# Colorization of Grayscale Images

Buddhika Ariyarathna
*Department of Computer Engineering*
*University of Peradeniya*
Sri Lanka
e20024@eng.pdn.ac.lk

Kavindu Methpura
*Department of Computer Engineering*
*University of Peradeniya*
Sri Lanka
e20254@eng.pdn.ac.lk

Amanda Senavirathna
*Department of Computer Engineering*
*University of Peradeniya*
Sri Lanka
e20366@eng.pdn.ac.lk

Bhagya Siriwardane
*Department of Computer Engineering*
*University of Peradeniya*
Sri Lanka
e20378@eng.pdn.ac.lk

Janith Wanasinghe
*Department of Computer Engineering*
*University of Peradeniya*
Sri Lanka
e20420@eng.pdn.ac.lk

***Abstract*—Colorization of grayscale images is a challenging problem with applications in image restoration, medical imaging, and digital arts. Traditional methods, including spatial and frequency domain transformations, often struggle with seamless color assignment and perceptual consistency. This project integrates classical image processing techniques with deep learning to enhance grayscale-to-color image translation.**

**Initially, intensity-based segmentation, Fourier domain processing, and heuristic color mapping were implemented. However, these approaches had limitations in structural detail preservation and natural color transitions. To improve performance, a hybrid Convolutional Neural Network (CNN)-based framework was introduced, constructed by integrating various components from existing methodologies while incorporating custom modifications. The CNN model follows an encoder-decoder architecture inspired by U-Net and autoencoders, utilizing convolutional and transposed convolutional layers. A perceptual loss function based on VGG networks was incorporated to enhance high-level feature retention.**

**Experiments during the project compare classical, hybrid, and deep learning-based colorization techniques using qualitative (visual inspection) and quantitative (PSNR, SSIM, and perceptual loss) evaluations in different milestones. Intermediate feature map visualization was implemented to analyze model activations, while dataset preprocessing, including grayscale-to-color transformation, was optimized using PyTorch's DataLoader. The integration of perceptual loss was customized using VGG11 with gradient-free feature extraction, and memory optimization techniques such as batch size tuning and gradient accumulation were applied to address GPU constraints.**

**Results indicate that the developed CNN-based model, built by combining multiple existing techniques with novel enhancements, outperforms purely heuristic methods, achieving improved realism and structural consistency. Additionally, classical preprocessing before CNN inference enhances efficiency and reduces artifacts. This framework provides a scalable grayscale image colorization solution, with applications in historical image restoration and content generation.**

***Index Terms*—Image Colorization, Grayscale-to-Color Conversion, Spatial Domain Processing, Frequency Domain Processing, Fourier Transform, OpenCV Colormap Mapping, Convolutional Neural Networks (CNNs), Deep Learning for Image Processing, Perceptual Loss in Image, Colorization, Automated Image Enhancement**

## I. Introduction

### A. Problem Statement & Motivation

Image colorization plays a crucial role in fields such as historical image restoration, medical imaging, and digital arts. Traditional grayscale-to-color conversion relies on heuristic-based transformations, which often fail to produce realistic results due to their lack of contextual understanding. Automating this process requires advanced techniques that can infer colors meaningfully while preserving structural integrity.

### B. Challenges

- **Color Ambiguity:** Multiple valid colorization possibilities exist for the same grayscale input.
- **Context Awareness:** Classical methods lack object recognition, leading to incorrect color assignments.
- **Computational Complexity:** Deep learning-based approaches require large datasets and significant computational power.

### C. Project Objectives

This project explores a hybrid approach:

1) Implement classical image processing techniques for colorization.
2) Construct a CNN-based deep learning model by integrating multiple existing techniques with custom enhancements.
3) Compare classical and CNN-based methods based on accuracy and efficiency.

## II. Background/Related Work

Previous works on grayscale-to-color image colorization can be categorized into:

- **Traditional Image Processing Methods:** Early methods focused on rule-based techniques such as intensity-based mapping, edge detection, and segmentation [1]. Fourier transform-based frequency domain techniques were also explored for texture enhancement and noise reduction

[2]. However, these methods lack semantic understanding, limiting their effectiveness for complex images.

- **Deep Learning-Based Methods:** With advancements in deep learning, data-driven approaches such as Convolutional Neural Networks (CNNs) have demonstrated superior results in colorization tasks. Zhang et al. [3] introduced an automatic colorization approach using CNNs trained on large-scale datasets. Additionally, generative adversarial networks (GANs) have been used in models like DeOldify [4] and Pix2Pix [5] to improve realism and color consistency.

This project builds on these approaches by integrating classical preprocessing with fine-tuned CNN models for improved performance.

## III. Approach

### A. Classical Techniques Implementation

- **Color Mapping:** OpenCV colormaps, intensity-based mapping, custom color gradients.

*1) Classical Techniques Walkthrough:*

- The dataset was created by applying grayscale transformation to images collected from Google and Kaggle, forming paired grayscale-color images.
- Images were preprocessed using:
  - Edge detection (Sobel, Canny)
  - Smoothing filters (Gaussian, Bilateral)
- Three different colorization methods were applied to the dataset:
  1) **Intensity-Based Color Mapping:** Maps grayscale intensity values to predefined RGB colors using rule-based mapping (if-else conditions).
  2) **OpenCV Color Mapping:** Uses OpenCV's built-in colormaps to assign colors based on intensity. The following code snippets were used:

```
import cv2
cv2.COLORMAP_JET
cv2.COLORMAP_TURBO
cv2.COLORMAP_VIRIDIS
cv2.COLORMAP_HOT
```

  3) **Custom Color Mapping:** Uses custom color gradients. The following syntax defines custom color ranges:

```
import matplotlib.colors
matplotlib.colors.
    LinearSegmentedColormap
```

### B. CNN Model Implementation

- **Constructed Model:** Built by integrating existing CNN components with modifications for improved performance.
- **Modifications:**
  - Custom intermediate feature map visualization.
  - Enhanced dataset sample printing for debugging.
  - Optimized perceptual loss integration using VGG11.
  - Memory-efficient training adjustments (smaller batch sizes, gradient accumulation).
- **Training Data:** Dataset expansion using Kaggle (2000 Data image pairs from Landscapes set) and Google-sourced grayscale-color image pairs.

*1) CNN Model Implementation Walkthrough:*

- The following modules were imported and used for implementation:
  - `torch`: The core library for PyTorch, used for tensor operations and neural networks.
  - `torch.nn`: Provides neural network layers and loss functions.
  - `torch.optim`: Contains optimization algorithms like Adam.
  - `torchvision.transforms`: Used for image preprocessing (e.g., resizing, converting to tensors).
  - `torch.utils.data`: Provides tools for creating datasets and data loaders.
  - `torchmetrics.image.SSIM`: A metric to compute the Structural Similarity Index Measure (SSIM) for image quality.
  - `PIL.Image`: Used for loading and manipulating images.
  - `matplotlib.pyplot`: For visualizing images and results.
  - `os`: For interacting with the file system (e.g., listing files in directories).

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import DataLoader,
    Dataset
from torchmetrics.image import
    StructuralSimilarityIndexMeasure as
    SSIM
from PIL import Image
import matplotlib.pyplot as plt
import os
```

- **Model Definition**
  - `ColorizationNet`: A neural network for colorizing grayscale images.
    - **Encoder**: Converts a grayscale image (1 channel) into a high-dimensional feature representation.
      - Uses convolutional layers (`Conv2d`) with ReLU activations.
      - The image size is reduced using `stride=2`.
    - **Decoder**: Converts the feature representation back into a color image (3 channels).
      - Uses transposed convolutional layers (`ConvTranspose2d`) to upsample the image.
      - The final layer uses a `Sigmoid` activation to ensure output values are in the range `[0, 1]` (valid RGB values).

- **Custom Dataset**
  `GrayscaleToColorDataset`: A custom dataset class for loading grayscale and corresponding color images.
  - **`gray_dir`**: Directory containing grayscale images.
  - **`color_dir`**: Directory containing color images.
  - **`transform`**: Preprocessing steps (e.g., resizing, converting to tensors).
  - **`__getitem__`**: Loads a grayscale image and its corresponding color image, applies transformations, and returns them as tensors.
- **Metrics Definition**
  - **`mse_loss`**: Computes the Mean Squared Error (MSE) between predicted and target images.
  - **`psnr`**: Computes the Peak Signal-to-Noise Ratio (PSNR), a metric for image quality.
  - **`colorfulness`**: Computes a metric for how colorful an image is.
  - **`R squared Score`**: How well the predicted colors match with the ground truth.
- **Visualization** Displays the grayscale input, predicted color image, and ground truth color image side by side.
- **Training Loop** Trains the model for a specified number of epochs. For each batch:
  - Moves data to the device (CPU/GPU).
  - Computes the model's output and loss.
  - Performs backpropagation and updates the model's weights.
- **Evaluation Loop**
  - Evaluates the model on the validation/test dataset.
  - Computes metrics (MSE, PSNR, SSIM, colorfulness) and visualizes the first batch of results.
- **Main Function Definition**
  - Sets up hyperparameters, device, and data transformations.
  - Creates the dataset and data loader.
  - Initializes the model, loss function, and optimizer.
  - Trains the model and evaluates it.
  - Saves the trained model to a file.
- **Enhancement**
  - Adding Perceptual Loss Function using the following code enhanced the accuracy of the model

```
    class PerceptualLoss(nn.Module):
def __init__(self):
    super(PerceptualLoss, self).
        __init__()
    self.vgg = models.vgg11(
        pretrained=True).features
        [:5].eval()  # Use first 5
        layers of VGG11
    for param in self.vgg.
        parameters():
        param.requires_grad = False

def forward(self, pred, target):
    with torch.no_grad():  #
        Disable gradient
        computation for perceptual
        loss
        pred_features = self.vgg(
            pred)
        target_features = self.vgg(
            target)
    return torch.nn.functional.
        mse_loss(pred_features,
        target_features)
```

    - PerceptualLoss: Uses a pretrained VGG11 network to compute perceptual loss.
    - Extracts features from the first 5 layers of VGG11.
    - Compares the features of the predicted and target images using MSE loss.
  - Modifying the Training Loop using Gradient Accumulation which provides the ability to simulate larger batch sizes.

## IV. Experiment

### A. Experimental Setup

- **Dataset:** A diverse collection of grayscale and color image pairs As shown in Fig. 1
- **Evaluation Metrics:** During the first milestone, the color mapping techniques mentioned in the III-A1 were done. The three methods had different PSNR, SSIM values as mentioned in the Table I.
  In the CNN implementation the model is trained for 20 and 30 epochs using the Adam optimizer with a learning rate of 0.001. Mean Squared Error (MSE) is used as the primary loss function, and the model is evaluated using the following metrics:

(a) Sample of Color Data Set

(b) Sample of Gray-scale Data Set

Fig. 1: Samples of the Dataset Used

- **MSE**: The pixel-wise difference between predicted and ground truth images is measured.
- **PSNR**: The quality of the predicted images relative to the ground truth is indicated.
- **SSIM**: The structural similarity between predicted and ground truth images is measured.
- **$R^2$**: How well the predicted colors match the ground truth is measured.

The evaluation metrics on the validation set are summarized in Table I.

| Method | PSNR | SSIM | Observations |
|---|---|---|---|
| Custom Color Mapping | 18.5 | 0.62 | Limited color diversity |
| Intensity Based Color Mapping | 19.3 | 0.65 | Better structure preservation |
| OpenCV Color Mapping | 19.3 | 0.65 | Better structure preservation |
| CNN Fine-Tuned | **23.14** | **0.93** | Most realistic colorization |

TABLE I: Comparison of Colorization Techniques

### B. Enhancement

Strong performance is achieved by the model on the validation set, as evidenced by the quantitative metrics and qualitative results. However, some limitations are observed:

- Fine details and textures in complex images are struggled with by the model.
- The colorization is observed to appear de-saturated in some cases.

The use of perceptual loss improved the visual quality of the predicted images.

### C. Quantitative Results

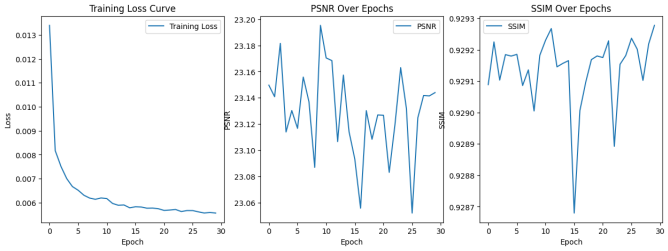
The numbers for PSNR, SSIM were directly obtained for the classical color mapping techniques. Those values did not depict the expected level of performance.

For the CNN approach, the training loss, loss over $R^2$ PSNR, and SSIM curves over 30 epochs with perceptual loss function integration, are shown in Figure 2. The training loss is observed to be decreased steadily and plateau after 20 epochs, indicating convergence. The PSNR and SSIM curves are seen to increase steadily, demonstrating that the model improves over time.

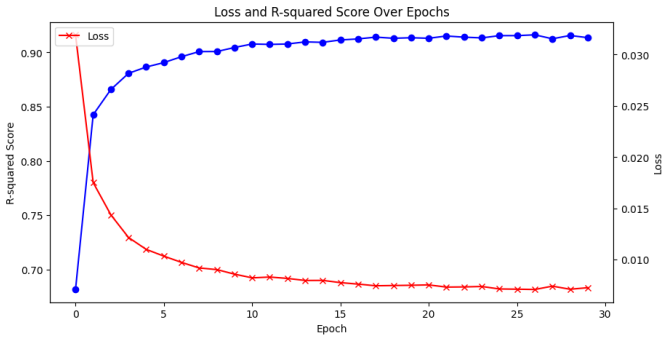
### D. Qualitative Results

Color Mapping technique felt short when it comes to proper colorization, a sample set of results can seen in Figure 3.

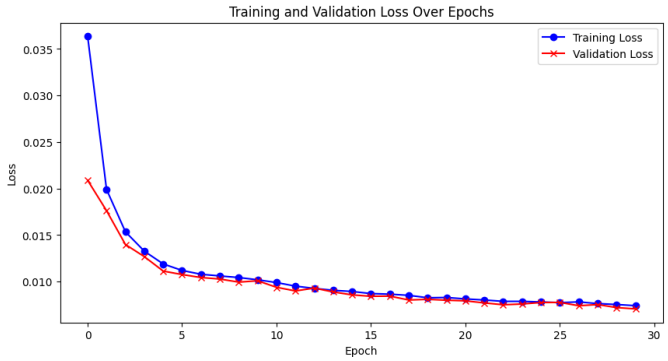
For the CNN Approach, example results of the model's predictions before and after the integration of perceptual loss function are shown in Figure 4 and Figure 5. The grayscale input, predicted color image, and ground truth color image are displayed side by side. Plausible colors are successfully predicted for most images, although some fine details are observed to be lost.




(a) Training loss, PSNR, and SSIM curves.




(b) Loss over $R^2$ score.




(c) Training and validation loss.

Fig. 2: Training performance metrics over epochs.

## V. Conclusion

### A. Key Findings

- Classical methods are effective for simple textures but struggle with complex objects.
- CNN-based models significantly improve realism by leveraging learned color distributions.
- Combining classical preprocessing with CNN training yields superior results.

### B. Future Work

- **Real-Time Implementation:** Optimize the model for video-based colorization by leveraging temporal consistency between frames. This could involve using recurrent neural networks (RNNs) or temporal convolutional networks (TCNs) to handle the sequence of frames efficiently, ensuring smooth and consistent colorization in real-time applications.

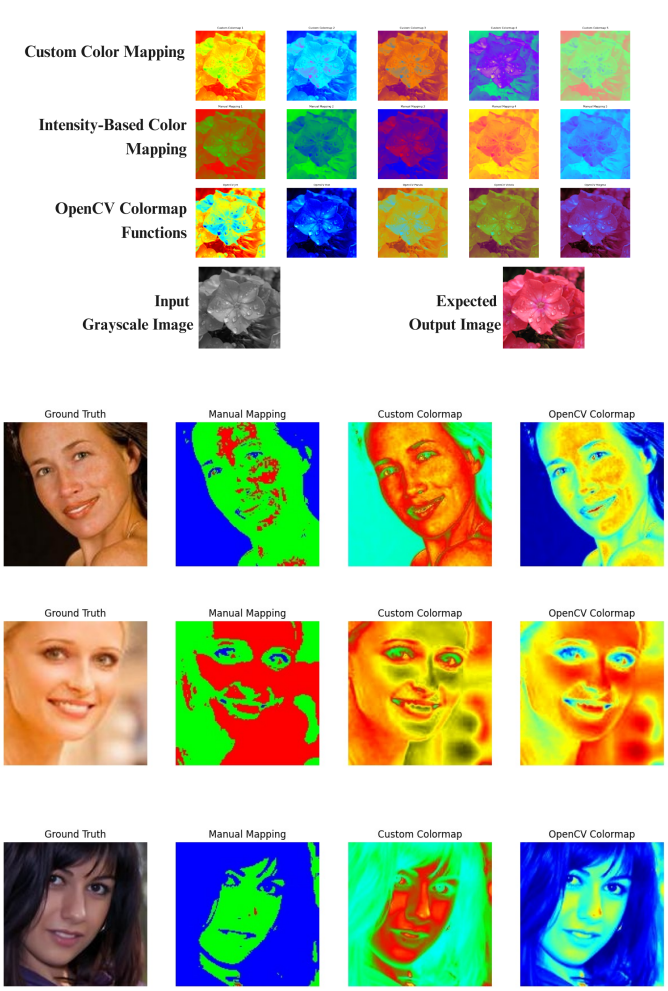
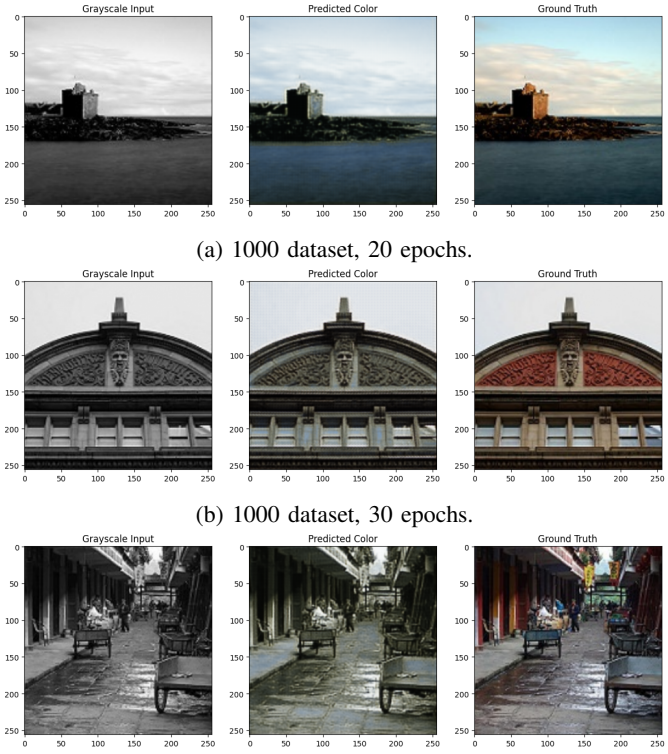



Fig. 3: Example results from Color Mapping Techniques

- **Dataset Expansion:** Train the model on a larger and more diverse dataset, incorporating a variety of image types, lighting conditions, and textures to improve the model's generalizability and robustness. Using high-quality and high-resolution datasets will allow the model to better handle a wider range of images and edge cases.
- **Adaptive Learning:** Implement reinforcement learning techniques to enable continuous improvement of the model. By incorporating a feedback loop where the model evaluates its predictions and adjusts its parameters based on user feedback or quality metrics, it can progressively enhance its performance in real-world applications.
- **Adversarial Learning:** Explore the use of Generative Adversarial Networks (GANs) to enhance the realism and detail of the colorized images. A GAN-based approach could involve training a discriminator to distinguish between real and generated colorized images, pushing the generator to produce more realistic and high-quality colorizations, especially for challenging or ambiguous regions.




(a) 1000 dataset, 20 epochs.




(b) 1000 dataset, 30 epochs.

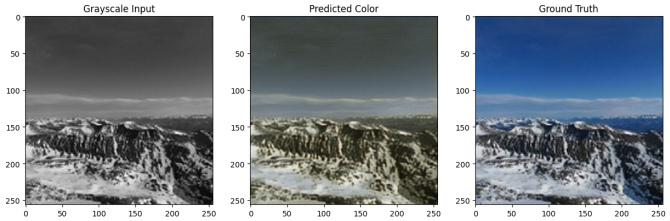



(c) 2000 dataset, 20 epochs.

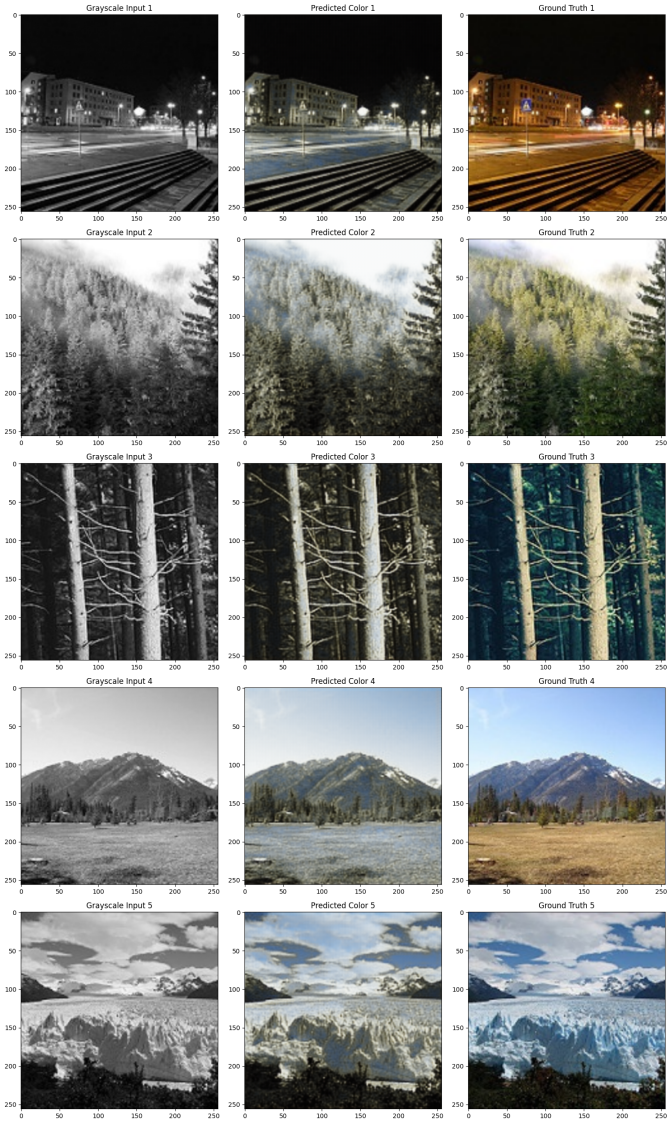
Fig. 4: Example results before perceptual loss integration: grayscale input (left), predicted color image (middle), and ground truth (right).

## REFERENCES

[1] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using Optimization," ACM Transactions on Graphics, vol. 23, no. 3, pp. 689–694, 2004.
[2] S. Gao, M. Zhang, et al., "Fourier Transform-based Image Enhancement Techniques," IEEE Transactions on Image Processing, vol. 28, no. 5, pp. 2169-2181, 2019.
[3] R. Zhang, P. Isola, and A. A. Efros, "Colorful Image Colorization," ECCV, 2016.
[4] J. Antic, "DeOldify: A Deep Learning-Based Project for Colorizing and Restoring Old Images," 2019. [Online]. Available: https://github.com/jantic/DeOldify
[5] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," CVPR, 2017.




(a) 1000 dataset with perceptual loss.




(b) 2000 dataset with perceptual loss.

Fig. 5: Example results after perceptual loss integration: grayscale input (left), predicted color image (middle), and ground truth (right).