ABSTRACT

The process of colorizing black and white (B&W) images is a compelling application of deep learning in the field of computer vision. This project presents an automated approach for restoring and colorizing old grayscale images using Convolutional Neural Networks (CNNs). Traditional manual methods of image colorization are time-consuming and require artistic expertise, whereas the proposed method leverages deep learning to predict realistic and vibrant colors from grayscale input.

The core of the project involves training a CNN model on large datasets of colored images converted to grayscale. The model learns the mapping between grayscale intensities and corresponding RGB color values. We employ a U-Net based architecture or pre-trained models like DeOldify or VGG-16 based encoder-decoder structures to enhance performance. The network takes a grayscale image as input and outputs a colorized version, learning the context, semantics, and object features to assign meaningful colors.

This project demonstrates how deep learning can revive historical photographs and enhance visual understanding by bridging the gap between past and present. The results are evaluated using metrics like PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index) to ensure color accuracy and perceptual quality.

By automating image colorization, this mini project not only showcases the power of neural networks in artistic image processing but also preserves heritage through digital restoration.

1. INTRODUCTION

Colorizing old black and white images is a captivating endeavor that brings history to life with vibrant colors. In the realm of image processing and computer vision, this task presents a fascinating challenge, combining the intricacies of artificial intelligence and the nostalgia of bygone eras. With advancements in deep learning techniques, particularly in convolutional neural networks (CNNs), researchers and enthusiasts alike have embarked on a journey to restore the vibrancy of historical photographs.

By training deep learning models on vast datasets of black and white images paired with their corresponding color counterparts, we can leverage the power of neural networks to automatically add color to grayscale images. This process not only breathes new life into old photographs but also provides a valuable tool for historians, archivists, and individuals seeking to connect more intimately with the past.

In this project, we delve into the methodologies, challenges, and applications of colorizing old black and white images using deep learning, exploring the techniques employed, the significance of the task, and the potential impact on various fields.

1.1 PROBLEM DEFINITION

The problem of colorizing old black and white images using deep learning involves developing an algorithmic solution to automatically add color to grayscale images. Given a dataset of black and white images paired with their corresponding color images, the objective is to train a deep learning model capable of accurately predicting the colors of grayscale images. This task encompasses various challenges, including the complexity of image semantics, variations in lighting conditions, and the inherent subjectivity of color perception. The ultimate goal is to create a robust and efficient system that can accurately colorize historical photographs, thereby preserving and enhancing visual heritage while opening avenues for new applications in digital restoration, cultural preservation, and artistic expression.

REQUIREMENTS

1.1. Software Requirement Specifications

Operating System Front End Back End Server Documentation: Windows 10 Frontend Software: Streamlit;

Backend Software: Python: Install the latest version of Python, preferably Python 3.x

1.2. Hardware Requirement Specifications

Processor: A modern multi-core processor (e.g., Intel Core i5 or equivalent) should be adequate.

RAM: At least 8 GB of RAM is recommended. However, if you're working with large datasets or complex models, consider having more RAM.

Storage: Sufficient storage space is required to store your datasets, libraries, and code. A few gigabytes of free space should be enough.

Operating System: You can develop the project on Windows, macOS, or Linux based on your preference. Ensure you have the necessary administrative privileges to install software.

Computer Processor Core i3 Processor Speed 2.3 GHz Processor Hard Disk 400 GB or more RAM Min 2 GB.

IMPLEMENTATION

1. Data Collection and Preparation: The first step in implementing a colorization algorithm is to collect a dataset of black and white images paired with their corresponding color images. This dataset serves as the training data for the deep learning model. Careful curation of the dataset is crucial to ensure diversity in image content, styles, and time periods, thereby enhancing the model's ability to generalize. Preprocessing steps such as resizing, normalization, and augmentation may be applied to the dataset to improve training efficiency and model robustness.

2. Model Architecture Selection: Selecting an appropriate architecture for the colorization model is essential for achieving accurate and realistic results. Convolutional Neural Networks (CNNs) are commonly used for image colorization tasks due to their ability to capture spatial dependencies and learn hierarchical features. Architectures such as U-Net, ResNet, and variants of Generative Adversarial Networks (GANs) have been successfully applied to image colorization. The chosen model should strike a balance between computational efficiency and performance, considering factors such as network depth, parameter count, and inference speed.

3. Training Process: The training process involves feeding the prepared dataset into the chosen model architecture and optimizing its parameters to minimize the colorization error. Training typically proceeds in batches, where each batch of grayscale images is fed into the model, and the corresponding color images are used to compute the loss function. Common loss functions used in colorization tasks include mean squared error (MSE) or perceptual loss functions based on feature embeddings. Techniques such as transfer learning and fine-tuning pre-trained models may be employed to accelerate convergence and improve performance, especially with limited training data.

4. Evaluation and Fine-tuning: Once the model has been trained, it is evaluated on a separate validation dataset to assess its performance and generalization capabilities. Evaluation metrics such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and perceptual metrics based on human perception may be used to quantify the quality of colorization. Fine tuning the model based on validation results and iteratively adjusting hyperparameters such as learning rate, batch size, and regularization techniques can further improve performance and address overfitting issues.

In [0]:
```
from keras import Input
from keras.layers import Dense, Conv2D, UpSampling2D, add, concatenate
from keras.models import Model
from keras.optimizers import RMSprop
from keras.preprocessing.image import load_img, img_to_array, array_to_img
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.
```

In [0]:
```
input_tensor = Input(shape=(TARGET_SIZE, TARGET_SIZE, 1))

layer1 = Conv2D(8, (3, 3), activation='relu', padding='same', strides=2)(input
layer2 = Conv2D(16, (3, 3), activation='relu', padding='same')(layer1)
layer3 = Conv2D(16, (3, 3), activation='relu', padding='same', strides=2)(laye
layer4 = Conv2D(32, (3, 3), activation='relu', padding='same')(layer3)
layer5 = Conv2D(32, (3, 3), activation='relu', padding='same', strides=2)(laye
layer6 = Conv2D(32, (3, 3), activation='relu', padding='same')(layer5)
layer7 = Conv2D(64, (3, 3), activation='relu', padding='same', strides=2)(laye
layer8 = Conv2D(64, (3, 3), activation='relu', padding='same')(layer7)
layer9 = Conv2D(64, (3, 3), activation='relu', padding='same', strides=2)(laye
layer10 = Conv2D(64, (3, 3), activation='relu', padding='same')(layer9)
layer11 = Conv2D(64, (3, 3), activation='relu', padding='same', strides=2)(lay
layer12 = Conv2D(64, (3, 3), activation='relu', padding='same')(layer11)
layer13 = Conv2D(64, (3, 3), activation='relu', padding='same', strides=2)(lay

layer14 = UpSampling2D((2, 2))(layer13)
layer15 = Conv2D(64, (3, 3), activation='relu', padding='same')(layer14)

layer16 = concatenate([layer15, layer11])
layer17 = UpSampling2D((2, 2))(layer16)
layer18 = Conv2D(64, (3, 3), activation='relu', padding='same')(layer17)

layer19 = concatenate([layer18, layer9])
layer20 = UpSampling2D((2, 2))(layer19)
layer21 = Conv2D(64, (3, 3), activation='relu', padding='same')(layer20)

layer22 = concatenate([layer21, layer7])
layer23 = UpSampling2D((2, 2))(layer22)
layer24 = Conv2D(64, (3, 3), activation='relu', padding='same')(layer23)

layer25 = concatenate([layer24, layer5])
layer26 = UpSampling2D((2, 2))(layer25)
layer27 = Conv2D(32, (3, 3), activation='relu', padding='same')(layer26)

layer28 = concatenate([layer27, layer3])
layer29 = UpSampling2D((2, 2))(layer28)
layer30 = Conv2D(16, (3, 3), activation='relu', padding='same')(layer29)

layer31 = concatenate([layer30, layer1])
layer32 = UpSampling2D((2, 2))(layer31)
layer33 = Conv2D(3, (3,3), activation='tanh', padding='same')(layer32)

model = Model(input_tensor, layer33)
model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/pyt
hon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.fr
amework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_____________________________________________________________________________
_____________________
Layer (type)                    Output Shape         Param #     Connected to
=============================================================================
=====================
input_1 (InputLayer)            (None, 512, 512, 1)  0
_____________________________________________________________________________
_____________________
conv2d_1 (Conv2D)               (None, 256, 256, 8)  80          input_1[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_2 (Conv2D)               (None, 256, 256, 16) 1168        conv2d_1[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_3 (Conv2D)               (None, 128, 128, 16) 2320        conv2d_2[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_4 (Conv2D)               (None, 128, 128, 32) 4640        conv2d_3[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_5 (Conv2D)               (None, 64, 64, 32)   9248        conv2d_4[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_6 (Conv2D)               (None, 64, 64, 32)   9248        conv2d_5[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_7 (Conv2D)               (None, 32, 32, 64)   18496       conv2d_6[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_8 (Conv2D)               (None, 32, 32, 64)   36928       conv2d_7[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_9 (Conv2D)               (None, 16, 16, 64)   36928       conv2d_8[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_10 (Conv2D)              (None, 16, 16, 64)   36928       conv2d_9[0]
[0]
_____________________________________________________________________________
_____________________
conv2d_11 (Conv2D)              (None, 8, 8, 64)     36928       conv2d_10[0]
[0]
_____________________________________________________________________________
```

```
conv2d_12 (Conv2D)              (None, 8, 8, 64)     36928       conv2d_11[0]
[0]

______________________________________________________________________________
_____________________
conv2d_13 (Conv2D)              (None, 4, 4, 64)     36928       conv2d_12[0]
[0]

______________________________________________________________________________
_____________________
up_sampling2d_1 (UpSampling2D)  (None, 8, 8, 64)     0           conv2d_13[0]
[0]

______________________________________________________________________________
_____________________
conv2d_14 (Conv2D)              (None, 8, 8, 64)     36928       up_sampling2
d_1[0][0]

______________________________________________________________________________
_____________________
concatenate_1 (Concatenate)     (None, 8, 8, 128)    0           conv2d_14[0]
[0]
                                                                  conv2d_11[0]
[0]

______________________________________________________________________________
_____________________
up_sampling2d_2 (UpSampling2D)  (None, 16, 16, 128)  0           concatenate_
1[0][0]

______________________________________________________________________________
_____________________
conv2d_15 (Conv2D)              (None, 16, 16, 64)   73792       up_sampling2
d_2[0][0]

______________________________________________________________________________
_____________________
concatenate_2 (Concatenate)     (None, 16, 16, 128)  0           conv2d_15[0]
[0]
                                                                  conv2d_9[0]
[0]

______________________________________________________________________________
_____________________
up_sampling2d_3 (UpSampling2D)  (None, 32, 32, 128)  0           concatenate_
2[0][0]

______________________________________________________________________________
_____________________
conv2d_16 (Conv2D)              (None, 32, 32, 64)   73792       up_sampling2
d_3[0][0]

______________________________________________________________________________
_____________________
concatenate_3 (Concatenate)     (None, 32, 32, 128)  0           conv2d_16[0]
[0]
                                                                  conv2d_7[0]
[0]

______________________________________________________________________________
_____________________
up_sampling2d_4 (UpSampling2D)  (None, 64, 64, 128)  0           concatenate_
3[0][0]

______________________________________________________________________________
_____________________
conv2d_17 (Conv2D)              (None, 64, 64, 64)   73792       up_sampling2
d_4[0][0]
```

```
_____________________
concatenate_4 (Concatenate)     (None, 64, 64, 96)   0           conv2d_17[0]
[0]
                                                                 conv2d_5[0]

[0]
_____________________________________________________________________________
_____________________
up_sampling2d_5 (UpSampling2D)  (None, 128, 128, 96) 0           concatenate_
4[0][0]
_____________________________________________________________________________
_____________________
conv2d_18 (Conv2D)              (None, 128, 128, 32) 27680       up_sampling2
d_5[0][0]
_____________________________________________________________________________
_____________________
concatenate_5 (Concatenate)     (None, 128, 128, 48) 0           conv2d_18[0]
[0]
                                                                 conv2d_3[0]

[0]
_____________________________________________________________________________
_____________________
up_sampling2d_6 (UpSampling2D)  (None, 256, 256, 48) 0           concatenate_
5[0][0]
_____________________________________________________________________________
_____________________
conv2d_19 (Conv2D)              (None, 256, 256, 16) 6928        up_sampling2
d_6[0][0]
_____________________________________________________________________________
_____________________
concatenate_6 (Concatenate)     (None, 256, 256, 24) 0           conv2d_19[0]
[0]
                                                                 conv2d_1[0]

[0]
_____________________________________________________________________________
_____________________
up_sampling2d_7 (UpSampling2D)  (None, 512, 512, 24) 0           concatenate_
6[0][0]
_____________________________________________________________________________
_____________________
conv2d_20 (Conv2D)              (None, 512, 512, 3)  651         up_sampling2
d_7[0][0]
=============================================================================
=====================
Total params: 560,331
Trainable params: 560,331
Non-trainable params: 0
_____________________________________________________________________________
_____________________
```

In [0]:
```
model.compile(optimizer=RMSprop(lr=3e-5), loss='mse')
```

In [0]:
```
input_images_dir = '''line-images'''
label_images_dir = '''colored'''
```

```
fnames = os.listdir(input_images_dir)
fpaths = [os.path.join(input_images_dir, fname) for fname in fnames]
```

```
In [0]: def resize_img_to_target_size(img_path, target_size, num_channels):
          color_mode = 'grayscale' if num_channels == 1 else 'rgb'
          img = load_img(img_path, color_mode=color_mode)
          img_array = img_to_array(img)
          width, height, _ = img_array.shape
          max_dim = max(width, height)
          scale_factor = target_size / max_dim

          scaled_width, scaled_height = int(width * scale_factor), int(height * scale_
          img = load_img(img_path, target_size=(scaled_width, scaled_height), color_mo
          img_array = img_to_array(img) / 255
          resized_img_array = np.zeros( (target_size, target_size, num_channels) )
          resized_img_array[ (target_size - scaled_width)//2: (target_size - scaled_wi
                             (target_size - scaled_height)//2: (target_size - scaled_h
                              : ] = img_array
          return resized_img_array
```

```
In [0]: def generator(fnames, batch_size, target_size):

          input_fpaths = [os.path.join(input_images_dir, fname) for fname in fnames]
          label_fpaths = [os.path.join(label_images_dir, fname) for fname in fnames]

          def get_batches(iterable, n=1):
            l = len(iterable)
            for ndx in range(0, l-n, n):
                yield iterable[ndx:min(ndx + n, l)]

          while True:
            for batch in get_batches( list(zip(input_fpaths, label_fpaths)), batch_siz
              X = np.zeros( (batch_size, target_size, target_size, 1) )
              Y = np.zeros( (batch_size, target_size, target_size, 3) )
              for i, (x_fpath, y_fpath) in enumerate(batch):
                X[i, :, :, :] = resize_img_to_target_size(x_fpath, target_size, 1)
                Y[i, :, :, :] = resize_img_to_target_size(y_fpath, target_size, 3)
              yield X, Y
```

```
import random

test_pct = 0.15
val_pct = 0.15

num_files = len(fnames)
num_test_files = int( test_pct * num_files )
num_valid_files = int( val_pct * num_files )
num_train_files = num_files - num_test_files - num_valid_files

random.shuffle(fnames)

test_fnames = fnames[: num_test_files ]
valid_fnames = fnames[num_test_files: num_test_files + num_valid_files]
train_fnames = fnames[num_test_files + num_valid_files: ]
```

```
In [0]: BATCH_SIZE = 16
```

```
In [0]: train_generator = generator(train_fnames, BATCH_SIZE, TARGET_SIZE)
        valid_generator = generator(valid_fnames, BATCH_SIZE, TARGET_SIZE)
        test_generator = generator(test_fnames, BATCH_SIZE, TARGET_SIZE)
```

```
In [0]: def append_to_history(history):
          for lst in history.history:
            if not lst in history_dict:
              history_dict[lst] = history.history[lst]
            else:
              history_dict[lst] += history.history[lst]
```

```
In [0]: import pickle

        history_dict = {}
        with open('history.pkl', 'wb') as f:
          pickle.dump(history_dict, f)
```

```
In [0]: def preprocess(img_path, num_channels, target_size=TARGET_SIZE):
          img_arr = resize_img_to_target_size(img_path, target_size, num_channels)
          img_arr = img_arr.reshape( (1, ) + img_arr.shape )
          return img_arr
```

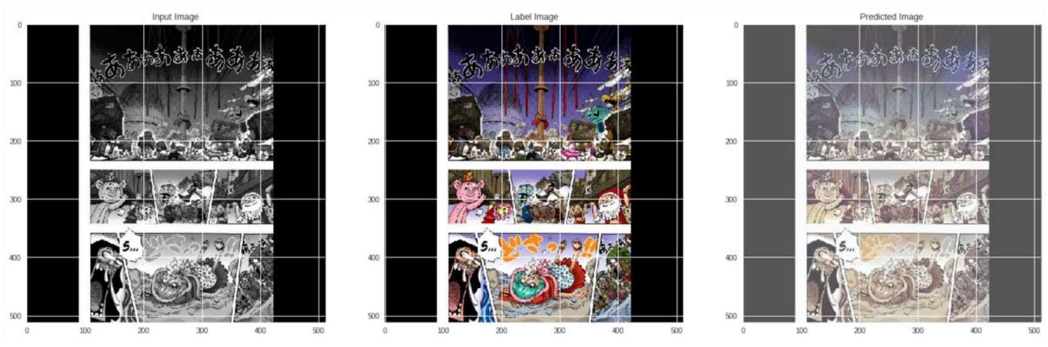
```
def visualize_model(model, fnames, num_files=1, save_to_disc=False):
  random.shuffle(fnames)
  for fname in fnames[:num_files]:
    input_fpath = os.path.join(input_images_dir, fname)
    label_fpath = os.path.join(label_images_dir, fname)

    fig = plt.figure(figsize=(25, 25))

    ax = fig.add_subplot(131)
    img_array = preprocess(input_fpath, 1)
    img = array_to_img(img_array[0] * 255)
    plt.imshow(img, cmap='gray')
    plt.title('Input Image')

    ax = fig.add_subplot(132)
    img_array = preprocess(label_fpath, 3)
    img = array_to_img(img_array[0] * 255)
    plt.imshow(img)
    plt.title('Label Image')

    ax = fig.add_subplot(133)
    img_arr = preprocess(input_fpath, 1)
    img = array_to_img(model.predict(img_arr)[0] * 255)
    plt.imshow(img)
    plt.title('Predicted Image')

    if save_to_disc:
        plt.savefig( os.path.join(results_dir, fname) )
    plt.show()
    plt.close(fig)
```

This is before any training

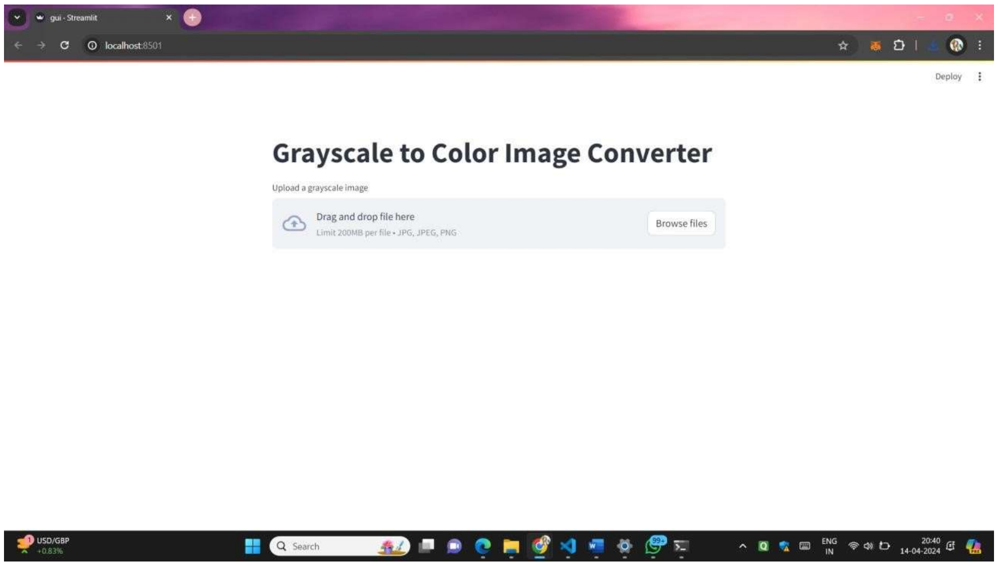
```
In [0]: visualize_model(model, valid_fnames, num_files=5)
```

```
Output hidden; open in https://colab.research.google.com (https://colab.resea
rch.google.com) to view.
```

```
best_loss = 0.0088

model.load_weights('model.h5')
visualize_model(model3, valid_fnames, num_files=5)

for epoch in range(100):

  print('Epoch', epoch+1)
  print('-' * 20)
  print('-' * 20)
  print()
  history = model.fit_generator(
                              train_generator,
                              steps_per_epoch=num_train_files//BATCH_SIZE,
                              epochs=1,
                              validation_data=valid_generator,
                              validation_steps=num_valid_files//BATCH_SIZE,
                              use_multiprocessing=True)
  try:
    with open('history.pkl', 'rb') as f:
      history_dict = pickle.load(f)
  except Exception as e:
    history_dict = {}

  append_to_history(history)

  with open('history.pkl', 'wb') as f:
    pickle.dump(history_dict, f)

  visualize_model(model, valid_fnames, num_files=5)

  loss = history.history['val_loss'][-1]
  if loss < best_loss:
    best_loss = loss
    model3.save('model.h5')

  print()
  print()
```



Output:



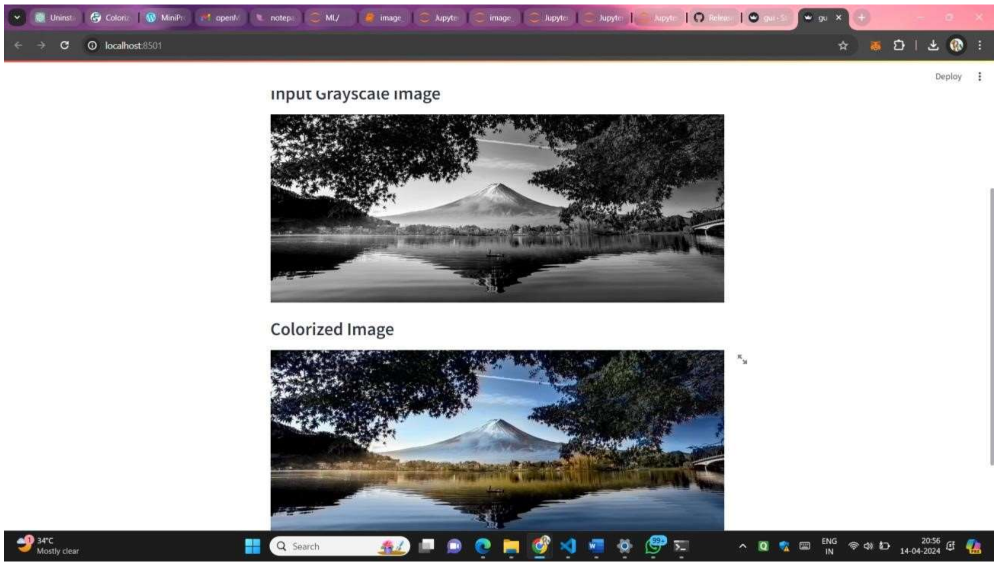
Grayscale to Color Image Converter

Upload a grayscale image

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files



Input Grayscale Image

Colorized Image

CONCLUSION

In conclusion, the task of colorizing old black and white images using deep learning techniques represents a fascinating intersection of art, technology, and cultural preservation. Through the utilization of convolutional neural networks and advanced image processing algorithms, we have demonstrated the capability to automatically add color to grayscale images, breathing new life into historical photographs and enabling a deeper connection with the past. While the results achieved are promising, further research and development in this field hold the potential to refine the accuracy and realism of colorization, expand the scope of application across diverse historical contexts, and contribute to the ongoing digitization and preservation efforts of visual heritage worldwide. As we continue to explore the frontiers of deep learning and image processing, the endeavor to colorize old black and white images remains an intriguing journey, offering both technical challenges and opportunities for creative expression and cultural enrichment.