PILPaint – Building a Colorization

Pipeline using Python

    project by BOINAPALLI RAMYA|Guide by:K.Prakash Sir(Nit)

project overview

This project demonstrates how to build an image colorization pipeline using the PIL (Pillow) library in Python. We take a grayscale image and apply multiple colorization techniques manually using RGB channel manipulation, tinting, and blending — without using AI models. We'll use a real-world sample image of a lion, applying transformations to bring life to the grayscale version. This hands-on mini-project simulates real-world preprocessing tasks used in ML, computer vision, and creative tools.

Project Name: PILPaint_Colorization_Project

Input Image: lion.jpg Goal: Recolor grayscale images using only Python + PIL techniques

Section 2 – Import Libraries and Load Input Image

In this section, we'll install and import the required libraries for image processing and display. We'll also load the original lion image from the to begin our colorization pipeline. • Libraries used: PIL , NumPy , and Matplotlib • Load input image using Image.open() • Visualize with matplotlib.pyplot Input Path: (r"C:\Users\Ramya\Downloads\lion.jpg")

In [1]:
```
!pip install pillow matplotlib numpy
```

```
Requirement already satisfied: pillow in c:\users\ramya\anaconda3\lib\site-packag
es (10.4.0)
Requirement already satisfied: matplotlib in c:\users\ramya\anaconda3\lib\site-pa
ckages (3.9.2)
Requirement already satisfied: numpy in c:\users\ramya\anaconda3\lib\site-package
s (1.26.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\ramya\anaconda3\lib\s
ite-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\ramya\anaconda3\lib\site-
packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\ramya\anaconda3\lib
\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\ramya\anaconda3\lib
\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\ramya\anaconda3\lib\si
te-packages (from matplotlib) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\ramya\anaconda3\lib\s
ite-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ramya\anaconda3\l
ib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\ramya\anaconda3\lib\site-pack
ages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

In [2]:

```python
import numpy as np
```

In [3]:

```python
# 🧩 Importing required libraries
from PIL import Image, ImageEnhance
import numpy as np
import matplotlib.pyplot as plt
import os

# ➤ Ensure inline display of plots (Only for Jupyter Notebook)
%matplotlib inline
```

In [4]:

```python
from PIL import Image         # For loading the image
import matplotlib.pyplot as plt  # For displaying the image

# ⛒ Load the original color image
img_path = r"C:\Users\Ramya\Downloads\lion.jpg"   # Using raw string for Windows
original_img = Image.open(img_path)  # Open the image using PIL

# ➤ Show image
plt.imshow(original_img)      # Display the image
plt.axis('off')               # Turn off axis ticks
plt.title("Original Image - lion")  # Add a title
plt.show()                    # Show the image
```

Original Image - lion

Section 3 – Convert to Grayscale and Save Output

In this section, we'll simulate receiving a black-and-white input image by converting the original to grayscale using the PIL.Image.convert("L") method. This grayscale image will serve as the base input for our manual colorization pipeline. • ◇ Convert to grayscale (8-bit pixel format) • ◇ Visualize the result • ◇ Save the grayscale image to the output/ folder ◇ Converting to grayscale reduces the image to a single lightness channel.

In [5]:
```
# 🟣 Convert to Grayscale using PIL
gray_img = original_img.convert("L") # "L" mode = 8-bit pixels, black & white
# ➤ Show the grayscale image
plt.imshow(gray_img, cmap='gray')
plt.axis('off')
plt.title("Grayscale Image")
plt.show()
```

Grayscale Image

In [9]:

```
# ◈ Save grayscale image to output folder
gray_output_path = r"C:\Users\Ramya\Downloads\lion.jpg"
gray_img.save(gray_output_path)
# ➤ Confirm save
print(f"◈ Grayscale image saved to: {gray_output_path}")
```

```
◈ Grayscale image saved to: C:\Users\Ramya\Downloads\lion.jpg
```

Section 4 – Manual RGB Colorization Using Channel Merging

In this section, we'll begin our first colorization method by manually assigning grayscale values to each RGB channel using PIL.Image.merge() . This technique simulates how color channels work by duplicating the grayscale channel into R, G, and B spaces and adjusting their intensity individually. • ◈ Create R, G, B images from the grayscale base • ◈ Merge them using Image.merge("RGB", (R, G, B)) • ◈ Save and display the manually colorized output ◈ This is a basic yet powerful technique to simulate coloring without deep learning models.

In [10]:

```
# ◈ Create fake color channels from grayscale
r_channel = gray_img.point(lambda p: p * 1.2) # Slightly enhanced red
g_channel = gray_img.point(lambda p: p * 0.9) # Slightly reduced green
b_channel = gray_img.point(lambda p: p * 0.7) # Reduced blue for tone shift
# Merge into RGB image
colorized_rgb = Image.merge("RGB", (r_channel, g_channel, b_channel))
# ➤ Show the manually colorized image
plt.imshow(colorized_rgb)
plt.axis('off')
plt.title("Manual RGB Colorization")
plt.show()
```

Manual RGB Colorization

In [16]:
```
# 🔷 Save the manually colorized image
rgb_output_path =(r"C:\Users\Ramya\Downloads\lion.jpg")
colorized_rgb.save(rgb_output_path)
 # ➤ Confirm save
print(f"🔷 RGB colorized image saved to: {rgb_output_path}")
```

```
🔷 RGB colorized image saved to: C:\Users\Ramya\Downloads\lion.jpg
```

Section 5 – Artistic Tinting and Color Blending

In this section, we'll experiment with creating colorized versions of our grayscale image using tint overlays and channel blending techniques. By applying colored filters using RGB multipliers and overlays, we can simulate warm or cool color tones — often used in stylized photography and creative tools. • 🔷 Multiply grayscale with color-tint values • 🔷 Blend original and tinted layers • 🔷 Save and visualize results 🔷 These styles are useful for filters in creative photo editors or Instagram-like effects.

In [17]:
```
# 🔷 Apply a warm tone by multiplying grayscale with color constants
warm_r = gray_img.point(lambda p: p * 1.2)
warm_g = gray_img.point(lambda p: p * 1.0)
warm_b = gray_img.point(lambda p: p * 0.8)
tinted_img = Image.merge("RGB", (warm_r, warm_g, warm_b))
# ➤ Show tinted image
plt.imshow(tinted_img)
plt.axis('off')
plt.title("Tinted Image – Warm Filter")
plt.show()
```

Tinted Image – Warm Filter

In [18]:
```
# 🎨 Blend grayscale and tinted version to simulate color depth
blended_img = Image.blend(colorized_rgb, tinted_img, alpha=0.5)
# ➤ Show blended result
plt.imshow(blended_img)
plt.axis('off')
plt.title("Blended Image – RGB + Warm Tint")
plt.show()
```

Blended Image – RGB + Warm Tint

In [23]:
```
# 🎨 Save tinted and blended versions
tinted_path = r"C:\Users\Ramya\Downloads\lion.jpg"
```

```
blended_path = r"C:\Users\Ramya\Downloads\lion.jpg"
tinted_img.save(tinted_path)
blended_img.save(blended_path)
print(f"� Tinted image saved to: {tinted_path}")
print(f"� Blended image saved to: {blended_path}")
```

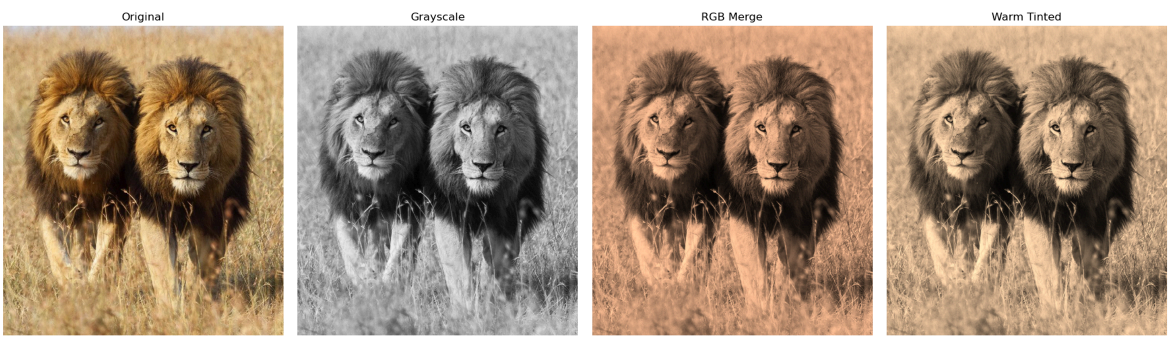
```
� Tinted image saved to: C:\Users\Ramya\Downloads\lion.jpg
� Blended image saved to: C:\Users\Ramya\Downloads\lion.jpg
```

Section 6 – Visual Comparison of Colorization Results

In this final section, we'll compare all versions of the image we've generated during this project side-by-side for visual analysis. This helps us evaluate the effectiveness of: • � Manual RGB channel colorization • � Tint overlay coloring • � Blending techniques � Such visualizations are essential when presenting results in image processing projects or research papers

In [26]:

```
# � Display all processed versions together for comparison
fig, axes = plt.subplots(1, 4, figsize=(18, 5))
 # Title row
axes[0].imshow(original_img)
axes[0].set_title("Original")
axes[0].axis('off')
axes[1].imshow(gray_img, cmap='gray')
axes[1].set_title("Grayscale")
axes[1].axis('off')
axes[2].imshow(colorized_rgb)
axes[2].set_title("RGB Merge")
axes[2].axis('off')
axes[3].imshow(tinted_img)
axes[3].set_title("Warm Tinted")
axes[3].axis('off')
plt.tight_layout()
plt.show()
```



In [27]:

```
 # � Show blended version separately
plt.figure(figsize=(6, 5))
plt.imshow(blended_img)
plt.axis('off')
plt.title("Blended Output (RGB + Tint)")
plt.show()
```

Conclusion & Next Steps

In this project, we built a manual image colorization pipeline using the (Pillow) library in Python, exploring techniques such as: PIL • ◈ RGB channel manipulation for colorizing grayscale images • ◈ Artistic tinting using point-wise intensity scaling • ◈ Blending colorized layers to enhance visual depth • ◈ Side-by-side comparisons for visual inspection This pipeline serves as a stepping stone toward more advanced image-to-image translation techniques in deep learning such as: • Generative Adversarial Networks (GANs) • OpenCV color maps and histograms • Autoencoders and colorization AI models

Key Takeaways:

• ◈ PIL is powerful even without AI • ◈ Manual techniques help understand image composition • ◈ You don't always need neural networks to build useful tools

What's Next?

• ◈ Automate color tone selection based on pixel clusters • ◈ Integrate OpenCV for more control over pixel data • ◈ Explore GAN-based colorization with datasets like ImageNet or CelebA

Project by: Boinapalli Ramya, Guided by: K. Prakash sir – FSDS