Image Colorization Using Hybrid SLR-CNN Model

ECS 171 Group 3

Anna Chan, Richard Lewins, Tiffany Tran, Alexandr Volkov, Jasmine Saldano, Qihan Guan, Karim Shami, Kaylie Lam, Steven Liu, Shreya Vallala, Navin Klein, Nathan Castellon

Fall 2025

Abstract

The transition from black and white to color images in media has created a longstanding interest in restoring old grayscale images through colorization. Although editing software has evolved to allow individuals to manually recolor images, the process still demands considerable time and expertise, making it difficult for non-experts and unsuitable for large-scale restoration efforts. The emergence of deep learning techniques, particularly Convolutional Neural Networks (CNNs), has made it possible to detect and learn complex spatial patterns within images, enabling us to infer chromatic information from the structural cues present in grayscale images. However, this approach was prone to inaccuracies, specifically with color desaturation and chroma-luminance misalignment (color drift); with a limited dataset, there was also some risk of overfitting. So to address these issues, we developed a hybrid structural linear regression (SLR) model that utilizes the CNN with a fixed set of spatial kernels or filters. By setting structural constraints, the hybrid model eliminates color drift artifacts, producing a colorization of images void of extensive leakage. Our comprehensive approach offers a more practical and accessible solution to restore grayscale images into colored ones.

1 Introduction

1.1 History of Image Colorization

While humans are skilled at interpreting visual content, computational approaches to image processing can enable tasks such as restoration, enhancement, and transformation on a scale that would be impractical when done manually. The implementation of automated, machine learning methods for interpreting and manipulating images has fundamentally transformed the field of image processing. These computational approaches date back to 1964, when they were first used to correct distortions and improve the resolution of original lunar photographs taken by NASA's Surveyor I project. [1]

Computer enchancment during this time was a necessity. Improvements in image colorization meant better data for scientists to work on and thus allow for better analysis from the images gathered. This need for better data and improved image colorization spanned across various industries and thus pushed others to enhance the technology for the betterment of the field. Since then, the field has seen significant research progress, leading to more sophisticated techniques.

1.2 Applications of Modern Techniques

Today, we often use convolutional neural networks (CNN), a type of artificial neural network that is particularly well suited for image processing. It is a feed-forward neural network that includes a convolutional layer, allowing it to learn features with kernel optimization. CNNs have been particularly helpful in the computer vision sector; utilizing its ability to classify and recognize images, audio, and speech. This serves as an important addition to several different industries, aiding in medical advancements and military surveillance, as well as the agriculture and production sectors. [2]

In the context of image editing, these advancements in technology have also enabled major software companies to automate and simplify editing tasks for the general public. For instance, CNNs are integrated into AI-powered features across Adobe's suite of image editing tools to help streamline the editing process for millions of users (e.g Neural Filters) [3] However, these implementations are still being revised and researched to provide greater accuracy. While met with some high praise, many users find faults within these tools, citing inconsistent outputs and the production of unnatural artifacts, resulting in necessary human intervention.

With this project, we are focusing on a subarea of image colorization, which involves converting grayscale images into accurate RGB images. Applications of this method are highly relevant in the following areas:

- Restoration: Colorizing old grayscale photographs into full-color ones can serve an important archival purpose as many historical moments can be preserved with greater detail and clarity.
- Storage and Recovery: Color images may be converted to grayscale or other formats to save storage space and communication bandwidth. Accurately reconstructing the original color values from this simplified representation is fundamental to realizing these benefits in storage and communication.

- Filters: Modifying the colors of landscapes and objects in an image can be useful for creative professionals who are looking to enhance and transform their work without reshooting or manual editing.

Using a variety of datasets–CIFAR, Oxford Flower, STL10–we trained a baseline convolutional neural network (CNN) model, which had issues with color drift and color desaturation. The CNN baseline colorization typically predicts chroma from the grayscale image. But this has two problems:

1. It doesn't have a constraint to respect the geometry of the luminance field, so the gradients from neighboring pixels leak into the center, which causes the model to smear color across edges.
2. The CNN's output is unconstrained on the whole 2D plane with no structural guidance, which leads to color bleeding and desaturation under MSE.

We formulated the following hypothesis to address these issues: Color should only be applied where the grayscale image provides the geometric evidence. To test this hypothesis, we use a Hybrid structural linear regression (SLR) + CNN model. This model does not predict color directly but rather, learns to combine a fixed set of pixel patterns and structural features. By constraining the output to the linear combination of these local features, we explored how our hybrid model ensures that color changes are only driven by these structural patterns in the grayscale image.

2 Methods

We implemented two different models in hopes of achieving an effective and efficient image colorizer.

2.1 Dataset

2.1.1 Our Datasets

We trained and tested our colorizer on the following datasets:

CIFAR10 dataset[1] :

- 60,000 labeled images, 10 classes, 6,000 images per class
- Classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
- Resolution: 32x32
- Notes: low quality dataset, so model does not understand features

Oxford Flower dataset[2] :

- Commonly occurring flowers in the UK
- 102 categories, 40-258 images per class, 10k total images
- Varied resolutions (around 500x500)
- Notes: dataset is too small and has a limited scope since it is only flowers

STL10 dataset[3] :

- 100,000 unlabeled images, 10 classes
- 5,000 labeled training images
- Classes: Airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck
- Resolution: 96x96
- Notes: good dataset with complex images, but the colors are not consistent.

2.1.2 Data Preprocessing

During data preprocessing, we needed to first convert our samples to grayscale, as all of our datasets contain images in RGB. To achieve this, we use a standard weight vector associated with the three RGB channels.

$$\mathbf{w} = [0.299 \;\; 0.587 \;\; 0.114]^\top$$

The above vector correlates to the way humans perceive different colors. We can see that green colors account for a majority of our perception of color brightness. Conversely, blue colors account for the least amount of the brightness that humans see. We can simply multiply the weight vector by each of the three color channels that make up an RGB pixel.

After converting all of our data samples to grayscale, we needed to establish a uniform set of dimensions that would be applied to all of our inputs. We resized each of our data samples to 144 pixels by 144 pixels using bilinear interpolation. Because the CIFAR10 and STL10 datasets contained images of smaller dimensions we attempted to retain the structure of the original image by center cropping the image with a size of 128 pixels by 128 pixels.

To increase the generality of our model, we also experimented with data augmentation by applying random flips and rotations ($\pm 10^\circ$) to the training data. Furthermore, we attempted to address concerns of overfitting by merging datasets to provide the model with a more diverse training set.

2.1.3 Evaluation Metrics

We evaluate our models primarily using Mean Squared Error (MSE). MSE measures the average squared difference between our model RGB vector and the actual RGB vector, making it a natural choice for

[1] https://www.cs.toronto.edu/ kriz/cifar.html
[2] https://www.robots.ox.ac.uk/ vgg/data/flowers/102/
[3] https://cs.stanford.edu/ acoates/stl10/

regression-based colorization. After each epoch, we recorded both training and validation MSE to verify that our architecture and hyperparameters were learning meaningful color representations. Tracking validation MSE also allowed us to diagnose overfitting or underfitting across the three datasets. In addition to MSE analysis, we performed visual inspection of the colorized outputs to evaluate perceptual quality, since numerically low MSE does not always correspond to visually realistic colors.

2.2 RGB Decomposition

To predict color from grayscale, we want to predict the pixel's chroma (hue and saturation) based on its luminance (brightness). We were inspired by Lab Color Space [8], which is a 3-dimentional space of 3 axes: L (luminance), a (red-green axis), and b (blue-yellow). In other words, each colored pixel is composed of 1 luminance value and 2 chroma values.

Rather than representing luminance and chroma in lab space, we decided to model our prediction in RGB space.

2.2.1 Grayscale Image

A 2D grayscale image is a matrix $Y \in \mathbb{R}^{H \times W}$, where H and W denote the image height and width in pixels, respectively. Each entry I_{ij} is a scalar representing the intensity (brightness) of the pixel located at the i-th row and j-th column, in the range of 0 (black) to 255 (white) inclusive.

2.2.2 Colored Image

A colored image is a 3D matrix (tensor) that stores three separate grayscale layers, one for each color channel: Red (R), Green (G), and Blue (B). Define:

$$V \in \mathbb{R}^{H \times W \times 3}$$

where H is image height (rows) and W is image width (columns). The 3rd dimension (size = 3) corresponds to the three color channels.

Each entry (pixel) $\mathbf{v}_{ij}$ is a 3D vector:

$$\mathbf{v}_{ij} = \begin{bmatrix} R_{ij} G_{ij} B_{ij} \end{bmatrix} \in \mathbb{R}^3$$

Each component gives that pixel's red, green, and blue intensities:

$$R_{ij},\ G_{ij},\ B_{ij} \in \{0, 1, \ldots, 255\}$$

Each entry like (255, 0, 0) is the RGB vector for one pixel (pure red).

2.2.3 RGB to Grayscale

To convert from grayscale to RGB, we can express the grayscale part Y of the RGB vector $\mathbf{v}$ as a linear combination of

$$y \ = \ \mathbf{w}^\top \mathbf{v} \ = = w_R R + w_G G + w_B B$$

Here, $\mathbf{w} = [0.299\ 0.587\ 0.114]^\top$ is a constant vector that defines brightness in RGB space. The RGB weights 0.299, 0.587, 0.114 come from how human vision perceives brightness. Our eyes are most sensitive to green, somewhat to red, and least to blue, so green contributes the most to perceived luminance while blue contributes the least. Early color-TV systems used this same perceptual model, so that color broadcasts could remain compatible with black-and-white TVs: the grayscale image (the $Y or lumachannel) had to look "correct" to human viewers.$

2.2.4 Luminance-Chroma Decomposition

Now that we know what grayscale/colored images look like, and the conversion from color to grayscale using $\mathbf{w}$, we must now figure out how to decompose the colored RGB image into luminance and chroma independently.

We first project the grayscale part Y of the RGB vector $\mathbf{v}$ as a linear combination of a constant weight vector $\mathbf{w}$ and RGB vector $\mathbf{v}$. While the $y = \mathbf{w}^\top \mathbf{v}$ mapping is helpful, it has infinite many solutions and different chroma $\mathbf{v}_\perp$ values produce the same y . So in order to actually model $\mathbf{v}$, we need to find a more useful $\mathbf{u}$ decomposition that gives more constraints.

With vector $\mathbf{v}$ and a nonzero vector $\mathbf{w}$, we can rewrite $\mathbf{v}$ as

$$\begin{aligned}
\mathbf{v} \ &= \ \mathrm{proj}_{\mathbf{w}}(\mathbf{v}) + (\mathbf{v} - \mathrm{proj}_{\mathbf{w}}(\mathbf{v})) \\
&= \ \frac{\mathbf{v} \cdot \mathbf{w}}{\mathbf{w} \cdot \mathbf{w}} \mathbf{w} + \left(\mathbf{v} - \frac{\mathbf{v} \cdot \mathbf{w}}{\mathbf{w} \cdot \mathbf{w}} \mathbf{w} \right) \\
&= \ \underbrace{\frac{y}{\mathbf{w}^\top \mathbf{w}} \mathbf{w}}_{\text{brightness part of } \mathbf{v}} + \underbrace{\left(\mathbf{v} - \frac{y}{\mathbf{w}^\top \mathbf{w}} \mathbf{w} \right)}_{\text{the missing part of } \mathbf{v}} \quad (\text{as } y = \mathbf{w}^\top \mathbf{v}) \\
&= \ \underbrace{y}_{\substack{\text{luminosity} \\ \text{scalar} \\ y \in \{0, \ldots, 255\}}} \underbrace{\left(\frac{1}{\mathbf{w}^\top \mathbf{w}} \mathbf{w} \right)}_{\text{3D vector}} + \underbrace{\left(\mathbf{v} - y \frac{1}{\mathbf{w}^\top \mathbf{w}} \mathbf{w} \right)}_{\substack{\text{the missing part of } \mathbf{v} \\ \text{without luminosity (chroma)}}} \\
&= \mathbf{v}_\parallel + \mathbf{v}_\perp
\end{aligned}$$

Where $\mathbf{v}_\parallel$ and $\mathbf{v}_\perp$ represent the grayscale and chroma parts of $\widehat{\mathbf{v}}$, respectively. Our goal is to predict $\mathbf{v}_\perp$. This also tells us another important constraint and confirms mathematically what we know: $\mathbf{w}$ is orthogonal to $\mathbf{v}_\perp$

Using the orthogonal constraint $\mathbf{w}^\top \mathbf{v}_\perp = 0$ we can expand the dot product. Solving for one of the variable, say v_B we get: $v_B = -\frac{w_R v_R + w_G v_G}{w_B}$. This allows us to re-express $\mathbf{v}_\perp$ as

$$\mathbf{v}_\perp = \begin{bmatrix} v_R \\ v_G \\ -\dfrac{w_R v_R + w_G v_G}{w_B} \end{bmatrix} = v_R \begin{bmatrix} 1 \\ 0 \\ -\dfrac{w_R}{w_B} \end{bmatrix} + v_G \begin{bmatrix} 0 \\ 1 \\ -\dfrac{w_G}{w_B} \end{bmatrix}.$$

This shows that $\mathbf{v}$ can be expressed as a linear combination of two basis vectors $\mathbf{u}_1, \quad \mathbf{u}_2$ that are

orthogonal to $\mathbf{w}$. Therefore our final modeled RGB vector is

$$\hat{\mathbf{v}} = \frac{Y}{\mathbf{w}^\top \mathbf{w}}\mathbf{w} + \underbrace{\alpha\,\mathbf{u}_1 + \beta\,\mathbf{u}_2}_{\mathbf{v}_\perp}$$

This models our RGB vector as a decomposition of luminance ($\mathbf{v}_\parallel$) and chroma ($\mathbf{v}_\perp$, our orthonormal color basis).

2.3 Baseline CNN

2.3.1 Baseline Formulation

To implement our RGB model, we use a standard CNN regression model. We used convolution kernals to learn the features $\mathbf{z}$ through backpropagtion and gradient descent. We used the ADAM optimizer to determine the weights of the model. ADAM combines the RMSProp and Momentum optimization algorithms resulting in an adaptable learning rate and global minimums during gradient descent. Then, we normalized using PyTorch's BatchNorm2D. And then, we used the ReLU activation function for non-linearity. The loss is mean squared error.

$$h^{(0)} = Y \in \mathbb{R}^{H\times W\times 1}$$

$$h_k^{(\ell)} = \mathrm{ReLU}\Big(z_k^{(\ell)}\Big) = \begin{cases} 0, & z_k^{(\ell)} \le 0, \\ z_k^{(\ell)}, & z_k^{(\ell)} > 0. \end{cases}$$

$$z_k^{(\ell)} = \mathrm{BN}\Big(\Big\langle K^{(\ell,k)},\, h^{(\ell-1)}\Big\rangle_F + b_k^{(\ell)}\Big)$$

$$\mathbf{z}^{(L)} = \begin{bmatrix} z_1^{(L)} \\ z_2^{(L)} \end{bmatrix} = \begin{bmatrix} \big\langle K^{(L,1)},\, h^{(L-1)}\big\rangle_F + b_1^{(L)} \\ \big\langle K^{(L,2)},\, h^{(L-1)}\big\rangle_F + b_2^{(L)} \end{bmatrix}$$

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} z_1^{(L)} \\ z_2^{(L)} \end{bmatrix}$$

$$\hat{\mathbf{v}} = \frac{Y}{\mathbf{w}^\top \mathbf{w}}\,\mathbf{w} \;+\; \alpha\,\mathbf{u}_1 \;+\; \beta\,\mathbf{u}_2$$

$$\mathcal{L} = \frac{1}{N}\sum_{i,j}\big\|v_{ij} - \hat{v}_{ij}\big\|^2$$

2.3.2 Baseline Architecture

We use the following CNN architecture:

- Input: Luminance L
- Encoder: 3x Conv(4x4, stride 2): 1→32→64→128
- Bottleneck: 2x Conv(3x3): 128→128→128
- Decoder: 3x ConvTranspose(4x4, stride 2): $128 \to 64 \to 32 \to 2$
- Output: Predicted chroma coefficients (α, β)

2.3.3 Addressing our Hypothesis

Our prediction model for the RGB vector is

$$\hat{\mathbf{v}} \;=\; \underbrace{\frac{Y}{\mathbf{w}^\top \mathbf{w}}\mathbf{w}}_{\mathbf{v}_\parallel} \;+\; \underbrace{\Big[(\hat{\alpha})\,\mathbf{u}_1 + (\hat{\beta})\,\mathbf{u}_2\Big]}_{\mathbf{v}_\perp}$$

Through testing our baseline CNN model, not only did we see recoloring inaccuracy, we also saw many cases of "leaking" and "color drifting".

We first hypothesized that it may have been due to the problem with the MSE loss function's regression to the mean, the datasets' lack of variety, and the unconstrained RGB space.

But then we hypothesized that it may have to do with the fundemental structure of our baseline CNN. Our chroma model output space is the entire 2D plane:

$\hat{\mathbf{v}}_\perp^{\mathrm{CNN}} \in \mathbb{R}^2 \quad \big((\alpha,\beta) \in \mathbb{R}^2\big)$

The output is geometrically unconstrained. Especially with MSE loss, this often results in desaturation (regression to the mean), as the network averages conflicting color modes rather than selecting a precise hue.

Through our research, we also found another potential structural problem that contributes this pixel drifitng problem, called "graident leakage"[10,9]. Consider a sub-region of the grayscale Y where the gradient at the center is zero, $\nabla Y_{ij} = 0$. Here ∇ denotes a discrete finite-difference gradient (e.g. $Y_{i+1,j} - Y_{i,j}$),

This means all immediate neighbors have identical brightness:

$$Y_{i,j} \approx Y_{i+1,j} \approx Y_{i,j+1} \approx \dots$$

While the absolute chroma value $\hat{\mathbf{v}}\perp$ may depend on the surrounding patch (to gather semantic context), the chroma variation $\nabla\hat{\mathbf{v}}\perp$ should follow the local luminance variation. In other words, we ideally require $\nabla Y_{ij} = \mathbf{0}$ implies $\nabla\hat{\mathbf{v}}_{ij} = \mathbf{0}$.

To analyaze this, we consider the first order linear approximation of the CNN. Although the full CNN is nonlinear, it's piecewise linear. So in a it's approximated regions around a given luminance patch, the mapping can $(\alpha(\hat{i},j), \beta(\hat{i},j))$ can be approximated by its Jacobian. This jacobian has the form of a convolution, so locally it behaves like an effective linear filter $\hat{\alpha}(i,j) \approx \langle W_{\text{eff}}^{\alpha}, \nabla Y_{\text{patch}}(i,j)\rangle$

Now consider a specific case where the center pixel has no luminance variation ($\nabla Y(i,j) = \mathbf{0}$), but one of its neighboring pixel contains an edge ($\nabla Y((i,j)+\mathbf{n}) \neq \mathbf{0}$). The spatial gradient of the prediction $\nabla\hat{\alpha}$ becomes (and by symmetry, β)

$$\begin{aligned}\nabla\hat{\alpha}(i,j) \;&=\; \langle W_{\text{eff}}^{(\alpha)},\, \nabla Y_{\text{patch}}(i,j)\rangle \\ &= \;\cdots\; + \; W_{\mathbf{0}}^{(\alpha)}\underbrace{\nabla Y(i,j)}_{\text{Center }(=\mathbf{0})} + W_{\mathbf{n}}^{(\alpha)}\underbrace{\nabla Y((i,j)+\mathbf{n})}_{\text{Neighbor }(\neq\mathbf{0})} + \;\cdots\end{aligned}$$

Ideally, $\nabla\hat{\alpha}$ and $\nabla\hat{\beta}$ should depend only on the local variation at the center pixel. However, the equation above

shows that in a standard CNN, $\nabla\hat{\alpha}$ and $\nabla\hat{\beta}$ are weighted sums of the gradients from the entire patch. Thus, even when the center pixel has no luminance variation ($\nabla Y_{ij} = \mathbf{0}$), the CNN produces $\nabla\hat{\mathbf{v}}_{\perp,ij} \neq \mathbf{0}$

The model has mathematically "leaked" the gradient information from the neighbor $((i,j) + \mathbf{n})$ into the prediction for the center (i,j), causing edge misalignment.

2.4 Hybrid Model

2.4.1 Hybrid Overview

To test our hypothesis of the baseline model's problem, we used a hybrid model that combines CNN and SLR,

In this Hybrid model, we still used MSE as our loss function and still trained on the same datasets. We also still used the same decomposition model, but except we define $\alpha = \mathbf{a}^\top \mathbf{z}$ $\beta = \mathbf{b}^\top \mathbf{z}$.

where the features $\mathbf{z}_{ij}$ are fixed with our handpicked kernels $K^{(k)}$:

$$\mathbf{z}_{ij} = \begin{bmatrix} \langle K^{(1)},\, Y_{\text{patch}}(i,j)\rangle_F \\ \langle K^{(2)},\, Y_{\text{patch}}(i,j)\rangle_F \\ \vdots \\ \langle K^{(m)},\, Y_{\text{patch}}(i,j)\rangle_F \end{bmatrix}.$$

So our model is

$$\widehat{\mathbf{v}}_{ij} \;=\; \frac{Y_{ij}}{\mathbf{w}^\top\mathbf{w}}\mathbf{w} \;+\; \Big[(\mathbf{a}_{ij}^\top \mathbf{z}_{ij})\,\mathbf{u}_1 + (\mathbf{b}_{ij}^\top \mathbf{z}_{ij})\,\mathbf{u}_2\Big]$$

In this model, we use a CNN to find the optimal weights $\mathbf{a}$ and $\mathbf{b}$ instead

$$\mathbf{a}_{ij} = g_\theta(Y_{\text{patch}}(i,j)), \quad \mathbf{b}_{ij} = h_\theta(Y_{\text{patch}}(i,j))$$

While $\mathbf{v}_\perp$ is still in the color plane spanned by $\mathbf{u}_1, \mathbf{u}_2$, how it moves in that plane is now constrained by $\mathbf{z}$, as we force α, β to be linear in the fixed features $\mathbf{z}_{ij}$. This is in contrast to the baseline CNN model, where the network must both discover where edges/textures are and decide what chroma to assign.

This theoretically would fix the misalignment problem in the baseline CNN. If $\nabla Y_{ij} = 0$ then

$$\nabla\hat{\alpha} = \underbrace{(\nabla\mathbf{a})\cdot\mathbf{z}}_{\text{Term 1}} + \underbrace{\mathbf{a}\cdot(\nabla\mathbf{z})}_{\text{Term 2}}.$$

Term 2 vanishes as $\nabla\mathbf{z} = \mathbf{0}$ (as $\nabla z \approx z_{i+1} - z_i$), and consequently $\nabla\mathbf{z} = \mathbf{0}$. Since the fixed kernels $K^{(k)}$ are zero-mean derivative-like filters, they output a constant response when the luminance patch is constant. So if Y is flat, i.e. $Y = [\ldots 10\ 10\ 10 \ldots]^\top$, then $\nabla z = 0$

Term 1 also vanishes as $\nabla\mathbf{z} = 0$. Since the filters output zero magnitude in the flat regions, the feature vectors $\mathbf{z}$ itself is the zero vector. Therefore regardless of the values $\mathbf{a}$. The dot product $(\nabla\mathbf{a})^\top$ is exactly zero. This enforces the desired constraint that chroma cannot change where the luminance does not change($\nabla Y_{ij} = 0 \implies \nabla\mathbf{v}_\perp \approx 0$.)

The specific predefined kernels we used are Sobel X for vertical edges, Sobel Y for horizontal edges, and Laplacian, identity for general edges detection.

$$\underbrace{\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}}_{\text{sobel } X}, \quad \underbrace{\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}}_{\text{sobel } Y}, \quad \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\text{Laplacian}}$$

2.4.2 Hybrid Formulation

$\mathbf{H}^L \in \mathbb{R}^{H\times W\times(2m)}$ where $2m$ is the number of

$$h^{(0)} = Y \in \mathbb{R}^{H\times W\times 1}$$

$$\text{for } \ell = 1, \ldots, L-1$$

$$h_k^{(\ell)} = \text{ReLU}\Big(\text{BN}\Big(\Big\langle K^{(\ell,k)},\, h^{(\ell-1)}\Big\rangle_F + b_k^{(\ell)}\Big)\Big),$$

$$z_{\text{fixed}}^{(m)} = \Big\langle K_{\text{fixed}}^{(m)},\, Y\Big\rangle_F$$

$$\mathbf{H}^{(L)} = \begin{bmatrix} a \\ b \end{bmatrix} = \Big\langle K^{(L)},\, h^{(L-1)}\Big\rangle_F + b^{(L)}$$

$$\hat{\mathbf{v}} = \frac{Y}{\mathbf{w}^\top\mathbf{w}}\,\mathbf{w} \;+\; (a\cdot\mathbf{z}_{\text{fixed}})\,\mathbf{u}_1 \;+\; (b\cdot\mathbf{z}_{\text{fixed}})\,\mathbf{u}_2$$

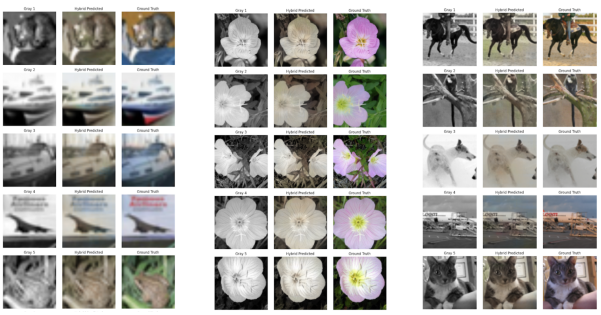
To summarize, by introducing fixed derivative filters, we explicitly encode luminance structure into the chroma predictor. This constrains the CNN so that chroma edges can only occur where luminance edges exist, preventing the baseline CNN's tendency to produce misaligned or hallucinated color boundaries. In Layman's terms, the hybrid model splits the work: the fixed filters lessen the learning curve for the CNN.

2.5 Training our Models

To train both our models, we used Google Colab's A100 GPU with a batch size of 64 and 50 epochs. For simplicity, this is pseudocode of our hybrid model implementation (same as formulation).

```
for each image:
    Y = convert_rgb_to_gray(V, w)
    z = extract_fixed_kernels(Y)
    a, b = cnn.predict(Y)
    v_perp = (a * z) * u[1] + (b * z) * u[2]
    v_predicted  = v_parallel + v_perp
    loss = mse(v_predicted, V_true)
```

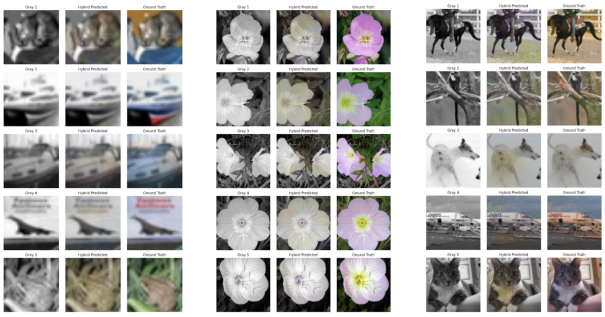
Our baseline model is similar, except we don't extract fixed kernels.



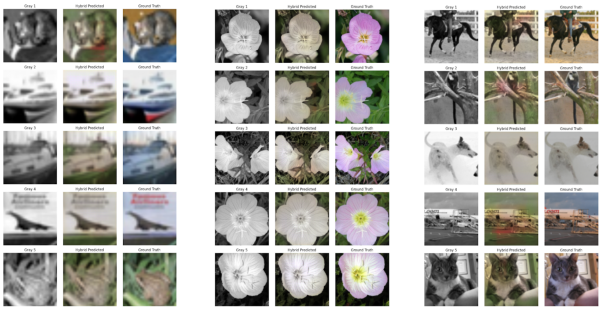
Hybrid Model Outputs (STL10)

Baseline Model Outputs (STL10)

Hybrid Model Outputs (CIFAR10)



Baseline Model Outputs (CIFAR10)



Hybrid Model Outputs (Flowers)

Baseline Model Outputs (Flowers)

Figure 1: Hybrid vs. Baseline Model Outputs Across All Three Datasets

3 Results

In this section, we compare the performance of our baseline CNN colorization model with our proposed hybrid structured linear regression (SLR) + CNN model across all three datasets — CIFAR10, Oxford Flowers, STL10.

Overall, as seen in Fig. 1, our results show that the hybrid model performed better than the baseline model across our evaluation metrics. While the overall accuracy is not perfect, the hybrid approach was better at capturing key visual features and producing more stable color predictions. These improvements indicate that the hybrid model is able to address several challenges the baseline struggled with, making it a more viable and promising option for automated image colorization. To evaluate and compare these models, we tested both methods on all three chosen datasets — STL10, CIFAR10, and Flowers.

Table 1 summarizes the overall performance of the hybrid model relative to the baseline across all datasets. The hybrid MSE to baseline ratio of 0.931 and an overall improvement of 6.93% indicates that, on average, the hybrid CNN produces more accurate colorizations than baseline CNN.

Formulas:

$$\text{Hybrid MSE / Baseline MSE Ratio} = \frac{\text{Hybrid MSE}}{\text{Baseline MSE}}$$

$$\text{Overall Improvement (\%)} = \left(1 - \frac{\text{Hybrid MSE}}{\text{Baseline MSE}}\right) \times 100$$

Metric	Value
Hybrid MSE / Baseline MSE Ratio	0.931
Overall Improvement (%)	6.93

Table 1: Overall performance summary across all datasets.

Table 2 breaks down the validation MSE and hybrid-to-baseline ratio for each individual dataset. It demonstrates that the hybrid model consistently reduces MSE compared to the baseline, with the most notable improvements on CIFAR10 and Flowers datasets.

Formula:

$$\text{Hybrid/Base Ratio (dataset)} = \frac{\text{Hybrid MSE (dataset)}}{\text{Baseline MSE (dataset)}}$$

Dataset	Base MSE	Hybrid MSE	Hybrid/Base Ratio
CIFAR10	0.007763	0.005827	0.751
Flowers	0.012417	0.009101	0.733
STL10	0.005774	0.005577	0.966

Table 2: Validation MSE & Hybrid/Base Ratio Across All Datasets.

Table 3 specifically focuses on the percentage improvement of the hybrid over the baseline for each individual dataset. It quantifies the development in accuracy, demonstrating substantial improvements for CIFAR10 (24.94%) and Flowers (26.73%). The improvement for STL10 is smaller (3.42%), yet overall these each reflect dataset-specific differences in performance.

Formula:

$$\text{Improvement (\%) (dataset)} = \frac{\text{Baseline MSE} - \text{Hybrid MSE}}{\text{Baseline MSE}} \times 100$$

Dataset	Improvement (%)
CIFAR10	24.94
Flowers	26.73
STL10	3.42

Table 3: Percentage Improvement of Hybrid Model Over Baseline Model.

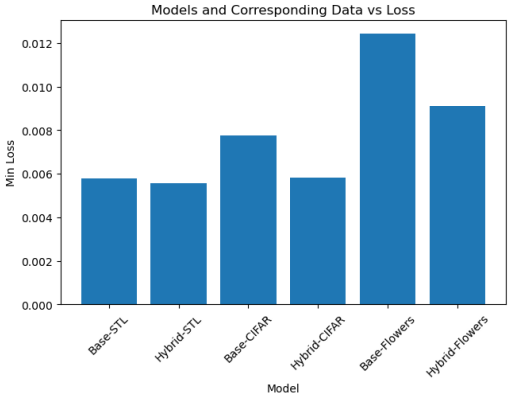


Figure 2: Data vs Loss (All Models)

3.1 Training on the STL10 Dataset

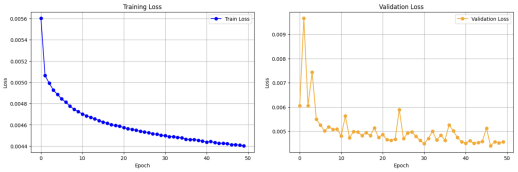


Figure 3: Hybrid Model Training and Validation Loss

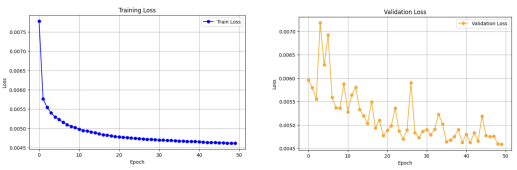


Figure 4: Baseline Model Training and Validation Loss

3.2 Training on the CIFAR10 Dataset

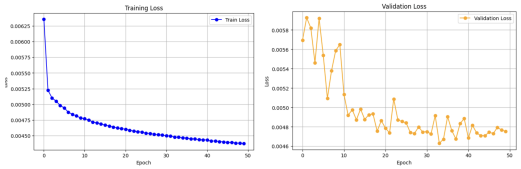


Figure 5: Hybrid Model Training and Validation Loss

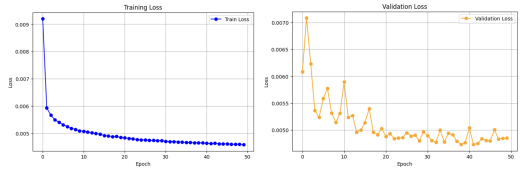


Figure 6: Baseline Model Training and Validation Loss

3.3 Training on the Flowers Dataset

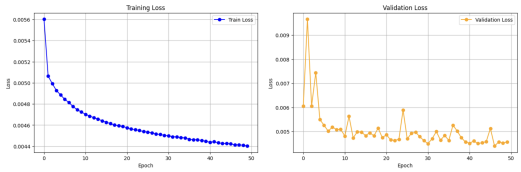


Figure 7: Hybrid Model Training and Validation Loss

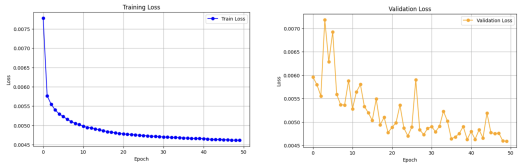


Figure 8: Baseline Model Training and Validation Loss

4 Discussion

4.1 Issues with the Dataset

To build a model capable of generalizing across a wide range of images, we intentionally trained on multiple datasets to help introduce more diversity in our inputs. However, each dataset came with inherent limitations that shaped model behavior and performance.

4.1.1 CIFAR10

We found that the CIFAR10 dataset faced challenges with the resolution of images. Although the dataset included 60,000 RGB images, they were only 32 x 32 pixels in size, which is not sufficient to capture detailed spatial information. To help mitigate this, we resized the images to 144 x 144 pixels using PyTorch's "resize" function (implements bilinear interpolation). However, the model performed worse when tested on true high-resolution data, showing that upsampling was not able to fully compensate for the absence of high-resolution detail in the original data.

4.1.2 Flowers102

This dataset was mainly limited by its relatively small quantity of images which led the model to overfit. To address this issue, we implemented data augmentation techniques involving random horizontal flips and rotations to help increase variability in the data. However, despite augmentation, the dataset's limited size remained a fundamental constraint, so we did not see significant improvements from this approach.

4.1.3 STL10

This dataset contained a large number of highly complex images, but it lacked consistency in the colorization of objects, which ultimately led to desaturated reconstructions of the images. The dataset also was intended for object classification and so, the images are most suitable for models focused on recognizing object outlines and structure. However, for proper reconstruction, it may be necessary to look at deeper heuristics, like surface textures and environmental cues, to help increase colorization accuracy.

4.2 Dataset Challenges

Due to the lack of datasets dedicated specifically for image colorization, we relied on publicly available datasets that were originally designed for tasks such as detection and classification as opposed to colorization. Therefore, we found that the quality and structure of these datasets were not well suited for the needs of colorization, particularly in the following four areas:

1. There was great variability in the colors of objects within the training images (e.g. buses, shirts, doors) which makes it difficult for the model to consistently select appropriate color outputs.

2. The small size of many images limited the model's ability to learn subtle variations in shades, which often resulted in a single dominant color being predicted for many objects (e.g. sky, trees).

3. We chose to convert RGB images to grayscale for training, but it is important to note that grayscale images do not necessarily correspond to corrected monochrome images. In this process, we may have lost or misrepresented certain color information.

4. Certain datasets could have introduced biases due to disproportions in the type of scenes represented. For example, models trained on images of indoor scenes, tended to bias predictions towards warmer, yellow

tones. Meanwhile images of outdoor scenes could bias predictions to more green and blue tones. These patterns could be affecting the model's ability to generalize and produce accurate colorizations across different image contexts.

Creating a dataset that satisifies these criteria's would ensure that color variablity, low resolution, and lighting variablity don't have as much of a negative impact on the model's performance.

4.3 Dataset Improvements

To improve model performance, we need to design a tailored dataset that directly addresses the challenges previously outlined. This will improve model performance via ensuring better data. The dataset needs to satisfy the following:

1. Contains a large pool of high-resolution images with dimensions of at least 256 x 256 pixels.
2. Has standardized colors (e.g. carrots should only be orange across the whole dataset)
3. Provides standalone RGB and grayscale images for each item, for which the grayscale images' luminosity is corrected for color temperature, lighting and saturation.

4.4 Model Limitations

The primary limitations our model faced was the oversimplification of the color in our formulation. In particular, our assumption was that since the RGB schema is based on three dimensions, we could transform back and forward from such dimensionality despite losing information. This is a naive assumption as color space is actually a lot more complex and importantly not euclidean. A movement in one direction will not display the same changes in strength as another. This means that our initial assumption resulted in an inherently limited model.

$$\hat{\mathbf{v}} = \mathbf{v}_{\|} + \left[(\mathbf{a}^\top \mathbf{z})\mathbf{u_1} + (\mathbf{b}^\top \mathbf{z})\mathbf{u_2}\right]$$

This is restricted and flawed from the start; we are limiting our output space to be a linear combination of our predefined patterns/features.

And while our model is mathmathically seperated and independent, this is only in RGB space: it doesn't match human's visual separation of brightness and color. A certain shade of color in greyscale can map to a large set of actual colors, but each of these actual colors may have a different shade that does not always correspond to the decomposed greyscale scalar. In our project this was good enough for approximations with high biased features, so we were still able to capture certain colors.

The second limitation of our model is that our loss function, MSE, makes the assumption that each pixel is independent of the others. In the context of the model, this encourages it to precisely map a color to a shape when in fact certain objects do not have a defined color. This is best explained by comparing two cars of the same brand that have different colors, and therefore there are two "correct" answers. For such reasons, color is often considered a multimodal problem and requires a model more complex than a simple CNN. When we end up backpropagating, we end up with an average of the two. Thus, we see a lot of average yellow colors with certain labels and low saturation, neutral tones.

4.5 Model Improvements

The most direct improvement we can make is to include more complex kernels that capture more complex patterns in our fixed features. Our currennt model only uses a few basic models. Another possible improvement we can make is to add a learned bias term derived from CNN. It could break the linearity constraint we enforced on our hybrid model, but we think it's a worthy tradeoff for recoloring accuracy.Another potential improvement is test with different loss functions and convert to Lab space. As we previously discussed, the RGB space is not perceptually seperated like the Lab space.

One direction for further research and development that we could take, could be implementing classification into our model. As discussed previously, one of the main challenges that our model faced was in situations where the CNN identified two different color possibilities and instead of picking one of two, it outputs an average of the two. This leads to inaccuracies and washed out results. Hwang and Zhou [5] show that the conversion from regression to classification can be an effective approach in addressing this problem. Instead of predicting chromatic values directly, one can create a CNN that groups specific chromatic values into separate classes. The model can then compute the probabilities of each of the classes and make a final decision based on those results. Furthermore, classification of image categories could be implemented as a means of assisting the model in narrowing down the set of possible colors to choose from. The model would ideally learn which subset of colors is most appropriate for specific categories of images, improving the overall accuracy.

Further research related to other deep learning architectures could also prove to be helpful in improving our model. For instance, we could expand on our current CNN architecture and integrate Generative Adversarial Networks (GAN). The addition of a generator and discriminator gives the model the ability to create artificial images. These images are output by the generator and passed to the discriminator. Given the created image and the ground truth image, the discriminator then attempts to distinguish which of the images is not a product of the generator. The network performs optimization on the loss of the discriminator to improve the output of the generator. Data suggests that the use of a GAN results in more accurate results thanuo et al [6] proposed a method of utilizing GANs to address challenges relating to edge

detection individual CNNs. We could apply this approach to our model as a means of addressing issues with color drift and the washing out of colors.

5 Conclusion

This project provides a way for everyday people to conveniently convert black and white images to RGB color with considerable accuracy. By leveraging Convolutional Neural Networks (CNNs), we are able to expedite this previously labor-intensive process. Our approach involved first converting a selected color photo dataset into grayscale photos which our models can then train on to learn and infer colors of both variations of the photo. This project also explores experimenting with multiple models, one with and without the use of the hybrid Structural Linear Regression model (SLR), to compare how it affects the model's learning and results.

Despite the limitations that this project represents, there is still much work to be done to improve the accuracy of our testing results. Our current training and testing datasets aren't consistent or numerous enough to confidently give a more accurate result. Having a larger dataset with a decently-sized set amount of subjects would allow the model to standardize colors with certain objects, increasing the model's accuracy overall. Another factor to consider for increasing accuracy is to make sure the training dataset consists of higher resolution images to help define shapes easier and further prevent color drift. On a more fundamental level, we could also experiment with implementing additional models to test with, such as the Generative Adversarial Network, to further experiment with different color prediction methods to try and attempt to find a more accurate approach to colorization. Despite having several possibilities to pivot to in the future, our system represents the beginnings of making colorization tasks streamlined and accessible.

References

[1] A.N. "Computer Process Brightens Surveyor Moon Pictures." Jet Propulsion Laboratory, California Institute of Technology, 09 August 1966, https://www.jpl.nasa.gov/news/computer-process-brightens-surveyor-moon-pictures

[2] R. Raj, and A. Kos. "An Extensive Study of Convolutional Neural Networks: Applications in Computer Vision for Improved Robotics Perceptions." MDPI, 9 February 2025, https://www.mdpi.com/1424-8220/25/4/1033.

[3] M. Abadi, A. Agarwal, P. Barham, et.al, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[4] J. Ramirez, "Photoshop Neural Filters powered by AI." Adobe, https://www.adobe.com/products/photoshop/neural-filter.html.

[5] J. Hwang, Y. Zhou, "Image Colorization with Deep Convolutional Neural Networks." Stanford.edu, 7 January 2016, https://cs231n.stanford.edu/reports/2016/pdfs/219_Report.pdf

[6] L. Guo, L. Liang, J. He, et al, "Gray Scale Image Coloring Method Based on GAN." Association for Computing Machinery, 26 August 2020, https://doi.org/10.1145/3414274.3414485

[7] J. Su, H. Chu, and J. Huang, "Instance-Aware Image Colorization." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 21 May 2020, pp. 7968–7977, https://doi.org/10.48550/arXiv.2005.10825

[8] M. N. Islam, A. Anand, M. Mandal, et al, "SAR Image Colorization Using cGAN and Gradient Difference in LAB Color Space." IEEE Xplore, 30 October 2025, https://doi.org/10.1109/ICAIET65052.2025.11211103.

[9] R. Zhang, P. Isola, A. A. Efros, "Colorful Image Colorization." University of California, Berkeley, 5 October 2016, https://doi.org/10.48550/arXiv.1603.08511

[10] H. Chang, O. Fried, Y. Liu, et al, "Palette-based Photo Recoloring." IEEE Digital Library, 27 July 2015, https://doi.org/10.1145/2766978

Author Contributions

Anna Chan: Oversaw group organization and coordination, wrote and finalized the report in LaTeX, and wrote and finalized the presentation.

Richard Lewins: Proposed the initial idea, organized the GitHub repository, and developed backend code/model loading for the demo website.

Tiffany Tran: Implemented the frontend of the demo website, created slides explaining the mathematical methodology, and revised the slides and report for quality.

Alexandr Volkov: Organized and coordinated the group, located datasets, trained hyperparameter models, performed dataset augmentation, and contributed to report writing.

Jasmine Saldano: Organized group and deadlines, implemented the frontend and backend for the demo website, and contributed to report writing.

Qihan Guan: Conducted process research, designed the hybrid model, and contributed to report and slide writing.

Karim Shami: Designed the baseline model, trained models, created data visualizations, and contributed to report and slide writing.

Kaylie Lam: Trained hyperparameter models and contributed to report and slide writing.

Steven Liu: Trained models, performed data preprocessing, and contributed to report and slide writing.

Shreya Vallala: Contributed to report and slide writing and revisions.

Navin Klein: Performed data preprocessing, data augmentation, data integration, identified effects of different datasets, tuned hyperparameters, and contributed to report writing.

Nathan Castellon: Contributed to report and slide writing and revisions.