixel to a specific bin using caffemodel or prototxt.

Overall result

I tested 6 different algorithms in my jupyter notebook, and they did not really perform as expected. For most of the algorithm, the image colorization process was able to colorize in blue and green, corresponding in majority of the sky and the trees respectively. It ignores the red and yellow colors. Two were based on already created algorithm that I modify to be applied with OpenCV processing. A classic cnn model with a fusion step gave the best result (for green and blue only) and the algorithm of Richard Zhang [ref.3] that I did not have the opportunity to make it work well because the majority of the image were still grayscale. I also built a simple model composed on only convolutional layers for a basic reference, it gives comparable results despite its simplicity. When researching in some papers, I decided to apply another one composed of convolutional layers with strides, and up sampling the image in the encoder part. The results were very similar to the classic cnn model. Finally, after some learning in class, I decided to test a model of split learning where the decoder is a shared layer, and the encoder are specific to each color channel. The result was better than the basic model. I also tried a model composed of large kernel for convolutional step with a L1 regularizer on them. The result was only grayscale image for this one.

Table des matières

Introduction

Problem

The project of coloring black and white images to produce an output with possible real colors from grayscale is a complex subject. With tools like OpenCV and TensorFlow, this project helped me gained experience in the idea of LAB color space, using OpenCV to apply transformation on images and applying an AI model of neural networks to train from a dataset. Many concepts will be explained more in detail later in the section "Background and related work".

I used 2 different datasets, principally of landscapes images. The split train/val/test that I use is 70/15/15. First a dataset consisting of 7129 landscape images (street, buildings, mountains, glaciers, trees, etc.) of size 150x150 [ref.1]. Secondly, a dataset consisting of 8800 images of travel and adventures of 256x256 pixels [ref.2]. The first dataset is easy to use but have a lot of blue/green background colors and represent well my type of images I wanted, a landscape. The second dataset will be used to perform training on bigger images with a dataset with more variety all related to travel: landscape, peoples, city, cultural, object, animals.

Reason

Developing an AI model about colorization of black and white images seems an interesting project where it could be applied to our old photos to bring back the original color. Automated image colorization is a topic gaining a lot of notice recently in the exploration of computer vision and deep learning.

Background information

It will be explaining more later, here a brief introduction to the cnn parameters. The model is generally split in 2 steps, an encoder, and a decoder. The encoder is, for the most application, conv2d layers with LeakyRelu or Relu activation function, sometimes with dilation, followed by batch normalization for some algorithms. The input is filtered either by stride=2 or Maxpooling. The decoder is always upsampling2d with classic conv2d layers. The loss functions used is either MAE (L1) or MSE(L2) or some other homemade specialized loss function. The Optimizer used is either ADAM or RMSPROP.

See figure 1 for an example of layers to use, the probability distribution step represents the classification task, where I did not try to implement because I favorized the regression task as the figure 2.

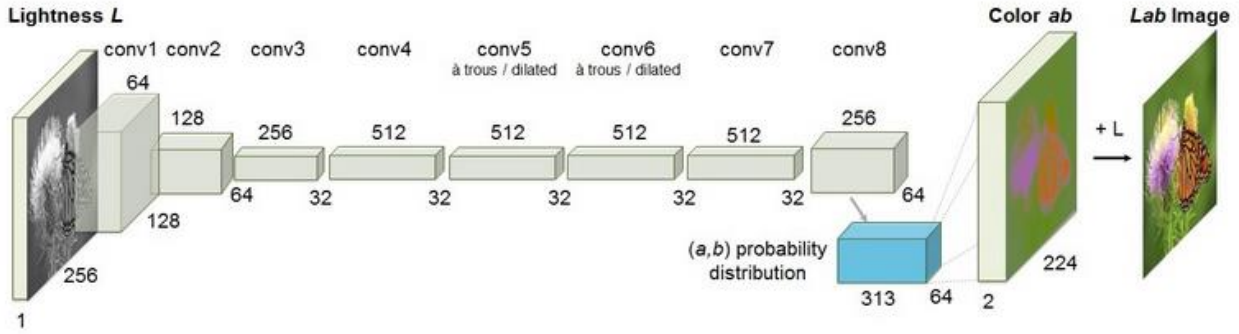


Fig1. Proposed model for classification task [ref.3]

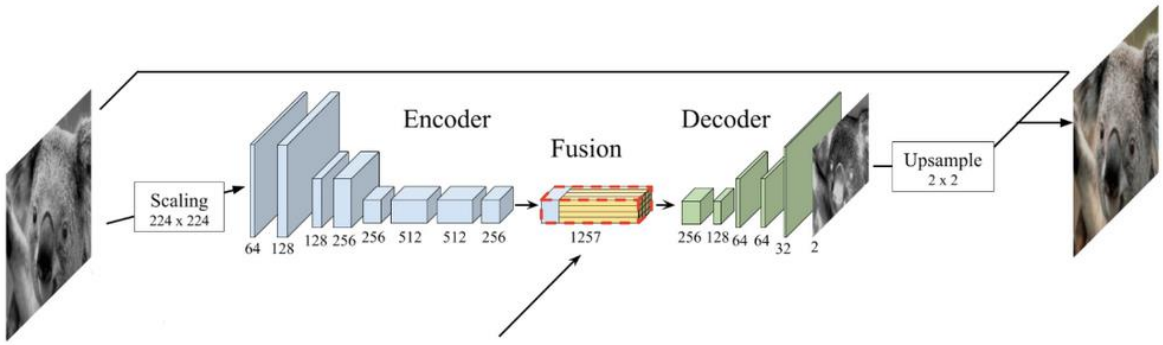


Fig2. Proposed model for regression task [ref.9]

Contribution

I did the project individually. I have completed the project proposal, project and concept research, dataset download, coding image processing, implementing models and compiling results.

Paper structure

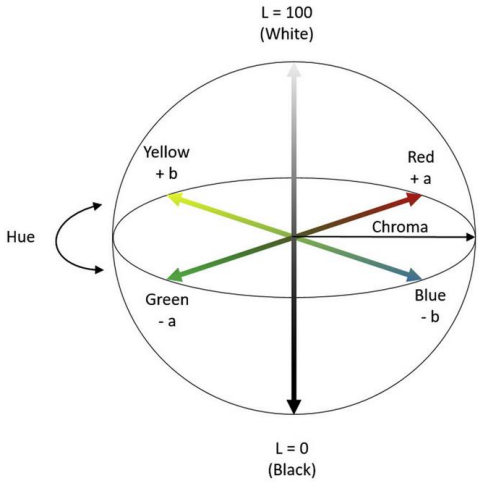
The remaining structure of the paper will cover the overall task I did to develop this project. It will go through the background and related work already done in this field of artificial intelligence and explaining different concepts and approach that may work or may not. Also, the proposed solution about image processing, model building, training, evaluating, testing, and hyper parameters evaluated. It will cover the results of my project followed by the discussion of the approach and a brief conclusion about the research of image colorization.

Background and related work

Concept

LAB color Space

In Lab color space, we have three channels (see figure 3). The L channel encodes the lightness of each pixel and provide a visualization as a black and white image. The A channel encodes how much green-red and B channel is for blue-yellow (see figure 4). This is the best color space for this project because to train a model for colorization, we need to give it a grayscale image and we want to output a colored image. When using Lab, we can give the L channel to the model and want it to predict the other two channels. After the prediction, we concatenate all the channels, and we get our colorful image.



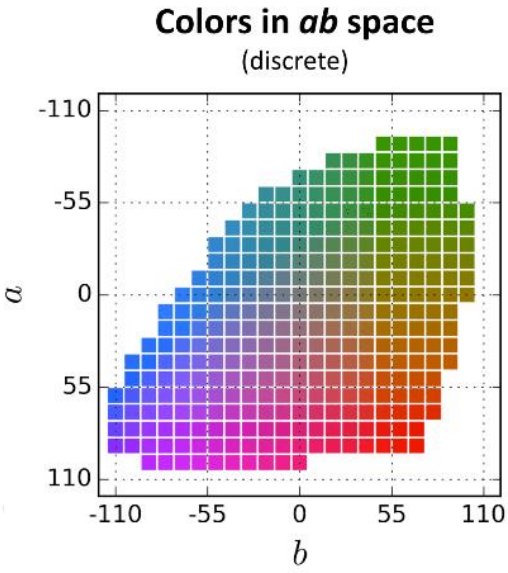


Fig3. LAB color space [ref.9]

Fig.4 AB color space value bin [ref.8]

NOTE : OpenCV rescale all those AB values in the range 0,255

CNN

Convolutional neural networks are composed of multiple hidden layers made of artificial neurons. They are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value. Each layer generates several activation functions that are passed on to the next layer. The first layer usually extracts basic features such as horizontal or diagonal edges. This output is passed on to the next layer which detects more complex. The bigger the network is, more it can identify complex features. [ref.15]

Research done and approach

The input of the model will be a grayscale image (in this project, it will be a landscape image) and the output will be the same image but with an LAB value for each pixel. During the last few years, many different solutions have been proposed to colorize images by using deep learning.

Colorful Image Colorization paper [ref.3] approached the problem as a classification task and they also considered the uncertainty of this problem (e.x. a car in the image can take on many different and valid colors and we cannot be sure about any color for it)

However, another paper approached the problem as a regression task (with different technic like GAN) [ref.6].

There are pros and cons to each approach. The problem with regression task is a more conservative result, so more grayish images. Personally, I dive into the regression approach. Approaching the problem as a classification of ab color in 313 different bins is a bit more complicated. We need to apply a specialized loss function and download some prebuilt model of detection to classify each pixel to a specific bin using caffemodel or prototxt.

The model is generally split in 2 steps, an encoder, and a decoder. The encoder is, for the most application, conv2d layers with LeakyRelu or Relu activation function, sometimes with dilation, followed by batch normalization for some algorithms. The input is filtered either by stride=2 or Maxpooling. The decoder is always upsampling2d with classic conv2d layers. The loss functions used is either MAE (L1) or MSE(L2) or some other homemade specialized loss function. The Optimizer used is either ADAM or RMSPROP.

Proposed solution

Image processing

First, a brief resume of this task, followed by the overall description.

We can resume the images preprocessing steps as:

1. With OpenCV, read the image as BGR color.
2. Resize to a default size 128x128 (or 256x256)
3. Make sure the image opened is uint8
4. Convert BGR to LAB
5. Set each value to float32
6. Normalize the pixel by dividing by 255
7. Split each channel to get L (lightness), A (green-red), and B (blue-yellow)
8. Return L channel for features and merge AB channel for labels

We can resume the reconstruction of the images as:

1. Merge the original L channel (grayscale) with the new predicted AB channel
2. Clip each pixel value to make sure they are in the range [0,1]
3. Multiply each pixel by 255
4. Set each value to uint8
5. Convert LAB to BGR to save the image or LAB to RGB to show the image

The python library used in this project is OpenCV, a tool well developed in image processing and low-level image manipulation. OpenCV work with images in the BGR format instead of RGB, meaning that we need to take some precaution to display the true color. For example, the library matplotlib used to display the image works with RGB pixels. For model training, we need to give as input a lot of images of the same size, so we resize them, convert them to Lab color, split the channels and normalize them to easily do the training steps. When saving or showing the new colored image from the model's prediction, we need to concatenate the original L channel with the new ab channel and do the reverse process of the preprocessing step.

Model building

To create a colorization model, we need to define the input and output first. So, the input is a tensor of size (height, width, 1) meaning we pass a grayscale image (L channel) and the output is a tensor of size (height, width, 2) indicating that we received an image with two channels, ab.

I built 6 different models based on already created algorithm or from some of my understanding of the task. I will explain them in order of my personal testing, without concern about the performance. The results will be explained later.

- **Classic**: based on a Kaggle notebook [ref.5], this was my reference for a possible good algorithm of encoder-fusion-decoder. The encoder was a sequence of 2D convolution layers with LeakyRelu activation, batch normalization and maximum pooling. The fusion step was a concatenation between the original layer and a convolution layer. The decoder was a sequence of up sampling layer, convolution layers with LeakyRelu activation and some batch normalization. The final layer used a tanh activation.
- **Basic**: The simplest deep learning model, consisting of only convolution layers with Relu activation function, that I used to establish a baseline.
- **Conv Sampling**: An algorithm based on [ref.7] and [ref.9]. The decoder is made of convolution layers with Relu activation and some with a stride=2 to reduce the size of parameters. The encoder is a sequence of up sampling layers and convolution layers with Relu activation. The final layer used a tanh activation, but I adapted it with a relu activation function.
- **Large Kernel Regularizer**: An idea that I wanted to try is to use convolution layers with a bigger kernel size (11x11 instead of 3x3) to increase the reach of layers in determining large features. Also, in the reading of some algorithms like in the GAN loss function from [ref.6], an additional regularizer in the convolution layers could be helpful, so some layers have a L1 regularizer. The decoder consists of conv2d with bigger kernel, a stride=2 and relu with batch normalization. The encoder is up sampling with conv2d and batch normalization. The final layer used a sigmoid activation.
- **RichardZhang**: I implemented a modified version of this famous algorithm for colorization in pytorch [ref.4]. It is a sequence of conv2d with relu, stride and dilatation rate, followed by layer normalization. The encoder starts with a conv2d transpose, conv2d and end with a bilinear up sampling. The final layer is generally

built with softmax activation for classification task, but because I worked with a regression task, I modified to a sigmoid. I also reduce the model size because it was taking too much time to run, or I was running out of gpu memory.

- **Split Learning**: An idea from the MTL framework, the shared decoder consists of conv2d with stride, relu and batch normalization. The decoder is split for channel a and b, made of conv2d and up sampling.

Model training

I applied different callbacks during the dataset training. An early stop of patience 5 epochs to restore the best weight if the loss did not reduce. A model checkpoint to save the weights in progress of the model if it fails during training. A TensorBoard to be able to compare the evolution of the loss and metrics through each epoch. A function to reduce learning rate when a plateau occurs after 2 epochs.

I defined a pipeline to do model training, starting with dataset loading, model building, model compilation and finally model fitting. The hyperparameters of the compilation and fitting will be presented later in the section "Hyperparameters".

Model evaluating and testing

With image colorization, evaluating the result of the model is complicated because the result can be a bit subjective. How can we define the possible colors and the representation of how much similar is two colors? With preexisting algorithm, the loss functions used is either MAE (L1) or MSE(L2) or some other homemade specialized loss function. The Optimizer used is either ADAM or RMSPROP.

Hyperparameters

Brief description of different parameters and their impact:

- BATCH SIZE: I did not have the opportunity to try different values, because I have 4gb of gpu ram and my maximum size before running out of memory is 8.
- EPOCH: Generally, between 20 and 25 epochs. With the early stop callback, it stops at the best number of epochs before overfitting
- LOSS: I tried two techniques, MSE vs MAE. It is hard to say which is best. MAE sometime create false color but produce more vibrant colors. MSE is stable but more grayish image (unsaturated). I have tested more with MSE at the beginning, but at the end more with MAE.
- OPTIMIZERS: I tried two versions, ADAM vs RMSPROP. ADAM is able to detect more specific details, but the two optimizers are similar in speed and performance. We may need to test both for different models. ADAM loss may have minor worst result but is more stable for each algorithm. I also tried SGD optimizer; it gives really bad result that we don't need to talk about it.
- LEARNING RATE: With the callback of lr_plateau, this hyperparameter has not a big impact. I keep the default starting value of 0.01 with a minimum of 1e-7. For starting value, tried 0.1 (don't learn much) and 0.001 (time consuming). For the minimum, I tried 1e-8, but it was time consuming.

- Number of hidden layers and units: I tested different alternative (explication in the model description)

Results

On my 6 different models, the ranking of best results is: classic, conv_sampling, split learning, basic, large kernel regularizer, and RichardZhang. Those algorithms, defined in my jupyter notebook, did not really perform as expected. For most of the algorithm, the image colorization process was able to colorize in blue and green, corresponding in majority of the sky and the trees respectively. It ignores the red and yellow colors.

First, I will mention problems that I got from my testing. For all the model that I built, the output images favorized the color blue and green, probably because my landscape dataset has more of those colors (unbalanced data). So, I decided to test on my second dataset with more variety. However, the outputs were bad because they were leaning way more to a grayish image. I decided to keep training on the first dataset. I also run out of gpu memory for my models, so I had to set the batch size to 8 or reduce the number of layers and units.

The classic cnn model with a fusion step gives the best result. I did 4 tests (adam+mse, adam+mae, rmsprop+mse, rmsprop+mae). The best images were with the adam optimizer and a mse loss (see figure 5). Like all the other images, there is a sur presentation of blue and green and you can refer to figure 6 to see the difference between adam and rmsprop. Because the loss values don't represent well the quality of the image, we can see that conv_sampling as a lower loss than this algorithm, but when looking the images, it was the best one.

loss E-4 (train/val/test)	adam_mse	adam_mae	rmsprop_mse	rmsprop_mae
basic	23/22/22	312/308/303	22/22/21	306/303/298
classic	21/20/19	301/290/287	22/20/20	303/290/286
conv_sampling	19/19/19	252/292/288	19/19/19	363/359/349

Fig.5 Result of the 3 first algorithms tested

The convolution layers and up sampling algorithm give similar results to the classic cnn. I did 4 tests (adam+mse, adam+mae, rmsprop+mse, rmsprop+mae). The best images were with the adam optimizer and a mae loss (see figure 5). The mae loss is more unstable than the mse and also create more false color, but it gives more vibrant color that we may found in a landscape image. You can refer to figure 6 to see the difference between mae and mse.

The split learning model gives good result (see figure 6). When comparing multiple images, this algorithm is similar to the classic cnn and the conv_sampling algorithm. This model used rmsprop and mae. The mae loss for training gave a value of 0.0304 and 0.0298 for validation. This model seems to have great potential to separate concretely the a channel from the b channel and do a good prediction on both.

The basic model gives comparable results despite its simplicity. Less feature detection is done, but it can add the blue and green color as correctly as the other algorithms. The rmsprop and

mse loss work best with this model as you can see on the figure 5 and 6. It is fast and simple, but I don't think it could give better results with more training.

The model from Richard Zhang did not work at all on my dataset and image preprocessing techniques. The output was simply a grayscale image and blue color in the sky of some images (see figure 6). I did a lot of modifications on this script because it used a classification task instead of my idea of regression task and did not work well with my limitation on gpu memory.

The large kernel regularizer did not work. The output is a grayscale image. It gives a loss 0.0343 with adam optimizer and mae loss.

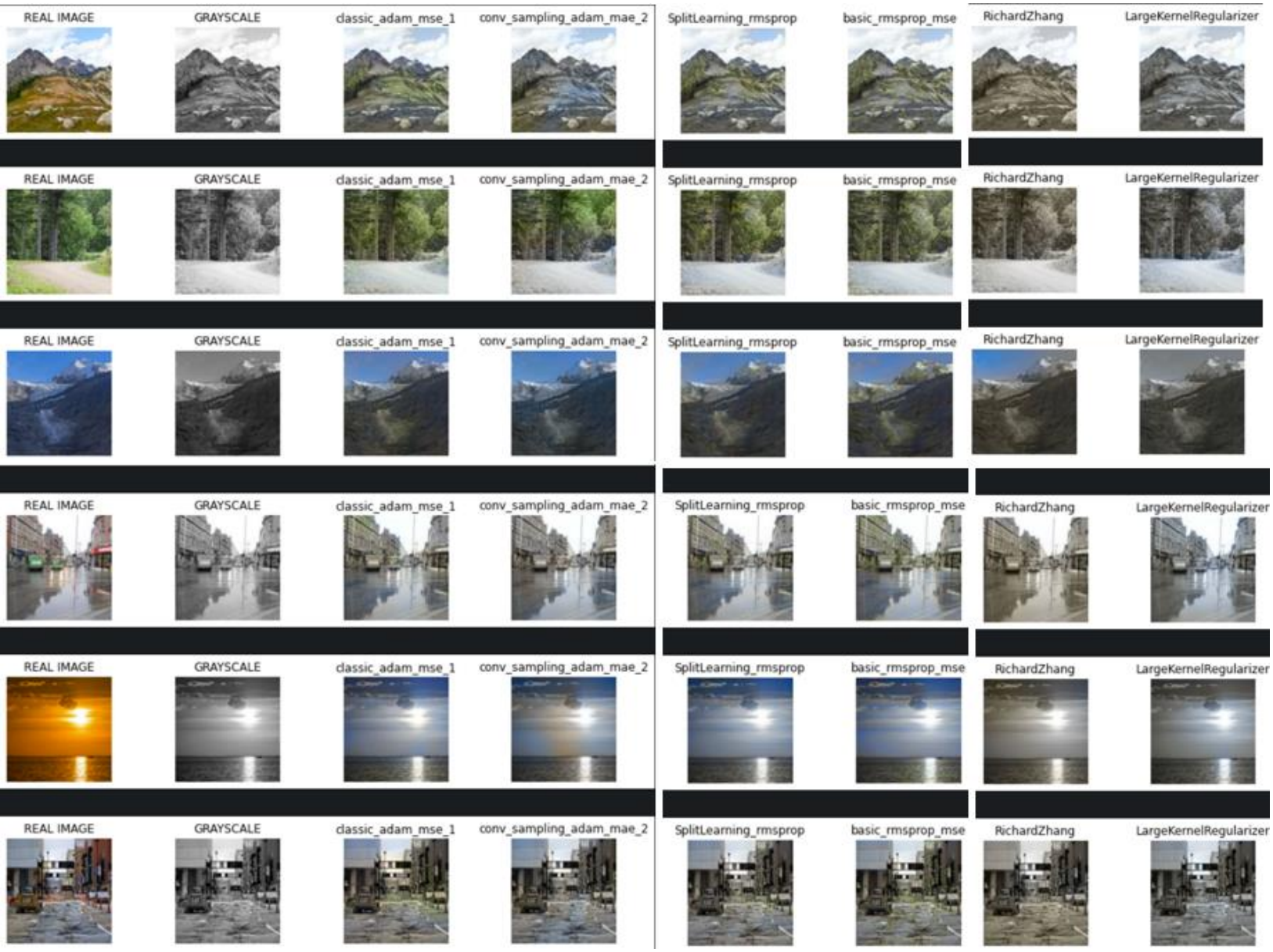


Fig.6 Result of the models, grayscale and ground truth included

Discussion

For all the model that I built, the output images favorized the color blue and green, probably because my landscape dataset has more of those colors (unbalanced data). So, I decided to test on my second dataset with more variety. However, the outputs were bad because they were leaning way more to a grayish image. I decided to keep training on the first dataset. When comparing my result with research papers, I could say that a classification approach instead of a regression one could be helpful in this case. The papers assign 313 bins each corresponding to a specific ab value. They also proposed alternative to put more weight on rarer color, for example red and yellow in my case. This method will be able to learn with bigger dataset compared to my method I implemented, because as I explained when training with my second dataset, all the images produced were more grayish. This can be explained by the fact that calculating the mse on a regression task will always tend for a more conservative value.

I decided to concentrate on images of landscape for this project, meaning that my dataset had an imbalanced of colors. For most of the algorithm, the image colorization process was able to colorize in blue and green, corresponding in majority of the sky and the trees respectively. To get better result, I could have used an image data generator to do image augmentation, and also increase the minority class (red and yellow colors). A good example in my dataset about an oversample of green is a tree without leaves (ex. In winters). My algorithm of convolutional layers and up sampling added green color all around the tree, despite its real color of grey/brown.

I tested 6 different algorithms for image colorization, and they did not really perform as expected. Two were based on already created algorithm that I modify to be applied with OpenCV processing. A classic cnn model with a fusion step gave the best result (for green and blue only) and the algorithm of Richard Zhang [ref.3] that I did not have the opportunity to make it work well because the majority of the image were still grayscale. When reading papers, this approach seems the best and well developed. So, later on, ill need to adapt it more to my dataset. I also built a simple model composed on only convolutional layers for a basic reference, it gives comparable results despite its simplicity. When researching in some papers, I decided to apply another one composed of convolutional layers with strides, and up sampling the image in the encoder part. The results were very similar to the classic cnn model. Finally, after some learning in class, I decided to test a model of split learning where the decoder is a shared layer, and the encoder are specific to each color channel. The result was better than the basic model. I also tried a model composed of large kernel for convolutional step with a L1 regularizer on them. The result was only grayscale image for this one.

Conclusion

The project of coloring black and white images to produce an output with possible real colors from grayscale is a complex subject. With tools like OpenCV and TensorFlow, this project helped me gained experience in the idea of LAB color space, using OpenCV to apply transformation on images and applying an AI model of neural networks to train from a dataset.

Colorful Image Colorization paper [ref.3] approached the problem as a classification task and they also considered the uncertainty of this problem (e.x. a car in the image can take on many different and valid colors and we cannot be sure about any color for it). However, another paper approached the problem as a regression task (with different technic like GAN) [ref.6]. There are pros and cons to each approach. The problem with regression task is a more conservative result, so more grayish images. Personally, I dive into the regression approach. Approaching the problem as a classification of ab color in 313 different bins is a bit more complicated. We need to apply a specialized loss function and download some prebuilt model of detection to classify each pixel to a specific bin using caffemodel or prototxt.

I built 6 different models based on already created algorithm or from some of my understanding of the task. The output images favorized the color blue and green. So, the results were not as expected. In the future, I would need to favorized the option of image colorization as a classification problem. Binning a list of colors will produce better vibrant color representation, because the regression problem will tend to an average of colors, meaning more pastel colors or a more grayish representation of the scenes.

References

1. Mamba, B. (2021, February 6). *Landscape Color and grayscale images*. Kaggle. Retrieved February 21, 2022, from https://www.kaggle.com/theblackmamba31/landscape-image-colorization
2. Dutta, D. (2018, September 24). *Google scraped image dataset*. Kaggle. Retrieved February 21, 2022, from https://www.kaggle.com/datasets/duttadebadri/image-classification
3. Richzhang. *Colorful image colorization*. Colorful Image Colorization. (n.d.). Retrieved February 21, 2022, from https://richzhang.github.io/colorization/
4. Richzhang. (n.d.). *Richzhang/colorization: Automatic colorization using deep neural networks. "Colorful image colorization." in ECCV, 2016.* GitHub. Retrieved February 21, 2022, from https://github.com/richzhang/colorization
5. basu369victor. (2020, March 28). *Image colorization basic implementation with CNN*. Kaggle. Retrieved February 21, 2022, from https://www.kaggle.com/basu369victor/image-colorization-basic-implementation-with-cnn
6. Shariatnia, M. (2020, November 18). *Colorizing Black & white images with U-Net and conditional Gan-A tutorial*. Medium. Retrieved February 23, 2022, from https://towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8
7. Wallner, E. (2017, October 29). *How to colorize black & white photos with just 100 lines of Neural Network Code*. Medium. Retrieved February 23, 2022, from https://emilwallner.medium.com/colorize-b-w-photos-with-a-100-line-neural-network-53d9b4449f8d
8. Nayak, S. (2018, July 29). *Convolutional neural network, CNN based image colorization using opencv*. LearnOpenCV. Retrieved March 2, 2022, from https://learnopencv.com/convolutional-neural-network-based-image-colorization-using-opencv/
9. Nurdialit, D. G. (2021, April 28). *Image colorization*. Image Colorization. Retrieved March 3, 2022, from https://algotech.netlify.app/blog/image-colorization/
10. Rosebrock, A. (2019, February 25). *Black and white image colorization with opencv and deep learning*. PyImageSearch. Retrieved March 4, 2022, from https://pyimagesearch.com/2019/02/25/black-and-white-image-colorization-with-opencv-and-deep-learning/
11. Charpiat, G. *Machine learning methods for automatic image colorization*. (n.d.). Retrieved March 8, 2022, from https://www.lri.fr/~gcharpia/colorization_chapter.pdf
12. Lukemelas. Image colorization with convolutional neural networks. (2018, May 15). Retrieved March 29, 2022, from https://lukemelas.github.io/image-colorization.html
13. Hwang, J., & Zhou, Y. *Image colorization with deep convolutional neural networks*. (n.d.). Retrieved April 4, 2022, from http://cs231n.stanford.edu/reports/2016/pdfs/219_Report.pdf
14. Lewinson, E. (2020, April 17). *Image colorization using convolutional autoencoders*. Medium. Retrieved April 4, 2022, from https://towardsdatascience.com/image-colorization-using-convolutional-autoencoders-fdabc1cb1dbe
15. *CNN for deep learning: Convolutional Neural Networks*. Analytics Vidhya. (2021, July 23). Retrieved April 13, 2022, from https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

Annex

Appendix A : Github Link

https://github.com/SimonPaquette/image_colorization

Appendix B : Jupyter Notebook

./image_colorization.ipynb

Appendix C : Python File

./image_colorization.py

Appendix D : Project Proposal

./CSI4106_ProjectProposal_SimonPaquette_300044038.docx

Appendix E : Models

./models/