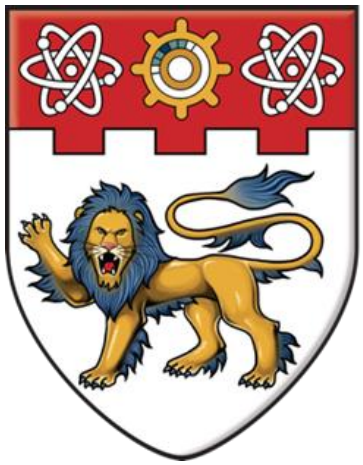


NANYANG TECHNOLOGICAL UNIVERSITY

SC4001/CE4042/CZ4042: Neural Network and Deep Learning

Assignment 2

Report

Title: Analysis of Deep Image Colourization Techniques

Name	Matric Number
Ang Boon Leng	U2022835L
Dhanyamraju Harsh Rao	U2023045C
Parashar Kshitij	U2023805J

Table of Contents

1. Introduction

1.1 Problem & Scope

Image colorization is the process of adding colors to greyscale images so that they look aesthetic and realistic when compared to the actual images. We focus on the subbranch of automatic image colorization, which is to colorize greyscale images without any prompt or guidance.

The difficulty lies in the limited context of a greyscale input. The neural network must learn natural and expected colors of nature, humans, buildings, etc. To be exact, the neural network will learn the traits that link greyscale images with colored ones. Such a task is significant for restoring historic images or improving medical images [1].

1.2 Color Spaces

RGB color space is a common representation of colors in digital images. It is an additive color model consisting of three-color channels, red (R), green (G), and blue (B). Each pixel of an image is a combination of the three-color channels with integer values ranging from 0 to 255.

CIELAB color space, also known as just LAB, is an alternative color representation based on the opponent color model of human vision. It is a perceptually uniform space that consists of three channels: one channel for luminance (L*) with values 0 to 100, two channels for chrominance, namely green-red opposite colors (a*), and yellow-blue opposite colors (b*), both with values -128 to 127. (From this point on, we will refer each channel as L, A, and B for clarity.)

LAB is often a more popular choice for computer vision tasks as compared to RGB because it mimics how human eyes perceive colors in natural images, allowing for more accurate results that capture subtle color variations and shading.

2. Default Model

The default model represents the starting point for all the experiments on the model architecture and is the model architecture for all experiments on color formats and loss functions. It is based on the Deep Koalarization Architecture [3]. The model will return the A and B channels of a LAB image as the output unless changed by the experiment.

2.1 Data Preparation

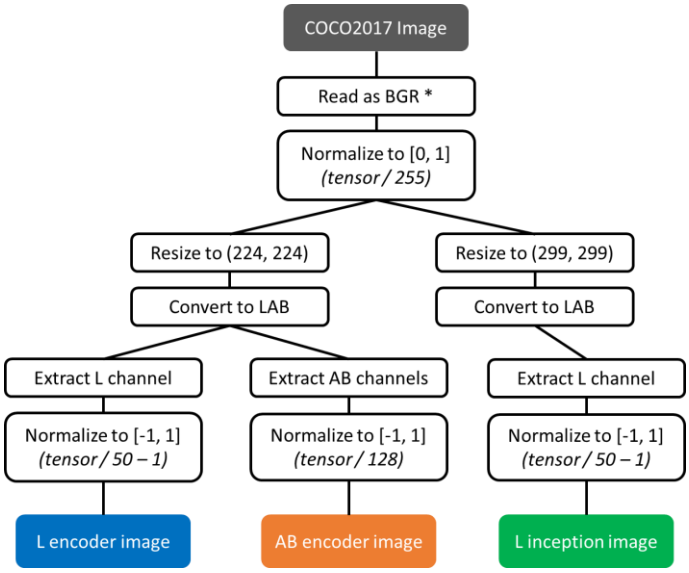
Common Objects in Context (COCO) dataset is a popular dataset for computer vision tasks such as object detection, segmentation, and captioning [4]. We decided to use this dataset for image colorization task due to its diverse range of real-world images with different scenes, objects, and lighting conditions. This allows the model to learn a wide spectrum of natural colors in the real-world. COCO dataset is also a widely used benchmark for comparing computer vision models.

We used the COCO 2017 dataset from Kaggle [5] for the Train and Test image set, each consisting of around 118,000 and 41,000 images respectively. We then performed pre-processing to downsize the dataset by reducing the width and height to 256 for all images.

Before training and testing, for every image in the COCO2017 dataset, three channels of image data will be prepared in a CustomDataset and loaded into a DataLoader.

1. **L encoder image** – the L channel of the image in LAB format, used as the first input for the default model to predict A and B channels.
2. **AB encoder image** – the A and B channels of the image in LAB format, used as the target in loss function.
3. **L inception image** – the L channel of the image in LAB format, used as input for the Inception-v3 model to obtain image embeddings, which then becomes the second input for the default model.

Here are the steps taken to process the three channels of image data.

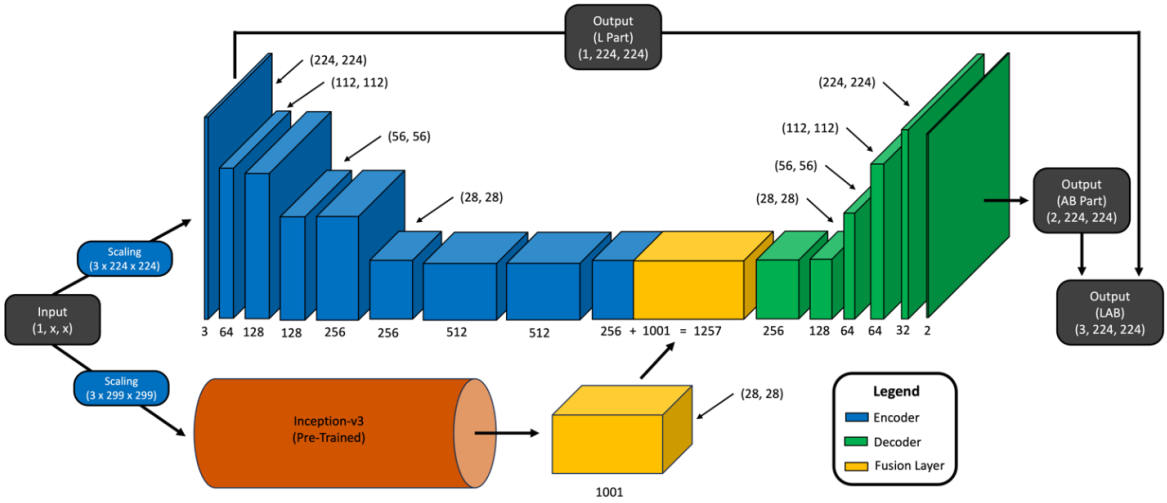


**Note: opencv-python library reads images as BGR format under the hood.*

During post-processing, the AB output of the test dataset will be denormalized and combined with the original L channel to form a complete LAB image. All colored images will be converted to RGB format for visualization.

2.2 Default Architecture

The default architecture is a slightly modified version of the Deep Koalarizer Architecture [3]. It is based on the implementation by Raju et al. [6]. Given below is the architecture diagram of the default architecture.



This model consists of 3 stages: Encoder (in blue), Fusion Layer (in yellow), and Decoder (in green). The encoder accepts an input of size (3, 224, 224) while the pre trained Inception-v3 [7] model accepts a size of (3, 299, 299). Since the input image has only one channel, the values are duplicated to form the other 2 channels. The first layer in the network is the encoder which takes in an input of size (3, 224, 224) and outputs a size of (256, 28, 28). The encoder aims to capture certain features in the grey image that can be used by subsequent layers to infer the color. Each element of the encoder consists of a 2D convolutional layer followed by 2D Batch Normalization and a Rectified Linear (ReLU) activation function. The fusion layer takes the Inception output of size (1001, 28, 28) and concatenates it with the output of the encoder layer of size (256, 28, 28). This will create 1257 filters of size (28, 28) which are fed into the decoder layer. The decoder layer now tries to infer the color of the image using the filters provided by the encoder and the fusion model. Each element of the decoder layer consists of a 2D convolutional layer followed by

2D Batch Normalization and a Rectified Linear (ReLU) activation function. An up-sample function is also used to double the resolution of the filters. The decoder outputs the A and B channels of the LAB output which are then merged with the input image (L channel) to form the final LAB output image.

2.3 Hyperparameters

All the subsequent models in the report were trained for 20 epochs. The average training time was 11 hours. The training was performed on the GPU cluster provided the School of Computer Science and Engineering and Nanyang Technological University, using a single GPU with a wall time of 6 hours.

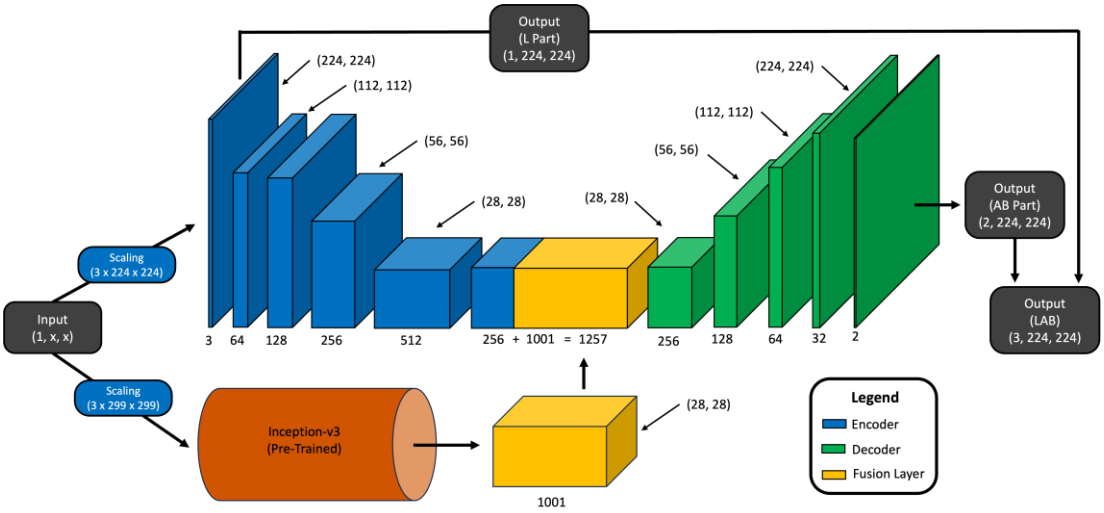
- Batch Size: 32
- Loss Function: Mean Square Error (MSE)
- Learning Rate: 10^{-3}
- Optimizer: Adam
- Scheduler: Reduce LR On Plateau

3. Experiment 1: Architecture

The purpose of this experiment is to propose new architectures which have different features from the default architecture and observe the effects of these changes on the final output. Other than the architecture itself all other parameters will be kept the same to ensure that any changes in the output are caused by the architecture itself.

3.1. Small Encoder Small Decoder (SESD)

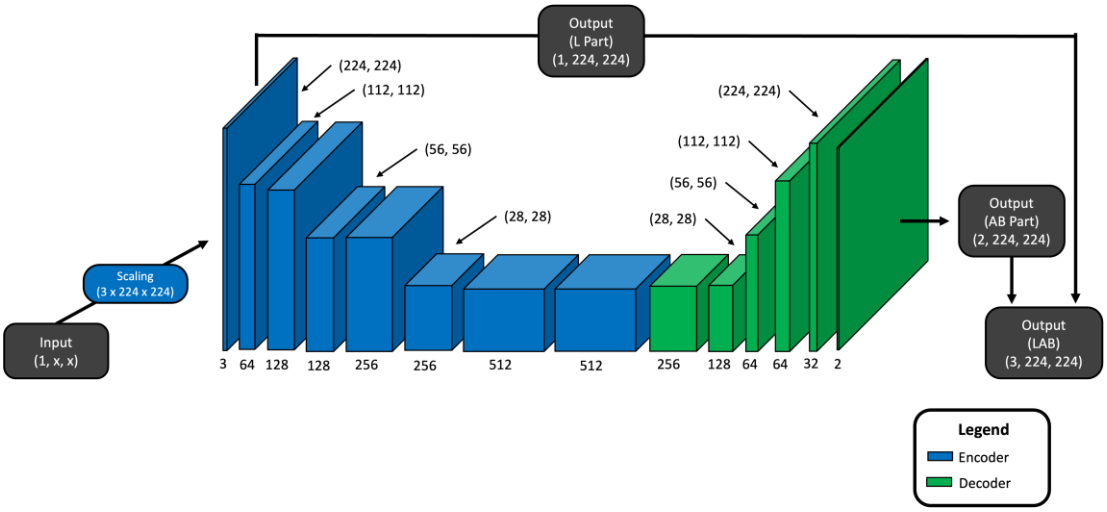
This architecture reduces the number of layers in the encoder and decoder of the default architecture.



In the default encoder 2 different layers are used to reduce size and increase channels, one which first reduces the size ((128, 112, 112) → (128, 56, 56)) and one which increases the channels ((128, 56, 56) → (256, 56, 56)). The SESD architecture does so in once step ((128, 112, 112) → (256, 56, 56)). The same also happens in the decoder just with an increase of size instead. This aims to reduce the number of parameters being trained which should help the model produce stronger colors in fewer epochs.

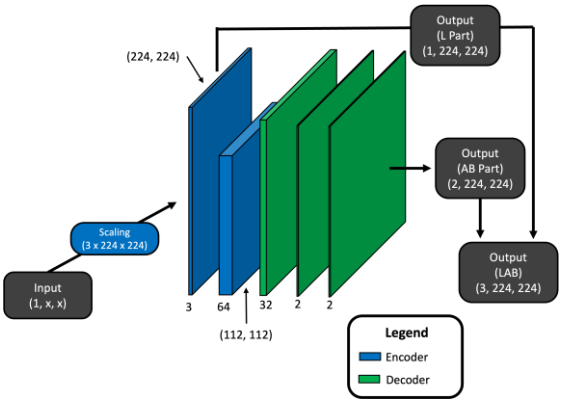
3.2 No Fusion

The No Fusion is as the name suggests just the Default Architecture without the Fusion layer. By comparing the results of no fusion with the default architecture, we aim to understand the role of Fusion layer in the final result.



3.3 Tiniest

As the name suggests, this architecture is significantly smaller than the rest. It consists of just one encoder layer and two decoder layers. By analyzing the results of this architecture, we aim to understand the limits of such types of architectures for the Image colorization task.

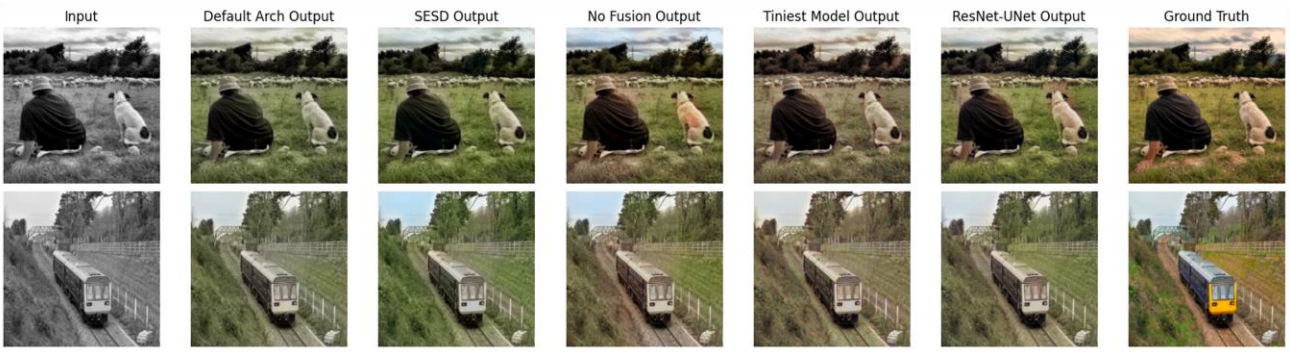


3.4 ResNet-UNet

U-Net is an Image-to-Image Architecture originally intended for Semantic Segmentation of Biological Images [8]. Recently U-Net has received some attention for the Image Colorization task [9]. By studying the results of this architecture, we aim to understand the impact of direct cross connections between the encoder and decoder. We use an implementation of U-Net that uses ResNet layers to make the code simpler [10],hence the name ResNet-UNet.

3.5 Results & Discussion

In this section we compare the results of the different model architectures.



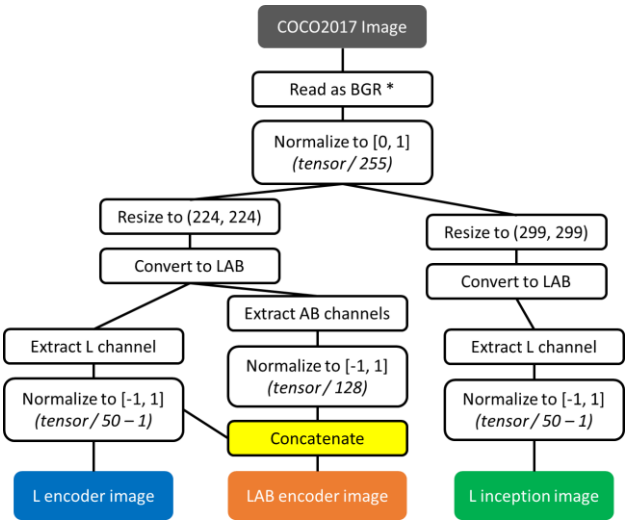
Here, we observe that SESD has slightly stronger, i.e., higher saturation, colors than the Default Architecture. This may indicate that smaller architectures are able to produce strong colors faster than their larger counterparts. The SESD output's sky also has a blue tinge, which the Default Arch Output does not. The No Fusion seems to have a larger variety of colors as compared to Default and SESD. However, it seems that it is having trouble understanding the exact boundary between the different colors. This behavior is observed even on the ResNet-UNet Output which also doesn't have a fusion layer. In fact, the effect of the wrong boundaries seems to be exacerbated in ResNet-UNet with both the man and dog having larger halos around their bodies (More on this in Appendix 2). The Tiniest model seems to be struggling with the task in general. In both cases it seems to color the entire image with average color, as the entire image seems to be of a different darkness from the same shade (The model only returns the A & B channel of LAB, which control shade, while the L channel controlling darkness comes from the input).

4. Experiment 2: Output Color Space

In this section, the output color space of the default model will be modified, that is, to output images in LAB or RGB format instead of just AB channels. As usual, all other parameters will be kept constant.

4.1 Output as LAB Space

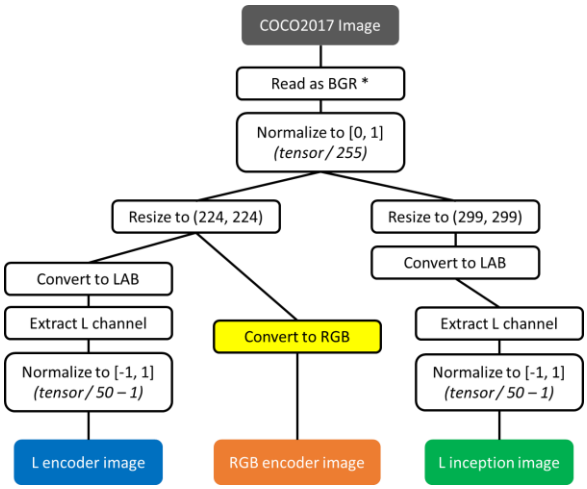
To facilitate this experiment, changes were made to the architecture and the data processing. Firstly, the last layer of the decoder was modified to have 3 out channels for L, A, and B channels instead of only 2 previously. Besides, the data processing was also modified to prepare the LAB encoder image as the target instead of AB encoder image. The new process is as follows:



**Note: opencv-python library reads images as BGR format under the hood.*

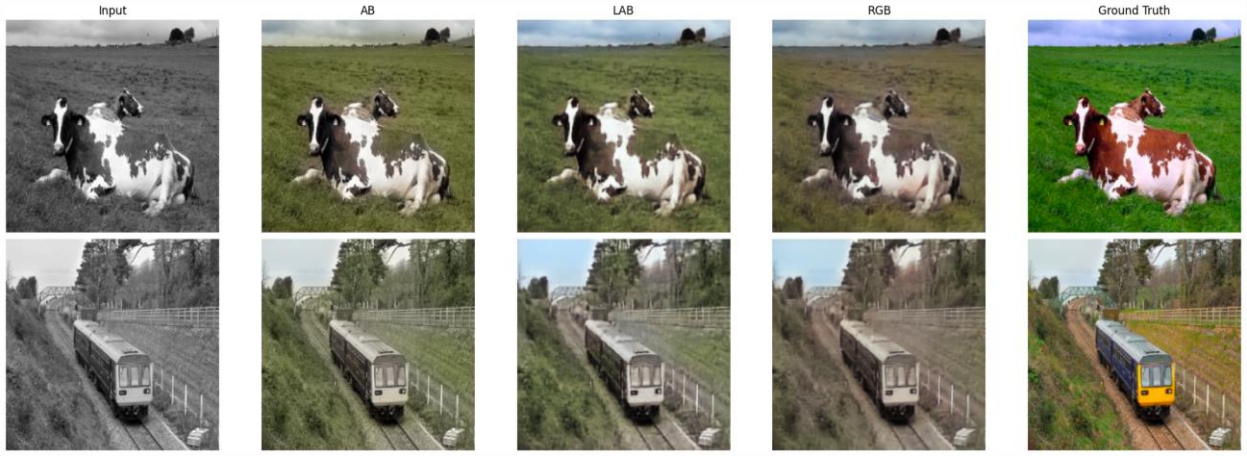
4.2 Output as RGB Space

To facilitate this experiment, changes were made to the architecture and the data processing. Firstly, the last layer of the decoder was modified to have 3 out channels for R, G, and B channels instead of only 2 previously. The final activation function was also modified to be Sigmoidal instead of Tanh, as we expect the predicted RGB values to be [0, 1]. Besides, the data processing was also modified to prepare the RGB encoder image as the target instead of AB encoder image. The new process is as follows:



**Note: opencv-python library reads images as BGR format under the hood.*

4.3 Results and Discussion



Reminder: The AB image is the output from the default model, which is combined with the L channel of the original (ground truth) image for visualization.

In terms of saturation, the LAB image is observed to have the highest saturation, followed by AB and RGB. This is evident in the contrast between the greener hue of the grass and the bluer hue of the sky in the LAB image. RGB model may have performed the worst due to it having to learn to predict three color channels as opposed to two color channels in AB and LAB models.

In terms of sharpness, the AB image is observed to have the sharpest image, followed by LAB and RGB. This is evident by the increasing blurriness of the image from AB image to LAB and RGB. The L channel of the LAB color space provides luminance and thus clarity to an image. In AB model, the L channel is retained from the original image, and hence has the closest color intensity to the original image. In LAB model, the image also has an L channel

but is now an output variable that could have deviated from the original L channel values. In RGB model, there is no luminance channel and hence this information is easily lost during training, resulting in a blurry image.

In terms of object boundaries, it seems that all three models have trouble distinguishing exact boundaries between the different colors. This is evident in the cow image, where some brown spots of the cows appear to merge with the grass in the background. In fact, the effect of the wrong boundaries seems to impact RGB the most, with both the cows and train having a halo around them.

5. Experiment 3: Loss Functions

In this section, the MSE loss function used in the training of the default model will be modified to more sophisticated ones. As usual, all other parameters will be kept constant.

5.1 Mean Square Error (MSE)

It is one of the simplest and most common loss functions. For an image X of size $H \times W$, the MSE is given by the equation:

$$MSE = \frac{1}{H \times W} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} [\bar{X}_{i,j} - X_{i,j}]$$

MSE will never be negative since we are squaring the error. MSE is great for ensuring that the trained model has no outlier predictions with huge errors since MSE puts larger weight on these errors due the squaring. The disadvantage of using MSE is that if the model makes a single very bad prediction, the squaring part of the function magnifies the error. In the context of image colorization, the purpose of MSE loss function is to distinguish the objective quality between generated image and the ground truth.

5.2 Learned Perceptual Image Patch Similarity (LPIPS)

This metric has gained popularity in areas such as frame interpolation and measuring the similarity between features of two images extracted from a pretrained network. It is a perceptual metric that quantifies the human perceived similarity between two images. Unlike traditional metrics such as PSNR (Peak Signal to Noise Ratio) and SSIM (Structural Similarity Index Measure) [11] [12], which calculate differences based on raw pixel values or simple transformations thereof, LPIPS leverages deep learning to better align with human visual perception. It uses the distance between features extracted by a convolutional neural network (CNN) pretrained on an image classification task as a perceptual metric. A low LPIPS score means that images are perceptual similar. The input and output images should be in RBG format.

Since LPIPS accepts image in RGB format, we used the modified output color space model outlined in Section 4.2 to generate colored images in RGB format. We used the implementation of LPIPS from TorchMetrics [12]. We specified the backbone network type to be VGG which is computationally more expensive but achieves a higher accuracy in capturing complex image features as compared to SqueezeNet and AlexNet. These are the parameters specified:

- net_type = ‘vgg’
- reduction = ‘mean’
- normalize = True

5.3 Structural Similarity Index Measure (SSIM)

This metric quantifies image quality degradation caused by processing such as data compression or by losses in data transmission. It measures the perceptual difference between 2 similar images.

SSIM separates out similarity measurements into three different components [13]:

- Luminance: $l(x,y) = \frac{2\mu_x\mu_y+c_1}{\mu_x^2+\mu_y^2+c_1}$
- Contrast: $c(x,y) = \frac{2\sigma_x\sigma_y+c_2}{\sigma_x^2+\sigma_y^2+c_2}$
- Structural: $s(x,y) = \frac{\sigma_{xy}+c_3}{\sigma_x\sigma_y+c_3}$
- μ_x is the pixel sample mean of x; μ_y is the pixel sample mean of y.
- σ_x^2 is the variance of x; σ_y^2 is the variance of y.
- σ_{xy} is the covariance of x and y.
- $c_1 = (k_1L)^2$; $c_2 = (k_2L)^2$; $c_3 = c_2/2$; These constants help to stabilize the division with weak denominator.
- L is the dynamic range of pixel values (typically, $L = 2^{\#\text{bits per pixel}}$ -1 and $k_1 = 0.01,\ k_2 = 0.03$)

SSIM is then a weighted combination of these comparative measures:

$$SSIM(x,y) = l(x,y)^{\alpha}.c(x,y)^{\beta}.s(x,y)^{\gamma}$$

Setting the weights α, β, γ as 1, SSIM can be defined as:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_3)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Since SSIM has an optimal value of 1 with a range of [-1, 1], we defined the loss function as loss(x, y) = 1 – SSIM(x, y). This makes the optimal value is zero and the range as [0, 2].

5.4 Multi-Scale SSIM

This metric is a generalization of Structural Similarity Index Measure by incorporating image details at different resolution scores [13]. The sampling density of image signal, distance from image plane to observer and the perceptual capability of the observer can change the perceivability of the image. It is shown by the formula:

$$MSSSIM(x,y) = [l_M(x,y)]^{\alpha_M}.\prod_{j=1}^{M}[c_j(x,y)]^{\beta_j}[s_j(x,y)]^{\gamma_j}$$

The original image is indexed as scale 1 and the highest scale is M. Luminance is computed only at scale M while contrast and structure are compared at j-th scale. Like SSIM, Since MSSSIM has an optimal value of 1 with a range of [-1, 1], we defined the loss function as loss(x, y) = 1 – MSSSIM(x, y). This makes the optimal value is zero and the range as [0, 2].

5.5 Results & Discussion

SSIM generates images with highest color saturation. However, it has a very high tendency to “leak” colors across boundaries, resulting in halo effects that are spread widely. Perhaps the model is too sensitive to boundaries and forms unnatural boundaries.

Multi-Scale SSIM has the worst performance as it generates uniform color shades across the entire image.

LPIPS has the most verdant shade of green because it is designed to capture the human perception of image quality better than traditional metrics such as SSIM, MSE, and Multi-scale SSIM. It performs exceedingly well in terms of moderate color saturation and clear distinction between subjects and backgrounds. This is because LPIPS is sensitive to structural and semantic information in images, which are often enhanced by color saturation and distinction between subjects and background. It is robust to the variations of luminance and contrast. This metric has been calibrated on a large dataset of human-judged perceptual similarity, which may reflect the preference of human observers.

6. Conclusion

In this project, we tackled the computer vision task of automatic image colorization. We made modifications to the Deep Koalarizer architecture (Encoder-Fusion-Decoder) in three main aspects: model architecture, output color space, and loss function. The model was trained and tested on COCO2017 Dataset. The performance of each modified model of the same aspect is then compared and evaluated in terms of various metrics including color saturation, luminance, sharpness, object boundary, and more.

In terms of model architecture, our results show that the SESD architecture has the highest color saturation and handles color boundaries well. No Fusion and ResNet-UNet architectures exhibit halo effects around subjects in the image. Tiniest Model performs the worst as it generally assigns uniform shades of the same color across the image.

In terms of output color space, our results show that LAB format has the best color saturation but loses a little image sharpness. RGB format performs poorly in both color saturation and image sharpness as well as creates a slight halo effect. Both models had trouble distinguishing the color boundaries between subjects and backgrounds.

In terms of loss function, our results show that LPIPS performs exceedingly well in terms of moderate color saturation and clear distinction between subjects and backgrounds. SSIM model generates images with highest color saturation and contrast among objects, but it creates widespread halo effects around subjects. MultiSSIM has the worst performance as it generates uniform color shades across the entire image.

Overall, the three optimal modifications: SESD architecture, LAB output, and LPIPS loss function individually managed to outperform the default model in color saturation and boundaries.

7. Possible Future Work

Given the remarkable progress we observed while using LPIPS, expanding on the use of deep learning models as a loss function may yield better results in the future. Models like ERGAS [13] [14], SDI, SAM, etc., could serve the same purpose. Generative Adversarial Networks (GANs) may also lead to better images. All the architectures discussed in this report can be used as effective generators in GANs.

Due to a constraint of time and computational resources, we were unable to try the different architectures using the 2 best loss functions in training. Instead, we had to rely on MSE for the comparison. Analyzing the different architectures in Section 3 with the best loss functions from Section 5 could be an area of expansion in the future.

Deep Image Colorization models often find it difficult to color objects with uncertain colors. Grass is green, the sky is blue, but a train could be white, yellow, blue, grey, or a combination of them. This may cause confusion in the model which may result in such objects appearing in grey in the final output. This was observed in the case of the train moving on tracks in a small depression. In such a situation providing some context to the model may be beneficial. This could either be through a text query or a context image.

Another area for future research is the potential use of pretrained image generation models like DALL-E and Stable Diffusion, in the place of the Inception v3 model here.

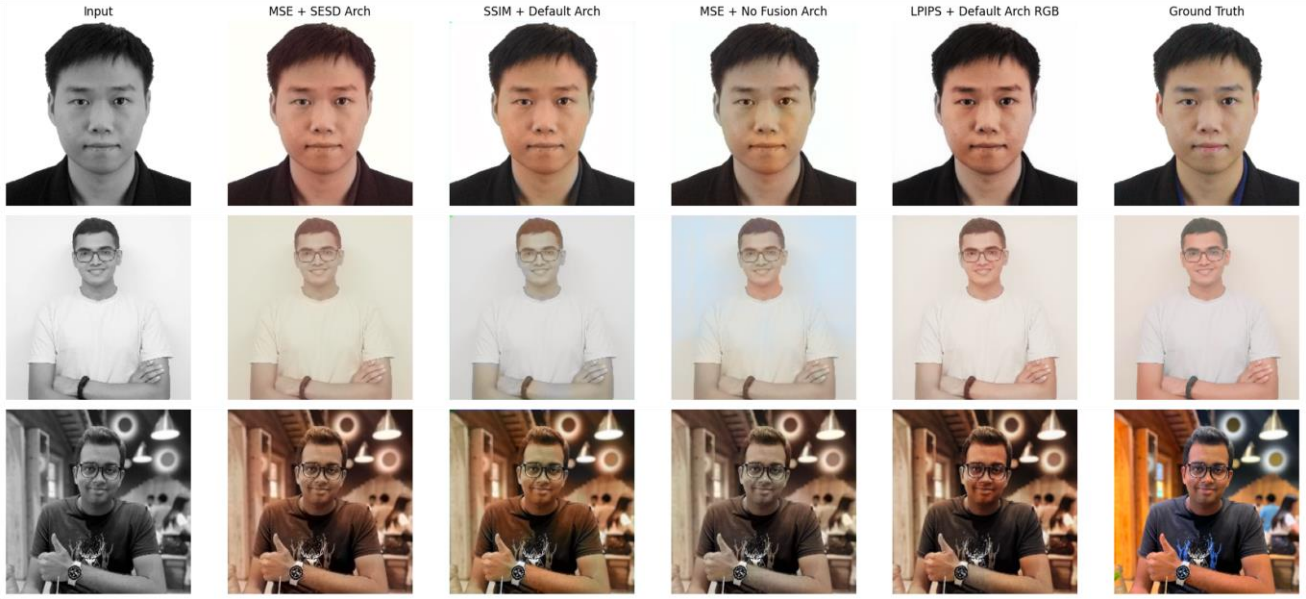
8. Appendix

8.1 Situation-based Testing

The purpose of this section is to test our models on specific types of images. Most of these images have been captured by the researchers for this very purpose. Here, we take a combination of the different architecture in Section 3 training using MSE Loss and the loss functions in Section 5 trained using the Default Architecture.

8.1.1 Portraits

Portraits are images focused on a single person. Here, we observe the performance of different models on portraits with 3 different backgrounds: pure white, off white, and compound.



While all models seem to perform well on the pure white background, that performance soon degrades on the off white and compound background. LPIPS + Default Arch RGB seems to be the only exception to this trend as it performs equally well across all three backgrounds, giving the most realistic skin tones of the group.

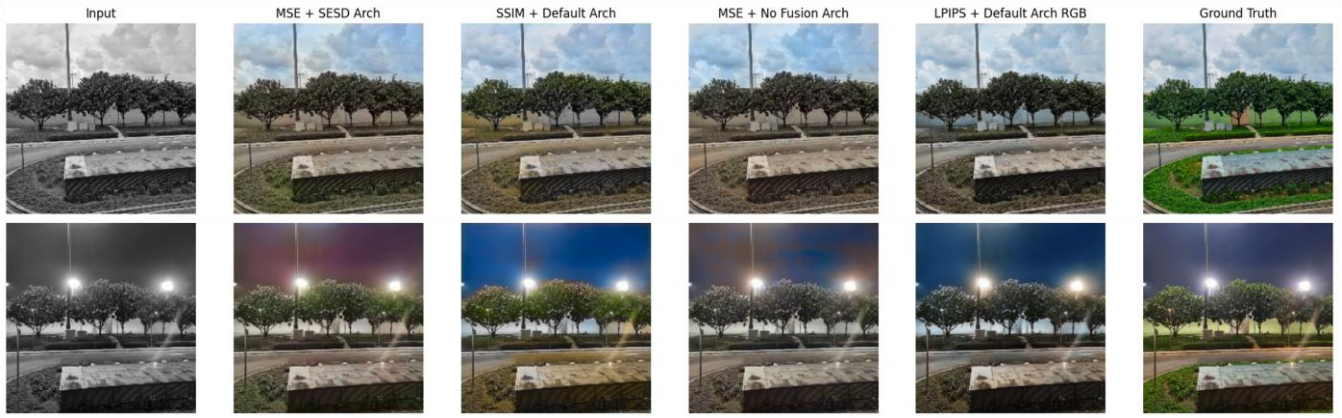
8.1.2 Nature

In these photos the focus is a group of trees. The images here are dominated by green and don't display much of a range of colors.

MSE + SESD Arch performs the best with accurate colors for the sky, trees, and grass. SSIM + Default Arch seems to have trouble maintaining a green hue of the grass in areas with and without shadows. MSE + No Fusion performs the worst with its brown shade of nature.

8.1.3 Day vs Night

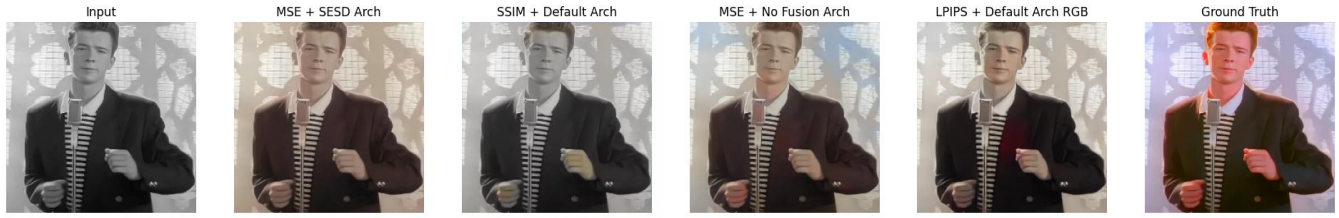
Here, we compare the performance of the models for the same picture when one was taken during the day, and one was taken at night.



It is interesting that both MSE + SESD and SSIM + Default Arch seem to have a higher saturation (stronger colors) for the trees at night. Each of the model architectures also seem to have a distinctly colored sky.

8.1.4 Heavily Edited

Here, we look at images that have been heavily edited. We aim to understand if the model will color the image to the edited version or to something more normal.



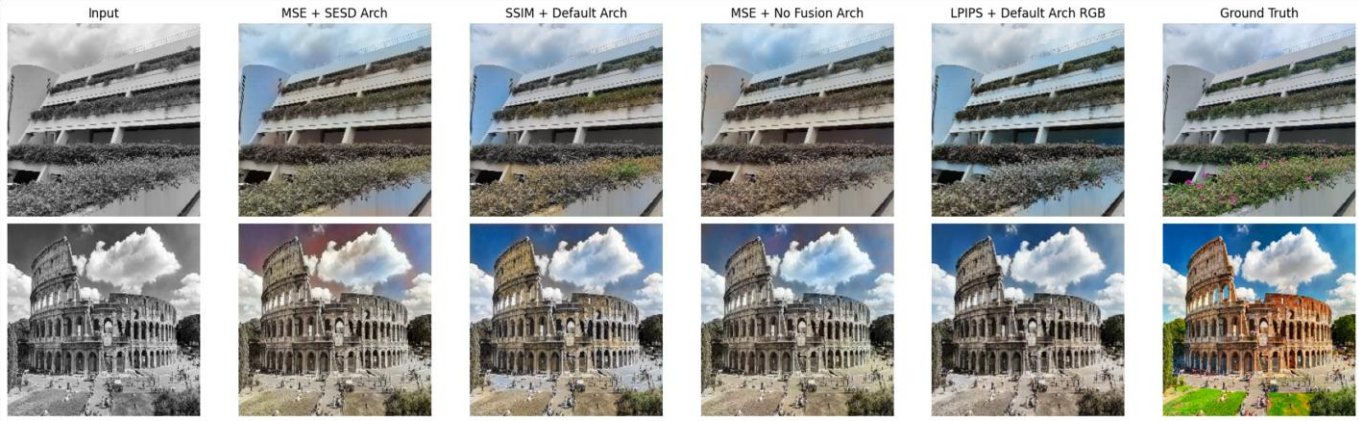
Apart from SSIM + Default Arch, all models seem to colorize the image as if it's a normal image instead of the heavily edited ground truth.

8.1.5 Food

Here, MSE + No Fusion Arch seems to fail miserably. Meanwhile, the other architectures seem to a good job, with LPIPS + Default Arch RGB doing especially well.

8.1.6 Buildings

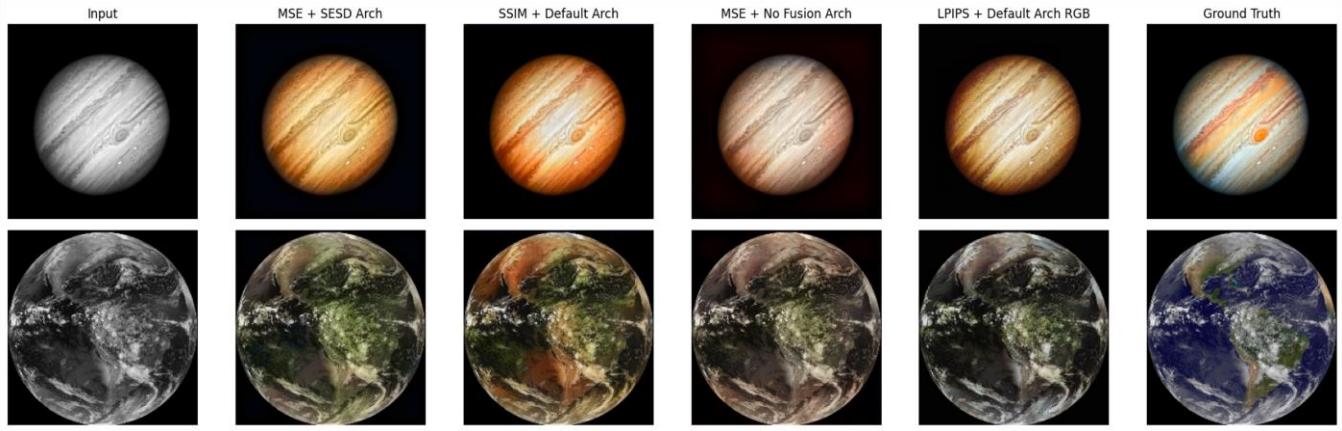
Buildings are difficult for colorization models to deal with as they often lack context and could be of any color.



While still a decent distance away from the Ground Truth all the models seem to be able to reasonably infer the color of the building. However, this uncertainty seems to affect the color of the plants in first image and the lawn in the second image as well.

8.1.7 Space

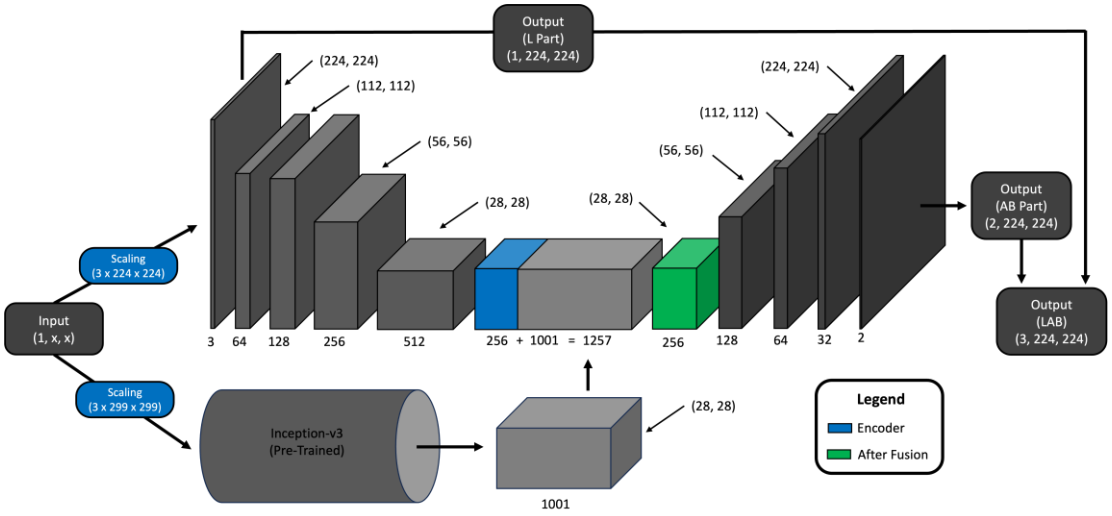
In this section we aim to use space images to test the out of domain (OOD) generalization capabilities of the model. There is not a single image from space in the training data, so the model will be dealing with completely unknown objects.



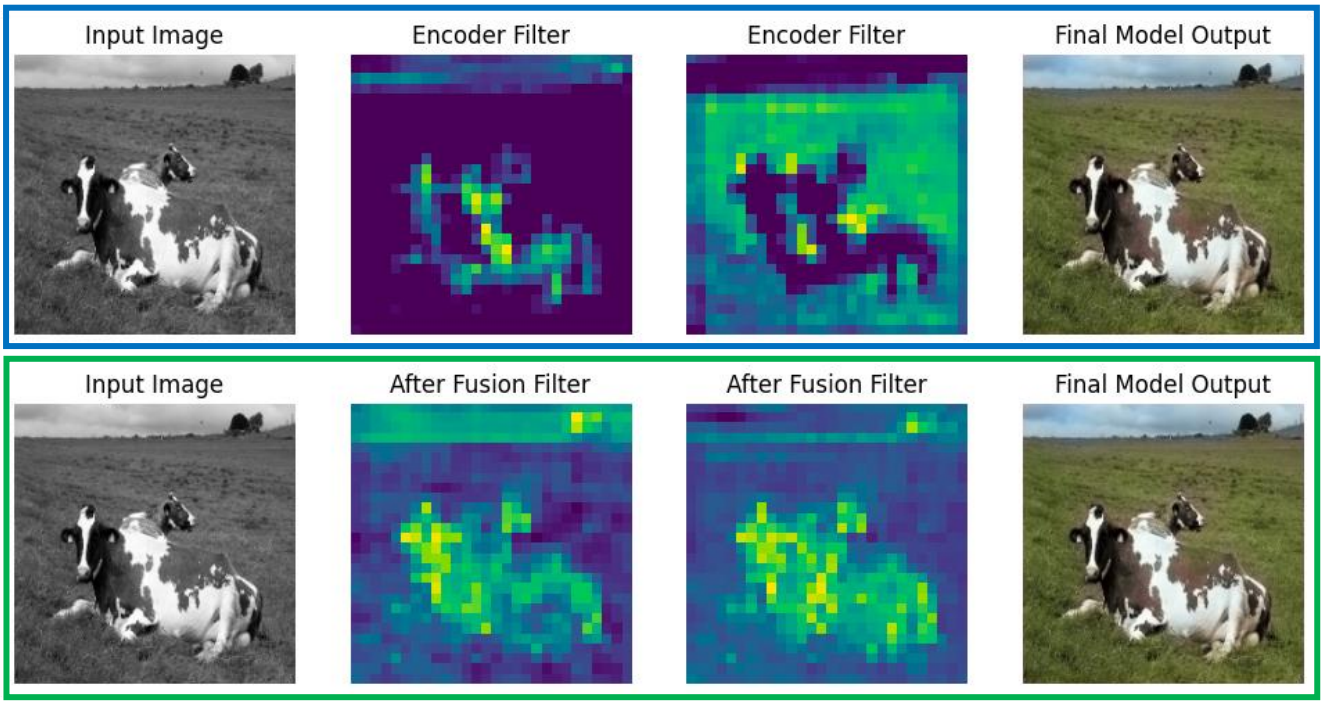
All the models fail to predict the accurate colors for Earth but make descent guesses for Jupiter. This shows that while the model can make out the approximate color from the grey image, it still needs some context to better estimate the exact shade of the color needed.

8.2 Filters

The purpose of this section is to understand how the model produces its outputs. For this part we will be using the SESD Architecture trained using MSE as a loss function.



Here we examine the filters at 2 locations: After the encoder (in blue) and after the fusion layer (in green).



In the first Encoder filter and After Fusion filter, the encoder filter manages to distinguish the white patches of the cow from the surroundings although no distinction is made between the darker cow patches and the grass. This may help explain why the No Fusion and ResNet-UNet architectures had a halo effect around the main subjects in their images.

In the second Encoder filter and After Fusion filter, the encoder manages to identify the grass and dark cow patches together, essentially making it like a negative of the first encoder filter. On the other hand, the second After Fusion filter continues to distinguish between the cow, the grass, and the sky. It can identify white areas with better accuracy on the cow but fails to identify the clear difference between the grass and sky unlike the first After Fusion filter.

9. References

[1] V. Basu, "Image Colorization basic implementation with CNN," Kaggle, [Online]. Available: https://www.kaggle.com/code/basu369victor/image-colorization-basic-implementation-with-cnn. [Accessed 2023].

[2] Wikipedia, "CIELAB color space," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/CIELAB_color_space. [Accessed 2023].

[3] F. Baldassarre, D. G. Morín and L. Rodés-Guirao, "Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2," 9 Decemeber 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1712.03400. [Accessed 2023].

[4] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, "Microsoft COCO: Common Objects in Context," 21 February 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1405.0312. [Accessed 2023].

[5] Awsaf, "COCO 2017 Dataset," [Online]. Available: https://www.kaggle.com/datasets/awsaf49/coco-2017-dataset.

[6] A. H. Raju and A. Ravishankar, "PyTorch Implementation of Deep Colorization," [Online]. Available: https://github.com/ahemaesh/Deep-Image-Colorization/blob/master/Colorization_with_only_encoder_and_decoder.ipynb.

[7] PyTorch, "INCEPTION_V3," [Online]. Available: https://pytorch.org/hub/pytorch_vision_inception_v3/.

[8] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 18 May 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1505.04597. [Accessed 2023].

[9] C. Ballester, A. Bugeau, Hernan Carrillo, M. Clément, R. Giraud, L. Raad and P. Vitoria, "Analysis of Different Losses for Deep Learning Image Colorization," 18 May 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2204.02980. [Accessed 2023].

[10] N. Usuyama and K. Chahal, "UNet/FCN PyTorch," [Online]. Available: https://github.com/usuyama/pytorch-unet.

[11] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," 2004. [Online]. Available: https://www.cns.nyu.edu/pub/lcv/wang03-preprint.pdf. [Accessed 2023].

[12] Lightning AI, "TorchMetrics," [Online]. Available: https://lightning.ai/docs/torchmetrics/stable/. [Accessed 2023].

[13] Z. Wang, E. P. Simoncelli and A. C. Bovik, "MULTI-SCALE STRUCTURAL SIMILARITY FOR IMAGE QUALITY ASSESSMENT," 2003. [Online]. Available: https://ece.uwaterloo.ca/~z70wang/publications/msssim.pdf. [Accessed 2023].

[14] Q. Du, N. H. Younan, R. King and V. P. Shah, "On the Performance Evaluation of Pan-Sharpening Techniques," 2007. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4317530. [Accessed 2023].