```
from google.colab import drive
drive.mount('/content/gdrive')
```

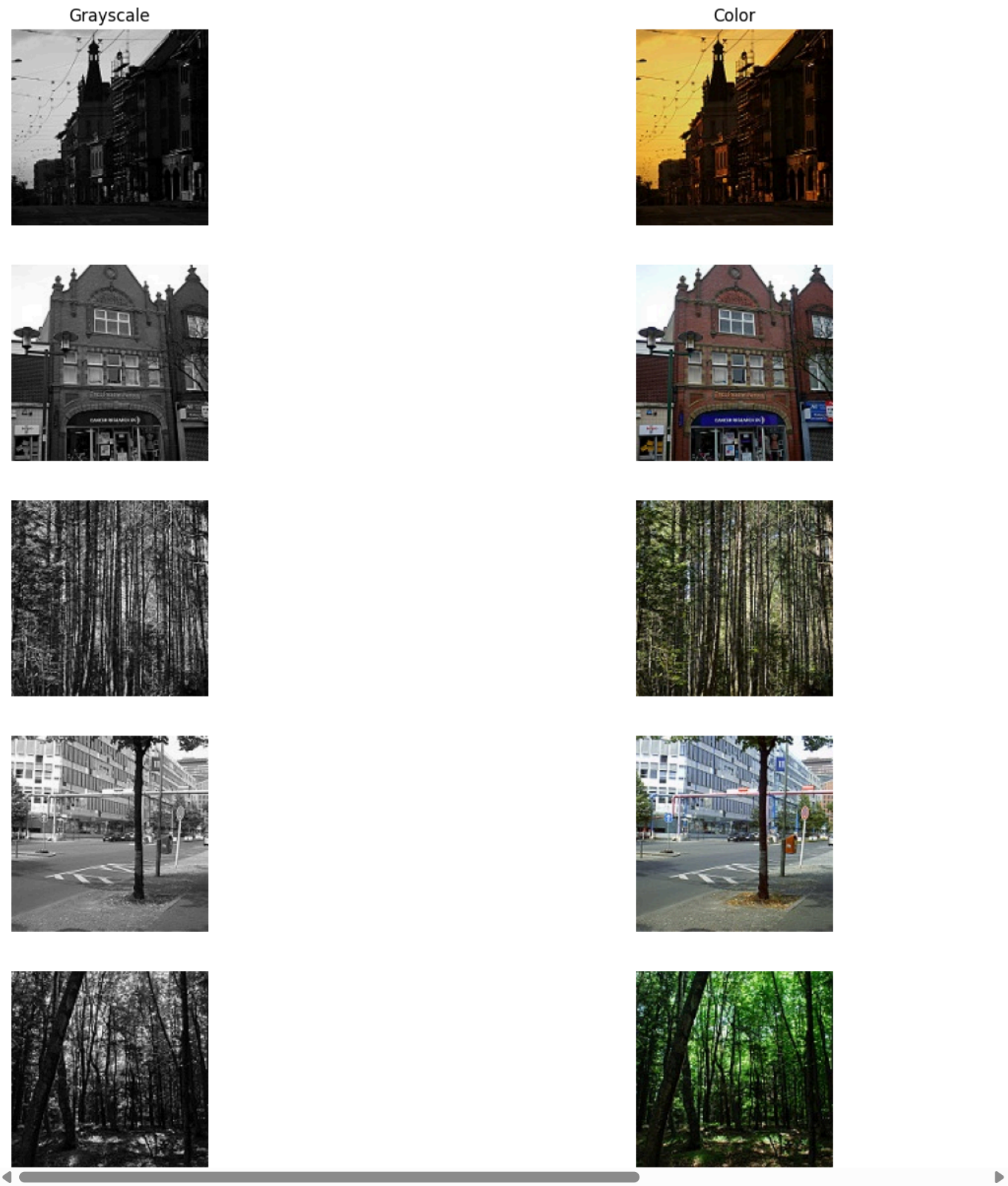
```
Mounted at /content/gdrive
```

```
!unzip "/content/gdrive/MyDrive/archive (3).zip"
```

```
  inflating: landscape Images/gray/947.jpg
  inflating: landscape Images/gray/948.jpg
  inflating: landscape Images/gray/949.jpg
  inflating: landscape Images/gray/95.jpg
  inflating: landscape Images/gray/950.jpg
  inflating: landscape Images/gray/951.jpg
  inflating: landscape Images/gray/952.jpg
  inflating: landscape Images/gray/953.jpg
  inflating: landscape Images/gray/954.jpg
  inflating: landscape Images/gray/955.jpg
  inflating: landscape Images/gray/956.jpg
  inflating: landscape Images/gray/957.jpg
  inflating: landscape Images/gray/958.jpg
  inflating: landscape Images/gray/959.jpg
  inflating: landscape Images/gray/96.jpg
  inflating: landscape Images/gray/960.jpg
  inflating: landscape Images/gray/961.jpg
  inflating: landscape Images/gray/962.jpg
  inflating: landscape Images/gray/963.jpg
  inflating: landscape Images/gray/964.jpg
  inflating: landscape Images/gray/965.jpg
  inflating: landscape Images/gray/966.jpg
  inflating: landscape Images/gray/967.jpg
  inflating: landscape Images/gray/968.jpg
  inflating: landscape Images/gray/969.jpg
  inflating: landscape Images/gray/97.jpg
  inflating: landscape Images/gray/970.jpg
  inflating: landscape Images/gray/971.jpg
  inflating: landscape Images/gray/972.jpg
  inflating: landscape Images/gray/973.jpg
  inflating: landscape Images/gray/974.jpg
  inflating: landscape Images/gray/975.jpg
  inflating: landscape Images/gray/976.jpg
  inflating: landscape Images/gray/977.jpg
  inflating: landscape Images/gray/978.jpg
  inflating: landscape Images/gray/979.jpg
  inflating: landscape Images/gray/98.jpg
  inflating: landscape Images/gray/980.jpg
  inflating: landscape Images/gray/981.jpg
  inflating: landscape Images/gray/982.jpg
  inflating: landscape Images/gray/983.jpg
  inflating: landscape Images/gray/984.jpg
  inflating: landscape Images/gray/985.jpg
  inflating: landscape Images/gray/986.jpg
  inflating: landscape Images/gray/987.jpg
  inflating: landscape Images/gray/988.jpg
  inflating: landscape Images/gray/989.jpg
  inflating: landscape Images/gray/99.jpg
  inflating: landscape Images/gray/990.jpg
  inflating: landscape Images/gray/991.jpg
  inflating: landscape Images/gray/992.jpg
  inflating: landscape Images/gray/993.jpg
  inflating: landscape Images/gray/994.jpg
  inflating: landscape Images/gray/995.jpg
  inflating: landscape Images/gray/996.jpg
  inflating: landscape Images/gray/997.jpg
  inflating: landscape Images/gray/998.jpg
  inflating: landscape Images/gray/999.jpg
```

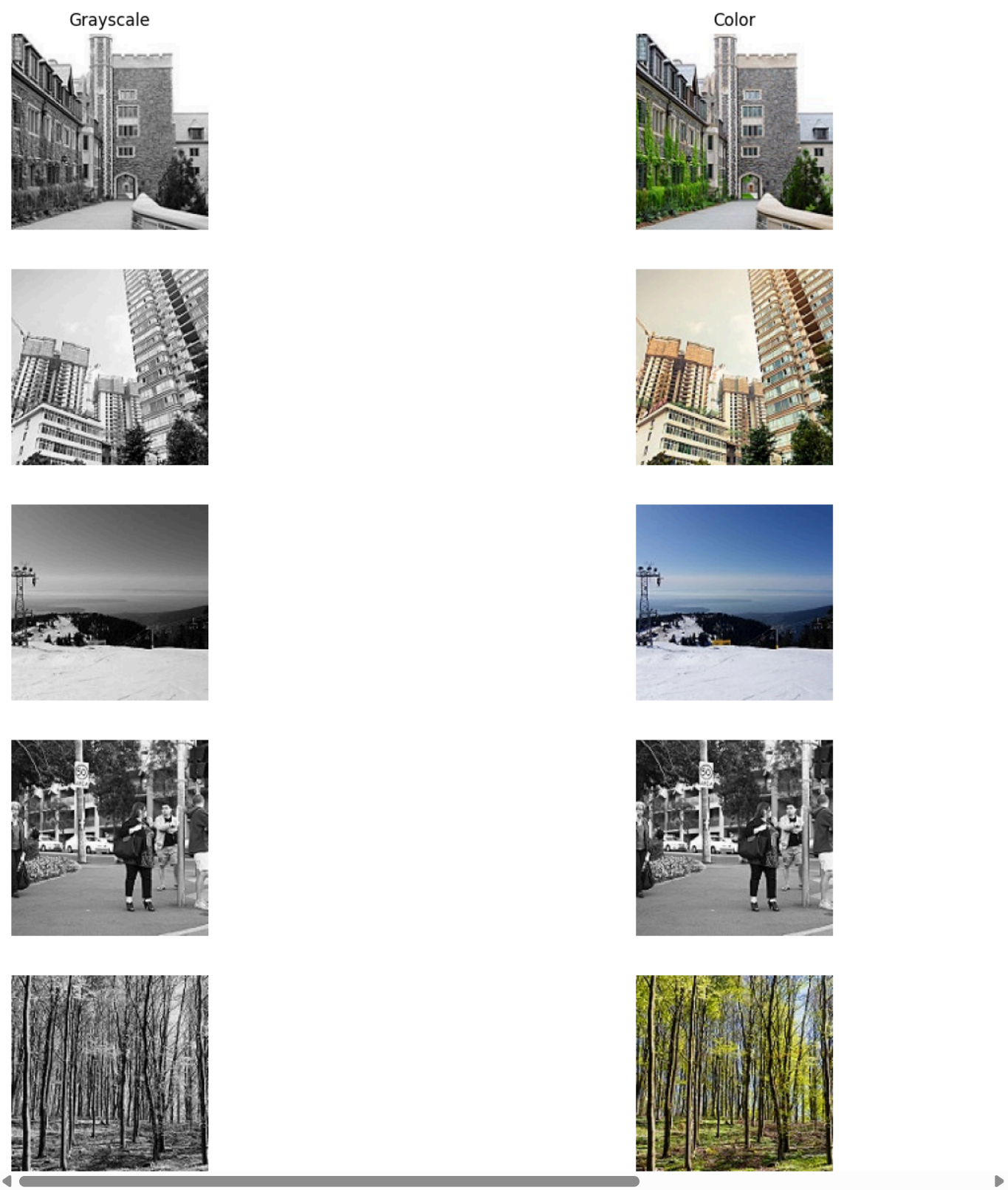
```
import os
import torch
import torchvision
from torchvision.io import read_image
from torch import nn
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision.datasets import ImageFolder
import torch.optim as optim
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
from PIL import Image
from tqdm.notebook import tqdm
```

```
MANUAL_SEED = 42
BATCH_SIZE = 32
SHUFFLE = True


class LandscapeDataset(Dataset):
  def __init__(self, transform=None):
    self.dataroot = './landscape Images'
    self.images = os.listdir(f'{self.dataroot}/color')
    self.transform = transform

  def __len__(self):
    return len(self.images)

  def __getitem__(self,idx):
    image_path = self.images[idx]

    color_img = read_image(f'{self.dataroot}/color/{image_path}') / 255
    gray_img = read_image (f'{self.dataroot}/gray/{image_path}') / 255

    if self.transform:
      color_img = self.transform(color_img)
      gray_img = self.transform(gray_img)

      return color_img, gray_img


transform = transforms.Compose([transforms.Resize((150, 150), antialias=False)])
dataset = LandscapeDataset(transform=transform)

train_set, test_set = random_split(dataset, [0.8,0.2], generator=torch.Generator().manual_seed(MANUAL_SEED))

trainloader = DataLoader(train_set, batch_size = BATCH_SIZE, shuffle=SHUFFLE)
testloader = DataLoader(test_set, batch_size = BATCH_SIZE, shuffle=SHUFFLE)


# Visualize some images

def show_images (color, gray):
    fig, axs = plt.subplots (5, 2, figsize=(15, 15))
    axs [0, 0].set_title('Grayscale')
    axs [0, 1].set_title('Color')
    for i in range(5):
        axs[i, 0].imshow(gray[i].permute (1,2,0), cmap='gray')
        axs[i, 0].axis('off')
        axs [i, 1].imshow(color[i].permute (1,2,0))
        axs[i, 1].axis('off')
    plt.show()

color, gray = next(iter(trainloader))
show_images(color, gray)
```




```
[26] # Training Variables

EPOCHS = 3
LEARNING_RATE = 0.001
MOMENTUM = 0.9
DEVICE= 'cuda' if torch.cuda.is_available() else 'cpu'


# Visualize some images

def show_images (color, gray):
    fig, axs = plt.subplots (5, 2, figsize=(15, 15))
```

```
        axs [0, 0].set_title('Grayscale')
        axs [0, 1].set_title('Color')
        for i in range(5):
            axs[i, 0].imshow(gray[i].permute (1,2,0), cmap='gray')
            axs[i, 0].axis('off')
            axs [i, 1].imshow(color[i].permute (1,2,0))
            axs[i, 1].axis('off')
        plt.show()

    color, gray = next(iter(trainloader))
    show_images(color, gray)
```



```
class ColorAutoEncoder(nn.Module):
  def __init__(self):
    super().__init__()
    #Input shape: Bx1x150x150
    #Conv2d(in_channels, out_channels, kernel_size, stride, padding)

    self.down1 = nn.Conv2d(1, 64, 3, stride=2, )
    self.down2 = nn.Conv2d(64, 128, 3, stride=2, padding=1)
    self.down3 = nn.Conv2d(128, 256, 3, stride=2, padding=1)
    self.down4 = nn.Conv2d(256, 512, 3, stride=2, padding=1)

    self.up1 = nn.ConvTranspose2d(512, 256, 3, stride=2, padding=1)
    self.up2 = nn.ConvTranspose2d(512, 128, 3, stride=2, padding=1)
    self.up3 = nn.ConvTranspose2d(256, 64, 3, stride=2, padding=1, output_padding=1)
    self.up4 = nn.ConvTranspose2d(128, 3, 3, stride=2, output_padding=1)

    self.relu = nn.ReLU()
    self.sigmoid = nn.Sigmoid()

  def forward(self, x):
    # Down sample
    d1 = self.relu(self.down1(x))
    d2 = self.relu(self.down2(d1))
    d3 = self.relu(self.down3(d2))
    d4 = self.relu(self.down4(d3))
    # Upsample
    u1 = self.relu(self.up1 (d4))
    u2 = self.relu(self.up2(torch.cat((u1,d3), dim=1)))
    u3 = self.relu(self.up3(torch.cat((u2, d2), dim=1)))
    u4 = self.sigmoid(self.up4(torch.cat((u3,d1), dim=1)))

    return u4


# Initialize the model
model=ColorAutoEncoder().to (DEVICE)
total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
f"Total Number of trainable parameters of this model are: {total_params:,}"
```

```
'Total Number of trainable parameters of this model are: 3,470,659'
```

```
# Initialize the optimizer and loss function
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)


for epoch in range(EPOCHS):
    model.train()
    running_loss = 0.0

    for idx, (color_img, gray_img) in tqdm(enumerate(trainloader), total=len(trainloader)):
        color_img = color_img.to(DEVICE)
        gray_img = gray_img.to(DEVICE)

        predictions = model(gray_img)  # Forward pass

        optimizer.zero_grad()
        loss = criterion(predictions, color_img)  # MSELoss expects [0,1] tensors
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    avg_loss = running_loss / len(trainloader)
    print(f"Epoch {epoch + 1}, Avg Loss: {avg_loss:.6f}")
```

```
100%                                                  179/179 [15:21<00:00,  3.83s/it]
Epoch 1, Avg Loss: 0.021204
100%                                                  179/179 [15:17<00:00,  3.92s/it]
Epoch 2, Avg Loss: 0.010252
100%                                                  179/179 [15:16<00:00,  3.88s/it]
Epoch 3, Avg Loss: 0.008566
```

```
total_loss = 0.0
with torch.no_grad():
```

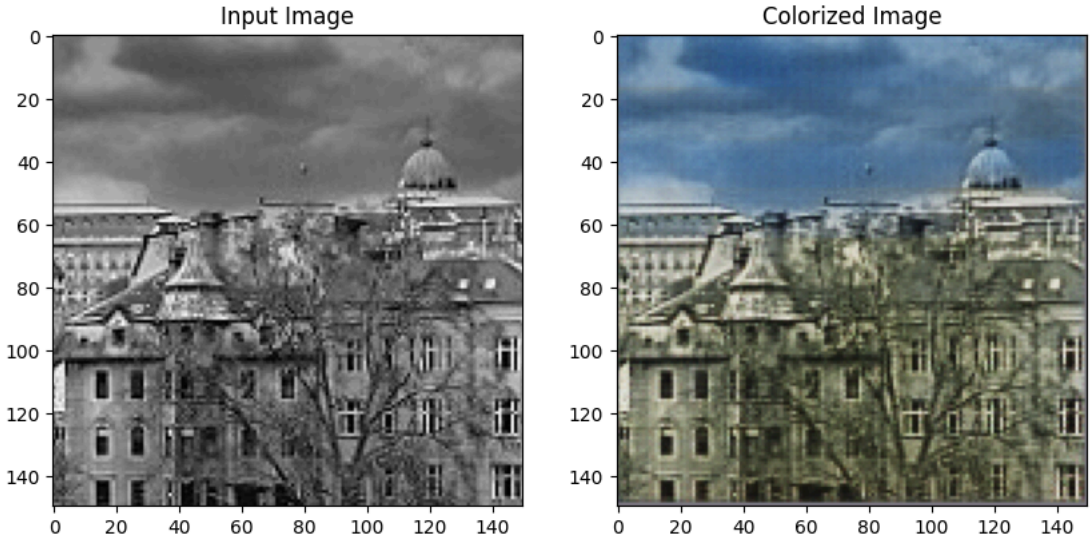
```
  for idx, (color_img, gray_img) in tqdm(enumerate (testloader), total=len(testloader)):
    color_img = color_img.to(DEVICE)
    gray_img = gray_img.to(DEVICE)

    prediction = model(gray_img)

    loss = criterion(prediction, color_img)
    total_loss += loss.item()

print(f"Total Testing loss is: {total_loss/ len(testloader):.3f})")
```

```
100%  45/45 [01:19<00:00,  1.44s/it]
Total Testing loss is: 0.008)
```

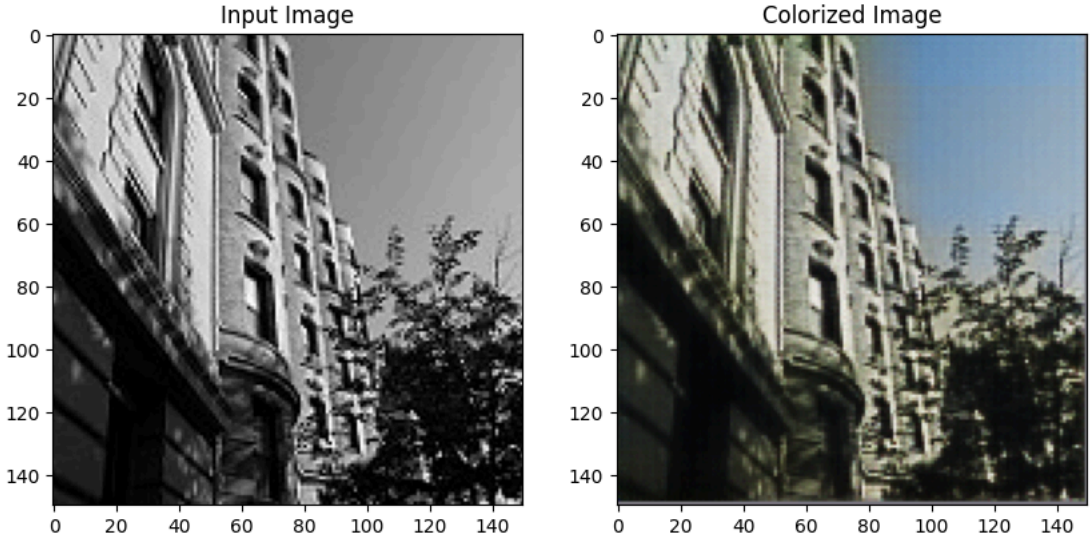
```
from torchvision.io import read_image
from torchvision import transforms
import matplotlib.pyplot as plt
import os

def colorize_image(image_path):
    """
    Colorizes a grayscale image using the trained model.

    Args:
        image_path (str): Path to the grayscale input image.

    Returns:
        None: Displays the input and colorized images.
    """
    # Load and preprocess the input image
    if not os.path.exists(image_path):
        print(f"Error: File not found at {image_path}")
        return

    input_image = read_image(image_path) / 255  # Normalize to [0, 1]
    transform = transforms.Resize((150, 150), antialias=False)  # Resize
    input_image = transform(input_image)

    # Add a batch dimension (B=1)
    input_image = input_image.unsqueeze(0).to(DEVICE)

    # Get the colorized prediction
    with torch.no_grad():
        output_image = model(input_image)

    # Remove batch dimension and move to CPU
    output_image = output_image.squeeze(0).cpu()
    input_image = input_image.squeeze(0).cpu()

    # Display the images
    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    axs[0].imshow(input_image.permute(1, 2, 0), cmap='gray')  # Input (grayscale)
    axs[0].title.set_text('Input Image')
    axs[1].imshow(output_image.permute(1, 2, 0))  # Output (colorized)
    axs[1].title.set_text('Colorized Image')
    plt.show()

# Example usage 1: Processing images from a specific folder
image_folder = '/content/landscape Images/gray/'  # Adjust if your folder name is different

# 1. Check if the folder exists
if os.path.exists(image_folder):
    print("Folder exists!")
else:
    print(f"Folder does not exist at {image_folder}! Please check the path.")
    # You might want to stop execution here if the folder doesn't exist
    # import sys; sys.exit(1)

# 2. Get a list of image files in the folder
image_files = [
    f for f in os.listdir(image_folder)
    if os.path.isfile(os.path.join(image_folder, f)) and f.lower().endswith(('.png', '.jpg', '.jpeg'))
]

# 3. Process 20 sample images or all if fewer than 20
num_images_to_process = min(20, len(image_files))
```

```
if num_images_to_process == 0:
    print(f"No image files found in the folder {image_folder}.")
else:
    print(f"Processing {num_images_to_process} sample images (out of {len(image_files)} found):")
    for i in range(num_images_to_process):
        image_path = os.path.join(image_folder, image_files[i])
        print(f"  - {image_path}")
        colorize_image(image_path)
```

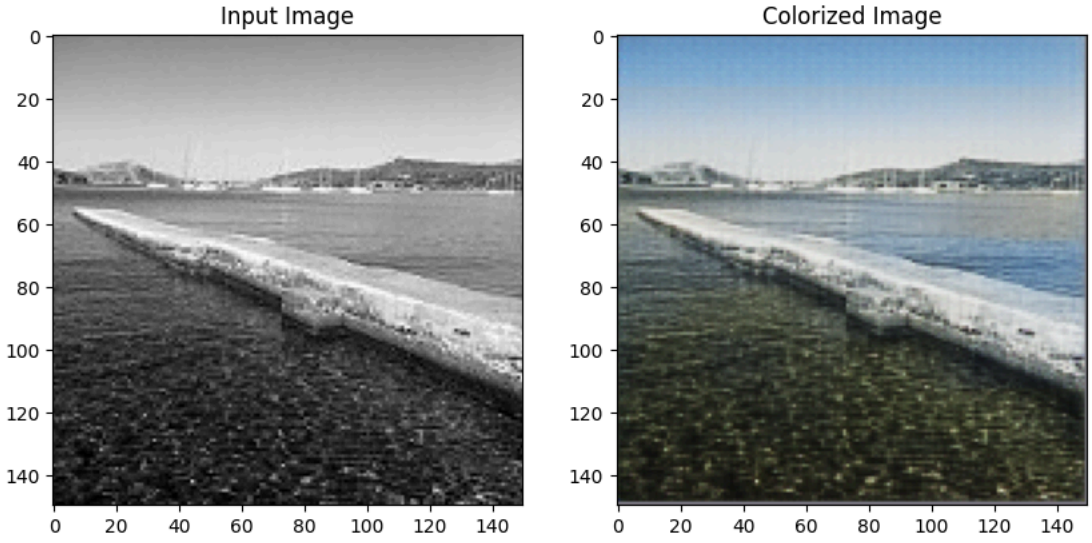
```
Folder exists!
Processing 20 sample images (out of 7129 found):
  - /content/landscape Images/gray/1686.jpg
```



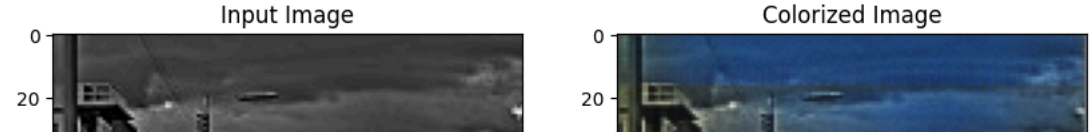

```
  - /content/landscape Images/gray/5464.jpg
```

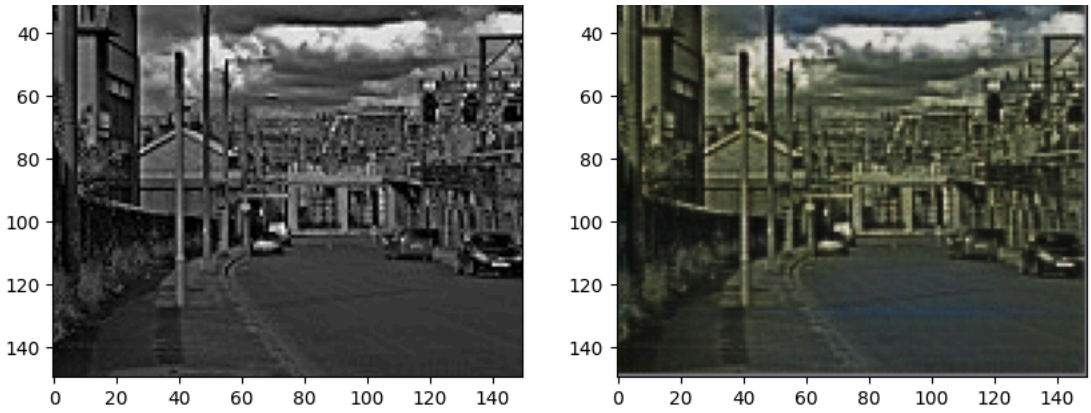



```
  - /content/landscape Images/gray/5805.jpg
```

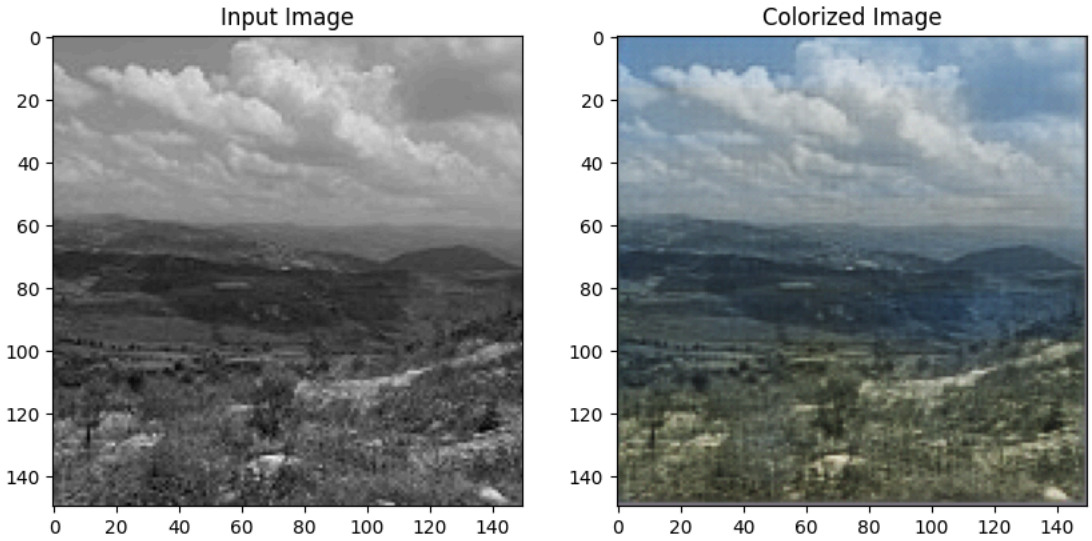



```
  - /content/landscape Images/gray/5950.jpg
```

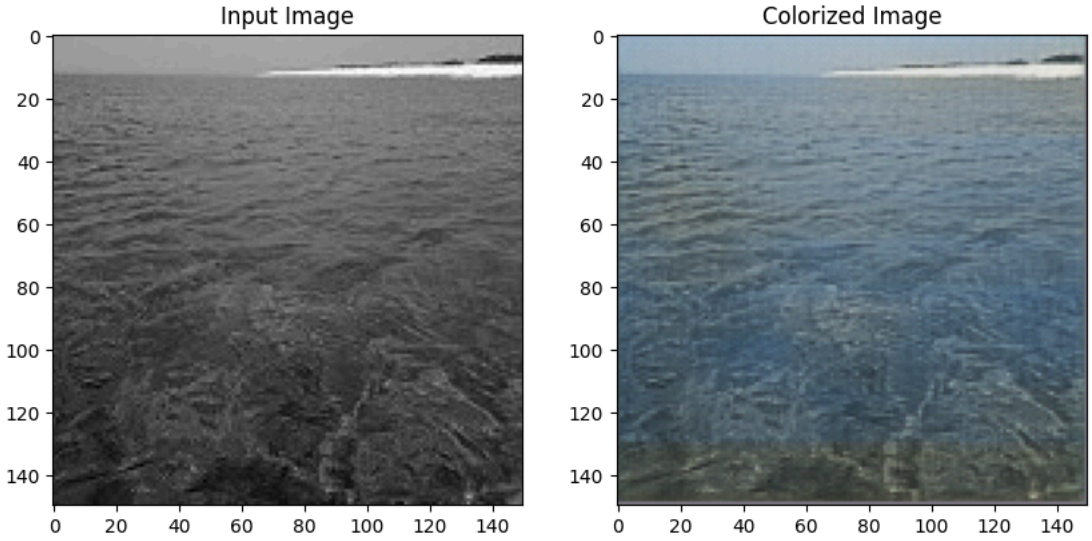




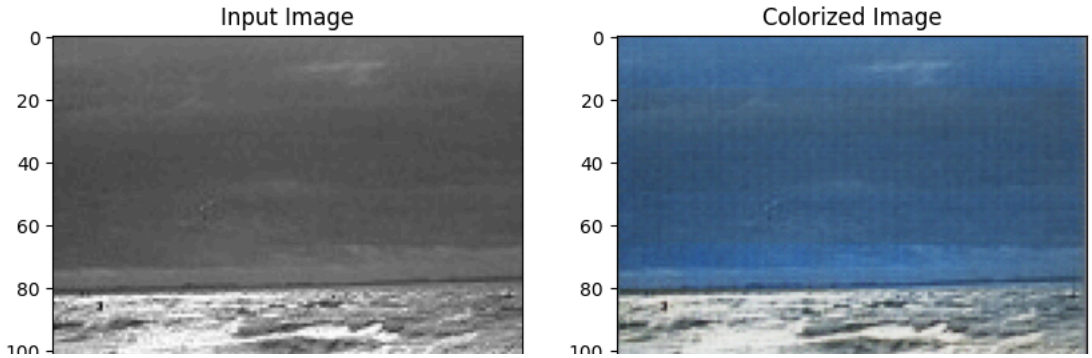
- /content/landscape Images/gray/4023.jpg

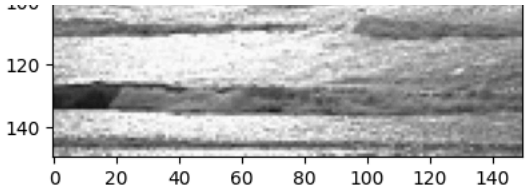


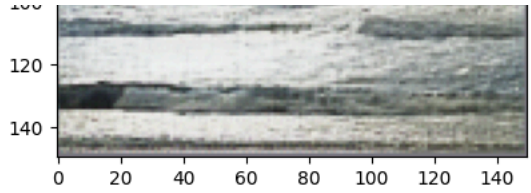
- /content/landscape Images/gray/4985.jpg



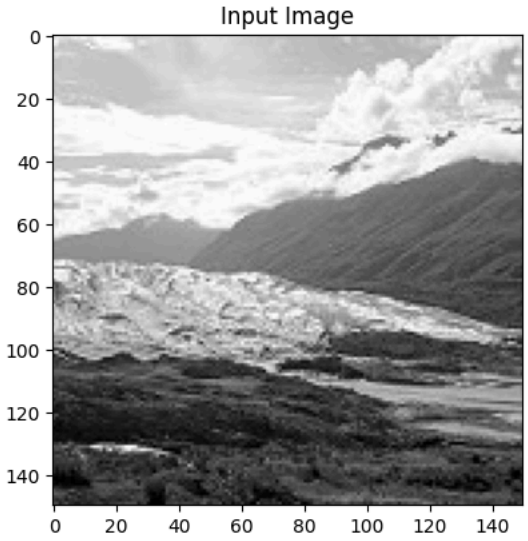
- /content/landscape Images/gray/1841.jpg

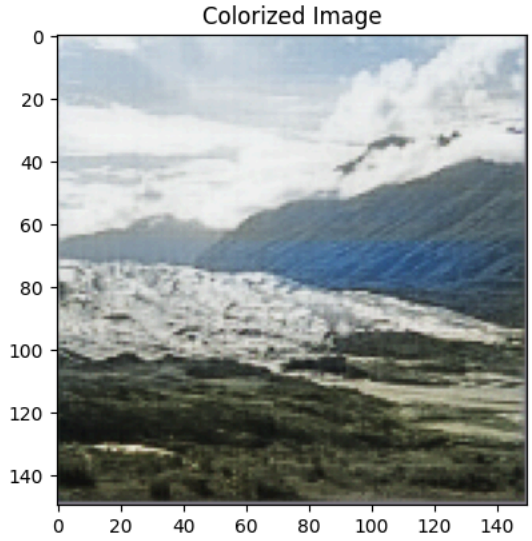




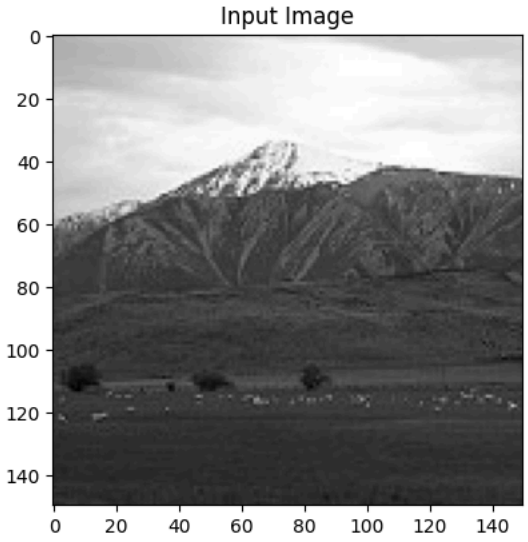


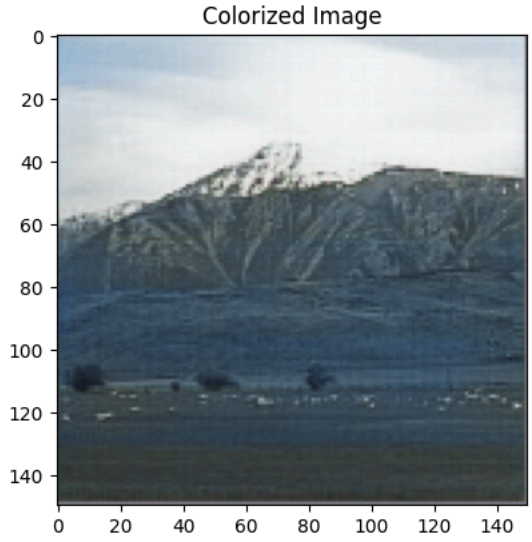
- /content/landscape Images/gray/5249.jpg



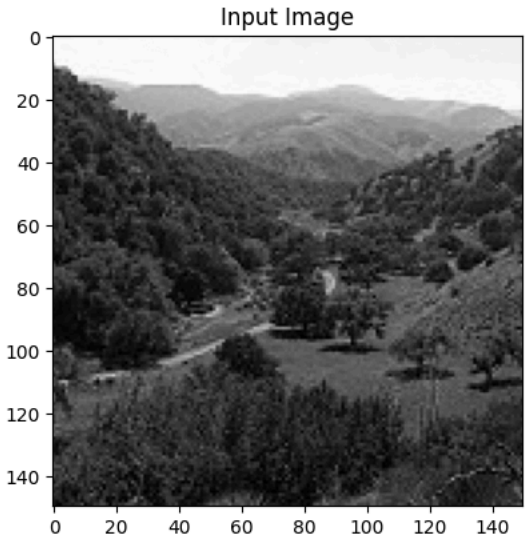


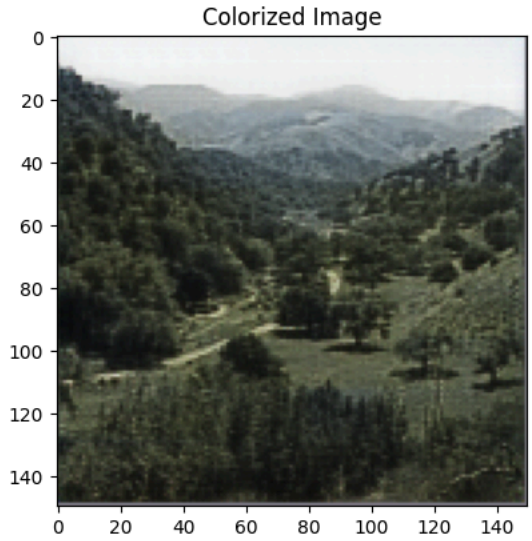
- /content/landscape Images/gray/145.jpg



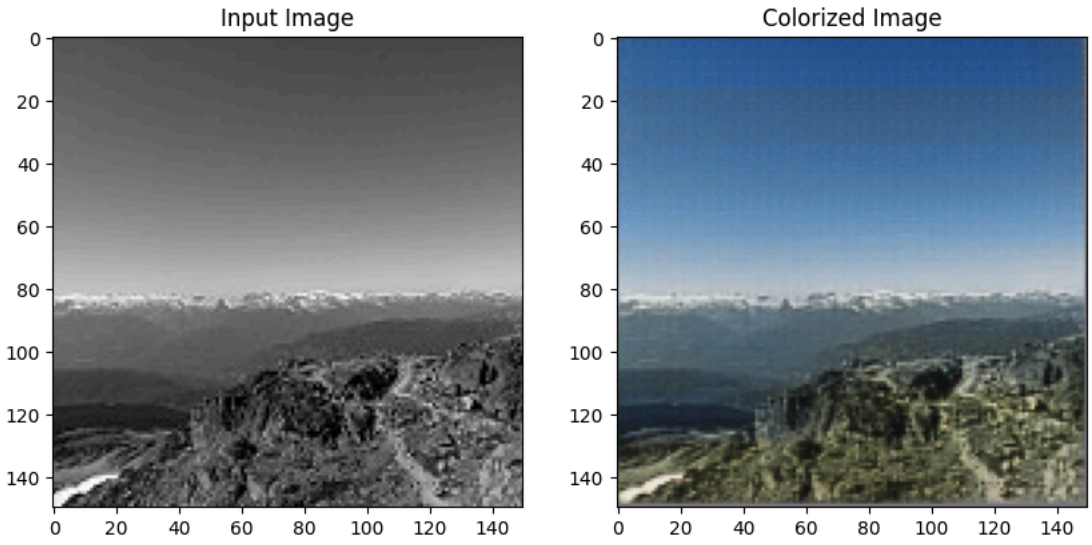


- /content/landscape Images/gray/1204.jpg

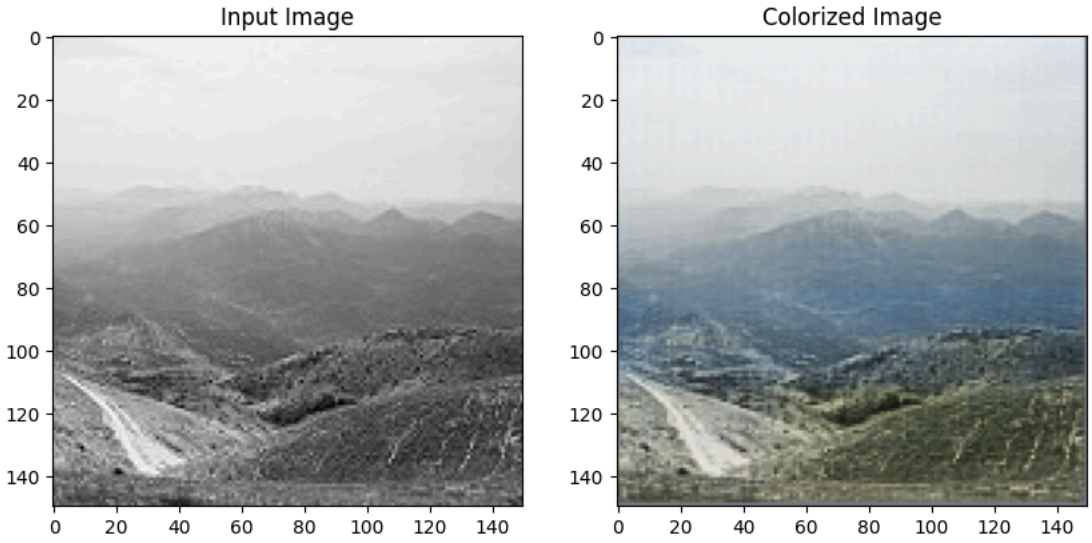




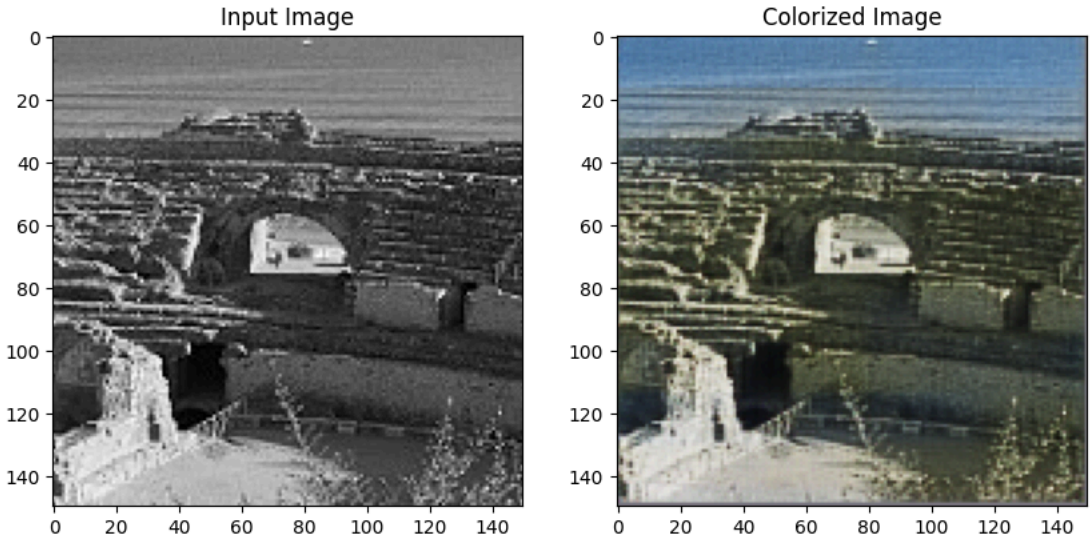
- /content/landscape Images/gray/3195.jpg



- /content/landscape Images/gray/20.jpg



- /content/landscape Images/gray/4484.jpg



- /content/landscape Images/gray/4441.jpg