# Black and White to RGB Color

ECS 171 Group 3
Fall 2025
Instructor Ilias Tagkopoulos

AUTO ISO 90 ISO 100 ISO 200 ISO 400 ISO 600

# Table of contents

# 01 Introduction

AUTO ISO **90** ISO **100** ISO **200** ISO **400** ISO **600**

# The Manual Approach

## Background

- Older photos and films exist only in black and white
- Colorization done by hand by referencing knowledge of natural colors and historical trends

## Problem

- Manual restoration methods are time-consuming and expensive
  - Limits accessibility and scalability

# Our Idea

Use concepts learned in class with computer vision ideas to make a Convolution Neural Network (CNN)

# 02 Background & Math

AUTO ISO 90 ISO 100 ISO 200 ISO 400 ISO 600



# How do we represent color?




# The Math: Grayscale

$$Y \in \mathbb{R}^{H \times W}$$

*Height (pixels)* *Width (pixels)*

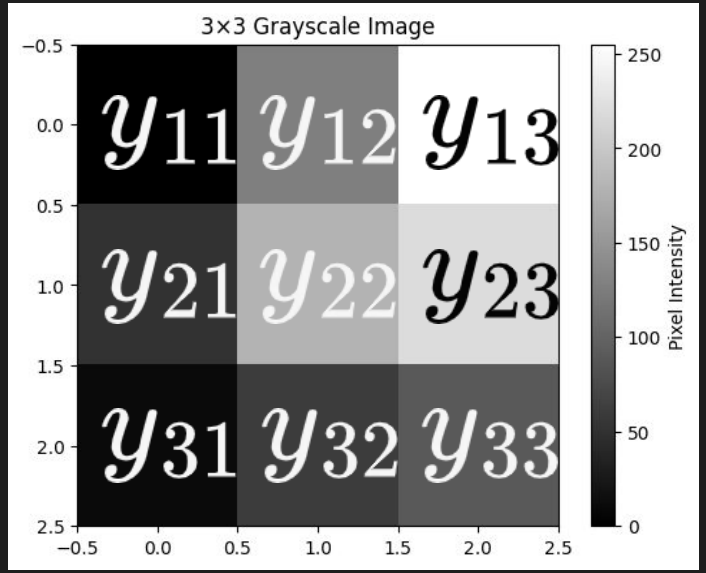
***2D Grayscale Image (matrix)***

Each pixel in the image is stored as a scalar in the matrix, which represents the brightness of the pixel. (ranges from 0: black → 255: white)

$$y_{ij} \in \{0, 1, \ldots, 255\}$$

***Intensity/Brightness of Pixel (scalar)***

$$Y^{3\times3} = \begin{bmatrix} 0 & 127 & 255 \\ 50 & 180 & 220 \\ 10 & 60 & 90 \end{bmatrix}$$



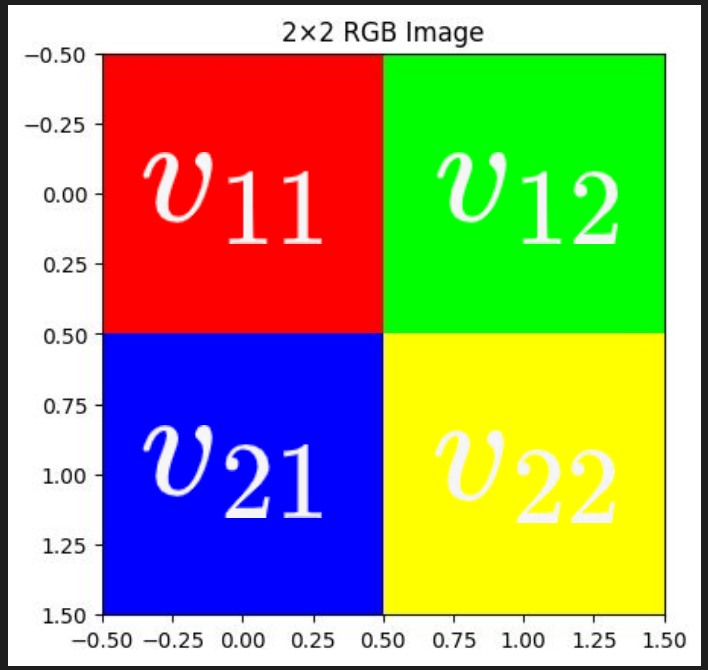
# The Math: Colored

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

*3D Colored Image (tensor)*

*Height (pixels)* · *Width (pixels)* · *RGB Color Channels*

$$V^{2\times2\times3} = \begin{bmatrix} [(255,0,0)] & [(0,255,0)] \\ [(0,0,255)] & [(255,255,0)] \end{bmatrix}$$




Each pixel is stored as a 3D vector, where each component represents the intensity of the red, green, and blue color channels, respectively.

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

*3D pixel vector*

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

*Color Intensity of Pixel (scalar)*

# From RGB to Grayscale

**We want:**

$$f(I_{\text{gray}}) \rightarrow I_{\text{color}}$$

$$f(\mathbb{R}^{H \times W}) \rightarrow \mathbb{R}^{H \times W \times 3}$$

***2D Grayscale Matrix*** ***3D Colored Tensor***

**Convert grayscale to RGB**

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

$$y_{ij} \in \mathbb{R} \qquad \mathbf{v}_{ij} \in \mathbb{R}^3$$

***2D Grayscale Matrix*** ***3D RGB Tensor***

---

We can project the grayscale part $y_{ij}$ as a linear combination of the RGB components of the colored pixel $\mathbf{v}_{ij}$.

$\mathbf{w}$ is a constant weight vector that specifies the contribution of each RGB channel to pixel brightness.

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

***given standard BT.601 weights***

$$\begin{aligned} y_{ij} &= \mathbf{w}^\top \mathbf{v}_{ij} \\ &= [w_R \quad w_G \quad w_B] \begin{bmatrix} R_{ij} \\ G_{ij} \\ B_{ij} \end{bmatrix} \\ &= w_R R_{ij} + w_G G_{ij} + w_B B_{ij} \end{aligned}$$



# How do we predict the colored vector?

# From Grayscale to RGB

**We want:**

Solve for RGB vector $\mathbf{v}$ given $y = \mathbf{w}^\top \mathbf{v}$.

$$\mathbf{v} = \text{proj}_{\mathbf{w}}(\mathbf{v}) + (\mathbf{v} - \text{proj}_{\mathbf{w}}(\mathbf{v}))$$

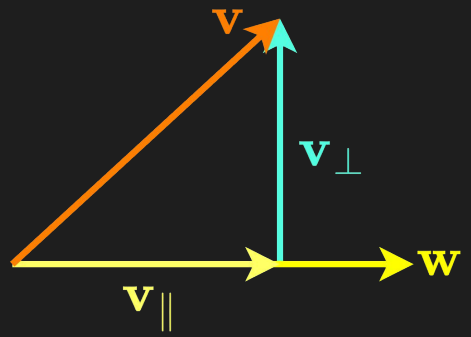
$$= \underbrace{\frac{\mathbf{w}^\top \mathbf{v}}{\mathbf{w}^\top \mathbf{w}} \mathbf{w}}_{\text{parallel to } \mathbf{w}} + \underbrace{\left(\mathbf{v} - \frac{\mathbf{w}^\top \mathbf{v}}{\mathbf{w}^\top \mathbf{w}} \mathbf{w}\right)}_{\text{orthogonal to } \mathbf{w}}$$

$$= \underbrace{\frac{y}{\mathbf{w}^\top \mathbf{w}} \mathbf{w}}_{\text{brightness part of } \mathbf{v}} + \underbrace{\left(\mathbf{v} - \frac{y}{\mathbf{w}^\top \mathbf{w}} \mathbf{w}\right)}_{\text{the missing part of } \mathbf{v}} \quad \text{as } y = \mathbf{w}^\top \mathbf{v}$$

$$= y \underbrace{\left(\frac{1}{\mathbf{w}^\top \mathbf{w}} \mathbf{w}\right)}_{\text{direction of } \mathbf{w}} + \underbrace{\left(\mathbf{v} - y\frac{1}{\mathbf{w}^\top \mathbf{w}} \mathbf{w}\right)}_{\text{color component of } \mathbf{v} \text{ without luminosity (chroma)}}$$

luminosity scalar $\in \{0, 1, \ldots, 255\}$

$$= \mathbf{v}_{\parallel} + \mathbf{v}_{\perp}$$




RGB vector $\mathbf{v}$ can be decomposed into a luminosity component $\mathbf{v}_{\parallel}$ (brightness) and a chroma component $\mathbf{v}_{\perp}$ (color without brightness).

This gives us the following constraint:

$$\mathbf{w}^\top (\mathbf{v} - \text{proj}_{\mathbf{w}}(\mathbf{v})) = \mathbf{w}^\top \mathbf{v}_{\perp} = 0$$

# From Grayscale to RGB

We have the constraint: $\mathbf{w}^\top \mathbf{v}_\perp = 0$ $\mathbf{w} = \begin{bmatrix} w_R \\ w_G \\ w_B \end{bmatrix}$ $\mathbf{v}_\perp = \begin{bmatrix} v_R \\ v_G \\ v_B \end{bmatrix}$

Expanding this, we get:

$$w_R v_R + w_G v_G + w_B v_B = 0$$

Then, we can solve for one variable, say $v_B$: $v_B = -\dfrac{w_R v_R + w_G v_G}{w_B}$

Rewriting $\mathbf{v}_\perp$ with this:

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

$v_R$ and $v_G$ are free parameters, while $v_B$'s value is constrained by orthogonality to $\mathbf{w}$.

The constraint defines a 2D subspace of $\mathbb{R}^3$, so $\mathbf{v}_\perp$ can be expressed as a linear combination of two linearly independent basis vectors.

# From Grayscale to RGB

**We want:**

Solve for RGB vector $\mathbf{v}$ given $y = \mathbf{w}^\top \mathbf{v}$.

Choose any two linearly independent vectors:

$$\mathbf{u}_1 = \begin{bmatrix} u_{1R} \\ u_{1G} \\ u_{1B} \end{bmatrix}, \ \mathbf{u}_2 = \begin{bmatrix} u_{2R} \\ u_{2G} \\ u_{2B} \end{bmatrix} \in \mathbb{R}^3 \quad \text{such that} \quad \mathbf{w}^\top \mathbf{u}_1 = \mathbf{w}^\top \mathbf{u}_2 = 0$$

**Orthogonality Constraint**

Then any possible chroma vector $\mathbf{v}_\perp$ can be represented as a linear combination of them.

$$\mathbf{v}_\perp = \alpha\,\mathbf{u}_1 + \beta\,\mathbf{u}_2, \quad \alpha, \beta \in \mathbb{R}$$

**So we can finally represent the RGB vector $\mathbf{v}$ as:**

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

^ Luminance ^ Chroma

# 03 Data and Methods

AUTO ISO 90 ISO 100 ISO 200 ISO 400 ISO 600

# Test Idea and Hypothesis

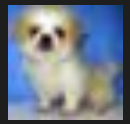
## Hybrid:

- No color bleeding and leakage, particularly through the center
- Performs better when the given image features captured in our fixed features **z**

## Baseline CNN:

- Perform better in more types of images (Lower validation loss)
- Color leakage, breaking boundary

# Datasets Used



CIFAR dataset: Large number of samples (60k images), but low quality (32x32)
Hard for models to learn features

Oxford flower dataset: Small dataset size (~10k images), high resolution (128x128)
Dataset of only UK flowers, not generalizable

STL10 dataset: Dataset inspired by CIFAR, higher resolution (96x96)
Large number of samples: 100k unlabeled images,
5k labeled training images
Good dataset with complex images, inconsistent colorization of images

# Data Preprocessing

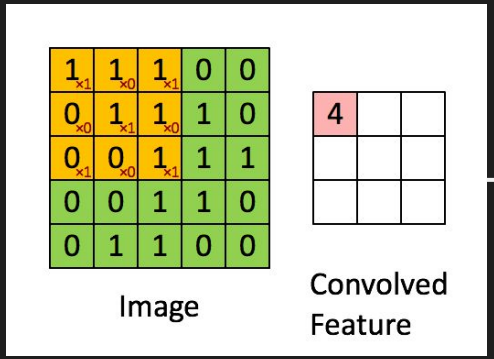
1. Conversion to RGB
   The images grayscale part are extracted using broadcast standards rgb weights ; then, colorization models are applied to analyze their performance.

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

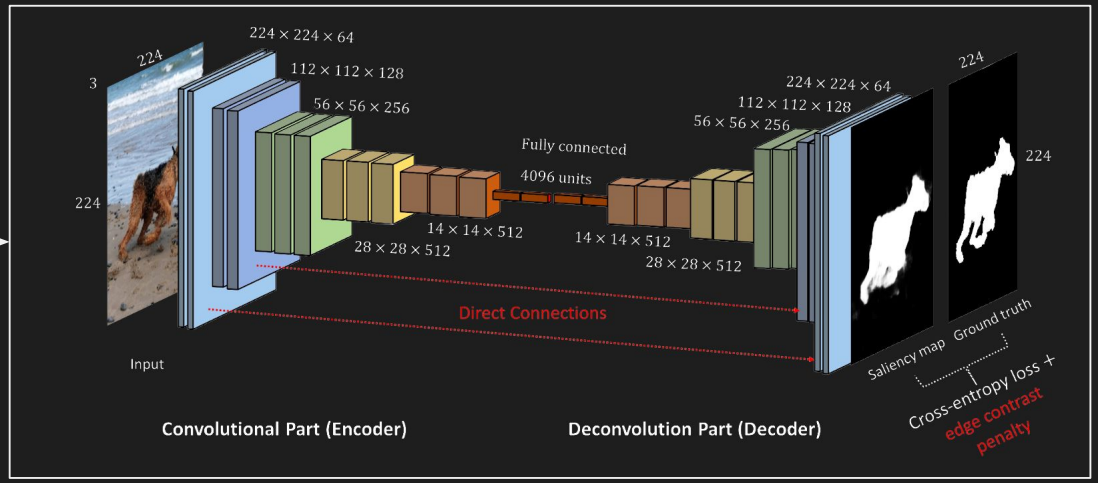
2. Resizing to fit tensor sizes: resize to 144×144, then center-crop to 128×128.
3. For better training:
   - Add random horizontal flips.
   - Add small random rotations (±10 degrees).

# CNN's and Encoder-Decoder Arch.

- CNNs can have Encoder-Decoder structures (somewhat like autoencoders)
- Convolution: Large features → compressed representation
  - Learn important parts of an image
- Deconvolution: Recreate image



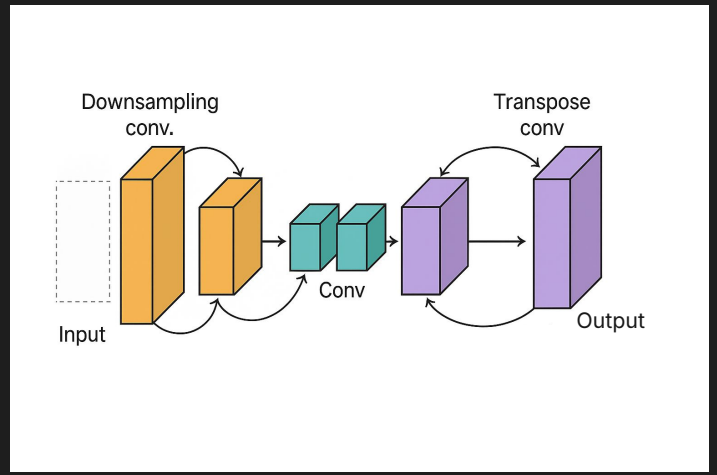

*Kernel Demonstration*




*Generalized CNN with image rebuilding*

# CNN Encoder-Decoder Arch.

- 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→64→32→2
- Output: predicted chroma coefficients (α, β)



# Optimizer

- Our model uses the Adam (Adaptive Moment Estimation) algorithm for optimizing the parameters
- Adam: 2 considerations
  - Momentum optimization (gradient descent that considers the intensity of previous gradients)
  - RMSProp optimization (dynamic learning rate using previous gradients with an emphasis on recent iterations)
- Combining the 2 results in:
  - Adaptable learning rate
  - More efficient optimization
  - Quicker gradient descent

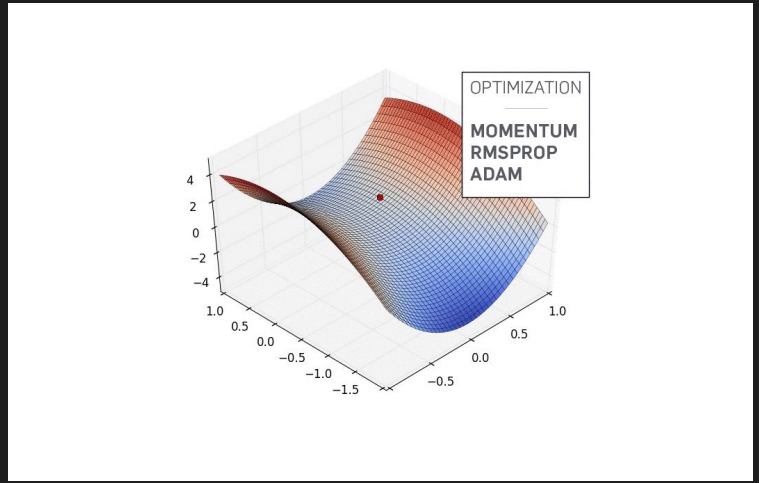



Diagram courtesy of Ayoosh Kathuria
https://www.digitalocean.com/community/tutorials/intro-to-optimization-momentum-rmsprop-adam

# Baseline CNN Formulation

Input:

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

Features Extraction and Normalization:
**For layer $\ell$ = 1,...., L - 1**

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

Activation function: Linear -> Non-Linear.
**For layer $\ell$ = 1,...., L - 1**

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

Loss Function(MSE)

$$\mathcal{L} = \frac{1}{N} \sum_{i,j} \left\| \mathbf{v}_{ij} - \widehat{\mathbf{v}}_{ij} \right\|^2$$

Output Layer:

$$\mathbf{z}^{(L)} = \begin{bmatrix} \mathbf{z}_1^{(L)} \\ \mathbf{z}_2^{(L)} \end{bmatrix} = \begin{bmatrix} \langle K^{(L,1)}, \mathbf{h}^{(7)} \rangle_F + b_1^{(L)} \\ \langle K^{(L,2)}, \mathbf{h}^{(7)} \rangle_F + b_2^{(L)} \end{bmatrix}$$

Parameter Extraction

$$\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1^{(L)} \\ \mathbf{z}_2^{(L)} \end{bmatrix}$$

Model

$$\widehat{\mathbf{v}} = \frac{Y}{\mathbf{w}^\top \mathbf{w}} \mathbf{w} + \boldsymbol{\alpha} \mathbf{u}_1 + \boldsymbol{\beta} \mathbf{u}_2$$

# Hybrid Formulation

Input:

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

Loss Function(MSE):

$$\mathcal{L} = \frac{1}{N} \sum_{i,j} \left\| \mathbf{v}_{ij} - \widehat{\mathbf{v}}_{ij} \right\|^2$$

Features Extraction and Normalization (hidden layer): **For layer $\ell$ = 1,...., L - 1**

$$h_k^{(\ell)} = \mathrm{ReLU}\left(\mathrm{BN}\left(\langle K^{(\ell,k)}, \mathbf{h}^{(\ell-1)} \rangle_F + b_k^{(\ell)}\right)\right) \qquad \mathbf{z}_{\text{fixed}}^{(m)} = \langle K_{\text{fixed}}^{(m)}, Y \rangle_F$$

Output Layer: *predicting a,b*

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

Model:

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

**Kernels:**

- Laplacian: Clusters
- Sobel X: vertical lines
- Sobel Y: horizontal lines

# Baseline CNN Model

## Pros

- Conditional logic (continuous function):
  - Can learn non-linear relationships.
    - Non-linear mapping from Luma to chroma
- Generalized Features
- Adaptive Kernels

## Cons

- Misalignment and Gradient leak
- Unconstrained
  - Entire search space is 2D plane
  - No Structural assumptions
  - Learn structure and color
    - Often results in desaturation under MSE

# Hybrid Model

## Pros

- Lower (MSE) error during training
- Pre-defined feature extraction layer
- Edge Alignment (No gradient leak)
- Tighter constraint
  - Search space is defined by **z**
  - Only need to learn the coefficients

## Cons

- Longer training time (more parameters)
- Linearity
- Can’t do conditional logic(continuous linear map)
- Noise sensitivity

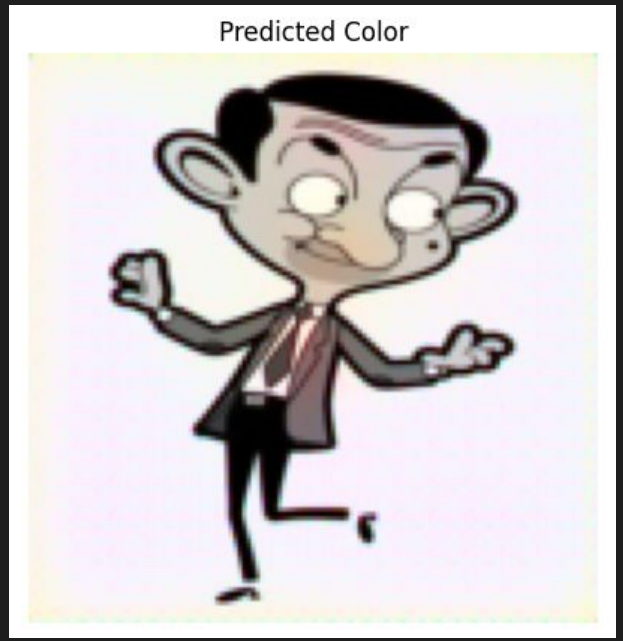
# 04 Results

AUTO ISO 90 ISO 100 ISO 200 ISO 400 ISO 600

# Hybrid (SLR + CNN) vs Base (CNN) Models

Gradient Leakage!

## Hybrid Model



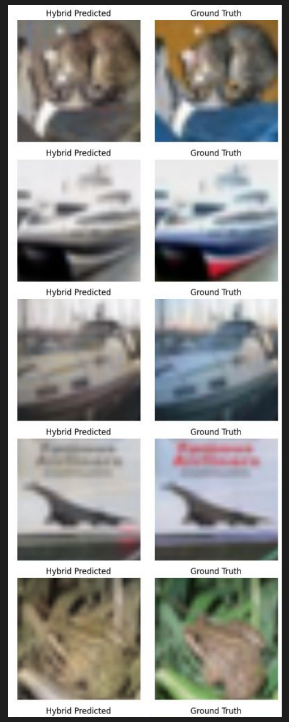



## Base Model

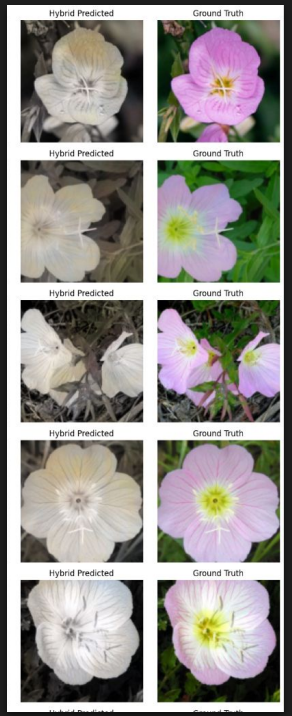
# Training on CIFAR10

## Hybrid Model

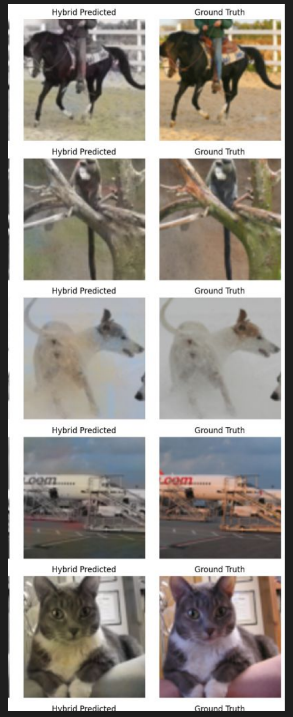
### CIFAR10




### Flowers

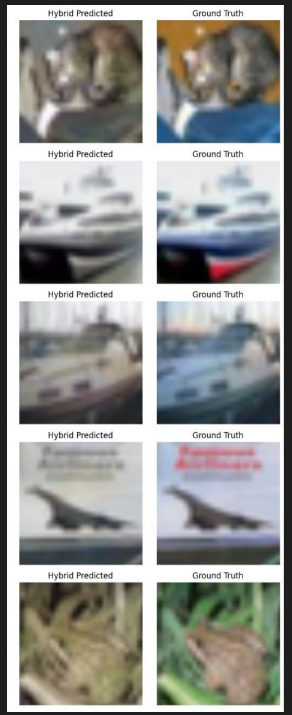



### STL10

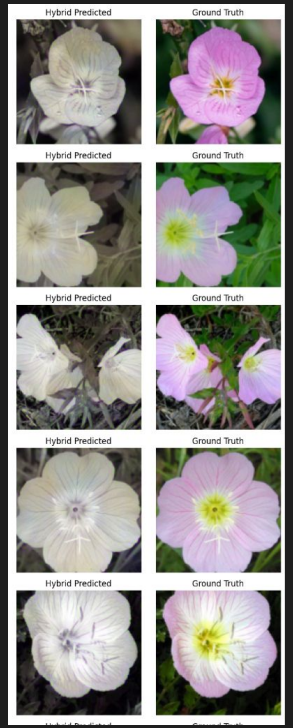



## Base Model

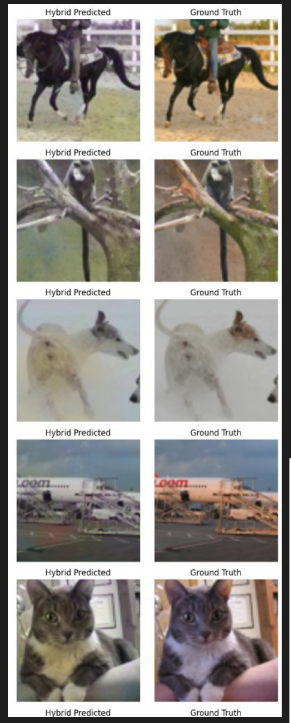
### CIFAR10




### Flowers




### STL10

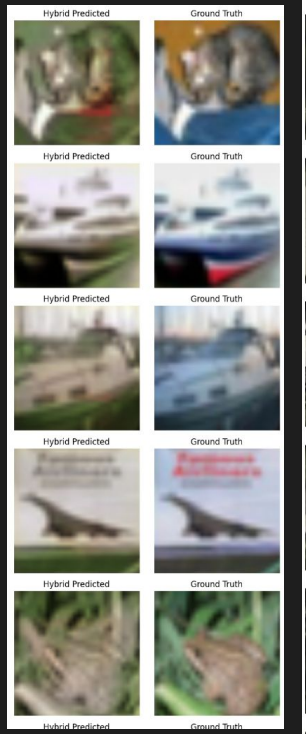


# Training on Flowers

Gradient Leakage!

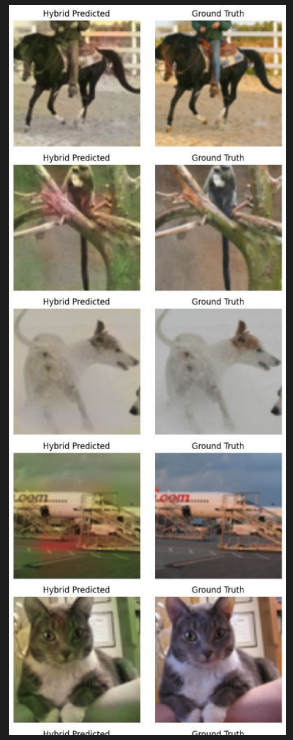
## Hybrid Model

### CIFAR10



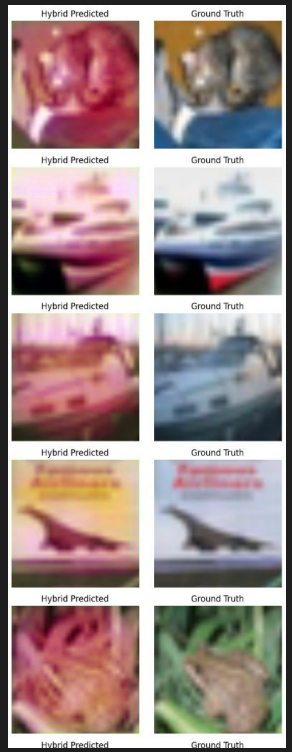

### Flowers


### STL10



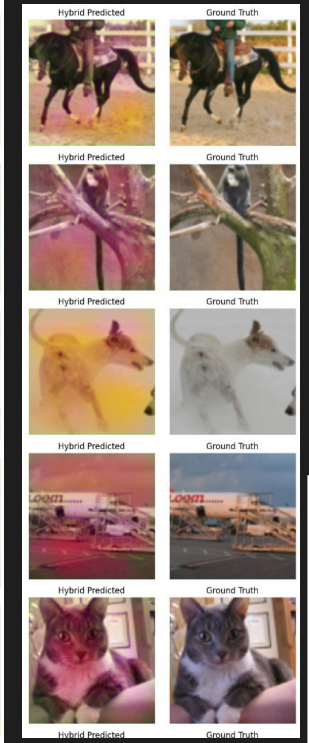

## Base Model

### CIFAR10




### Flowers


### STL10



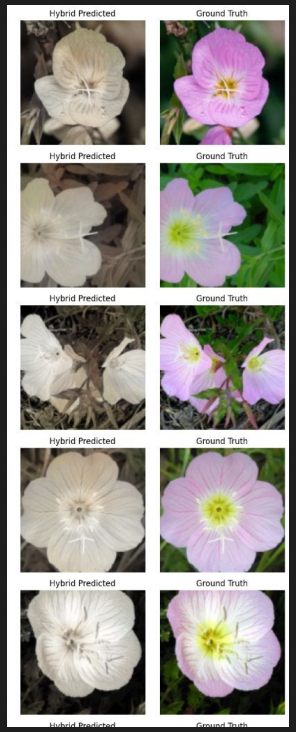
Best Performing

# Training on STL10

## Hybrid Model

### CIFAR10

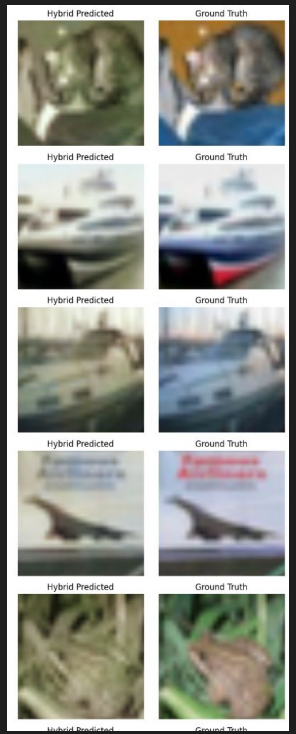

### Flowers



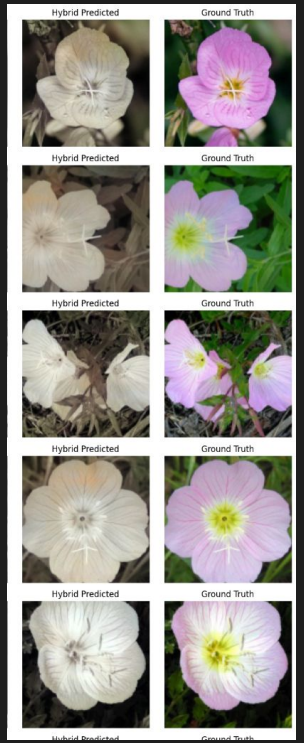

### STL10


## Base Model

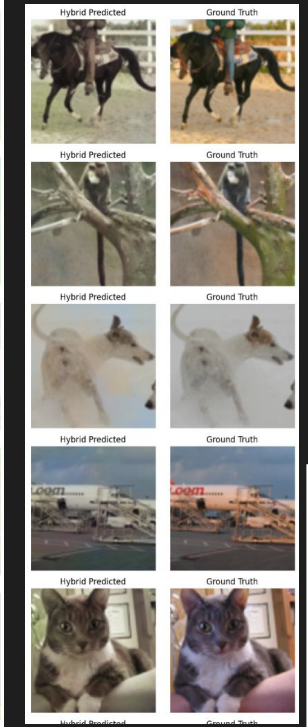
### CIFAR10




### Flowers




### STL10

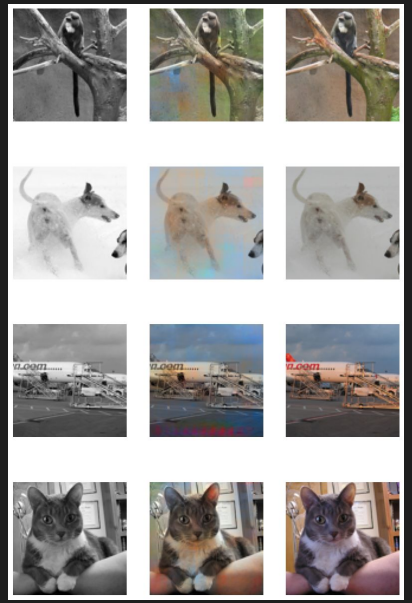


# Effect of Data Augmentation

## Hybrid Model

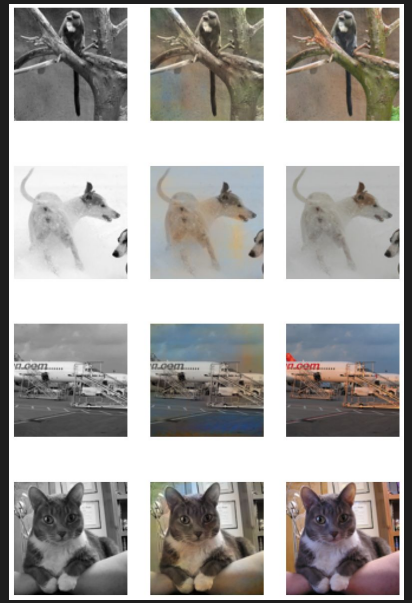
### Without

*Grayscale* *Predicted* *Ground Truth*



### With Data Aug

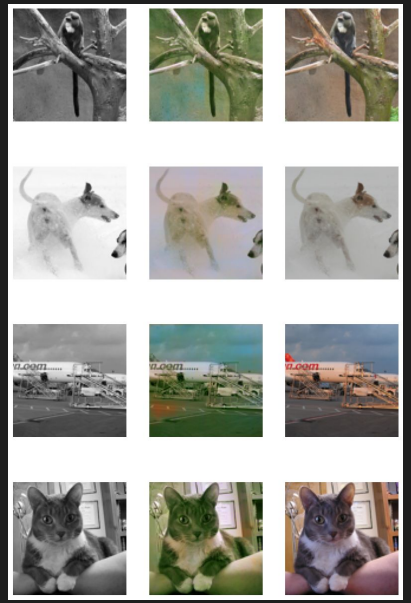
*Grayscale* *Predicted* *Ground Truth*



## Base Model

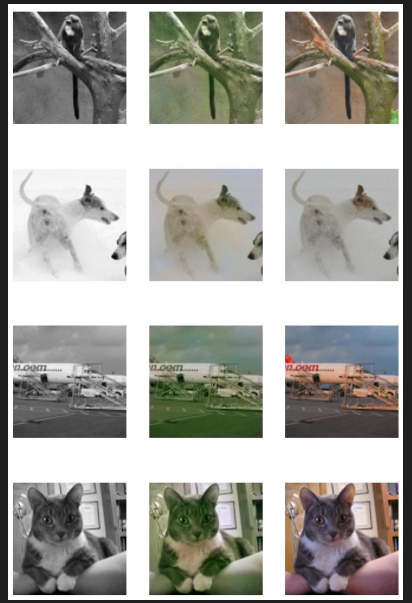
### Without

*Grayscale* *Predicted* *Ground Truth*

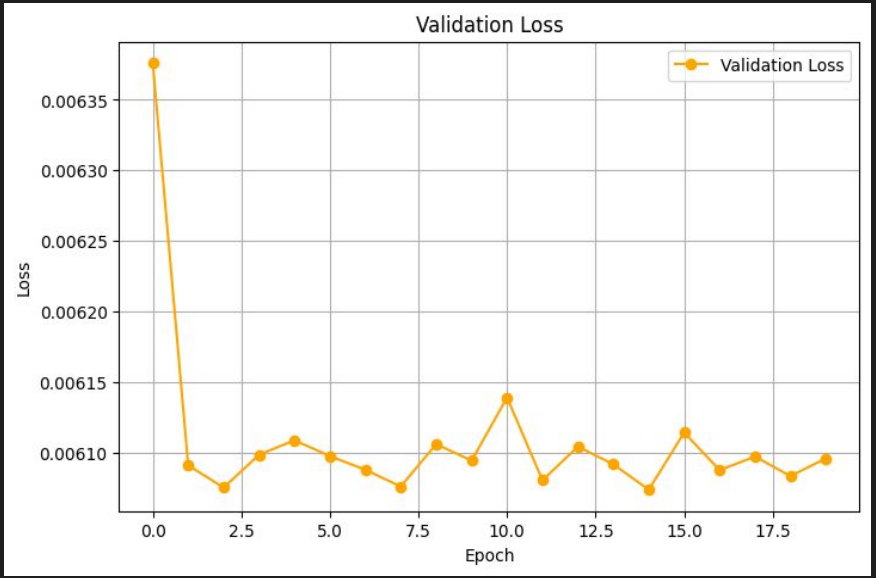


### With Data Aug

*Grayscale* *Predicted* *Ground Truth*

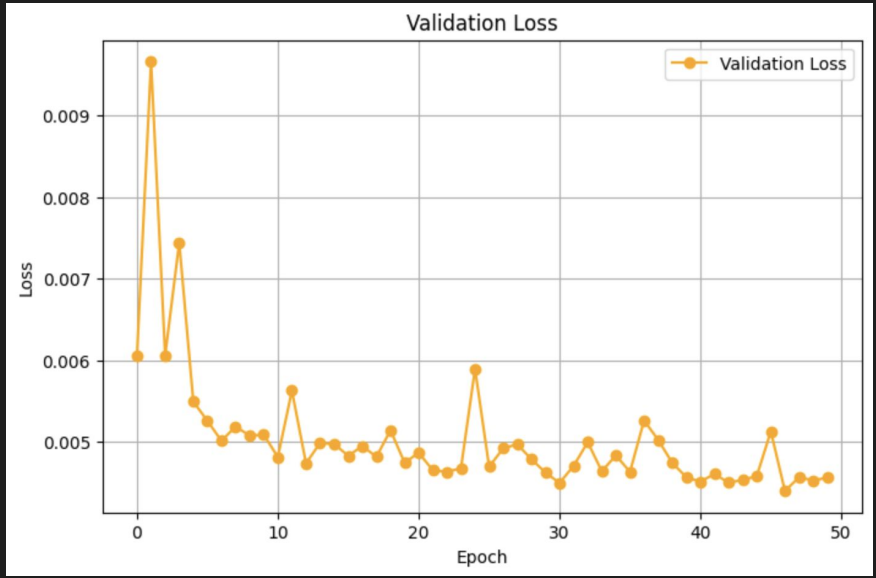


# Effect of no. layers on performance of CNN




1 layers CNN with hybrid structure

**Underfitting**



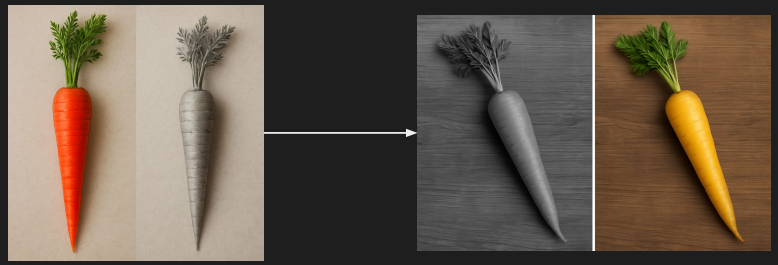

8 layers CNN with bottleneck layers, computed at 128 x 128

**Validation loss converges**

# 05 Discussion

AUTO ISO 90 ISO 100 ISO 200 ISO 400 ISO 600

# General Issues with Datasets



We relied on public datasets for tasks like detection and classification, not image colorization. This introduced a few notable issues:

- The colors of objects were not consistent
- Images were not large enough for our model to learn specific color shades
- We made grayscale images from RGB ones, but grayscale ≠ corrected monochrome images

There is a need to craft a proper benchmarking dataset that:

1. Ensures standardized colors across objects
2. Contains images of sufficient resolution
3. Has standalone RGB and grayscale images for each item

# Specific Issues with Datasets

- CIFAR10:
  - Required extensive upsampling (bilinear interpolation)
- Oxford Flowers:
  - Small amount of samples led to overfitting
- STL10:
  - Large amount of samples led to inconsistency amongst objects within training images

# Possible Improvements

- Ensemble learning
  - Combine multiple models together
- Test with a different module
  - Generative Adversarial Network to predict colors differently
- Have a more robust Dataset
  - Small datasets lead to overfitting
  - More high resolution images for more accurate testing and predictions
- Use Lab Space
- Different Loss Functions
- Classification
- Apply post-processing to remove artefacts using
  - Joint bilateral filtering
  - Semantic labels

# Our Takeaways

- The SLR + CNN model is effective at using familiar local patterns and structural features
- Makes “average” guesses
- Confirmed our hypothesis that the SLR + CNN model is a potential solution for the gradient leakage problem (no color bleeding)
- Both models have limitations when it comes to prediction accuracy, but show potential

# 06 Conclusion

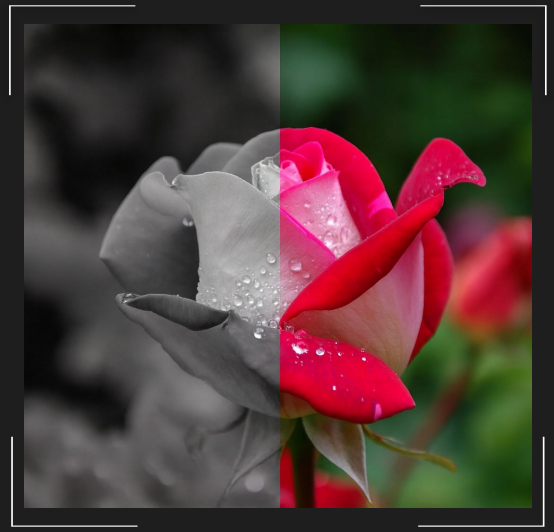
AUTO ISO 90 ISO 100 ISO 200 ISO 400 ISO 600

# Conclusion

- The use of automated methods for image colorization can help make this task more convenient and accessible

*Our hybrid SLR + CNN model shows potential for image colorization with its improvements in identifying key features within images and predicting RGB values from grayscale inputs, addressing the color drift and saturation issues within the baseline model.*

- Our findings emphasize the value in advancing image colorization methods to streamline not only archival and restoration practices, but also assist creative professionals in enhancing their work



# Demo

http://localhost:5000/

AUTO ISO 90 ISO 100 ISO 200 ISO 400 ISO 600

# Thank You

Questions?

AUTO ISO 90 ISO 100 ISO 200 ISO 400 ISO 600