Image Colorization

Sean Bremer & Aidan Szilagyi

December 15, 2025

Motivation

Problem
- Many archival images, historical photos, and creative projects are presented in grayscale
- Limits impact and accessibility

Solution
- Colorization can restore and enhance images, making them more engaging and informative

Target Task

- Infer realistic color channels from grayscale input
- Generative task

Related Work

original grayscale

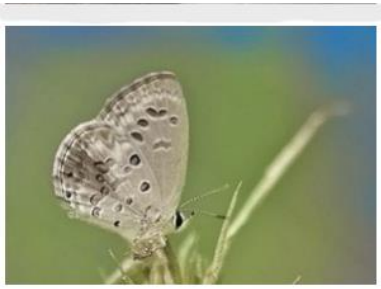
Anwar et al. (2024) attempts to survey and benchmark various colorization methods

Larsson et al. (2016) Identified Five Common Failure Cases:

Too Desaturated

Inconsistent Chroma

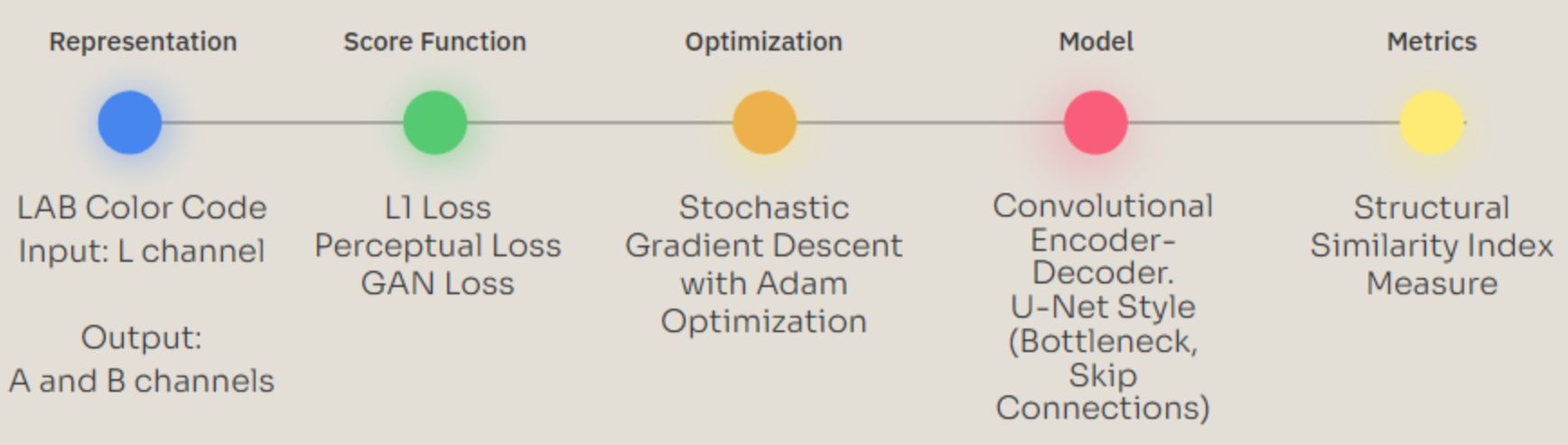
Inconsistent Hue



Edge Pollution

Color Bleeding

Proposed Solution



Representation (LAB Color Codes)
- Three axes color codes: greyscale, green-red, and blue-yellow
- LAB better models how the human eye perceives differences in color

Model (U-Net Encoder Decoder)
- Encoder extracts low-leaver features, decoder reconstructs high-level features

Optimization (Generative Adversarial Network)
- Generator takes grayscale input, generating colorized output
- Discriminator evaluates if the colorization is realistic, helping generator improve output

Metrics (Structural Similarity Index Measure)
- Compares structural similarity between predicted and ground truth colorized images
- Accounts for luminance, contrast, and structure

Implementation

ResNet-34 Encoder -> Our Decoder
Decoder:
Conv Channels: 512 -> 256 -> 128 -> 64
Each Conv has kernel size of channels/2 x channels/2

```
class ColorizationUNet(nn.Module):
    def __init__(self, pretrained=True):
        super().__init__()
        self.encoder = ResNet34Encoder(pretrained=pretrained)

        self.up4 = nn.ConvTranspose2d(512, 256, 2, stride=2)
        self.conv4 = nn.Sequential(
            nn.Conv2d(256 + 256, 256, 3, padding=1),
            nn.ReLU(inplace=True)
        )

        self.up3 = nn.ConvTranspose2d(256, 128, 2, stride=2)
        self.conv3 = nn.Sequential(
            nn.Conv2d(128 + 128, 128, 3, padding=1),
            nn.ReLU(inplace=True)
        )
```

Discriminator:
Conv Channels: 5 -> 64 -> 128 -> 256 -> 512 -> ReLU
Each Channel has 4x4 kernel with stride=2, halving the width and height each layer

```
class PatchGANDiscriminator(nn.Module):
  def __init__(self, in_channels = 5):
    super().__init__()

    self.conv1 = nn.Conv2d(in_channels, 64, kernel_size = 4, stride = 2, padding = 1)
    self.conv2 = nn.Conv2d(64, 128, kernel_size = 4, stride = 2, padding = 1)
    self.conv3 = nn.Conv2d(128, 256, kernel_size = 4, stride = 2, padding = 1)
    self.conv4 = nn.Conv2d(256, 512, kernel_size = 4, stride =2, padding = 1)
    self.conv5 = nn.Conv2d(512, 1, kernel_size = 4, stride = 1, padding = 1)

    self.leaky_relu = nn.LeakyReLU(0.2, inplace = True)
```

Data Summary

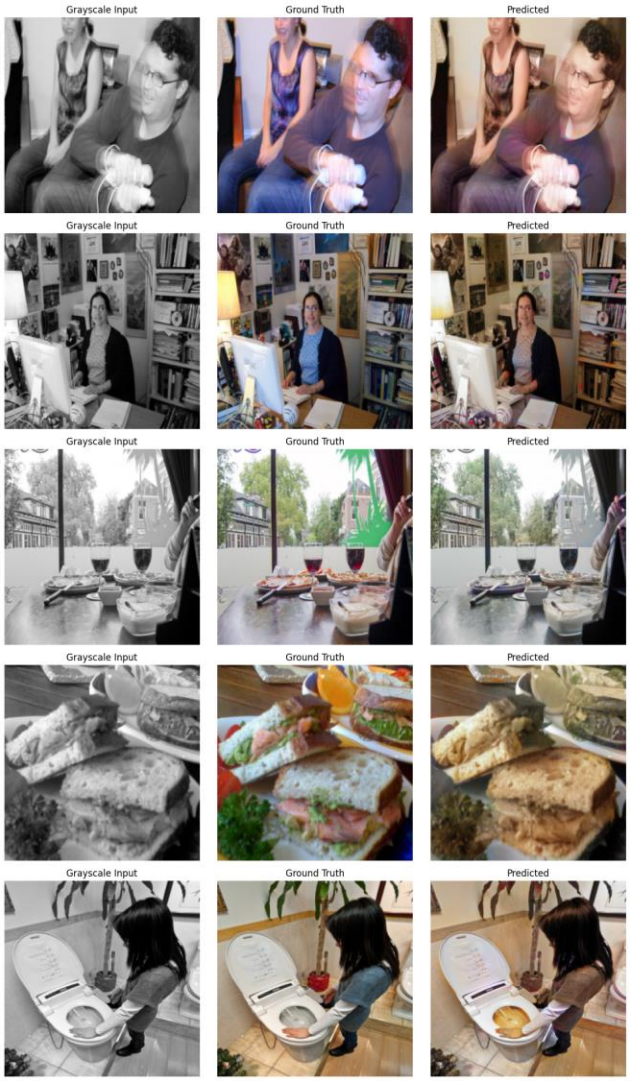
- Due to the large size of our training dataset, we trained our model on a randomized 10% of the dataset (roughly 370 images)
- For each epoch (10), we had the terminal print results every 100 steps (out of 370), depicting discriminator & generator loss (D & G loss)
- Sample run results:

```
Epoch [1/10], Step [1/370], D Loss: 1.3863, G Loss: 33.4629
Epoch [1/10], Step [101/370], D Loss: 1.2811, G Loss: 36.1519
Epoch [1/10], Step [201/370], D Loss: 1.3369, G Loss: 36.0646
Epoch [1/10], Step [301/370], D Loss: 1.1108, G Loss: 33.8154
Epoch [1/10], Step [370/370], D Loss: 1.2225, G Loss: 29.8015
Validation Results - L1 Loss: 0.0676, SSIM: 0.4171
Epoch [2/10], Step [1/370], D Loss: 1.2222, G Loss: 30.8697
Epoch [2/10], Step [101/370], D Loss: 0.6844, G Loss: 50.3742
Epoch [2/10], Step [201/370], D Loss: 0.8445, G Loss: 38.0917
Epoch [2/10], Step [301/370], D Loss: 0.8631, G Loss: 37.0352
Epoch [2/10], Step [370/370], D Loss: 1.0712, G Loss: 41.3892
Validation Results - L1 Loss: 0.0747, SSIM: 0.3219
Epoch [3/10], Step [1/370], D Loss: 0.9538, G Loss: 38.6591
Epoch [3/10], Step [101/370], D Loss: 0.7869, G Loss: 37.3652
Epoch [3/10], Step [201/370], D Loss: 0.6831, G Loss: 41.3719
Epoch [3/10], Step [301/370], D Loss: 0.5297, G Loss: 48.4916
Epoch [3/10], Step [370/370], D Loss: 0.8284, G Loss: 41.0376
Validation Results - L1 Loss: 0.0761, SSIM: 0.3259
Epoch [4/10], Step [1/370], D Loss: 0.7538, G Loss: 37.5349
Epoch [4/10], Step [101/370], D Loss: 0.6509, G Loss: 43.4072
Epoch [4/10], Step [201/370], D Loss: 0.7111, G Loss: 39.6706
Epoch [4/10], Step [301/370], D Loss: 1.1508, G Loss: 41.6996
Epoch [4/10], Step [370/370], D Loss: 0.9055, G Loss: 42.4359
Validation Results - L1 Loss: 0.0800, SSIM: 0.2707
Epoch [5/10], Step [1/370], D Loss: 0.8789, G Loss: 41.9285
Epoch [5/10], Step [101/370], D Loss: 0.8301, G Loss: 37.3074
Epoch [5/10], Step [201/370], D Loss: 0.8663, G Loss: 36.6406
Epoch [5/10], Step [301/370], D Loss: 1.0479, G Loss: 39.5751
Epoch [5/10], Step [370/370], D Loss: 0.8986, G Loss: 41.4412
Validation Results - L1 Loss: 0.0756, SSIM: 0.3513
```

```
Epoch [6/10], Step [1/370], D Loss: 1.0798, G Loss: 38.1668
Epoch [6/10], Step [101/370], D Loss: 1.1846, G Loss: 38.6542
Epoch [6/10], Step [201/370], D Loss: 1.0926, G Loss: 38.5350
Epoch [6/10], Step [301/370], D Loss: 0.8998, G Loss: 39.4373
Epoch [6/10], Step [370/370], D Loss: 1.1667, G Loss: 39.6570
Validation Results - L1 Loss: 0.0753, SSIM: 0.3149
Epoch [7/10], Step [1/370], D Loss: 1.0604, G Loss: 36.5707
Epoch [7/10], Step [101/370], D Loss: 1.2494, G Loss: 41.8965
Epoch [7/10], Step [201/370], D Loss: 1.2136, G Loss: 32.9834
Epoch [7/10], Step [301/370], D Loss: 1.0631, G Loss: 34.9567
Epoch [7/10], Step [370/370], D Loss: 1.0102, G Loss: 38.4644
Validation Results - L1 Loss: 0.0778, SSIM: 0.3572
Epoch [8/10], Step [1/370], D Loss: 1.4380, G Loss: 37.0367
Epoch [8/10], Step [101/370], D Loss: 1.0286, G Loss: 33.9818
Epoch [8/10], Step [201/370], D Loss: 1.1444, G Loss: 34.3082
Epoch [8/10], Step [301/370], D Loss: 1.1996, G Loss: 35.8631
Epoch [8/10], Step [370/370], D Loss: 1.3167, G Loss: 37.1101
Validation Results - L1 Loss: 0.0741, SSIM: 0.3730
Epoch [9/10], Step [1/370], D Loss: 1.0472, G Loss: 36.7639
Epoch [9/10], Step [101/370], D Loss: 1.2753, G Loss: 38.6405
Epoch [9/10], Step [201/370], D Loss: 1.2802, G Loss: 36.4594
Epoch [9/10], Step [301/370], D Loss: 1.0957, G Loss: 35.6124
Epoch [9/10], Step [370/370], D Loss: 1.2364, G Loss: 33.5947
Validation Results - L1 Loss: 0.0745, SSIM: 0.3703
Epoch [10/10], Step [1/370], D Loss: 1.0817, G Loss: 37.9398
Epoch [10/10], Step [101/370], D Loss: 1.1042, G Loss: 33.1013
Epoch [10/10], Step [201/370], D Loss: 1.0150, G Loss: 39.6250
Epoch [10/10], Step [301/370], D Loss: 1.0480, G Loss: 36.0761
Epoch [10/10], Step [370/370], D Loss: 0.9759, G Loss: 38.8151
Validation Results - L1 Loss: 0.0730, SSIM: 0.3871
```

Experimental Results

Validation Results –
L1 Loss: 0.0671
SSIM: 0.4375



Experimental Analysis

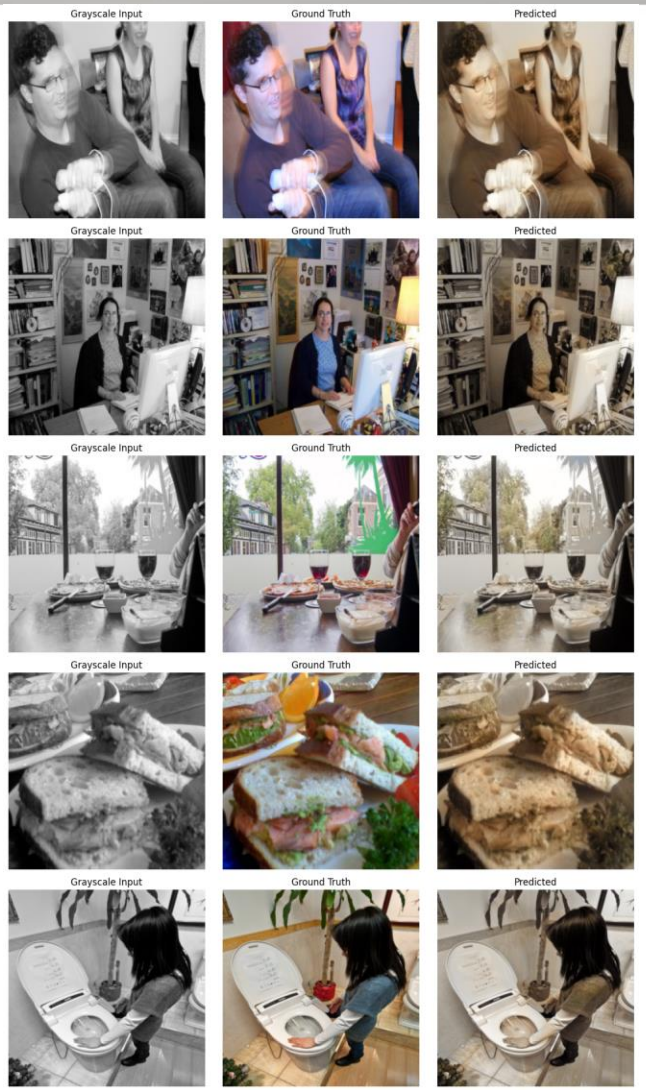
Loss

- Discriminator Loss
  - The discriminator appeared to fluctuate a loss from roughly 0.7 – 1.2, indicating the discriminator was very effective with discrimination
- Generator Loss
  - The generator appeared to fluctuate a loss from roughly 30 - 41
  - Could be a result of poor balance between the loss weights and the discriminator and generator complexity
  - Also, could be a result of a smaller dataset, with less training epochs
- L1 loss maintained a very low number
- SSIM did not have the greatest results, which ties into the potential challenges with the generator
- Ultimately, the model generated an image with relatively similar color; however, the colors were not as vibrant and were slightly off in certain regions

For Hinge GAN

Hinge Gan is a combination of the hinge loss function and cross-entropy loss. It performs slightly worse by the SSIM metric, but visually, colors appeared significantly more muted

- Validation Results - L1 Loss: 0.0675, SSIM: 0.4165

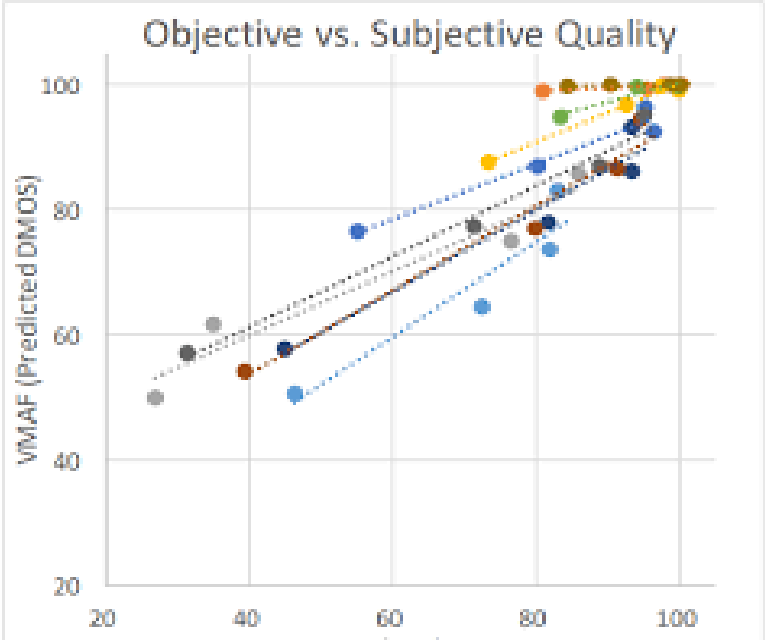


Conclusion and Future Work

Was (under)stated in Anwar et al., but image colorization has a problem with performance metrics misaligning with subjective results.

Some models that we trained had similar SSIM results, but one model's outputs were significantly more saturated than the other model's and looked much better because of it.

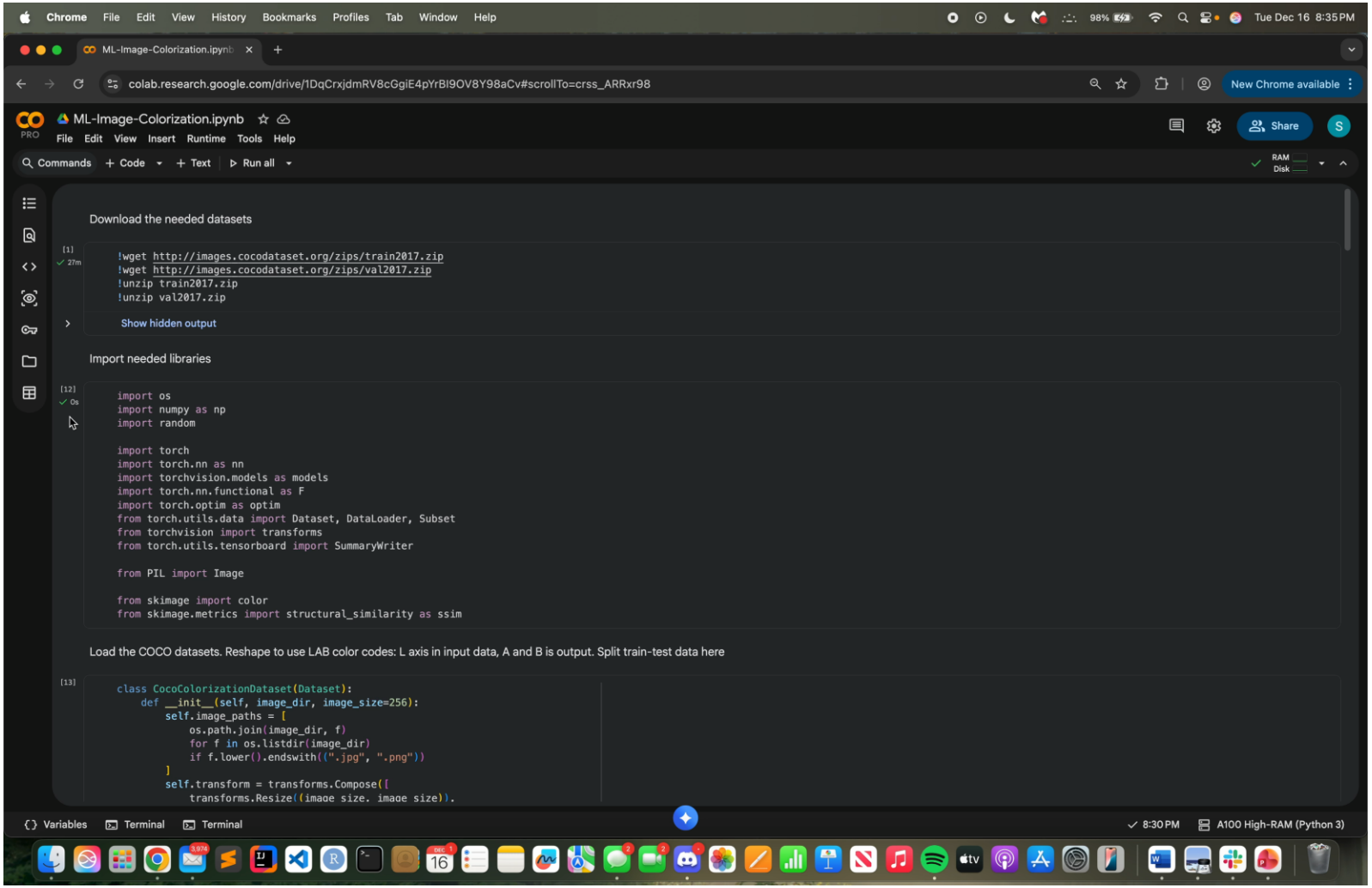
Models tend to take the average of the choices it has seen, rather find the most likely choice (L1 Loss)



Who did what?

- Find and preload dataset
  - Aidan
- Build pretrained model
  - Aidan
- Build decoder
  - Aidan
- Build discriminator and generator
  - Sean
- Build metric calculator and evaluator
  - Sean
- Build result visualizer
  - Sean
- Train and hyper tune model
  - Both

Video Demo


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Dec 16 8:35 PM
ML-Image-Colorization.ipynb
colab.research.google.com/drive/1DqCrxjdmRV8cGgiE4pYrBl9OV8Y98aCv#scrollTo=crss_ARRxr98
New Chrome available
ML-Image-Colorization.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
Download the needed datasets
[1]
27m
!wget http://images.cocodataset.org/zips/train2017.zip
!wget http://images.cocodataset.org/zips/val2017.zip
!unzip train2017.zip
!unzip val2017.zip
Show hidden output
Import needed libraries
[12]
0s
import os
import numpy as np
import random
import torch
import torch.nn as nn
import torchvision.models as models
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, Subset
from torchvision import transforms
from torch.utils.tensorboard import SummaryWriter
from PIL import Image
from skimage import color
from skimage.metrics import structural_similarity as ssim
Load the COCO datasets. Reshape to use LAB color codes: L axis in input data, A and B is output. Split train-test data here
[13]
class CocoColorizationDataset(Dataset):
def __init__(self, image_dir, image_size=256):
self.image_paths = [
os.path.join(image_dir, f)
for f in os.listdir(image_dir)
if f.lower().endswith((".jpg", ".png"))
]
self.transform = transforms.Compose([
transforms.Resize((image_size, image_size)),
Variables Terminal Terminal
8:30 PM
A100 High-RAM (Python 3)

References

Larsson, G., Maire, M., Shakhnarovich, G.: Learning representations for automatic colorization.In: European Conference on Computer Vision, pp. 577–593. Springer (2016)

Goni, O., Arka, H. S., Halder, M., Shibly, M. M. A., & Shatabda, S. (2025, March 24). *HINGERLC-gan: Combating mode collapse with Hinge loss and RLC regularization*. arXiv.org. https://arxiv.org/abs/2503.19074

Anwar, S., Tahir, M., Li, C., Mian, A., Li, C., Khan, F. S., & Muzaffar, A. W. (2024, September 2). *Image Colorization: A Survey and Dataset*. Image colorization: A survey and Dataset. https://arxiv.org/html/2008.10774v4

Zanforlin, M., Munarett, D., Zanella, A., & Zorzi, M. (n.d.). *SSIM-based video admission control and resource allocation algorithms*. Second International Workshop on modeling. https://dl.ifip.org/db/conf/wiopt/wiopt2014/ZanforlinMZZ14.pdf