Image colorization using CNNs

Aditya Sri Prasad Appana
Northeastern University
Boston,MA
appana.a@northeastern.edu

Abstract

This project is an implementation of an Image colorization network using a Convolutional Neural Network and a pre-trained model on PyTorch. The implementation is based on the paper Deep Koalarization by Federico et. al. [1]. In this implementation, we study how using various feature extractors affects the performance of the model, and how a different loss function can be used to attain better results.

1. Introduction

In various computer vision applications, though color images are widely represented using the RGB color space, many other color spaces depict the image information better. Even better than RGB space. These include color spaces like HSV, Lab, YCbCr, YUV, etc., which separate the luma, or the image intensity, from chroma or the color information.

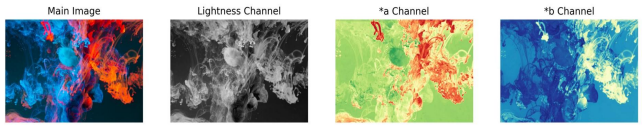

Figure 1 : A sample figure which shows the color ranges in the LAB color space. Lightness channel is a grayscale image. a* channel holds the Redness and greenness information. the b* channel holds the blueness and yellowness information.

From Figure 1, we see that the Lightness channel is basically a grayscale image. In every colorization research paper so far, the problem of colorization is solved by passing a gray scale image to the network and tuning the network to predict color channels. One of the main reasons LAB color space is used is because grayscale image is readily available as one of the channels. If we want to use an RGB image, there is an overhead of converting the RGB to a grayscale image, and then post processing the model's predicted output back to a colored RGB image. The model also has to predict 3 channels. This is inefficient considering in LAB space, the model just has to predict 2 channels. Using the same logic, the authors of the paper [1] decided to choose the LAB color space for their experimentation. The L channel in LAB depicts the lightness channel which is a grayscale image, it just contains the contrast between the lightest and darkest points in the image. It has values ranging from 0 to 100. the A and B channels are the color dimensions. A channel represents greenness to redness with values of -128 to +127 and b represents blueness to yellowness also with values from -128 to +127. So the network in this paper[1] tries to predict the A and B channels of an image, given its L channel(grayscale image).

Mathematically, the objective of the project is:

$$F: (X_L) \rightarrow (X_A, X_B) \quad (1)$$

where $X = (X_L, X_A, X_B)$ is a color image with 3 channels.

The task of "colorizing" a grayscale image involves assigning three-dimensional (RGB) pixel values to an image which varies along only one dimension (luminance or intensity). Since different colors may have the same luminance value but vary in hue or saturation, the problem of colorizing grayscale images has no inherently "correct" solution. Due to these ambiguities, human interaction usually plays a large role in the colorization process.[12]

1.1 Motivation

Designing and implementing an end to end solution to convert any grayscale image to a color image has been an important research study ever since the boom of Neural Networks. This research area, often referred to as Image colorization, is difficult because the network has to basically fool the human eye. This below implementation is able to colorize the image upto an extent combining Convolutional Neural Networks as encoder and decoder, and a pre trained feature extractor to extract high level image details.

2. Background:

2.1 Model Architecture

The model suggested in the paper [1] consists of an encoder, Feature Extractor, a fusion layer, and a decoder.

They used a pre-trained Inception-Resnet-v2 model as a high-level feature extractor, while the Deep CNN model is trained from scratch. The feature extractor provides information about the high level image contents that can help their colorization (it enhances the coloring process). An embedding of the gray-scale image is retrieved from the Inception model's last layer. The encoder and the feature extraction components obtain mid and high-level features, which are then fused in the fusion Layer. Finally, the Decoder uses these features to estimate a color image.

Encoder Network			Decoder Network		
Layer	**Kernels**	**Stride**	**Layer**	**Kernels**	**Stride**
Conv	64 x (3x3)	2x2	fusion conv	256 x(1x1)	1x1
Conv	128 x (3x3)	1x1	Conv	64 x (3x3)	2x2
Conv	128 x (3x3)	2x2	Upsample	-	-
Conv	256 x (3x3)	1x1	Conv	64 x (3x3)	2x2
Conv	256 x (3x3)	2x2	Conv	64 x (3x3)	2x2
Conv	512 x (3x3)	1x1	Upsample	-	-
Conv	512 x (3x3)	1x1	Conv	32 x (3x3)	2x2
Conv	256 x (3x3)	1x1	Conv	2 x (3x3)	2x2
			Upsample	-	-

Preprocessing: Before the images are passed to the Network, few preprocessing steps are implemented to ensure the images flow through the network properly. They are first scaled, and centered using the torchvision Transforms module.

Encoder: Encoder takes in a HxW x1 grayscale image and returns a (H/8) x (H/8) x 512 tensor. The encoder has 8 convolutional layers with 3x3 kernels, and padding of 1 to maintain the layer's input size. A Stride of 2 is used at layer 1 ,3 and 5 to reduce the number of computations.

Feature Extractor. The architecture proposed in the paper [1] uses Inception-Resnet-v2. But a couple more of my implementations are discussed in detail in the 3.1 methods section. But in any model, we extract the embedding of the input image, by extracting the layer before softmax. Based on the number of classes, we have a num_classes x 1x1 tensor. In case of Inception resnet v2, it is a 1001x1x1 tensor, These tensors are stacked in both height and width dimensions to create a fusion block. To be precise, this tensor is replicated H*W/64 times. This method was introduced by [7]. This approach obtains a single volume with the encoded image and the mid-level features of shape H/8×H/8×1257. (1257 because 1001+256 where 256 is the depth of the output tensor from the encoder.).

Decoder: This Fusion block then undergoes through 256 convolutions of (1x1) kernels, resulting in a H/8xW/8x256 tensor. This tensor then goes through a series of convolutions and upsampling and gives out a H x Wx 2 tensor. This represents the predicted a*b* channels.

2.2. Literature Survey

Image Colorization is the process of adding plausible color information to monochrome photographs or videos. Colorization is a highly undetermined problem, requiring mapping a real-valued luminance image to a three-dimensional color-valued one that has no unique solution[4]. In [5] Deshpande et. al. conceived the coloring problem as a linear system problem. With the advent of CNNs, there have been many approaches trying to solve the colorization problem. In [6], Cheng Z. et. al. proposed a deep neural network using image descriptors (luminance, and semantic features) as inputs. In [7] 2016, Iizuka, Serra et. al. proposed a method based on using global-level and mid-level features to encode the images and colorize them. In 2017, Larsson et al. [8], proposed a method in which a fully convolutional version of VGG-16 with the classification layer discarded was used to build a color probability distribution for each pixel. Also, in 2017, Zhang et al. [9] presented an end-to-end CNN approach, providing a color recommender system to help novice users and claiming to have enabled real-time use of their colorization system. This current paper [1] being implemented was proposed in 2017 showing robust results compared to all other papers mentioned above. In 2018 P. Isola et. al have proposed an interesting approach to colorization using Adversarial Networks. They call it Image-to-Image Translation with Conditional Adversarial Networks, and claim that it is effective at synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images. More recently in 2021, There is an even better approach to the colorization problem developed by Manoj et. al. called the Colorization transformer[9] based on Self-attention. They call it the ColTran, and claim that their approach gives diverse high fidelity image colorization results based on self-attention. Their architecture uses a conditional autoregressive transformer to produce a low resolution coarse coloring of the grayscale image first, and then two subsequent fully parallel networks upsample the coarse colored low resolution image into a finely colored high resolution image. The results are remarkable.

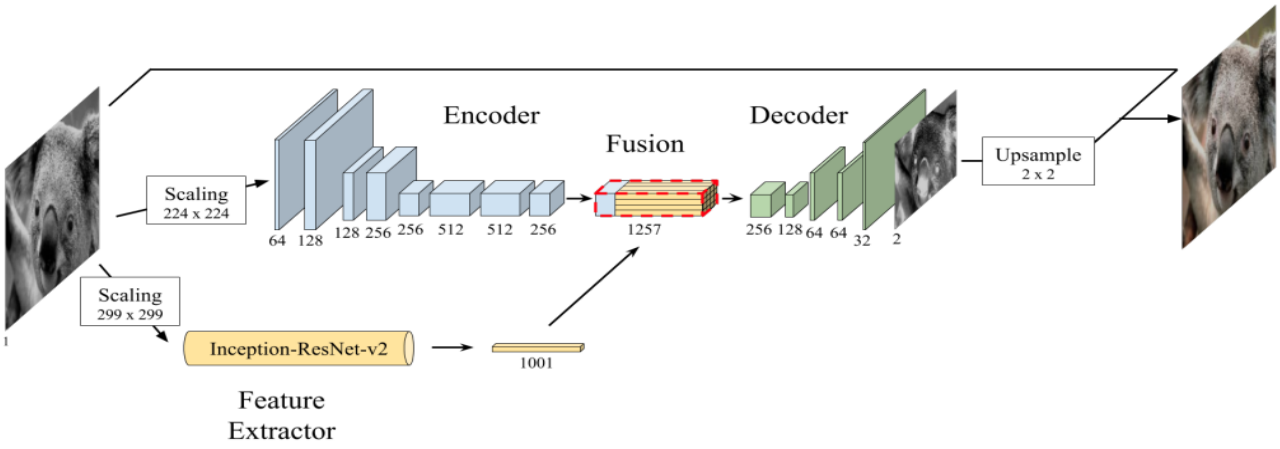

Figure 2 **: Proposed Model Architecture**

3. Methods:

The method proposed by the Deep Koalarization[1] paper uses an Inception-Resnet v2 pretrained model as the feature extractor. With the advancement of computer vision techniques and the availability of compute, there have been a lot of new Deep learning models which perform way better than the Inception-Resnet v2 model. Below are a few different feature extractors I used in my implementation.

a) SE-ReNet-152 as a feature Extractor: After some research, I found that the Squeeze and Excitation networks[10], combined with Resnets performed well on various tasks including image classification, scene classification and semantic segmentations.

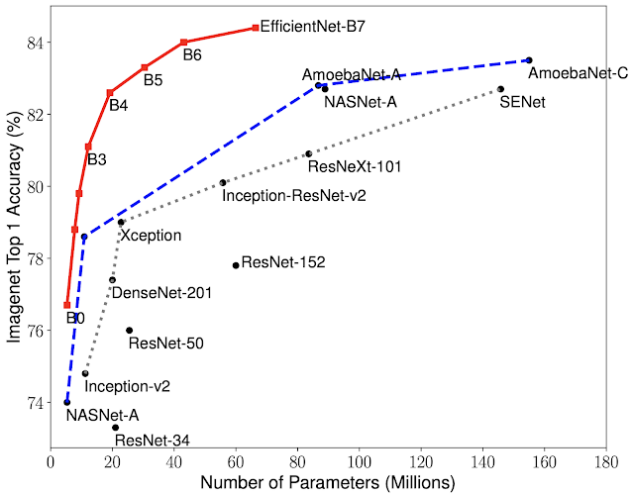

Figure 3**: Model Size vs. ImageNet Accuracy**

Figure 3 depicts the growth of deep learning models's accuracy for Classification on the ImageNet dataset. An immediate observation would be, as the number of parameters increased, the accuracy increases proportionally.

	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [10]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [10]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [10]	23.0	6.7	22.42	6.34	11.30	$21.57_{(0.85)}$	$5.73_{(0.61)}$	11.32
ResNeXt-50 [47]	22.2	-	22.11	5.90	4.24	$21.10_{(1.01)}$	$5.49_{(0.41)}$	4.25
ResNeXt-101 [47]	21.2	5.6	21.18	5.57	7.99	$20.70_{(0.48)}$	$5.01_{(0.56)}$	8.00
VGG-16 [39]	-	-	27.02	8.81	15.47	$25.22_{(1.80)}$	$7.70_{(1.11)}$	15.48
BN-Inception [16]	25.2	7.82	25.38	7.89	2.03	$24.23_{(1.15)}$	$7.14_{(0.75)}$	2.04
Inception-ResNet-v2 [42]	$19.9^{\dagger}$	$4.9^{\dagger}$	20.37	5.21	11.75	$19.80_{(0.57)}$	$4.79_{(0.42)}$	11.76

Figure 4: Single-Crop Error Rates (%) on ImageNet Validation Set

Figure 4 shows the top 1 and top 5 error rates for various model architectures in the first and second columns, and shows the error rate when Squeeze excitation blocks are added to these networks. Following the pattern, I chose the SENet (Squeeze and Excitation networks) along with the Inception Resnet-v2 as it had the lowest error rate. However, I could not find a pretrained model for it. Instead, chose the Resnet 152. The SE-Resnet 152 pretrained model is trained on ImageNet for 1000 classes. It takes in an input of the dimension [3, 224, 224], where 3 is the number of channels, 224 is the Height and Width. It's input is in range [0,1]. The Last layer before the softmax is extracted, hence we get a 1000x1x1 tensor. This tensor is an embedding of the input image(L channel).

b) Using EfficientNet-B7 as a feature Extractor: From Figure 2, It is evident that the EfficientNet-B7 model gives the highest Accuracy with a minimum number of parameters on the ImageNet classification task. Pytorch has a pretrained model for EfficientNet-B0 to B7. EfficientNet takes in an image of shape [3,224,224].

c) Using the default method proposed by the authors but with a different loss function: The reference paper [1] uses Mean Squared Error (MSE) loss to quantify the error between the estimated pixel colors in a*b* and their real value. For an image X, the MSE is calculated as shown in

Equation (2).

$$C(\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2HW} \sum_{k \in \{a,b\}} \sum_{i=1}^{H} \sum_{j=1}^{W} (X_{k_{i,j}} - \tilde{X}_{k_{i,j}})^2 \quad (2)$$

Here θ represents all the parameters in the model. And Xk(i,j) and X˜k(i,j) denote the i,j:th pixel value of the k:th component of the target and reconstructed image, respectively. While training, this MSE loss is back propagated to update the model parameters θ using Adam Optimizer with an initial learning rate of 0.001.
But, this loss function is a bit problematic for colorization problems as it penalizes high pixel prediction values. So the network chooses unsaturated pixel colors, which are less probable to be very wrong over vibrant, saturated colors. To overcome this issue, I'm using the Mean Squared Logarithmic Error (MSLE). As we may not want to penalize the model too much for predicting unscaled quantities directly. Relaxing the penalty on huge differences can be done with the help of Mean Squared Logarithmic Error.

Mean Squared Logarithmic Error (MSLE) Loss function is defined as :

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (\log(y_i + 1) - \log(\hat{y}_i + 1))^2 \quad (3)$$

where y_i is the actual pixel value and $\hat{y}_i$ is the predicted pixel value. The 1 is added for mathematical convenience as y could be 0 and log(0) is not defined.

4. Experiments:

Below are the various experiments performed,

a) Default Network with the Inception-resnetv2 as a feature extractor and with Mean Squared error (MSE) loss.
b) Network with the default Inception-resnetv2 as a feature extractor and with Mean Squared Logarithmic error (MSLE) loss.
c) Network with SE-Resnet-152 as a feature extractor and MSLE as a loss function.
d) Network with EfficientNet-B7 as a feature Extractor and MSLE as a loss function.

Various approaches trying to solve the colorization problem utilize the ImageNet dataset for training their network. The ImageNet dataset is huge with over 14 Million Images. The authors in the reference paper [1] used 60k images from ImageNet with 10% of it as validation data. In my implementation, I'm using 245k images from the ImageNet. This 245k is further split into training,validation and test sets where the validation data percentage can be changed as a hyper parameter. For now, I have allocated 20% of 200k images as a validation set. So the training samples are over 160k, validation samples are over 40k, and the test samples are over 45k.

Results:

Initially, The training was done on my personal laptop (HP omen, AMD Ryzen 7, 4000 series cpu and NVIDIA GTX 1660Ti). The training for 50 epochs took almost a day, but this was with a very very small portion of the Imagenet dataset. (training + validation set = 10k images and testing set = 2k images.)

Below is the plot for Training and Validation loss curves over 50 epochs.

Figure 5 : Plot of the training and validation losses over 50 Epochs. The X-axis shows the number of epochs, and the Y-axis shows the loss values.

Clearly, from Figure 5 it can be seen that the training loss keeps reducing, but the validation loss flattens from the 5th epoch. This means the model is overfitting to the data from the 5th epoch. Ideally, One should stop training the model once you observe the flattening of validation loss or if the validation loss rarely decreases, but I just left it to run overnight.

The results from this model were very bad. This model did not learn much with the very small dataset. This can be seen in Figure 6.

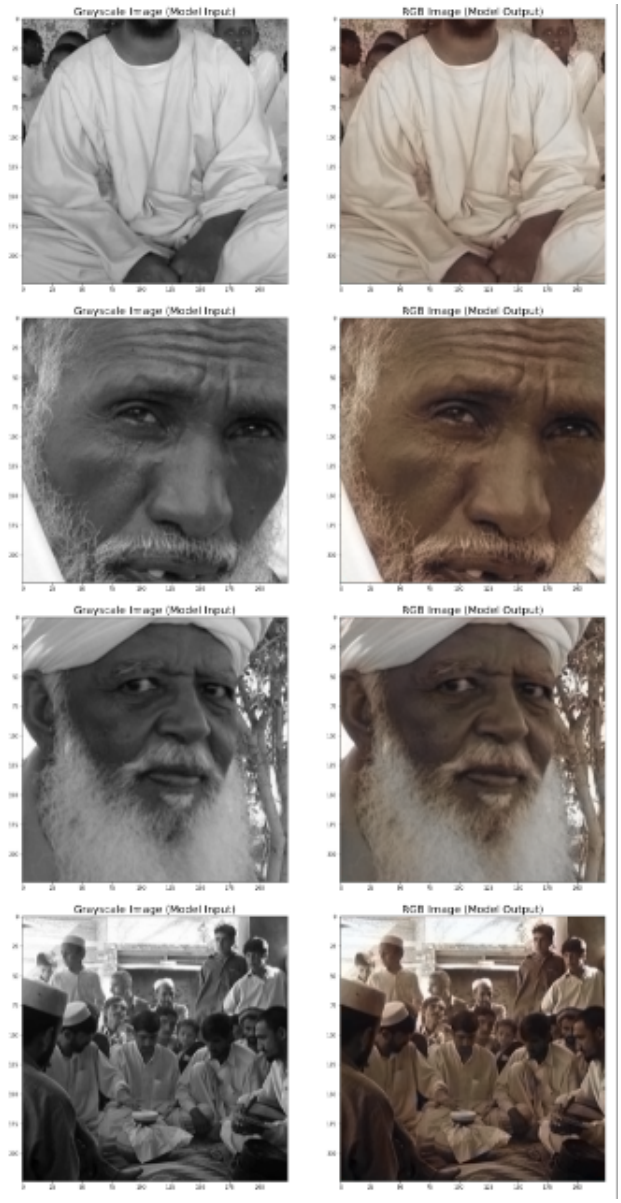

Figure 6: Outputs from model trained with just 10k Imagenet images.

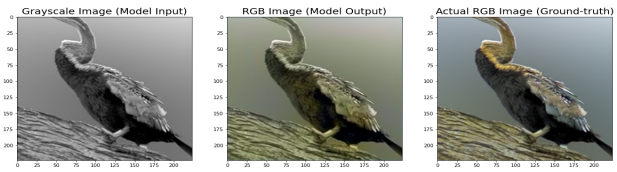

Figure 7: Comparison of the Input image to model, the model's output image, and the Ground truth Image.

Next, with access to the Discovery cluster, I tried to train various models mentioned in the Experiments section.

All the models were trained on Nvidia tesla v100-sxm2 with 20Gb VRAM and 12 CPU's. As the access to the cluster is limited to 8hrs, I was able to train each model in chunks by storing the best model after every 8 hrs. Also, sometimes the discovery cluster clears the workspace randomly which has led to even longer training times.

Below are the results for various experiments:

a) Default Network with the Inception-resnetv2 as a feature extractor and with Mean Squared error (MSE) loss.

Batch Size :100 (As defined in the paper [1])
epochs trained: ~40 epochs
Training time: 25hrs+

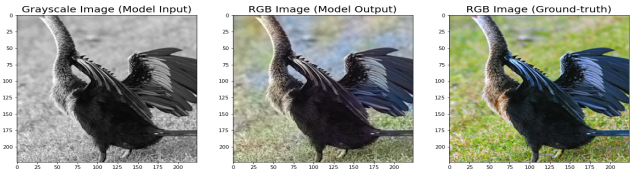

Figure 8: Comparison of the Input image to model, the model's output image, and the Ground truth Image. This image was taken from the Test set.

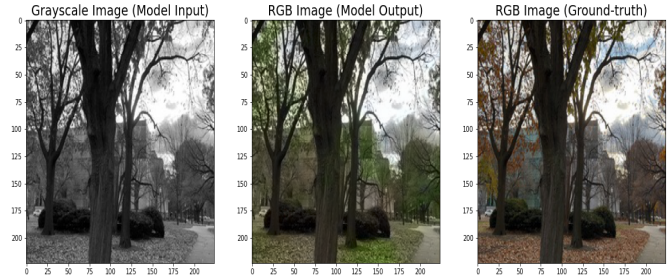

Figure 9: Comparison of the Input image to model, the model's output image, and the Ground truth Image. This image was taken in the Fall season, on my phone near the Museum of Fine Arts.

Figure 9 is a very complex image the model has never seen. There are green, brown, and a tinge of yellow colors in the image. We can infer that as the model was not trained on trees images during the fall season, so it is "biased" to predicting leaves and grass as always green. This can be fixed by adding more trees images from the fall season into the training dataset. Also, the model is able to differentiate the sky, clouds from the buildings in the background.
Training Loss: 0.001836.
Validation Loss: 0.001991.

b) Network with the default Inception-resnetv2 as a feature extractor and with Mean Squared Logarithmic error (MSLE) loss.

Batch Size :250
epochs trained: ~40 epochs
Training time: 24hrs+

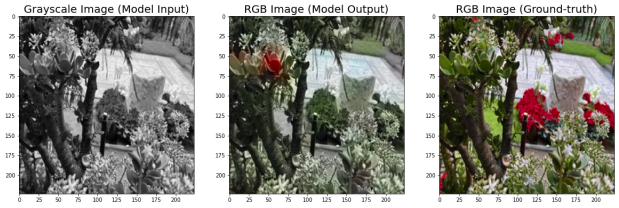

Figure 10: Comparison of the Input image to model, the model's output image, and the Ground truth Image. This image was taken in the Isabella Stewart Museum on my phone.

From Figure 10 we can infer that the model is just beginning to learn the red colors. The same input image, when passed to the model with Mean Squared error (MSE) loss function, has duller greens and does not predict any red colors in the image. That model was settling for pixel values which might not be wrong, ie., it doesn't risk by predicting high contrast colors. Whereas the Mean Squared Logarithmic Error (MSLE) doesn't penalize the model too much for predicting such pixel values. This I believe could be the reason to see a bit brighter and contrastive colors on this models' output images.

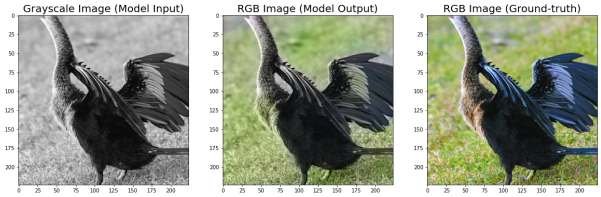

Figure 11: Comparison of the Input image to the model with Inception-resnetv2 pre trained network and MSLE loss, the model's output image, and the Ground truth Image. This image was taken from the Test set.

It can be seen that for the same number of epochs trained, the output of the model(Figure 11) with MSLE loss is much better than the output from the model with default MSE loss (Figure 8).

Figure 12 : Plot of the training and validation losses over 15 Epochs. The X-axis shows the number of epochs, and the Y-axis shows the loss values.

Figure 12 shows the plot for the network with Inception-resnetv2 as a feature extractor and with Mean Squared Logarithmic error (MSLE) loss function. I was able to store the loss values only for the first 15 epochs. Also, as per my understanding, this plot is smoother because of the higher batch size.
Training Loss: 0.024273.
Validation Loss: 0.027985.

c) Network with SE-Resnet-152 as a feature extractor and MSLE as a loss function.
Batch Size :128
epochs trained: ~48 epochs
Training time: 20hrs+

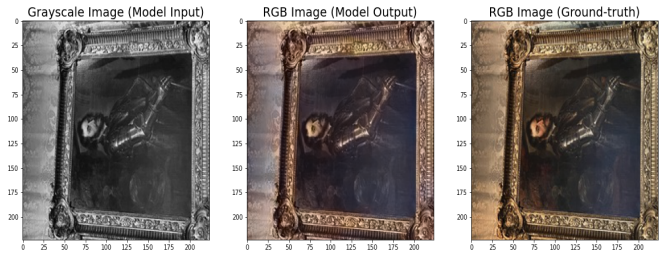

Figure 13 : Comparison of the Input image to the model with SE-resnet 152 pre-trained network and MSLE loss, the model's output image, and the Ground truth Image. This image was taken in the Isabella Stewart Museum on my phone.

The model output in Figure 13 is generated by the Network with SE-resnet 152 as the pre-trained network, and after just around 48 epochs. This model seems to converge to the right colors faster than the other models. Must be due to the high number of learned parameters in the Feature extractor. Also, it is observed that this network had taken less time to train than any of the other networks. I was able to train for ~48 epochs in 20-25hrs.

Figure 14 : Comparison of the Input images to the model with SE-resnet 152 pre trained network and MSLE loss, the model's output image, and the Ground truth Image. These images were taken from the Test set.

The outputs from this model were just brilliant. For a few images I couldn't tell the difference between the original and the predicted images. From Figure 14 we can see that the model accurately predicts even the blue color on the bird's wings.

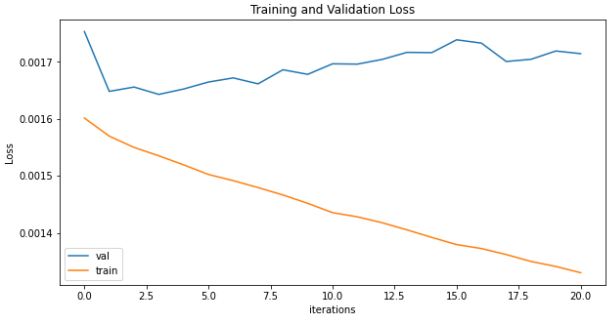

Figure 15 : Plot of the training and validation losses over 20 Epochs. The X-axis shows the number of epochs, and the Y-axis shows the loss values.

Figure 15 shows the plot for the network with SE-resnet 152 as a feature extractor and with Mean Squared Logarithmic error (MSLE) loss function. I was able to store the loss values only for the epochs 2 to 20. Also, as per my understanding, this plot is very smooth because of the higher batch size.
Training Loss: 0.001023.
Validation Loss: 0.001318.

d) Network with EfficientNet-B7 as a feature Extractor and MSLE as a loss function.
Batch Size :64
epochs trained: ~48 epochs
Training time: 20hrs+

Based on Figure 3 we can see that the number of learned parameters in Efficient Net is very less compared to all the other models, but it's classification Top 1 accuracy is relatively high. But as per my observations, the model does not perform very well compared to other models above.

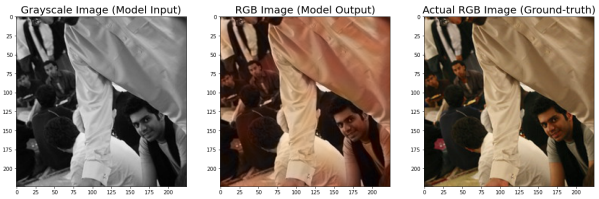

Figure 16: Comparison of the Input image to model with EfficientNet-B7 as a feature Extractor, the model's output image, and the Ground truth Image. This image was taken from the Test set.

From figure 16 we can infer that even though the output looks good, there is an orangish-red tinge across the image. Maybe training for a few more epochs will result in better predictions, but for the same number of epochs, the other models perform much better.
Training Loss: 0.004453
Validation Loss: 0.004380

5. Conclusion:

5.1 Discussion:

This implementation of the proposed end to end deep learning model performs reasonably well. But the results observed in the original paper[1] were far better than my implementations. I think it could be because of the type of training data. It is not clearly mentioned how the 60k images are sampled from the ImageNet dataset in the original paper. It could be a case that they took few images from every class and created their custom dataset so that the model would have seen many classes while training and could be performing well. In my implementation, the 245k images are extracted from the first 500 classes of the imagenet dataset so my model performs well for some images, but very badly for some other images. Maybe proper sampling of the data could help in better performance. It is also observed that the same architecture performs differently under a different loss function. The proposed Mean squared error loss punishes the model for predicting brighter and high contrast pixel values. Whereas the Mean squared logarithmic error predicts brighter pixel values. Maybe a few more epochs and proper sampling of input training data could result in much better results on unseen data. It is also observed that the task of mapping between luminance and a*b* channels is better achieved through probabilistic approaches. Using Variational Autoencoders

seems to be giving better results as per the Colorful image colorization paper by Zang et. al. [14].

5.2. Future Work:

This project is an implementation of the model discussed in the paper [1]. However, I believe this architecture could be improved to see better performance, better results and faster convergence. For example, there is no regularization implemented in this model; regularization could have prevented the overfitting observed in all of the above experiments. Also, there is no implementation of batch normalization in this architecture, adding batch normalization layers could have added some regularization, and we could have used larger learning rates, which would have helped speed up the model training process.

References:

[1] Federico Baldassarre, D. G. Morın, and L. Rodes-Guirao, "Deep koalarization: Image colorization using cnns and inception resnet-v2," arXiv preprint arXiv:1712.03400, 2017. 1, 2.

[2] https://dsp.stackexchange.com/questions/2687/why-do-we-use-the-hsv-colour-space-so-often-in-vision-and-image-processing

[3] Ironi, R., Cohen-Or, D., Lischinski, D.: Colorization by example. In: Rendering Techniques, Citeseer (2005) 201–210

[4] P. Vitoria, L. Raad, and C. Ballester, "Chromagan: Adversarial picture colorization with semantic class distribution," in The IEEE Winter Conference on Applications of Computer Vision, 2020, pp. 2445– 2454.

[5] Deshpande, A., Rock, J., Forsyth, D.: Learning large-scale automatic image colorization. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 567–575.

[6] Cheng, Z., Yang, Q., Sheng, B.: Deep colorization. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 415–423.

[7] Iizuka, S., Simo-Serra, E., Ishikawa, H.: Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. ACM Transactions on Graphics (TOG) 35(4) (2016) 110

[8] Larsson, G., Maire, M., Shakhnarovich, G.: Learning representations for automatic colorization. In: European Conference on Computer Vision, Springer (2017) 577– 593

[9] M. Kumar, D. Weissenborn, and N. Kalchbrenner, "Colorization transformer," in ICLR, 2021.

[10] Jie Hu, Li Shen, Gang Sun; Squeeze-and-Excitation Networks: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 7132-7141

[11] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. arXiv preprint arXiv:1709.01507, 2017.

[12] Welsh, T., Ashikhmin, M., Mueller, K.: Transferring color to grayscale images. In: ACM Transactions on Graphics (TOG). Volume 21., ACM (2002) 277–280

[13] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to image translation with conditional adversarial networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017

[14] Zhang, R., Isola, P., Efros, A.A.: Colorful image colorization. In: European Conference on Computer Vision, Springer (2016) 649–666