Performance Evaluation of the Two Color Detection Algorithms

COMP 482 - Computer Vision Project #2

Date: 03.06.2022
Student: Nadide Beyza Dokur
Instructor: Muhittin Gökmen
Institution: MEF University

1. Introduction

The purpose of picture colorization is to create a colorful image from a grayscale input image. Because this challenge is multimodal, a single grayscale image can correlate to a variety of plausible colored images. As a result, previous models frequently required a lot of human input in addition to a grayscale image.

Deep neural networks have recently demonstrated exceptional performance in automatic picture colorization, converting grayscale images to color without the need for extra human input. This success might be attributed in part to their capacity to acquire and apply semantic information (i.e., what the image is) in colorization, while we are yet unsure what makes these models perform so well.

In this project perform a performance evaluation of the two image color detection models. These machine learning models automatically turn grayscale images into colored images. The first model is called Image Colorization Basic Implementation with CNN and was built by Victor Basu(https://www.kaggle.com/code/basu369victor/image-colorization-basic-implementation-with-cnn/notebook). The second model is called Colorizing B&W Photos with Neural Networks and was built by Emill Warner(https://blog.floydhub.com/colorizing-b-w-photos-with-neural-networks/).

2. Models

2.1. Color Detetction Algorithm - 1: Image Colorization Basic Implementation with CNN by Victor Basu

In the GAN network, the auto-encoder network that is described in this model also serves as a generator.

The goal of this model is to see how well the most basic auto-encoder performs against image colorization. This auto-encoder model is used in conjunction with higher-level or more advanced approaches to get outstanding picture colorization outcomes.

The image was colorized using Lab value, where L refers to brightness and a and b stand for the color spectra green–red and blue-yellow, respectively. It converts the RGB picture to a LAB image, then extracts the L and ab values from the image before training the model to predict the ab value.

Since the model dataset contains only colored images while the model is trained on black and white images, it converts the colored image to grayscale and then to RGB format to achieve complete black and white image conversion. Opencv was also used to read the image, however, because Opencv reads the image in BGR format, it must first convert the image to RGB before converting it to grayscale.

2.2. Color Detetction Algorithm - 2: Colorizing B&W Photos with Neural Networks by Emill Warner

The model was constructed in three stages.

The first portion deconstructs the central reasoning. As an "Alpha" colorization bot, it creates a simple 40-line neural network.

The next phase is to build a generalizable neural network (the "Beta" version). The model is able to color photos that have never been viewed before by the bot.

It integrates the neural network with a classifier in the "final" form. It employs a Resnet V2 from Inception that has been trained on 1.2 million photos.

3. Performance Evaluations & Results

In this part of the project performance, evaluations of the two models will be completed in three steps. The first step is to evaluate the model losses during the epochs. The second step is to perform MSE comparisons as a loss function. The final step is to visual inspection comparisons of the two models.

3.1. The Model Losses During The Epochs

To assess the colorization autoencoders' performance, look at the evolution of the model losses over time for each of the models independently. There are 10 epochs and 1 step for each epoch.

3.1.1. Color Detetction Algorithm - 1

Figure 1 shows the model losses during the epochs.

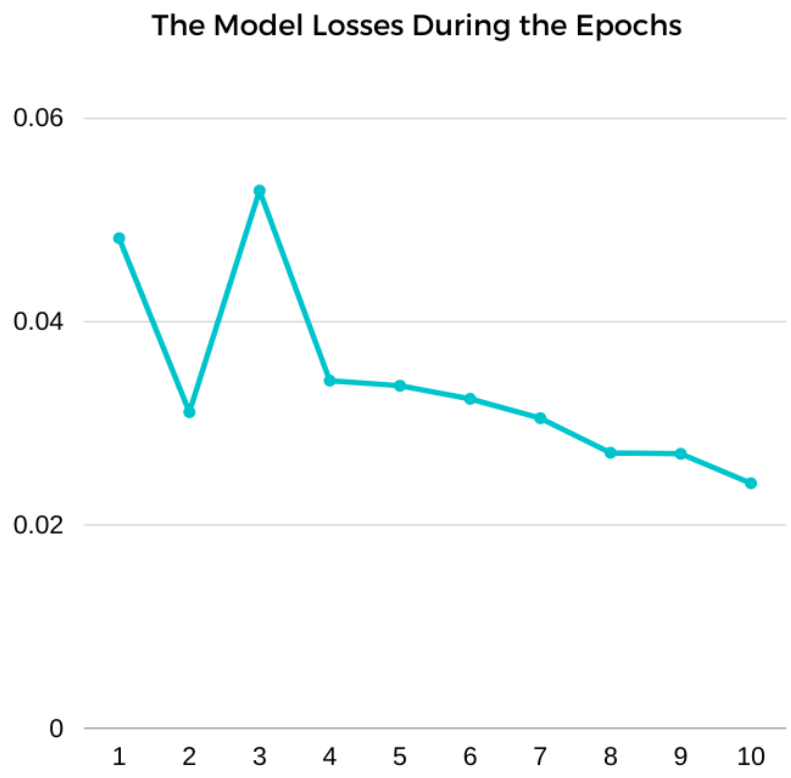
```
def GenerateInputs(X_,y_):
    for i in range(len(X_)):
        X_input = X_[i].reshape(1,224,224,1)
        y_input = y_[i].reshape(1,224,224,2)
        yield (X_input,y_input)
Model_Colourization.fit_generator(GenerateInputs(X_,y_),epochs=10,verbose=1
```

```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning

1/1 [==============================] - 2s 2s/step - loss: 0.0482
Epoch 2/10
1/1 [==============================] - 2s 2s/step - loss: 0.0311
Epoch 3/10
1/1 [==============================] - 2s 2s/step - loss: 0.0529
Epoch 4/10
1/1 [==============================] - 2s 2s/step - loss: 0.0342
Epoch 5/10
1/1 [==============================] - 2s 2s/step - loss: 0.0337
Epoch 6/10
1/1 [==============================] - 2s 2s/step - loss: 0.0324
Epoch 7/10
1/1 [==============================] - 2s 2s/step - loss: 0.0305
Epoch 8/10
1/1 [==============================] - 2s 2s/step - loss: 0.0271
Epoch 9/10
1/1 [==============================] - 2s 2s/step - loss: 0.0270
Epoch 10/10
1/1 [==============================] - 2s 2s/step - loss: 0.0241
<keras.callbacks.History at 0x7f20d3ba6710>
```

Figure 1. The Model Losses During the Epochs

Graph 1 shows the model losses during the epochs.

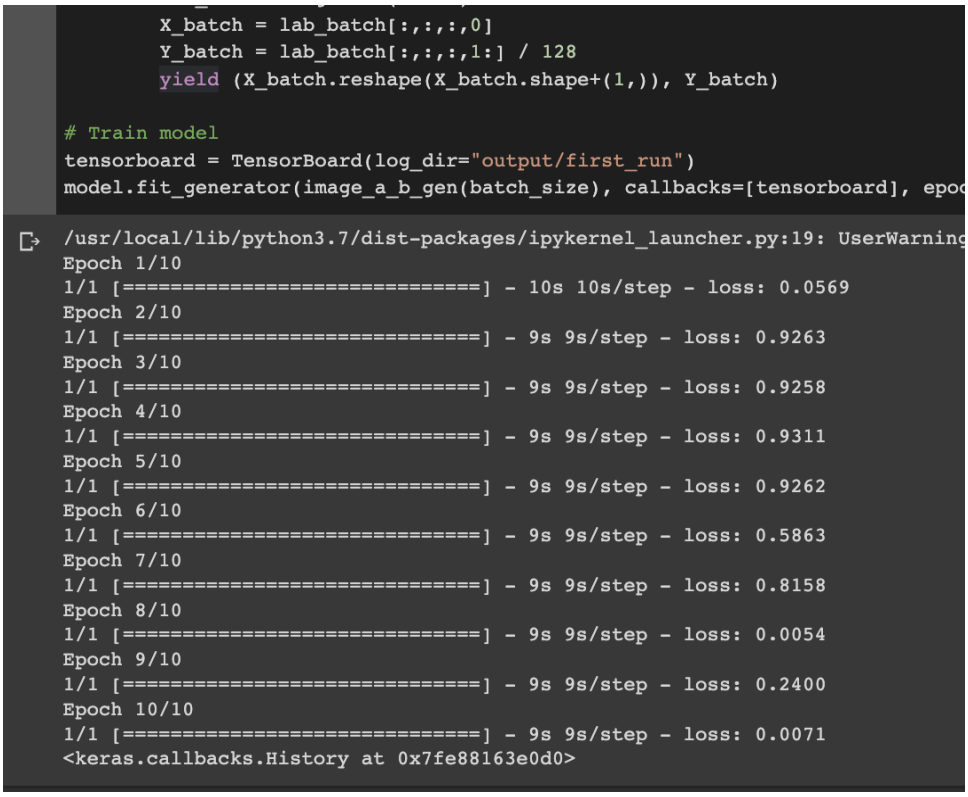


Graph 1. The Model Losses During The Epochs

In the image we can see that the loss started to plateau, however, it would probably still decrease if trained for more epochs. Also, there are one weird decrease and an increases in the validation loss.

3.1.2. Color Detetction Algorithm - 2

Figure 3 shows the model losses during the epochs.



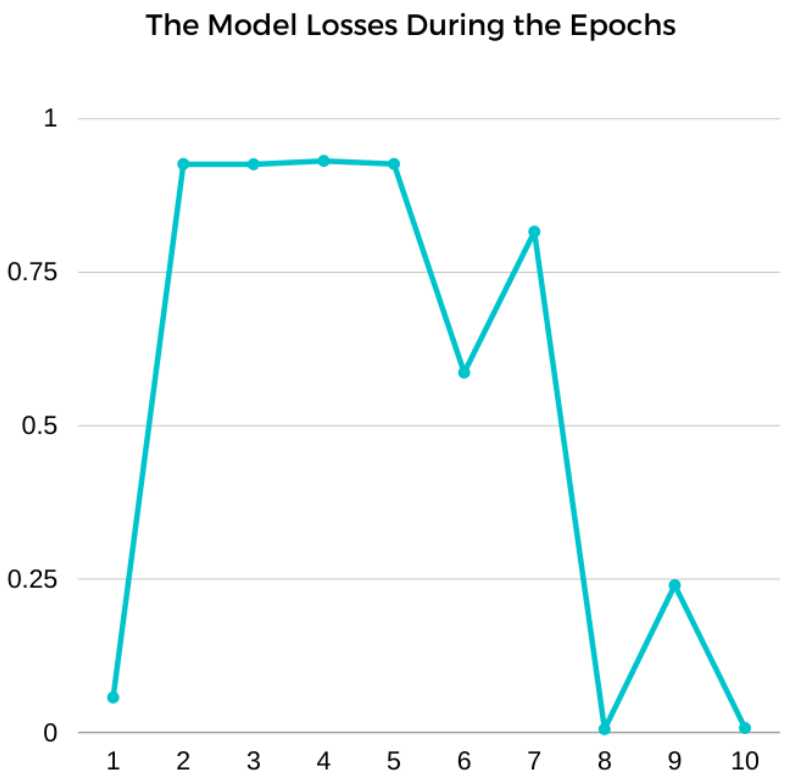
```
        X_batch = lab_batch[:,:,:,0]
        Y_batch = lab_batch[:,:,:,1:] / 128
        yield (X_batch.reshape(X_batch.shape+(1,)), Y_batch)

# Train model
tensorboard = TensorBoard(log_dir="output/first_run")
model.fit_generator(image_a_b_gen(batch_size), callbacks=[tensorboard], epoc
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: UserWarning
Epoch 1/10
1/1 [==============================] - 10s 10s/step - loss: 0.0569
Epoch 2/10
1/1 [==============================] - 9s 9s/step - loss: 0.9263
Epoch 3/10
1/1 [==============================] - 9s 9s/step - loss: 0.9258
Epoch 4/10
1/1 [==============================] - 9s 9s/step - loss: 0.9311
Epoch 5/10
1/1 [==============================] - 9s 9s/step - loss: 0.9262
Epoch 6/10
1/1 [==============================] - 9s 9s/step - loss: 0.5863
Epoch 7/10
1/1 [==============================] - 9s 9s/step - loss: 0.8158
Epoch 8/10
1/1 [==============================] - 9s 9s/step - loss: 0.0054
Epoch 9/10
1/1 [==============================] - 9s 9s/step - loss: 0.2400
Epoch 10/10
1/1 [==============================] - 9s 9s/step - loss: 0.0071
<keras.callbacks.History at 0x7fe88163e0d0>
```

Figure 3. The Model Losses During the Epochs

Graph 2 shows the model losses during the epochs.



Graph 2. The Model Losses During The Epochs

The model started at a higher loss than the V0 model, probably due to the much larger number of learnable parameters.

Graph 3 shows the comparison of the two models losses during the epochs. Red line represents Color Detection Algorithm-1 and green line represents Color Detection Algorithm-2.

Graph 3. Losses of the Two Models During The Epochs

The only thing that stands out in the plot above for the most sophisticated model (also in terms of the amount of learnable parameters) is the massive spike in the validation loss. It would be fascinating to learn the precise cause for this.

3.2. MSE Comparisons as Loss Function

The second step is to compare all of the models' validation set MSE at the 1th, 5th, and 10th.

Model / Epoch	1	5	10
Color Detetction Algorithm - 1	0.0482	0.0337	0.0241
Color Detetction Algorithm - 2	0.0569	0.9262	0.0071

Figure 4. MSE Comparisons as Loss Function

Overall, the Color Detection Algorithm-1 scored the lowest validation loss compare to Color Detection Algorithm-2.

3.3. Visual Inspection Comparisons

The results must then be presented for visual assessment.

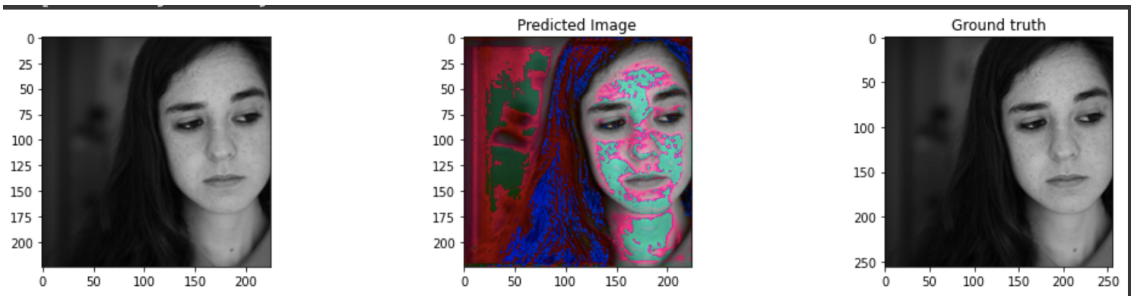


Figure 5. Output of the Color Detection Algorithm-1

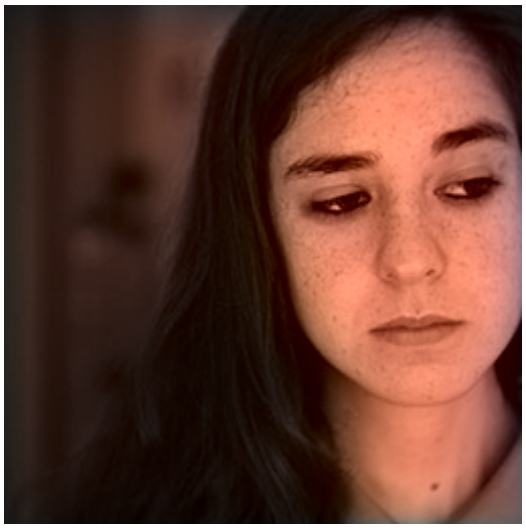


Figure 6. Output of the Color Detection Algorithm-2

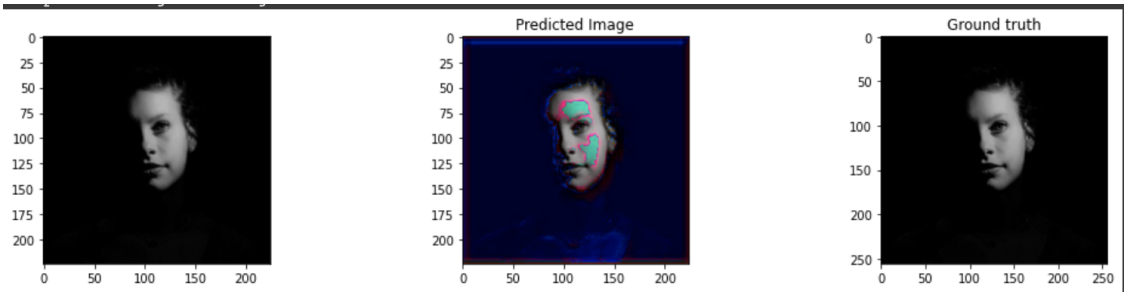


Figure 7. Output of the Color Detection Algorithm-1

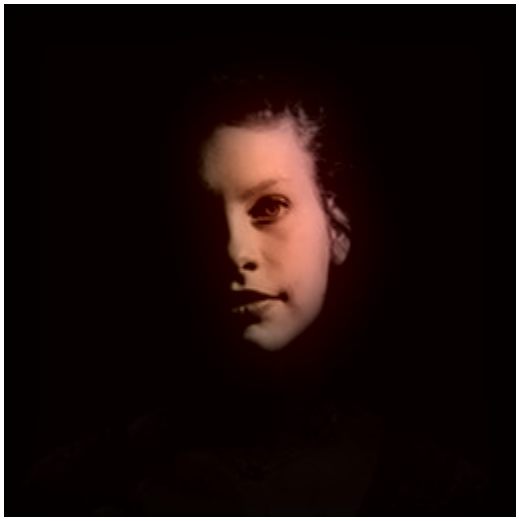


Figure 8. Output of the Color Detection Algorithm-2

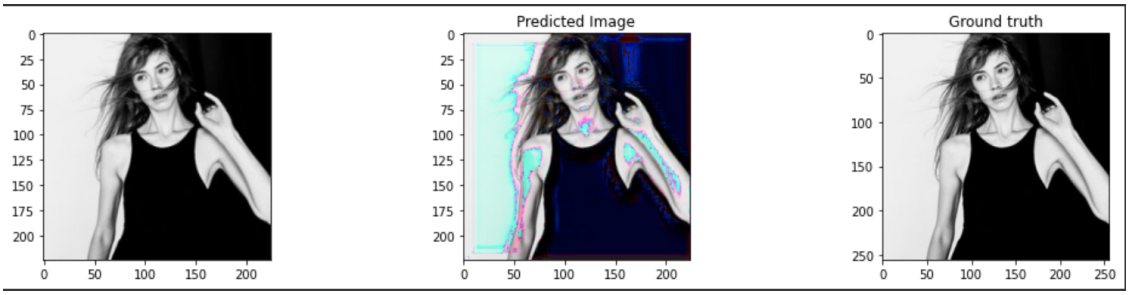


Figure 9. Output of the Color Detection Algorithm-1

Figure 10. Output of the Color Detection Algorithm-2

The Color Detection Algorithm-1 does a good job of capturing the image's color changes, but it colors itself wrongly. It also did the best job coloring the map; the V2 image is a little "trippy," with a lot of bright patches.

The Color Detection Algorithm-2 model captured the backdrop well, however, the Color Detection Algorithm-1 model is having trouble deciding which color to utilize.

All of the models were able to capture the images' faces.

The models attempted to colorize the above image, but the results were less than ideal. One reason could be that the input photographs are dark.

Summing up, the models are doing quite a decent job of colorizing the faces on the images. The neural networks seem to have human portrait pattern. However, the Color Detection Algorithm-1 is quite unstable and often results in bright and mismatched colors.

4. Conclusion

I had a great time working on this project and learned a lot. The project is far from comprehensive and exhaustive. Because of time limits, I did not explore all of the aspects I had in mind when it came to image colorization.
As a result, I looked into several possibilities for future expansions:

- Apply data cleaning to photos with a high fraction of totally white/black pixels, for example. This could indicate that these are some screen transitions, and that including them in the dataset will cause the model to be overly noisy.

- Use transfer learning from a network that has already been trained.

- Play around with GANs.

5. References

[1]"Image Colorization basic implementation with CNN", *Kaggle.com*, 2022. [Online]. Available: https://www.kaggle.com/code/basu369victor/image-colorization-basic-implementation-with-cnn/notebook. [Accessed: 03- Jun- 2022]

[2]"Colorizing B&W Photos with Neural Networks", *FloydHub Blog*, 2022. [Online]. Available: https://blog.floydhub.com/colorizing-b-w-photos-with-neural-networks/. [Accessed: 04- Jun- 2022]

[3]*Cs230.stanford.edu*, 2022. [Online]. Available: https://cs230.stanford.edu/projects_spring_2018/reports/8290982.pdf. [Accessed: 03- Jun- 2022]

[4]*Lri.fr*, 2022. [Online]. Available: https://www.lri.fr/~gcharpia/colorization_chapter.pdf. [Accessed: 03- Jun- 2022]

[5]"Image Colorization with Convolutional Neural Networks", *Lukemelas.github.io*, 2022. [Online]. Available: https://lukemelas.github.io/image-colorization.html. [Accessed: 03- Jun- 2022]

[6]"Color Classification With Support Vector Machine", *Medium*, 2022. [Online]. Available: https://medium.com/deepvision/color-classification-with-support-vector-machine-755360c96037. [Accessed: 03- Jun- 2022]

[7]"Image Colorization using Convolutional Autoencoders", *Medium*, 2022. [Online]. Available: https://towardsdatascience.com/image-colorization-using-convolutional-autoencoders-fdabc1cb1dbe. [Accessed: 03- Jun- 2022]