Image Colorization Using Autoencoders

Akhil Varma Vegesna[1], Teja Naidu Chintha[2]

[1,2]Indiana University Bloomington

akveges@iu.edu, tnchinth@iu.edu

Abstract

Image Colorization is a process of converting a grayscale image to color images. There are many potential uses for image colorization. We have explored a variety of neural networks, methods, and architectures that can be applied to the development of an image colorization model, including ImageNet, VGG16, and Capsule network. However, because the convolutional autoencoders are so effective at teaching the abstract representation of any type of data, we chose to implement it. And it requires little to no direct human interaction.

Introduction

An autoencoder is a form of neural network used for feature learning and dimensionality reduction. It consists of two components: a decoder that maps the lower-dimensional representation back to the original input space from the encoder's lower-dimensional representation of the input data. Autoencoders are trained by feeding them input data and trying to reconstruct the original input from the lower-dimensional representation. This forces both the encoder and the decoder to learn how to reconstruct the original input from the compressed representation of the input data. It can be trained using unsupervised, semi-supervised, or supervised learning methods.

Image colorization is an interesting problem because it involves using machine learning to add color to grayscale images in a way that accurately represents the original image. Autoencoders are well-suited for this task because they can learn the structure of an image, which allows them to generate new data that is consistent with that structure. Additionally, solving this problem has the potential to provide benefits in various applications, such as improving the visual quality of old black and white photos or making it easier to analyze satellite images. It can be used to colorize medical scans and images, making them simpler to comprehend and study in the field of medicine. It can be used to colorize old records and artifacts in the study of history, giving a more realistic depiction of how they appeared at the time of creation.

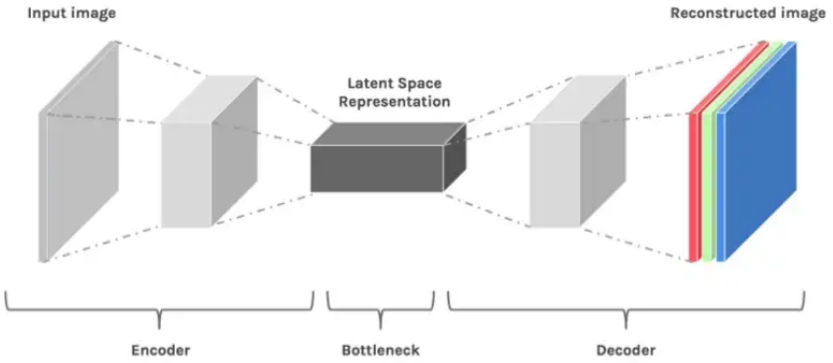

Fig 1: Autoencoder Diagram [7]

Dataset and Preprocessing

We used a Kaggle dataset which contains the grayscale and their corresponding color images. We divided them into train, validation, and test data - 5000, 500, 500 pictures respectively. We use cv2 library to convert color images to red, green, and blue channels on a scale of 0 to 255 and making them a size of 160x160x3. and grayscale of size 160x160x1 matrices.

Instance of Dataset

Fig 2: Dataset instance

Methodology

We used convolutional autoencoders to perform colorizing grayscale images. We used an architecture inspired from U-Net which is famous for biomedical image segmentation and is designed to perform semantic segmentation. Below is the U-Net architecture.

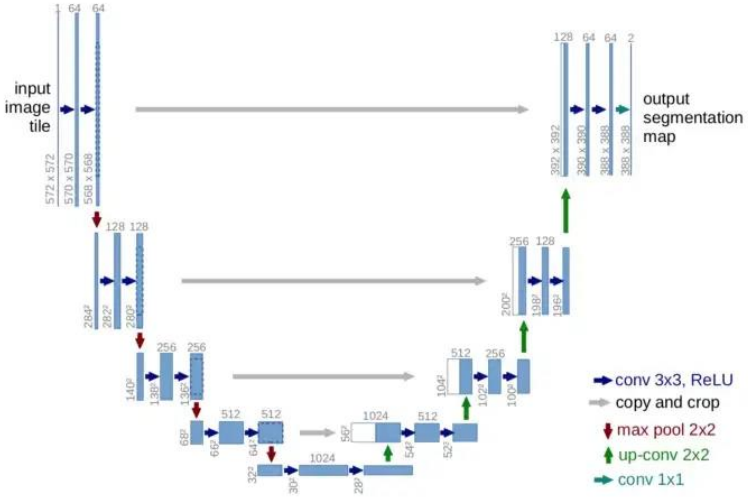

Fig 3: U-Net architecture [3]

As biomedical problems need to not only classify but also locate the abnormality, U-Net achieves this by implementing classification on every pixel of the image and distinguish boundaries, so that the output and input share the same dimensions.

U-Net has an architecture of symmetric U-shape and has two main parts, the left side performs contraction(encoding) using convolutional layers and the right side performs expansion(decoding) using transpose convolutional layers. The images are downscaled during contraction and upscaled during expansion. To avoid feature loss in the process, each encoder is linked to corresponding decoder using concatenate layer. Concatenation of feature maps give localization information which makes the semantic segmentation possible.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 160, 160, 1)	0	
sequential (Sequential)	(None, 80, 80, 128)	1280	input_1[0][0]
sequential_1 (Sequential)	(None, 40, 40, 128)	147584	sequential[0][0]
sequential_2 (Sequential)	(None, 20, 20, 256)	296192	sequential_1[0][0]
sequential_3 (Sequential)	(None, 10, 10, 512)	1182208	sequential_2[0][0]
sequential_4 (Sequential)	(None, 5, 5, 512)	2361856	sequential_3[0][0]
sequential_5 (Sequential)	(None, 10, 10, 512)	2359808	sequential_4[0][0]
concatenate (Concatenate)	(None, 10, 10, 1024)	0	sequential_5[0][0] sequential_3[0][0]
sequential_6 (Sequential)	(None, 20, 20, 256)	2359552	concatenate[0][0]
concatenate_1 (Concatenate)	(None, 20, 20, 512)	0	sequential_6[0][0] sequential_2[0][0]
sequential_7 (Sequential)	(None, 40, 40, 128)	589952	concatenate_1[0][0]
concatenate_2 (Concatenate)	(None, 40, 40, 256)	0	sequential_7[0][0] sequential_1[0][0]
sequential_8 (Sequential)	(None, 80, 80, 128)	295040	concatenate_2[0][0]
concatenate_3 (Concatenate)	(None, 80, 80, 256)	0	sequential_8[0][0] sequential[0][0]
sequential_9 (Sequential)	(None, 160, 160, 3)	6915	concatenate_3[0][0]
concatenate_4 (Concatenate)	(None, 160, 160, 4)	0	sequential_9[0][0] input_1[0][0]
conv2d_5 (Conv2D)	(None, 160, 160, 3)	51	concatenate_4[0][0]

Total params: 9,600,438
Trainable params: 9,597,878
Non-trainable params: 2,560

Fig 4. Model Summary

Our model has two blocks. The first layer is an input layer which takes the image size 160x160x1 (grayscale) and is followed by 5 Conv2D layers with strides 2, padding set as same which ensures output and input image have same dimensions and increasing filter size which is shown above. We applied Batch Normalization on 3rd, 4th and 5th Conv2D layers. During this process, each hidden layer's output is normalized using Batch Norm. We have bottleneck at 5th Conv2D layer which has a size of 10x10x1024.

The second block has 5 Conv2D Transpose layers with strides 2, padding set as same and decreasing filter size. We added dropout layer which gets initialized if dropout gets defined. Each Conv2D Transpose layer is followed by a concatenate layer which connects a Conv2D and a corresponding Conv2D Transpose layer which allows to avoid feature loss. After the 5th Conv2D Transpose layer is concatenated to input layer, an output layer performs Conv2D and brings the output size of 160x160x3. This last layer has strides = 1 and filter size 3 to bring the RGB channels. All the layers are given a kernel size of (3,3) except the output layer with (2,2).

Experimental Setup

Activation function:

All the layers use LeakyReLU activation function. To overcome any possible vanishing gradient problem, which happens when output is negative resulting in 0 when derivative is calculated.

Optimizer and Learning rate:

With careful research and experimentation, we chose Adam optimizer and 0.001 learning rate. Adam optimizer is said to have advantages of both RMSProp and AdaGrad and achieves good results faster than other optimizers. We have seen that 0.001 learning rate yielded better results.

Loss Function:

The loss function used is 'Mean Absolute Error' and accuracy metric. Although Mean Absolute error is a popular solution for our problem it might face issues as colorizing is a multimodal problem, there are various possible color versions which are acceptable for each pixel. This can lead to models trying to choose colors with low vibrance to avoid high mean absolute error penalty leading to desaturated colorizations.

Batch size, Epochs:

When training we used a batch size of 50 and experimented on several numbers of epochs. We finalized having 100 epochs as the loss already reached a plateau for epochs more than 100. We added an EarlyStopping callback, monitoring loss to avoid overfitting the model. Lastly, we used verbose of 0 and validation data of color and grayscale images which was produced when splitting the dataset to monitor progress and loss. We used Google Colab to run the code and used GPU for faster training and evaluating.

Results

When fitting model history.history modules stored the training and validation accuracies and losses. To evaluate the autoencoder's performance we plot training and validation losses over epochs and training and validation accuracy over epochs

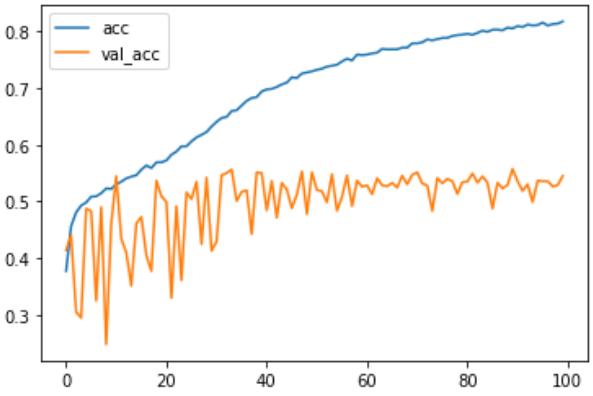

Fig 5. Accuracies over epochs

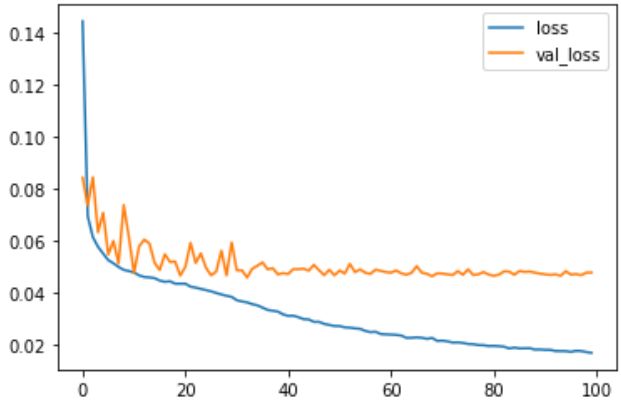

Fig 6. Loss over epochs

For the accuracy plot we see that training accuracy kept increasing and almost reached plateau whereas validation accuracy has a lot of fluctuations and reached plateau way before training accuracy. We achieved a lower validation accuracy but received good training accuracy.

For the loss plot we see training loss kept decreasing and almost reached plateau whereas validation loss reached plateau before training loss. We see a substantial decrease in training loss than the validation loss. The gap between training and validation lines can be explained by validation data not being a proper representation of the entire dataset.

When evaluating the model using test dataset, we achieved an accuracy of 54% and loss of 48%. Accuracy does not seem like a good measure to evaluate performance of autoencoders are there are various acceptable colors for each pixel. So, checking the model's performance by seeing the predicted results is a better way of evaluation. After evaluation function we plotted color image, respective grayscale image and the predicted color image from grayscale image.

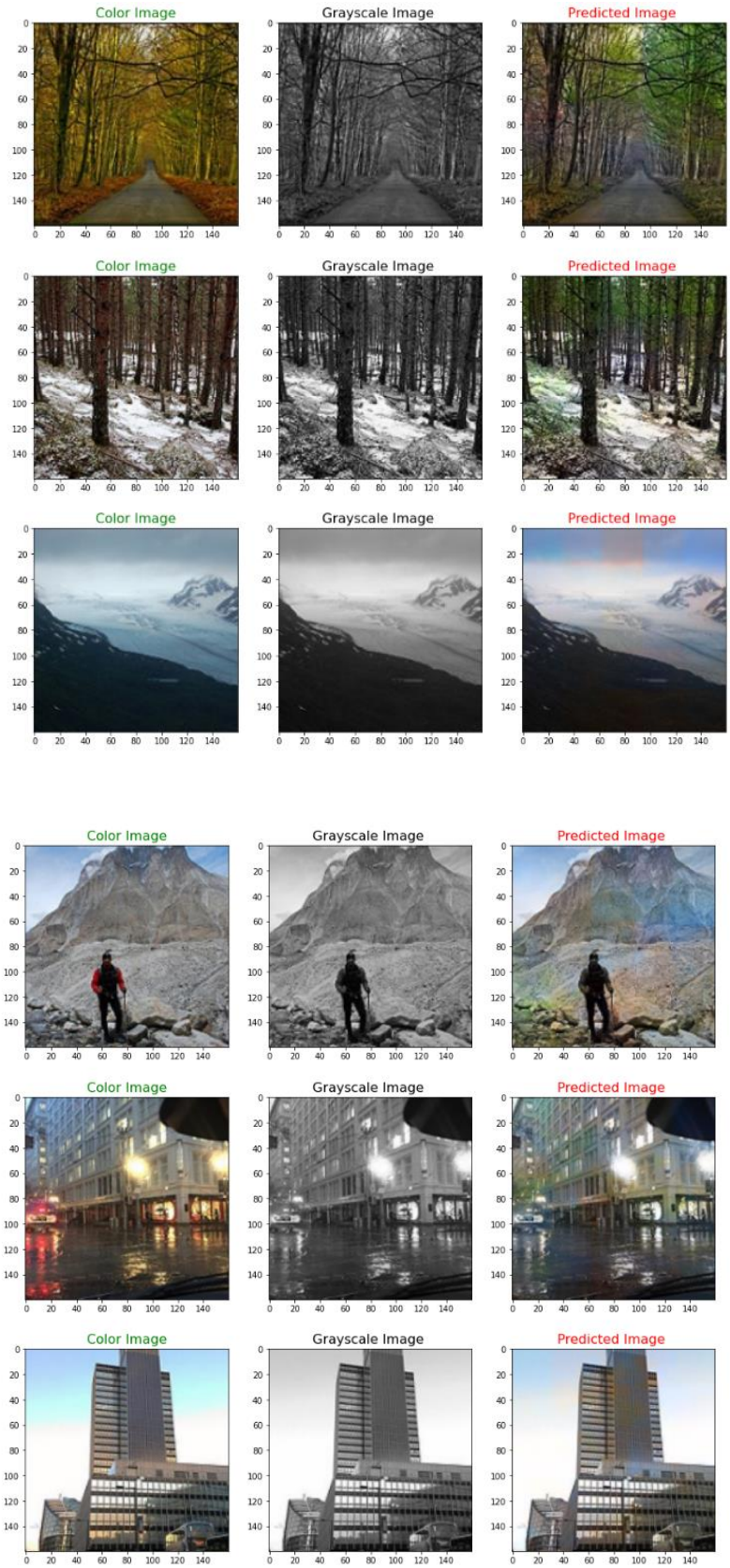

Fig 7. Test Results

Conclusion

We have successfully created a convolutional autoencoder to colorize grayscale images. We converted color images to RGB color space before training the model. We achieved a good prediction, and we plotted losses and accuracies to evaluate the model performance.

The fluctuations and gap between training and validations lines in plots could be because of uneven representation between validation and train sets. We didn't receive low saturated output images, so Mean Absolute error did a good job. The model also did a great job in capturing backgrounds as well as foregrounds. The model also delivered output images in good resolution. The model did a good job in sematic segmentation but struggled a little in deciding which color to use.

Both teammates contributed equally to the project.

Future Scope

In the future, would like to experiment using other color spaces like Lab, YCbCr which drastically reduce the complexity as total parameters decrease. For Lab space, a and b represent the color spectrum from green to red and blue to yellow respectively. So, complexity reduces a lot as channels become two instead of three. We would also like to use better datasets which have equal representation of all types of pictures and colors.

References

1. https://towardsdatascience.com/image-colorization-using-convolutional-autoencoders-fdabc1cb1dbe
2. https://www.kaggle.com/datasets/aayush9753/image-colorization-dataset
3. https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/
4. https://towardsdatascience.com/image-colorization-using-convolutional-autoencoders-fdabc1cb1dbe
5. https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5
6. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/
7. https://sefiks.com/2018/03/23/convolutional-autoencoder-clustering-images-with-neural-networks/