CIS 520 Final Project Report

Deep Image Colorization using Transfer Learning

Pankti Hitesh Parekh, Kedar Prasad Karpe, Pavel Shusharan
{pankti81, karpenet, pavs}@seas.upenn.edu

November 2021

1 Abstract

In this project, we propose a deep learning model to colorize grayscale images based on some semantic features. The goal of our work is to generate visually deceptive colorized versions of a given input image. We present a transfer learning approach to solve this problem of colorizing grayscale images using the concept of Autoencoders. We develop a machine learning model and train it on a dataset of 25000 grayscale images. Additionally, we also present performance measures of our approach and also compare it with existing colorization techniques.

2 Motivation

In image colorization, our goal is to produce a colored image using a given grayscale input image. This problem is challenging mainly due to the complexity and the variation of image conditions that need to be analysed by a given model. Even though some image semantics can be learnt, for example apples are red and the sky is blue, a larger subset of the problem is multi-modal. That is, a grayscale pixel may correspond to a multitude of plausible colors without changing the visual meaning of the image. For example, a t-shirt can be colored white, black, yellow, green but convey the same semantic meaning.

Traditional models have generally relied on user input for solving colorization problem. However, we're interested in a deep learning approach which in recent years have proved to outperform traditional models in the sense that they do not rely on any external user inputs other than the image itself.

3 Related Work

Image Colorization has been the focus of significant research in the field of Machine Learning over the past few decades. There is on-going research taking place to mainly focus on colorizing semantics of the images using deep convolutional neural networks to color minute details of images.

Deep colorization [2] can be regarded as the first work to have attempted to incorporate convolutional neural net- works (CNNs) for the colorization of images. However, this method does not fully exploit the CNNs; instead, it also includes joint bilateral filtering as a post-processing step to remove artifacts introduced by the CNN network.

Colorful image Colorization CNN2 [4] was one of the first attempts to colorize grayscale images. The network takes as input a grayscale image and predict 313 "ab" pairs of the gamut showing the empirical probability distribution, which are then transformed to "a" and "b" channels of the "Lab" color space. The network stacks convolutional layers in a linear fashion, forming eight blocks. Each block is made up of either two or three convolutional layers followed by a ReLU layer and Batch Normalization (BatchNorm [20]) layer. Striding instead of pooling is used to decrease the size of the image. The input to the network is 256×256, while the output is 224×224; however, it is resized later to the original image size.

Deep depth colorization $(DE)^2CO$ [1] employs a deep neural network architecture for colorizing depth images utilizing pre-trained ImageNet [23]. The system is primarily designed for object recognition by learning the mapping from the depths to RGB channels. The weights of the pre- trained network are kept frozen, and only the last fully connected layer of (DE)2CO is trained that classifies the objects with a softmax classifier. The pre-trained networks are merely used as feature extractors.

After evaluating some of the above papers, we conclude that image colorization has been done using CNNSs where convolution is performed *n* number of times where the ReLU function is applied to the model so give better classification of colours. The channels in the end are downsampled and upsampled. The main take away from the previous work is semantics is taken into consideration as mentioned above in the motivation.[3] Euclidian distance is considered to evaluate the loss between input and the output received.

The key considerations to be analysed while executing the project are:

1. Perceptual Realism (How compelling the colours look to human observer).
2. Semantics Interpretability.
3. Raw Accuracy (compute the percentage of predicted pixel colors within a threshold L2 distance of the ground truth in AB color space).

4 Dataset

We use Image Colorization dataset from Kaggle which is based on the MIRFLICKR25k dataset of 25k randomly sized colored images. All the images in this dataset are in the CIELab colorspace. The lightness and the ab values are stored in two separate files such that one can be used as input samples and the latter can be used as labels. The dataset consists the following files:

1. **ab.zip:** This contains 3 .npy files consisting of a and b dimensions of LAB color space images, of the MIRFLICKR25k randomly sized colored image dataset. Each file consists a list of *2-channel* ab components of the LAB colorspace of shape 224x224x2.
2. **l.zip:** This file consists of a gray_scale.npy file which is the grayscale version of the MIRFLICKR25k dataset. Each file consists a list of *2-channel* grayscale components of the LAB colorspace of shape 224x224.

Luckily the dataset in already present in the Lab colospace and does not have to be converted. Additionally, all images are of the dimension 224×224 pixels. However, there are preprocessing steps necessary to use this dataset on our model. Show below is a sample image from the dataset. Both the grayscale version and the colorized label is shown.

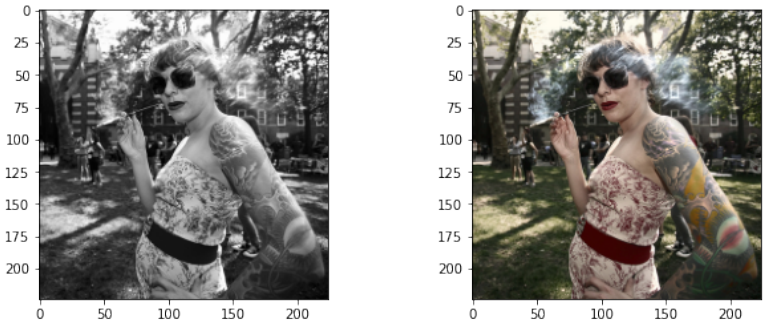

Figure 1: The image on the left shows the grayscale version and the image on the right shows the colorized label of a single image from the dataset.

4.1 Preprocessing

RESNET models accept images of the dimension $224 \times 224 \times 3$ Since our grayscale input image consists only one channel, we modify the first convolutional layer of RESNET to accept a single channel image.

Additionally, since RESNET demands normalised data, we perform a normalization step on our input images by diving all pixels values with 255.

4.2 Postprocessing

We also develop a method to convert Lab images into RGB since the image plotting functions of *matplotlib* cannot directly plot images in the Lab colorspace. We use color functions from *sklearn* to convert a 3 channel Lab image to the RGB colorspace.

5 Problem Formulation

We aim to generate a fully-colorized image, which has 3 channels per pixel, that is, lightness, saturation, and hue, from a grayscale image, which has only 1 lightness channel per pixel. For simplicity, we will only work with images of size 224 x 224, so our inputs are of size 224 x 224 x 1 (the lightness channel) and our outputs are of size 224 x 224 x 2 (the other two channels).

Rather than working with images in the RGB format, we will work with them in the CIELAB colorspace (Lightness, A, and B). This colorspace contains exactly the same information as RGB, but it will make it easier for us to separate out the lightness channel from the other two channels A and B. Additionally, THE CIELAB colorspace is interesting since it was intended to be a perceptually uniform colorspace with respect to the human vision. The A and B channels contain four unique colors of human vision: red, green, blue, and yellow.

We try to predict the color values of the input image directly, that is, we do regression rather than using classification techniques used previously in the literature.

5.1 Model Representation

In this project, we have used the principle of autoencoders to learn the colors from a compressed representation of an image. Our understanding of colorizing an image is that it consists of two steps:

1. Extracting semantic information from the input image.
2. Learning color models of the semantics.

The task of extracting semantic information can essentially be seen as a task of reducing dimensions into a feature space represented by the semantics of the image. We propose the use of encoders to transform the input features into this feature space of semantics.

In the second step of our algorithm, we use the information about these features to predict a full image in CIE Lab Colorspace. Essentially, this step involves using the reduced feature set to reconstruct an image of the original input size. We propose to use convolutional decoders to learn this information about colors from semantic features of the input image.

Our autoencoder model has the following structure:

1. **Input Layer Feature Size:** 224x224x1
2. **Bottleneck Layer Feature Size:** 128
3. **Output Layer Feature Size:** 224x224x2

5.2 Loss Function

Since we are doing regression on the input images, we used a mean squared error loss function. We minimize the squared distance between the color value of every pixel we try to predict, and the labelled color value of that pixel. The mean squared error loss function is slightly problematic for image colorization due to the multi-modality of the problem discussed in the previous sections. For example, if a gray shirt could be blue or green, and if our model picks the wrong color, it will be harshly penalized. As a result, our model will usually choose desaturated colors that are less likely to be wrong than bright, vibrant colors.

6 Methods

6.1 Transfer Learning

Since training a deep neural network can be computationally taxing, we propose leveraging pre-trained models for extracting semantic information.

RESNET-18 is a pre-trained Convolutional Neural Network trained on the Imagenet dataset of more than a million images. The model consists of 4 convolutional layers followed by a fully connected layer which classifies an image into 1000 predefined image classes. We replace the encoder architecture with all layers except the last Fully-Connected layer of RESNET-18 to generate embeddings for the model. We believe these embeddings essentially encode semantic information about the image structure and can thus be exploited to train a model about the color information of the image.

We then pass these embedding through a convolutional decoder to reconstruct the colorized version of the image. At every convolutional layer of the decoder we upscale the image until we get the desired dimension of $224 \times 224 \times 2$.

6.2 Model Architecture

The figure below shows the transfer learning model architecture we have followed in this project. Instead of an encoder, we use embeddings from a pretrained RESNET18 model as shown in the figure.

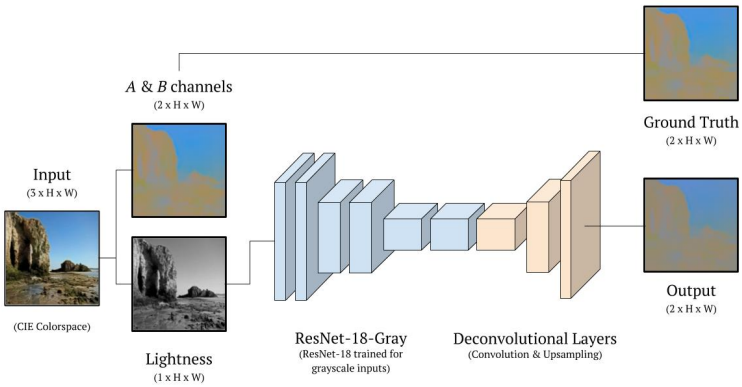

Figure 2: Proposed Transfer Learning Architecture using RESNET-18

6.3 Baseline Methods

- **Color Conversion:** cv2.cvtColor(lab_im, cv2.COLOR_Lab2RGB) method was used to convert the lab image to rgb.
- **Model:** RESNET layers used as the encoder are frozen and help the model identify essential features for the images. In the baseline model the output of the resnet is then upsampled with 2 convolutional layers and nn.PixelShuffle method.
- **Loss Function:** Baseline Loss function is set to be MSELoss, which is used to compare the ab channels of the base image against the output of the model.
- **Optimizer:** Baseline optimizer is set to torch.optim.Adam allowing for efficient gradient computation during training.
- **Learning Rate:** Baseline learning rate is set to 0.001. Since the values of in the tensors vary from 0 to 1 having a learning rate of 0.001 is large enough to ensure fairly fast convergence while also small enough to allow for some finer value fine tuning.
- **Sample size:** Sample size was set to 1000 to allow quicker iterative testing and providing large enough sample size. The data is then randomly split 8:2 for training and validation with
- **Batch size:** the batch size set to 128.
- **Training Loop:** After the data is extracted it is then passed to the training loop for 100 epochs where the average loss for every epoch is recorded, and output images are displayed periodically during the training process.
- **Miscellaneous:** The model is then saved locally on the drive for potential later use. After training has been done the validation loop is used to assess the performance of the model against the validation set.

6.4 Other Methods

Different color conversion methods were used during data optimization process.

7 Experiments and Results

7.1 Baseline Experiments

Using the baseline parameters described below get the following images on the during training.

- **Sample Size:** 25000 images
- **Epochs:** 100
- **Batch Size:** 128
- **Loss Function:** MSE Loss
- **Optimizer:** Adam
- **Learning Rates:** 0.001

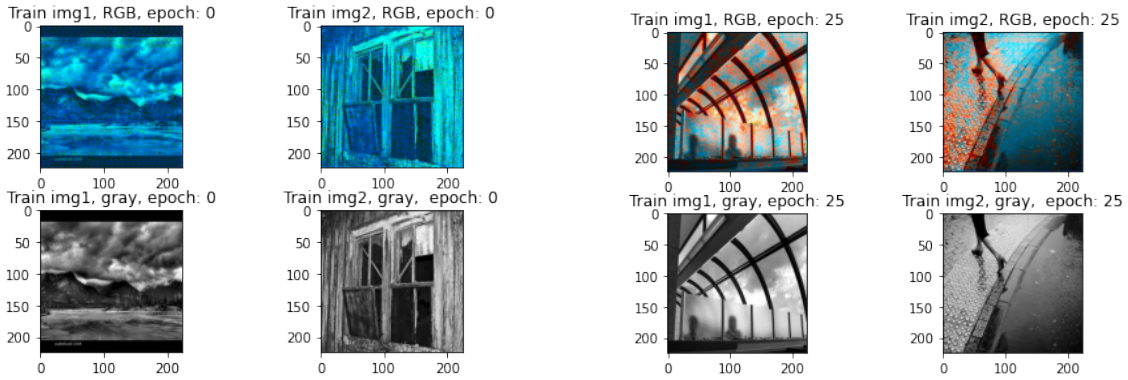

Figure 3: Input and Colorized images at epoch 0 and epoch 25 respectively.

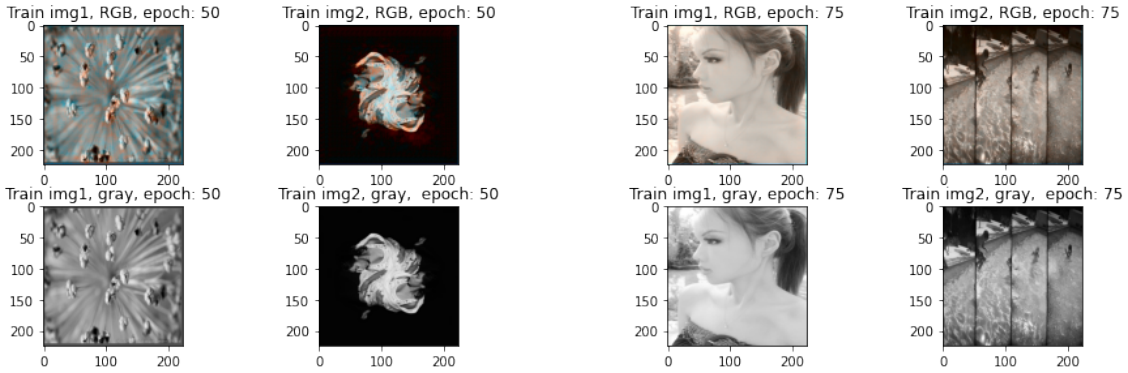

Figure 4: Input and Colorized images at epoch 50 and epoch 75 respectively.

7.1.1 Observation

Predictably the images eventually converge to a somewhat monotone image in the end. This is most likely due to two factors, one was because the learning rate was simply too high and second due to the nature of the loss function, since MSE loss gives us a per pixel average of the color if a range so the average color value is going to be the average of the possible pixel values making.

7.2 Adding More Convolutional Layers

Adjusting the learning rate to 0.0001 and adding a basic RESNET block as additional convolutional layers gives the following results:

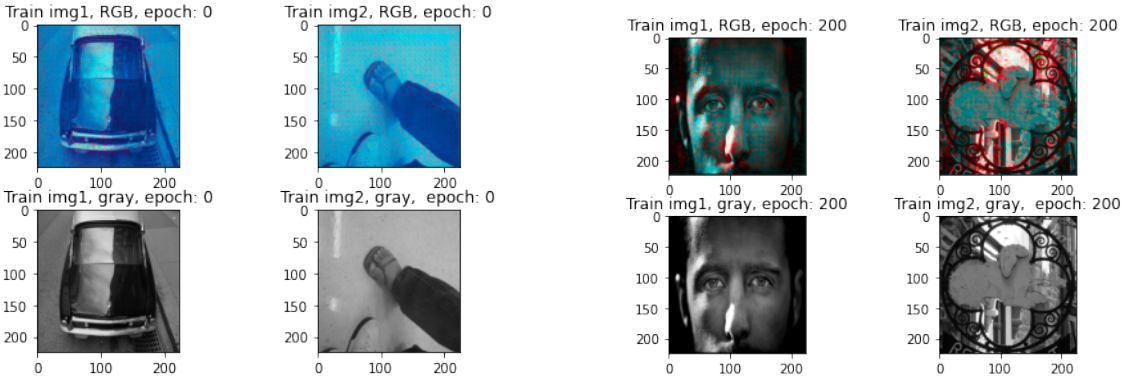

Figure 5: Input and Colorized images at epoch 0 and epoch 200 respectively.

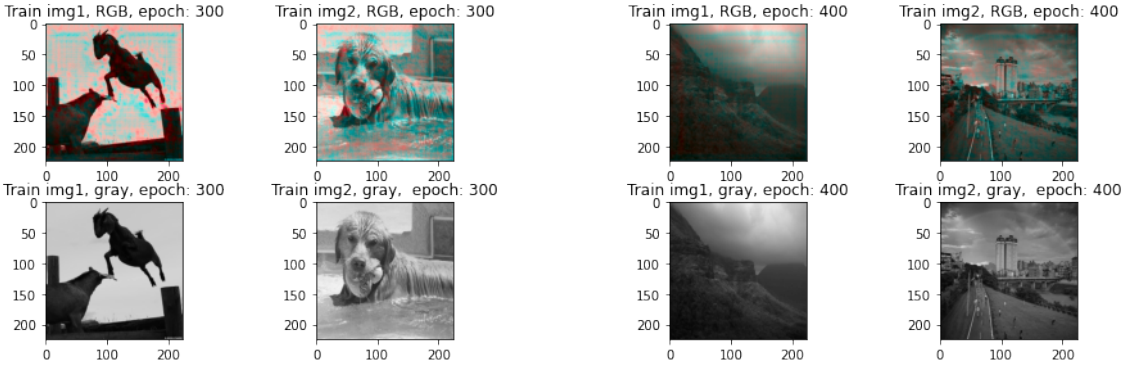

Figure 6: Input and Colorized images at epoch 300 and epoch 400 respectively.

7.2.1 Validation Images

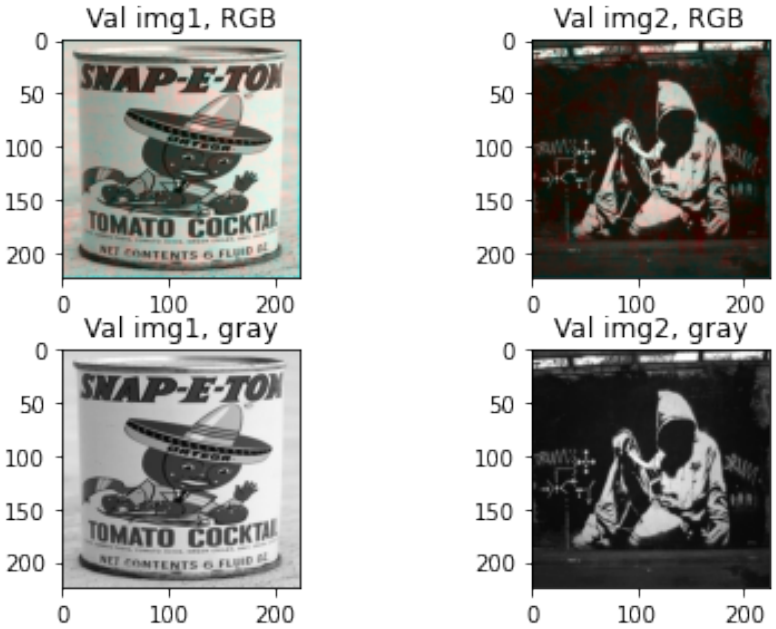

Figure 7: Colorized image generated by the model.

7.2.2 Loss Per Epoch

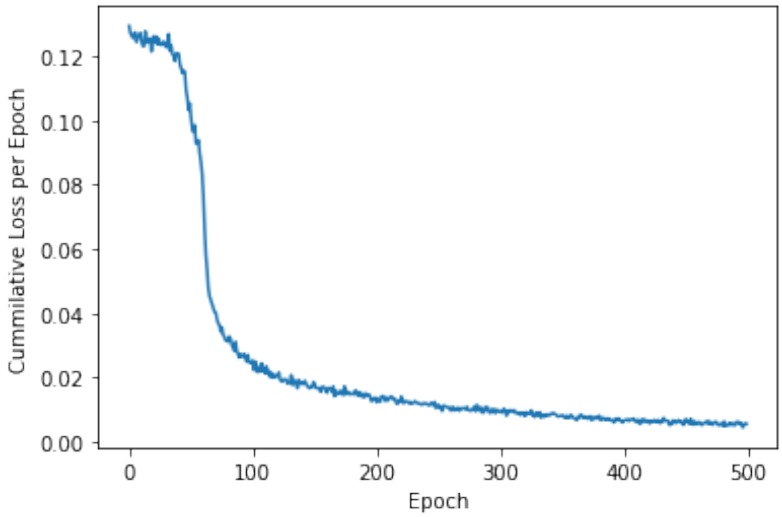

Figure 8: Loss Per Epoch

7.2.3 Observation

Reducing the learning rate as well as adding more unfrozen convolutional layers no longer forces the model to converge to a monotone layer, however the color scheme is still not ideal.

7.3 Learning Rates

In this experiment, we try to modify the learning rate for each training iteration and observe the loss function for the model. We track *Running MSE Loss* across epochs and plot it with its corresponding epoch. The following parameters were used for this experiment:

- **Sample Size:** 1000 images
- **Epochs:** 1000
- **Batch Size:** 128
- **Loss Function:** MSE Loss
- **Optimizer:** Adam
- **Learning Rates:** [0.0005, 0.001, 0.005, 0.01]

The plot below shows the relation between learning rate and epoch:

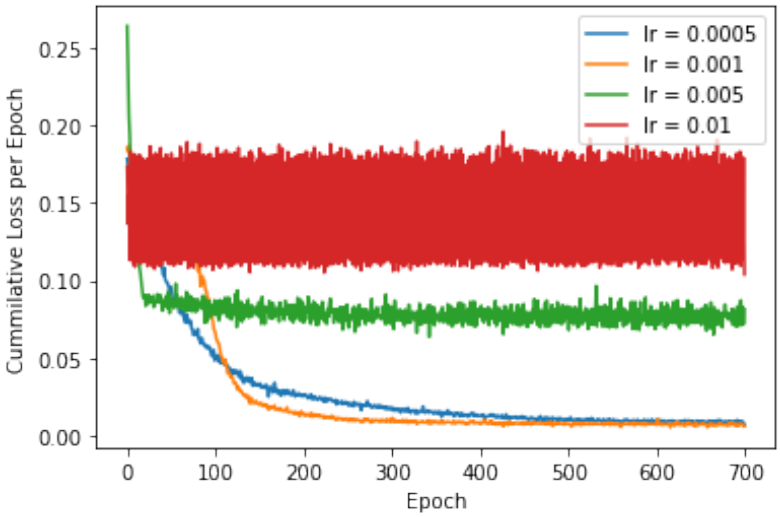

Figure 9: Loss vs Epoch for given set of learning rates.

7.4 Loss Functions

In this experiment, we try to modify the loss function for each training iteration and observe the loss for the model. We track *L1 Loss, Running MSE Loss, Cross Entropy Loss, KL Divergence Loss* across epochs and plot it with its corresponding epoch. The following parameters were used for this experiment:

- **Sample Size:** 1000 images
- **Epochs:** 1000
- **Batch Size:** 128
- **Loss Functions:** [L1 Loss, MSE Loss, Cross Entropy Loss]
- **Optimizer:** Adam
- **Learning Rate:** 0.001

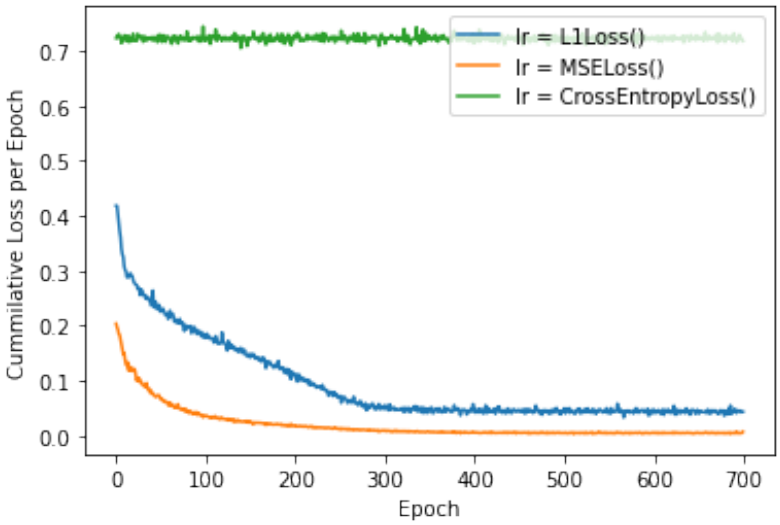

Figure 10: Loss vs Epoch for given set of learning rates.

7.4.1 Validation Images

Figure 11: Images at epoch 75 for L1 Loss, MSE Loss and Cross Entropy Loss respectively.

7.4.2 Observation

We observe from the loss vs epoch plot that the MSE Loss converges the fastest. However we also observe the monotonic nature of images generated using model trained over MSE loss.

8 Conclusion and Discussion

This project presents a supervised machine learning model to colorize grayscale images. We have used autoencoder architecture and followed the transfer learning approach to build and train our model. We used a pretrained RESNET18 model and used embeddings from its semi-last convolutional layer to reconstruct a fully colorized image. From our experiments presented in the sections above, we note that our model starts to perform well on the image dataset.

However, we observe from our experiements that the model has its own drawback given the parameters that we used to train it. For example, we not that the baseline model generates monotonic images. But, adding more unfrozen convolutional layer solves this issue of monotones in the prediction. We also compare the performance of the model at different learning rates and different loss functions.

Based on experiments, we can conclude that our model starts performing well with the tuned hyperparameters and with larger amounts of dataset.

8.1 Future Work

We are mainly interested in comparing other model architectures to understand how different models understand image complexity. We are also interested in understanding if the stuctural complexity of an image determines how well a particular model performs.

It is noted from the literature that GANs provide a lucrative basis of learning structural complexities in images much better than Convolutional Neural Networks. We wish to perform a comparative study of these models and try to understand the larger question to image colorization problems: How do deep networks learn multi-modal features in complex images?

References

[1] Fabio Maria Carlucci, Paolo Russo, and Barbara Caputo. "$(DE)^2CO$: Deep Depth Colorization". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2386–2393. DOI: 10.1109/LRA.2018.2812225.

[2] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. "Deep Colorization". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 415–423. DOI: 10.1109/ICCV.2015.55.

[3] Hasan Sheikh Faridul et al. "A Survey of Color Mapping and its Applications." In: *Eurographics (State of the Art Reports)* 3.2 (2014), p. 1.

[4] Richard Zhang, Phillip Isola, and Alexei A. Efros. "Colorful Image Colorization". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 649–666. ISBN: 978-3-319-46487-9.