TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM

KHOA CƠ KHÍ CHẾ TẠO MÁY

MÔN HỌC: THỊ GIÁC MÁY

BÁO CÁO CUỐI KỲ

ĐỀ TÀI: KHÔI PHỤC ẢNH XÁM VÀ CHUYỂN SANG ẢNH MÀU KẾT HỢP XỬ LÝ ẢNH VÀ AI

MHP:	**MAVI332529_03**	
SVTH:	**Diệp Khải Hoàn**	**20146090**
	Nguyễn Lê Phong	**20146516**
	Hồ Đăng Tú	**20146150**
GVHD:	**TS.Nguyễn Văn Thái**	

Tp Hồ Chí Minh, Tháng 6 năm 2023

NHIỆM VỤ MÔN HỌC XỬ LÝ ẢNH

Họ tên sinh viên:	**Diệp Khải Hoàn**	**MSSV: 20146090**
	Nguyễn Lê Phong	**MSSV: 20146516**
	Hồ Đăng Tú	**MSSV: 20146150**
Khóa:	**2020**	**Lớp: 201461(A,B)**

I. TÊN ĐỀ TÀI:

KHÔI PHỤC ẢNH XÁM VÀ CHUYỂN SANG ẢNH MÀU KẾT HỢP XỬ LÝ ẢNH VÀ AI

II. NHIỆM VỤ VÀ NỘI DUNG:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

III. NGÀY GIAO NHIỆM VỤ: 20/05/2023

IV. NGÀY HOÀN THÀNH NHIỆM VỤ: 00/00/2023

V. GIẢNG VIÊN HƯỚNG DẪN: **TS.Nguyễn Văn Thái**

Lời cảm ơn

Trong suốt quá trình thực hiện đề tài, mặc dù gặp nhiều khó khăn nhưng được sự hỗ trợ giúp đỡ kịp thời từ quý Thầy Cô và các bạn nên Final Project đã hoàn thành đúng tiến độ. Nhóm em xin chân thành cảm ơn thầy **Nguyễn Văn Thái** đã hướng dẫn cũng như tạo điều kiện cho bọn em trong suốt quá trình tìm hiểu và nghiên cứu đề tài.

Em cũng xin được gửi lời cảm ơn chân thành đến các thành viên trong lớp học **Thị Giác Máy** đã có những ý kiến đóng góp, bổ sung, cũng như động viên khích lệ giúp chúng em hoàn thành tốt đề tài.

Mặc dù nhóm thực hiện đề tài đã cố gắng hoàn thiện được báo cáo cuối kỳ, nhưng trong quá trình soạn thảo, cũng như kiến thức còn nhiều hạn chế nên có thể dẫn đến những thiếu sót. Vì thế nhóm thực hiện đề tài mong nhận được sự đóng góp ý kiến của quý thầy cô cùng các bạn sinh viên, để đồ án có thể hoàn thiện hơn nữa.

Sau cùng nhóm thực hiện xin chúc Thầy cô sức khoẻ, thành công và tiếp tục đào tạo những sinh viên giỏi đóng góp cho đất nước. Chúc các bạn sinh viên học tập thật tốt để không phụ công lao của Thầy Cô đã giảng dạy, hướng dẫn. Nhóm thực hiện xin chân thành cảm ơn.

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp. HCM, ngày ... tháng ... năm 2023

Giáo viên hướng dẫn

TS. Nguyễn Văn Thái

DANH SÁCH SINH VIÊN THAM GIA ĐỀ TÀI HỌC KỲ II NĂM HỌC 2022 - 2023

NHÓM 03 – STT: 02

Đề tài: Khôi phục ảnh xám và chuyển sang ảnh màu kết hợp xử lý ảnh và AI

STT	HỌ TÊN	MSSV	HOÀN THÀNH
1	Diệp Khải Hoàn	20146090	100%
2	Nguyễn Lê Phong	20146516	100%
3	Hồ Đăng Tú	20146150	100%

Ghi chú:

- Tỉ lệ % = 100%

Mục Lục:

Chương 1. TỔNG QUAN

1.1 Đặt vấn đề

Với thời gian trôi qua và những khó khăn của cuộc sống, những bức ảnh cũ thường bị mờ đi, mất màu, hay thậm chí bị hư hỏng vật lý. Tuy nhiên, với sự phát triển của xử lý ảnh và thuật toán trí tuệ nhân tạo, những bức ảnh quý giá của những chiến sĩ và người mẹ Việt Nam Anh hùng có thể được phục hồi và tái sinh một cách ấn tượng.

Những chiến sĩ và người mẹ Việt Nam Anh hùng đã dành cả thanh xuân và hết lòng hiến dâng cho đất nước. Họ là những người anh hùng, niềm tự hào của cả quốc gia. Nhưng thời gian trôi qua, những bức ảnh ghi lại những khoảnh khắc quý giá của họ dần mất đi sự rõ nét và sắc màu. Điều đó đã dẫn đến việc những kỷ niệm và sự tưởng nhớ đối với những người anh hùng này bị mờ mờ đi, mất đi sự rõ nét và không còn sống động như trước.

Tuy nhiên, nhờ vào sự tiến bộ của công nghệ phục hồi ảnh và trí tuệ nhân tạo, chúng ta có thể đem những bức ảnh đó trở về với vẻ đẹp và sự sống đầy màu sắc ban đầu.

Quá trình phục hồi ảnh cũ bằng xử lý ảnh và trí tuệ nhân tạo thường bắt đầu bằng việc quét hoặc nhập ảnh cũ vào máy tính. Sau đó, thông qua việc áp dụng các thuật toán và công nghệ xử lý ảnh, các vết nứt, mờ mờ và mất màu trên bức ảnh được phát hiện và khắc phục, tìm cách tái tạo màu sắc ban đầu, làm sắc nét hình ảnh và tái tạo các chi tiết quan trọng để mang lại cho bức ảnh sự sống động và rõ ràng.

Phục hồi ảnh cũ bằng xử lý ảnh và trí tuệ nhân tạo không chỉ là việc khôi phục những bức ảnh đã mờ đi, mất màu, mà còn là cách để chúng ta tôn vinh và tưởng nhớ những chiến sĩ và người mẹ Việt Nam Anh hùng hoặc có thể những tấm hình của ông cha ta ngày xưa. Nhờ công nghệ này, chúng ta có thể gìn giữ và lan tỏa những giá trị quý giá của quá khứ, để không bao giờ quên được những khoảnh khắc kỷ niệm thời xa xưa ấy.

1.2 Lý do chọn đề tài

Phục hồi ảnh cũ trắng đen thành ảnh có màu không chỉ là quá trình tái tạo, mà còn là một cách để khơi lại quá khứ, tôn vinh giá trị và kỷ niệm, tăng cường tương tác và hiểu biết, và khai thác tiến bộ công nghệ và trí tuệ nhân tạo.

Bằng việc phục hồi, chúng ta mang lại sự sống động cho những bức ảnh cổ, tái hiện lại màu sắc chân thực và hình dung rõ nét về quá khứ. Điều này giúp chúng ta tạo kết nối sâu sắc với lịch sử và di sản của chúng ta, đồng thời tôn vinh những người anh hùng và sự đóng góp của họ.

Bên cạnh đó, việc phục hồi ảnh cũ thành ảnh có màu mở ra khả năng tương tác và hiểu biết sâu hơn. Màu sắc đem đến sự rõ ràng và chi tiết, cho phép chúng ta nhìn thấy và cảm nhận về quá khứ một cách trực quan hơn. Điều này giúp chúng ta khám phá và tiếp cận lịch sử một cách đa chiều và sâu sắc.

Tóm lại, việc phục hồi ảnh cũ trắng đen thành ảnh có màu không chỉ là một quá trình tái tạo, mà còn là một cách để tái hiện lại quá khứ, tôn vinh giá trị và kỷ niệm, tăng cường tương tác và hiểu biết, và tận dụng tiến bộ công nghệ và trí tuệ nhân tạo.

1.3 Mục tiêu đề tài

Xây dựng được một chương một chương trình phục hồi ảnh cũ trắng đen và chuyển ảnh trắng đen sang ảnh màu bằng Tkinter của Python.

Củng cố và vận dụng lý thuyết xử lý ảnh đã học, nghiên cứu và tìm hiểu các thuật toán xử lý ảnh để xây dựng được một chương trình xử lý ảnh áp dụng được vào thực tế trong cuộc sống. Từ đó đưa ra đánh giá về kết quả nhận được và những hạn chế của chương trình.

1.4 Phương pháp nghiên cứu

+ Tìm hiểu các phương pháp cải thiện ảnh bằng lập trình python.
+ Tìm hiểu về các thuật toán trong xử lý ảnh để tái tạo màu sắc cho ảnh trắng đen.
+ Tìm hiểu về việc huấn luyện mô hình chuyển ảnh trắng đen sang ảnh màu.
+ Tham khảo giảng viên hướng dẫn và bạn bè.

1.5 Bố cục của đề tài

Chương 1: Tổng quan

Chương 2: Xây dựng chương trình

Chương 3: Hoạt động của chương trình

Chương 4: Kết luận và hướng phát triển của đề tài

Chương 2. XÂY DỰNG CHƯƠNG TRÌNH

2.1 Các thư viện dùng trong chương trình

- **OpenCV:** OpenCV là tên viết tắt của open source computer vision library – có thể được hiểu là một thư viện nguồn mở cho máy tính. Cụ thể hơn OpenCV là kho lưu trữ các mã nguồn mở được dùng để xử lý hình ảnh, phát triển các ứng dụng đồ họa trong thời gian thực. OpenCV cho phép cải thiện tốc độ của CPU khi thực hiện các hoạt động real time. Nó còn cung cấp một số lượng lớn các mã xử lý phục vụ cho quy trình của thị giác máy tính hay các mô hình học máy khác.

- **PIL:** Python cho phép giải quyết vấn đề các về xử lý hình ảnh thông qua thư viện Imaging (PIL)**.** Thư viện này hỗ trợ nhiều định dạng tập tin, và cung cấp khả năng xử lý hình ảnh và đồ hoạ mạnh mẽ.

- **Tkinter:** Tkinter là thư viện GUI tiêu chuẩn cho Python. Tkinter trong Python cung cấp một cách nhanh chóng và dễ dàng để tạo các ứng dụng GUI. Tkinter cung cấp giao diện hướng đối tượng cho bộ công cụ Tk GUI.

- **Module Keras:** Keras là một open source cho Neural Network được viết bởi ngôn ngữ Python. Giúp hỗ trợ xây dựng CNN, RNN hoặc cả hai. Thường được sử dụng vì sự đơn giản, dễ nắm bắt hơn các thư viện khác.

2.2 Các thư viện dùng trong chương trình

2.2.1. Tiền xử lý dữ liệu

```
10  # Lọc nhiễu muối tiêu
11  def Loc_TKTT_trung_vi(img, ksize):
12      m, n = img.shape
13      img_ket_qua_anh_loc_Trung_vi = np.zeros([m, n])
14      h = (ksize - 1) // 2
15      padded_img = np.pad(img, (h, h), mode='reflect')
16      for i in range(m):
17          for j in range(n):
18              vung_anh_kich_thuoc_k = padded_img[i:i+ksize, j:j+ksize]
19              gia_tri_TV = np.median(vung_anh_kich_thuoc_k)
20              img_ket_qua_anh_loc_Trung_vi[i, j] = gia_tri_TV
21      return img_ket_qua_anh_loc_Trung_vi
```

Giải thích:

+ Với ảnh đầu vào bị nhiều chấm trắng, đen do thời gian làm ảnh xuất hiện những dấu hiệu trên, để giải quyết vấn đề trên nhóm đã đề xuất sử dụng bộ lọc "**thống kê thứ tự trung vị**" để cải thiện ảnh lọc đi những vết chấm trắng đen xuất hiện trên ảnh.

+ Ta xây dựng một hàm **"LOC_TKTT_trung_vi"**

```
def Loc_TKTT_trung_vi(img, ksize):
```

+ Hàm nhận vào ảnh đầu vào img và kích thước cửa sổ ksize.

+ Lấy kích thước ảnh img và lưu vào biến m và n.

```
m, n = img.shape
```

+ Tạo một ảnh mới **img_ket_qua_anh_loc_Trung_vi** có cùng kích thước với ảnh đầu vào, ban đầu tất cả các pixel được gán giá trị 0.

```
img_ket_qua_anh_loc_Trung_vi = np.zeros([m, n])
```

+ Tính toán số hàng và cột được thêm vào ảnh ban đầu để tạo viền (padding). Biến h được tính bằng cách lấy giá trị trung bình số lẻ lớn nhất của ksize trừ 1, sau đó chia cho 2. Sử dụng hàm np.pad để thêm viền cho ảnh đầu vào, sử dụng chế độ phản xạ (mode='reflect'). Viền này giúp xử lý các pixel ở biên của ảnh.

```
h = (ksize - 1) // 2
padded_img = np.pad(img, (h, h), mode='reflect')
```

+ Bắt đầu từ hàng và cột thứ 0, lặp qua từng pixel trong ảnh đầu vào.

+ Đối với mỗi pixel, tạo ra một vùng ảnh kích thước ksize x ksize (cửa sổ) tại vị trí pixel đó trong ảnh đã được thêm viền.

```
vung_anh_kich_thuoc_k = padded_img[i:i+ksize, j:j+ksize]
```

+ Sử dụng hàm **np.median** để tính giá trị trung vị của các pixel trong cửa sổ.

```
gia_tri_TV = np.median(vung_anh_kich_thuoc_k)
```

+ Gán giá trị trung vị vừa tính được vào pixel tương ứng trong ảnh kết quả **img_ket_qua_anh_loc_Trung_vi.**

```
img_ket_qua_anh_loc_Trung_vi[i, j] = gia_tri_TV
```

+ Sau khi xử lý tất cả các pixel, hàm trả về ảnh đã lọc trung vị **img_ket_qua_anh_loc_Trung_vi.**

```
return img_ket_qua_anh_loc_Trung_vi
```

+ Kết quả thu được:

ảnh gốc bị nhiễu muối tiêu

ảnh sau khi lọc nhiễu muối tiêu

+ Sau khi lọc xong ta nhận thấy ảnh khá mờ, ta sẽ làm nét ảnh bằng bộ lọc trung vị

“loc_trung_vi”

+ Hàm loc_trung_vi(img) nhận vào ảnh đầu vào img.

```
def loc_trung_vi(img):
```

+ Lấy kích thước của ảnh img và gán cho biến m và n.

```
 img_new = np.zeros([m, n])
```

+ Tạo một ảnh mới img_new có cùng kích thước với ảnh đầu vào, ban đầu tất cả các pixel được gán giá trị 0.

```
img_new = np.zeros([m, n])
```

+ Bắt đầu từ hàng và cột thứ 1 và lặp qua từng pixel trong ảnh, loại trừ các pixel ở biên.

```
for i in range(1, m - 1):
    for j in range(1, n - 1):
```

+ Đối với mỗi pixel, tạo ra một danh sách temp chứa giá trị của các pixel lân cận trong cửa sổ 3x3.

```
for i in range(1, m - 1):
    for j in range(1, n - 1):
        temp = [img[i - 1, j - 1],  #Giá trị pixel nằm ở hàng trên và cột trái so với pixel hiện tại
                img[i - 1, j],      #Giá trị pixel nằm ở hàng trên và cùng cột với pixel hiện tại.
                img[i - 1, j + 1],  #Giá trị pixel nằm ở hàng trên và cột phải so với pixel hiện tại.
                img[i, j - 1],      #Giá trị pixel nằm ở cùng hàng và cột trái so với pixel hiện tại.
                img[i, j],          #Giá trị pixel hiện tại.
                img[i, j + 1],      #Giá trị pixel nằm ở cùng hàng và cột phải so với pixel hiện tại.
                img[i + 1, j - 1],  #Giá trị pixel nằm ở hàng dưới và cột trái so với pixel hiện tại.
                img[i + 1, j],      #Giá trị pixel nằm ở hàng dưới và cùng cột với pixel hiện tại.
                img[i + 1, j + 1]]  #Giá trị pixel nằm ở hàng dưới và cột phải so với pixel hiện tại.
```

➔ danh sách temp được sử dụng để tính toán giá trị trung vị của các pixel trong cửa sổ 3x3 xung quanh pixel hiện tại. Sau khi có danh sách này, nó được sắp xếp theo thứ tự tăng dần bằng cách sử dụng hàm **sorted(temp)**. Cuối cùng, phần tử thứ 4 trong danh sách **(temp[4])** được chọn làm giá trị trung vị và gán cho pixel tương ứng trong ảnh kết quả.

+ Các giá trị này sau đó được sắp xếp theo thứ tự tăng dần bằng cách sử dụng hàm **sorted(temp).**

```
temp = sorted(temp)
```

+ Gán giá trị trung vị (phần tử ở giữa) từ danh sách temp cho pixel tương ứng trong ảnh **img_new**.

```
img_new[i, j] = temp[4]
```

+ Sau khi xử lý tất cả các pixel, hàm trả về ảnh đã lọc trung vị **img_new**.

```
return img_new
```

ảnh gốc bị nhiễu muối tiêu

ảnh sau khi lọc nhiễu muối tiêu

ảnh sau khi làm sắc nét

2.2.2. Tạo dữ liệu và train AI

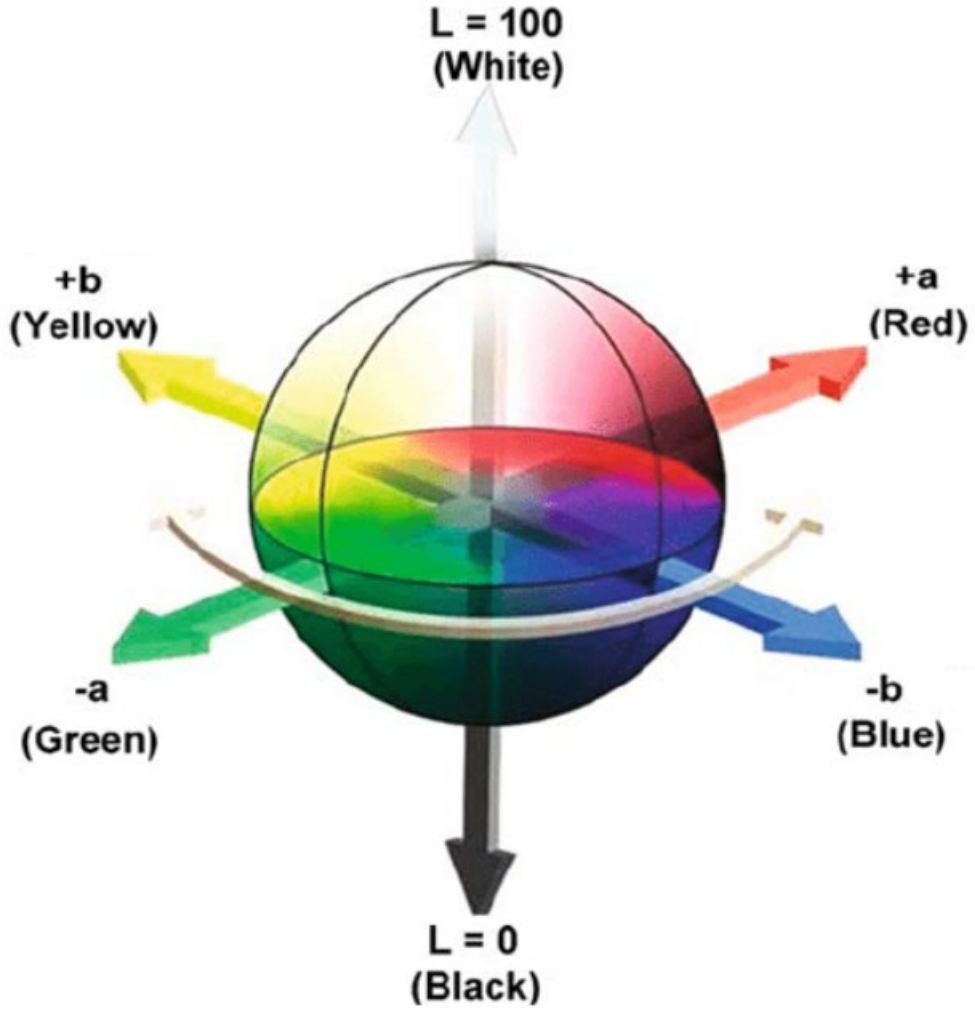
*Tìm hiểu về hệ màu lab

Việc sử dụng màu là một trong những yếu tố quyết định đến chất lượng của thành phẩm trong quá trình in ấn cũng như một vài quá trình khác. Bên cạnh hệ màu RGB, CMYK thì hệ màu Lab cũng đang được sử dụng một cách phổ biến. CIE là chữ viết tắt tiêu đề tiếng Pháp của. Ủy ban Quốc tế về Chiếu sáng "The Commission Internationale d'Eclairage". Hệ màu CIE L*a*b* (CIE Lab) ra đời năm 1976 là sự cải tiến của hệ màu Hunter Lab. Mô hình CIE L*a*b được xây dựng dựa trên khả năng cảm nhận màu sắc của mắt người. Các giá trị Lab mô tả tất cả các màu mà mắt của một người bình thường có thể nhìn thấy được. Hệ màu Lab là mô hình dạng hình cầu được biểu diễn bằng tổ hợp 3 kênh xử lý (3 trục):

Trục L: là trục thẳng đứng biểu diễn độ sáng của màu. Có giá trị từ đen (0) đến trắng (100)

Trục a: Chứa các giá trị màu từ màu xanh lá cây (âm) đến màu đỏ (dương)

Trục b: Chứa các giá trị màu từ màu xanh dương (âm) đến màu vàng (dương)



*Tạo dữ liệu và tiến hành train cho mô hình (sử dụng google colab)

```
from keras.preprocessing.image import ImageDataGenerator
from skimage.color import rgb2lab
import numpy as np
import os
from keras.utils import load_img, img_to_array

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

Thư viện ImageDataGenerator từ keras.preprocessing.image được sử dụng để tạo ra các biến đổi dữ liệu hình ảnh trong quá trình huấn luyện mô hình. Nó cho phép tạo ra các phiên bản biến đổi của các hình ảnh gốc để tăng cường dữ liệu huấn luyện, như xoay, cắt cụt, lật ngang, thu phóng, và nhiều biến đổi khác.
Thư viện rgb2lab từ skimage.color được sử dụng để chuyển đổi không gian màu RGB của các hình ảnh sang không gian màu Lab.
Thư viện numpy (imported as np) là một thư viện phổ biến trong Python dùng để làm việc với mảng và ma trận.
Thư viện os cung cấp các phương thức để làm việc với hệ điều hành, bao gồm thao tác trên các tệp và thư mục. Trong đoạn code này, nó được sử dụng để làm việc với các đường dẫn tệp và thư mục, đảm bảo quá trình đọc dữ liệu hình ảnh từ thư mục được thực hiện đúng đường dẫn.
Thư viện load_img và img_to_array từ keras.utils được sử dụng để đọc và chuyển đổi hình ảnh thành dạng mảng numpy. load_img được sử dụng để tải hình ảnh từ một đường dẫn cụ thể và trả về một đối tượng hình ảnh. Sau đó, img_to_array được sử dụng để chuyển đổi đối tượng hình ảnh thành mảng numpy.

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

Dùng để kết nối google colab với driver để lấy dữ liệu từ driver

```
folder = '/content/drive/My Drive/Colab Notebooks/dataset1/'
X = []
for imagename in os.listdir(folder):
    X.append(img_to_array(load_img(folder+imagename)))
X = np.array(X, dtype=float)

# Set up train and test data
split = int(0.95*len(X))
Xtrain = X[:split]
Xtrain = 1.0/255*Xtrain
```

folder = '/content/drive/My Drive/Colab Notebooks/dataset1/': Đây là đường dẫn tới thư mục chứa các tệp hình ảnh trong Google Drive. Đường dẫn này được lưu vào biến folder.

X = []: Tạo một danh sách rỗng X để lưu trữ dữ liệu hình ảnh.
for imagename in os.listdir(folder):: Với mỗi tệp hình ảnh trong thư mục folder, được lấy ra theo danh sách tệp tin sử dụng os.listdir, tiến hành các bước sau:
X.append(img_to_array(load_img(folder+imagename))): Đọc tệp hình ảnh bằng load_img, chuyển đổi thành mảng numpy bằng img_to_array, và sau đó thêm mảng này vào danh sách X.
X = np.array(X, dtype=float): Chuyển đổi danh sách X thành một mảng numpy. Đồng thời, kiểu dữ liệu của các phần tử trong mảng được đặt là kiểu số thực (float).
split = int(0.95*len(X)): Xác định chỉ số split bằng 0.95 nhân với số lượng phần tử trong mảng X (được tính bằng len(X)) và chuyển đổi kết quả thành số nguyên.
Xtrain = X[:split]: Gán mảng con X từ đầu đến chỉ số split cho biến Xtrain. Điều này tạo ra một mảng con chứa dữ liệu huấn luyện. (95% dùng để train còn 5% dùng để test)
Xtrain = 1.0/255*Xtrain: Chuẩn hóa dữ liệu huấn luyện Xtrain bằng cách chia mỗi phần tử của mảng cho 255. Điều này giúp đưa dữ liệu về phạm vi từ 0 đến 1 để tối ưu quá trình huấn luyện mô hình.

```
#CNN model
from keras.layers import Conv2D, UpSampling2D
from keras.callbacks import TensorBoard
from keras.models import Sequential

model = Sequential()

#Input Layer
model.add(Conv2D(64, (3, 3), input_shape=(256, 256, 1), activation='relu', padding='same'))

#Hidden Layers
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))
```

```
model.add(UpSampling2D((2, 2)))

model.summary()

#Compiling the CNN
model.compile(optimizer='rmsprop', loss='mse', metrics = ['accuracy'])
```

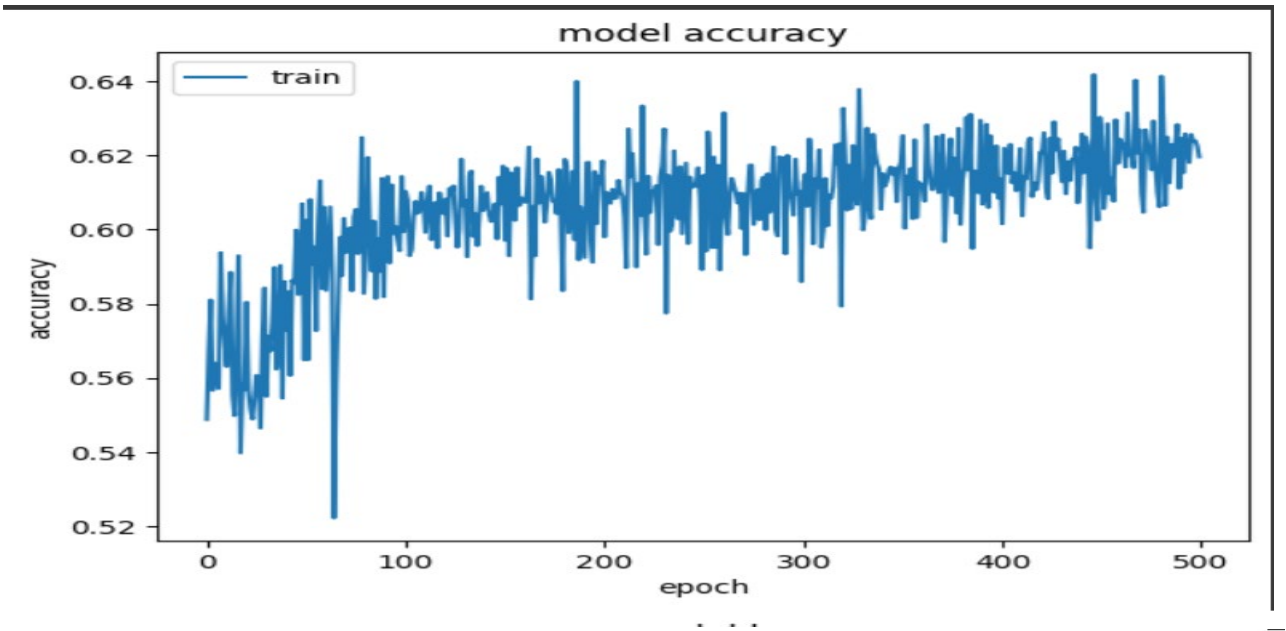
Đoạn mã trên định nghĩa một mô hình CNN (Convolutional Neural Network) bằng cách sử dụng các lớp của thư viện Keras:
model = Sequential(): Tạo một mô hình tuần tự (sequential model) bằng cách khởi tạo đối tượng Sequential().
model.add(Conv2D(64, (3, 3), input_shape=(256, 256, 1), activation='relu', padding='same')): Thêm một lớp Conv2D với 64 bộ lọc kích thước (3, 3). Đây là lớp đầu tiên trong mạng và được sử dụng làm lớp đầu vào. Kích thước đầu vào của hình ảnh là (256, 256, 1) (chiều cao, chiều rộng, số kênh), và hàm kích hoạt được sử dụng là ReLU.
Các lớp tiếp theo là các lớp Conv2D được thêm vào mô hình. Các lớp này được kết nối tuần tự và tạo thành các lớp ẩn trong mạng CNN. Mỗi lớp Conv2D có các tham số như kích thước bộ lọc, hàm kích hoạt, và cách tính đệm (padding) và bước đi (stride). Một số lớp còn sử dụng lớp UpSampling2D để tăng kích thước hình ảnh.
model.summary(): In ra tổng quan về kiến trúc của mô hình, hiển thị số lượng tham số trong mô hình và kích thước của các lớp.
model.compile(optimizer='rmsprop', loss='mse', metrics=['accuracy']): Biên dịch mô hình với các tham số như thuật toán tối ưu hóa (optimizer), hàm mất mát (loss function), và các độ đo (metrics) để đánh giá hiệu suất của mô hình trong quá trình huấn luyện. Trong trường hợp này, thuật toán tối ưu hóa là RMSprop, hàm mất mát là mean squared error (mse), và độ đo là accuracy.

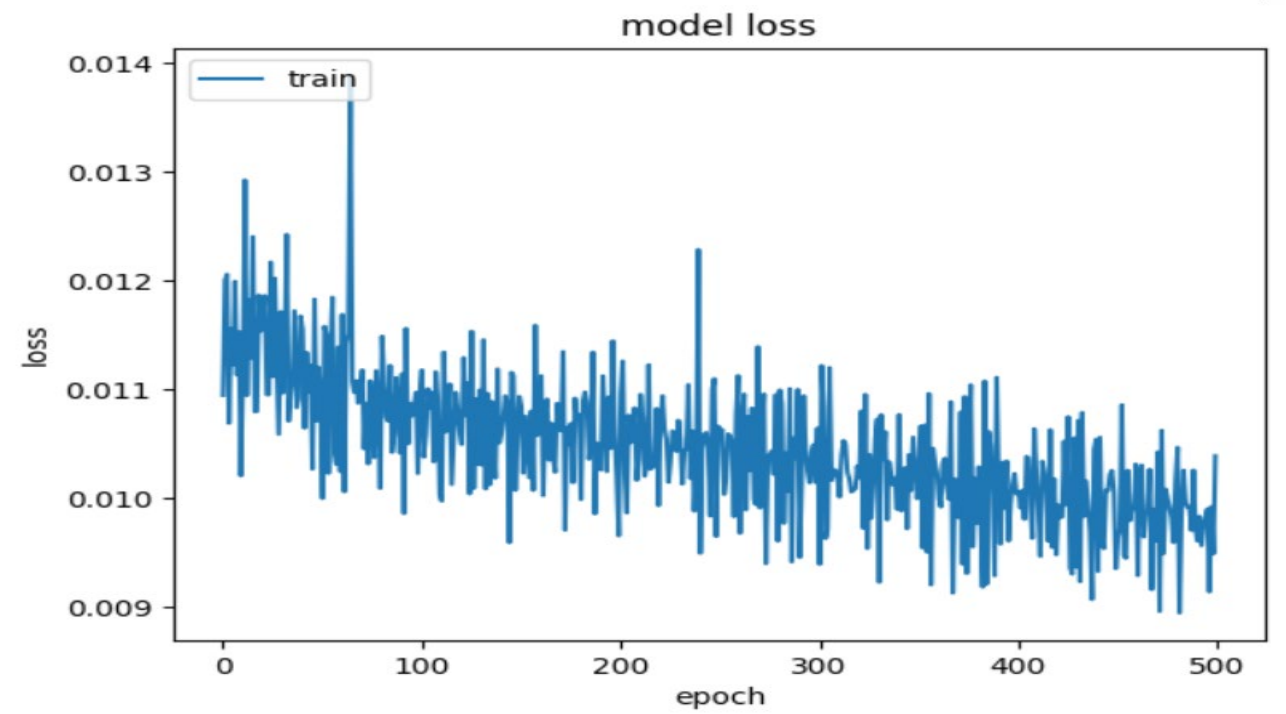
```
# Image transformer
datagen = ImageDataGenerator(
        shear_range=0.2,
        zoom_range=0.2,
        rotation_range=20,
        horizontal_flip=True)

# Generate training data
batch_size = 10
def image_a_b_gen(batch_size):
    for batch in datagen.flow(Xtrain, batch_size=batch_size):
        lab_batch = rgb2lab(batch)
        X_batch = lab_batch[:,:,:,0]
        Y_batch = lab_batch[:,:,:,1:] / 128
        yield (X_batch.reshape(X_batch.shape+(1,)), Y_batch)
```

Đoạn mã trên sử dụng lớp ImageDataGenerator từ thư viện Keras để tạo dữ liệu huấn luyện cho mô hình CNN.
datagen = ImageDataGenerator(...): Đối tượng datagen được tạo ra từ lớp ImageDataGenerator. Đây là một công cụ để tạo ra các biến đổi ảnh trong quá trình huấn luyện.
shear_range=0.2: Tham số này xác định phạm vi biến đổi cắt cụt (shear transform) cho các ảnh.
zoom_range=0.2: Tham số này xác định phạm vi biến đổi thu phóng (zoom in/out) cho các ảnh.
rotation_range=20: Tham số này xác định phạm vi biến đổi xoay (rotation) cho các ảnh.
horizontal_flip=True: Tham số này xác định xem ảnh có bị lật ngang hay không.
def image_a_b_gen(batch_size): Định nghĩa một hàm generator có tên là image_a_b_gen, nhận vào tham số batch_size (kích thước gói dữ liệu).
Trong vòng lặp for batch in datagen.flow(Xtrain, batch_size=batch_size), hàm generator này lặp qua từng gói dữ liệu được tạo ra bởi datagen dựa trên dữ liệu huấn luyện Xtrain và kích thước gói batch_size.
Các bước xử lý dữ liệu được thực hiện trên từng gói dữ liệu:
lab_batch = rgb2lab(batch): Chuyển đổi không gian màu của các ảnh trong gói dữ liệu từ RGB sang LAB.
X_batch = lab_batch[:,:,:,0]: Lấy kênh đầu tiên (độ sáng) của ảnh làm đầu vào (X_batch).
Y_batch = lab_batch[:,:,:,1:] / 128: Lấy các kênh còn lại (độ màu) của ảnh chia cho 128 để chuẩn hóa giá trị về khoảng [-1, 1].
yield (X_batch.reshape(X_batch.shape+(1,)), Y_batch): Trả về một cặp dữ liệu huấn luyện là đầu vào (X_batch) và đầu ra (Y_batch). Đầu vào được reshape lại để thêm một chiều thứ tư (1) để phù hợp với kích thước của mạng CNN.
tensorboard = TensorBoard(log_dir="/output/beta_run")
trainedmodel = model.fit_generator(image_a_b_gen(batch_size), callbacks=[tensorboard],epochs=500, steps_per_epoch=30)
Trong đoạn mã trên:
tensorboard = TensorBoard(log_dir="/output/beta_run"): Đối tượng tensorboard được tạo ra từ lớp TensorBoard trong Keras. Nó được sử dụng để ghi log và tạo báo cáo cho quá trình huấn luyện của mô hình. Tham số log_dir xác định đường dẫn nơi lưu trữ các tệp log và báo cáo của TensorBoard.
trainedmodel = model.fit_generator(image_a_b_gen(batch_size), callbacks=[tensorboard], epochs=500, steps_per_epoch=30): Phương thức fit_generator() được sử dụng để huấn luyện mô hình trên dữ liệu được tạo ra bởi image_a_b_gen(batch_size). Các tham số:
image_a_b_gen(batch_size): Đây là hàm generator được sử dụng để tạo dữ liệu huấn luyện.

callbacks=[tensorboard]: Danh sách các đối tượng callback được áp dụng trong quá trình huấn luyện. Trong trường hợp này, chỉ có một callback là tensorboard, được sử dụng để ghi log và tạo báo cáo TensorBoard.
epochs=500: Số lượng epochs (vòng lặp) trong quá trình huấn luyện.
steps_per_epoch=30: Số lượng bước huấn luyện trong mỗi epoch.





Câu lệnh trên được sử dụng để lưu mô hình mạng nơ-ron và trọng số của nó vào các file riêng biệt.

Đầu tiên, `model.to_json()` được sử dụng để chuyển đổi mô hình thành định dạng JSON, tức là biểu diễn dạng văn bản của cấu trúc mô hình. Kết quả của chuyển đổi này là một chuỗi JSON.

Sau đó, câu lệnh `with open("model.json", "w") as json_file` mở một tệp có tên là "model.json" để ghi nội dung JSON. Tệp này sẽ chứa thông tin về kiến trúc của mô hình.

Tiếp theo, `json_file.write(model_json)` được sử dụng để ghi chuỗi JSON vào tệp "model.json".

Cuối cùng, `model.save_weights("model.h5")` được sử dụng để lưu trọng số của mô hình vào tệp "model.h5". Trọng số được lưu dưới dạng một tệp nhị phân có định dạng HDF5.

2.2.3. Tiến hành tạo màu cho ảnh xám từ model đã được train

Sau khi ảnh đầu vào đã được xử lý, ta tiến hành chuyển đổi dữ liệu sao cho phù hợp với đầu vào của model.

```
imgTV_3x3 = imgTV_3x3.astype(np.uint8)
# cv2.imwrite('daura_1.jpg', imgTV_3x3)

imgTV_3x3_3 = cv2.cvtColor(imgTV_3x3, cv2.COLOR_GRAY2BGR)
imgTV_3x3_3 = Image.fromarray(imgTV_3x3_3)
resized_image = imgTV_3x3_3.resize((256, 256))

colorize = []
colorize.append(img_to_array(resized_image))
colorize = np.array(colorize, dtype=float)
colorize = rgb2lab(1.0/255 * colorize)[:, :, :, 0]
colorize = colorize.reshape(colorize.shape + (1,))
```

+Tiến hành chuyển đổi ảnh sang dạng np.uint8 mục đích để chuyển đổi từ ảnh xám 1 kênh màu sang ảnh xám có đủ 3 kênh màu

+ Chuyển đổi sang dạng image để tiến hành resize

+Tạo một mảng và thêm ảnh kích thước chuẩn vào, chuyển mảng sang dạng float và chuyển sang kênh lab và lấy kênh màu lab và đưa về dạng đầu vào chuẩn cho model (1,256,256,1)

```
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("model.h5")
```

Tiến hành load model lên.

```
output = loaded_model.predict(colorize)
output = output * 128
```

Tiến hành dự đoán 2 kênh màu còn lại, do cả 2 kênh đều có giá trị từ -128 đến 128 mà đầu ra từ -1 đến 1 nên tiến hành nhân với 128.

```
cur = np.zeros((256, 256, 3))
```

Tạo một mảng mới với 3 kênh

```
sharpened_image_L = cv2.GaussianBlur(colorize[:,:,:,0], (0, 0), 3)
colorize[:, :, :, 0] = cv2.addWeighted(colorize[:, :, :, 0], 1.5,
sharpened_image_L, -0.5, 0)
cur[:, :, 0] = colorize[:, :, :, 0]
```

- `sharpened_image_L = cv2.GaussianBlur(colorize[:,:,:,0], (0, 0), 3)`: Ở đây, ảnh L* (độ sáng) của ảnh được lấy từ mô hình dự đoán màu sắc được lọc bằng GaussianBlur với kernel có kích thước (0, 0) và độ lệch chuẩn là 3. Kỹ thuật GaussianBlur làm mờ ảnh để loại bỏ các chi tiết nhỏ và làm nổi bật các cạnh.
- `colorize[:, :, :, 0] = cv2.addWeighted(colorize[:, :, :, 0], 1.5, sharpened_image_L, -0.5, 0)`: Ở đây, ảnh L* sau khi được làm mờ được trộn (blend) lại với ảnh L* ban đầu bằng cách sử dụng hàm addWeighted của OpenCV. Trong công thức này, ảnh L* ban đầu được nhân với hệ số 1.5, ảnh L* sau khi làm mờ được nhân với hệ số -0.5, và sau đó hai ảnh này được cộng lại với nhau. Mục đích của việc này là tăng độ sắc nét của ảnh L* sau khi được dự đoán.
- `cur[:, :, 0] = colorize[:, :, :, 0]`: Ở đây, giá trị ảnh L* sau khi đã được tăng cường sắc nét được gán vào kênh L* của ảnh kết quả cuối cùng (cur).

```
for x in range(256):
    for y in range(256):
        cur[x, y, 1] = output[:, x, y, 0] * 2
        cur[x, y, 2] = output[:, x, y, 1] * 2
```

Gán giá trị cho 2 kênh màu còn lại, nhân với hệ số 2 giúp tăng cường màu sắc cho bức hình.

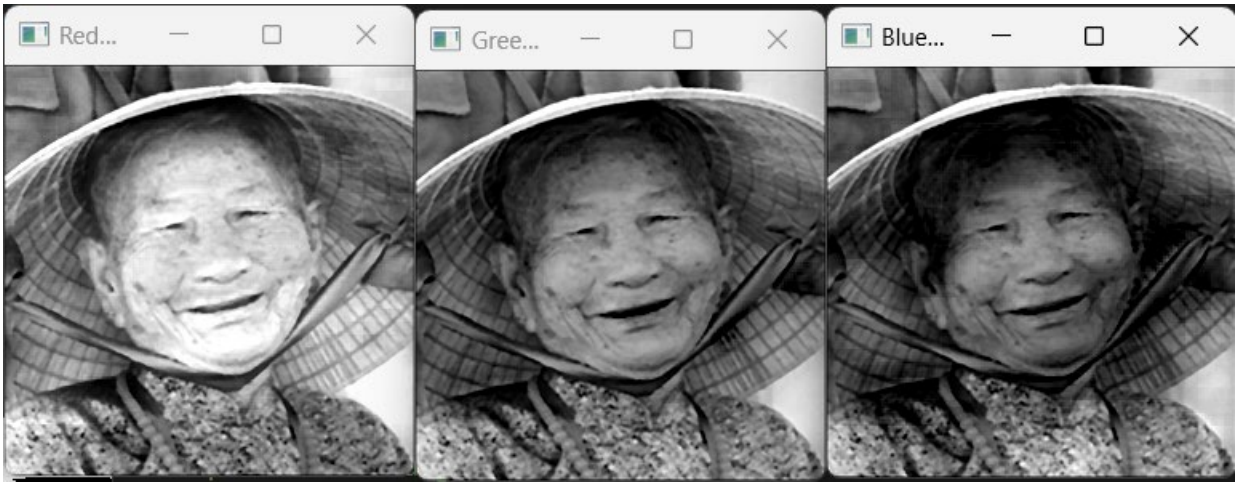
```
resImage = lab2rgb(cur)
resImage = (resImage * 255).astype(np.uint8)
```
Tiến hành chuyển đổi hệ lab sang RGB và tiến hành đưa về dạng pixel chuẩn từ 0 đến 255

Như vậy ta đã có đầu ra hoàn chỉnh đã tô màu.

2.2.4. Lọc nhiễu

```
red_channel = resImage[:, :, 0]
green_channel = resImage[:, :, 1]
blue_channel = resImage[:, :, 2]
```

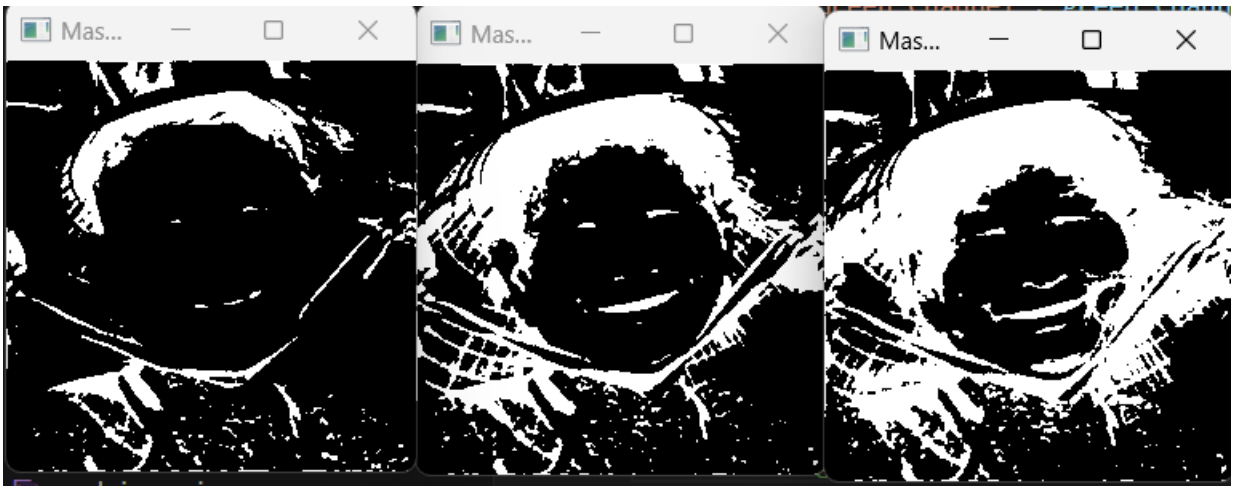
Tiền hình trích xuất kênh màu của từng kênh (do đang là ảnh RGB nên kênh RED sẽ là 0), ảnh của các kênh sẽ được hiển thị dưới dạng ảnh xám.



Ta thấy một số phần bị nhiễu đen, do vậy ta sẽ tiến hành xử lí những chỗ này.

```
_, mask_red = cv2.threshold(red_channel, 60, 255, cv2.THRESH_BINARY_INV)
_, mask_green = cv2.threshold(green_channel, 60, 255, cv2.THRESH_BINARY_INV)
_, mask_blue = cv2.threshold(blue_channel, 60, 255, cv2.THRESH_BINARY_INV)
```

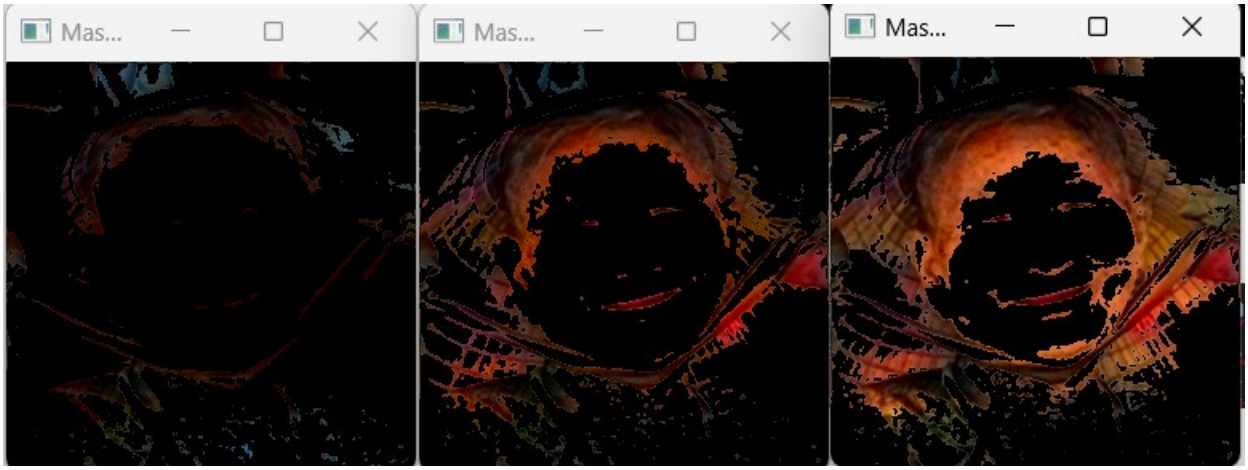
Tiến hành ngưỡng hóa, nếu các ảnh có giá trị pixel thì sẽ được gán màu trắng và ngược lại, ta được 3 mask sau:




```
masked_image_red = cv2.bitwise_and(resImage_BGR,resImage_BGR, mask=mask_red)
masked_image_green = cv2.bitwise_and(resImage_BGR,resImage_BGR, mask=mask_green)
```

```
masked_image_blue = cv2.bitwise_and(resImage_BGR,resImage_BGR, mask=mask_blue)
```

Tiến hành và 2 ảnh gốc tại những nơi mặt nạ mask có pixel khác 0, ta được:



Những vùng có màu là những vùng tương ứng với từng ảnh, kênh màu của nó tại đó đang bị nhiễu đen, ta sử dụng and trên ảnh gốc mục đích giúp giữ lại được giá trị pixel tại những vị trí đó để tiến hành xử lý.

*Hàm nội suy tuyến tính:

```
def linear_interpolation_channel_x(channel_image,threshold,pixel,x,y,dividend):
    pixel_extra=pixel
    neighbor_x, neighbor_y = x, y
    while (pixel_extra > threshold):
        if(neighbor_x>=255):break
        neighbor_x += 1
        _,_,pixel_extra = channel_image.getpixel((neighbor_x,neighbor_y))
    _,_,neighbor_value = channel_image.getpixel((neighbor_x,neighbor_y))
    pixel_return_x = np.interp(x, [x, neighbor_x], [pixel, neighbor_value])
    pixel_return_x = pixel_return_x/dividend
    return float(pixel_return_x)
```

Ta sẽ tạo ra một hàm nội suy tuyến tính, có đầu vào là kênh màu đang xét, ngưỡng cho là bị nhiễu, giá trị pixel đang xét, tọa độ x,y và số bị chia.

Lúc đầu t sẽ lưu lại vị trí tọa độ của pixel bị nhiễu (pixel>threshold)(vì sau đó ta sẽ xét các ảnh mask_image có viền đen, nên sẽ sẽ xem là các giá trị bị nhiễu sẽ lớn hơn threshold đồng thời để tránh được một số vị trí có màu xẫm tự nhiên)

Sau đó, ta sẽ tăng giá trị x (xét pixel theo hàng ngang) mà tịa vị trí đó không còn nhiễu, lúc này ta sẽ tuyến tính hóa giá trị pixel tại điểm x nhiễu so với vị trí x không nhiễu và giá trị pixel không nhiễu ở đó ta sẽ được giá trị pixel thay thế, ta sẽ thêm vào đó 1 bộ chia mục đích có thể điều chỉnh giá trị pixel trả về linh hoạt hơn.

Tương tự ta cũng sẽ có 1 hàm tương tự nhưng theo trục y.

```
for x in range(256):
    for y in range(256):
        b_r,g_r,r_r = masked_image_red.getpixel((x,y))
        b_g,g_g,r_g = masked_image_green.getpixel((x,y)) #anh BGR
        b_b,g_b,r_b = masked_image_blue.getpixel((x,y))

        r,g,b=resImage_PIL.getpixel((x,y)) # anh BGR
```

Tiến hành lấy giá trị pixel của 3 ảnh mask_image (ảnh có viền đen và có màu ở những chỗ được cho là nhiễu đen) và lấy các giá trị pixel ở ảnh gốc.

```
if(r_b>100):
            b_x=
linear_interpolation_channel_x(masked_image_blue,100,r_b,x,y,3.03)
            b_y =
linear_interpolation_channel_y(masked_image_blue,100,r_b,x,y,3.03)
            b=(b_x+b_y)/2
            b=(np.uint8(b))
            # b=255
        if(r_g>100):
            g_x =
linear_interpolation_channel_x(masked_image_green,100,r_g,x,y,2.05)
            g_y =
linear_interpolation_channel_y(masked_image_green,100,r_g,x,y,2.05)
            g=(g_x+g_y)/2
            g=(np.uint8(g))
            # g=255
        if(r_r>10):
            r_x = linear_interpolation_channel_x(masked_image_red,10,r_r,x,y,0.9)
            r_y = linear_interpolation_channel_y(masked_image_red,10,r_r,x,y,0.9)
            r=(r_x+r_y)*0.5
            r=(np.uint8(r))
            # r=255
```

Nếu r_g >100, xem như nhiễu, ta tiến hành nội suy tuyến tính và lấy giá trị trung bình của 2 giá trị đó và gán nó cho pixel g đã lấy trước đó của ảnh gốc.

Tương tự với các kênh còn lại.

Như vậy giá trị r,g,b đã được lọc nhiễu tại những vị trí có nhiễu đen.

```
result.putpixel((x,y),(b,g,r))
```

Tiến hành gán lại giá trị pixel cho ảnh đầu ra cuối cùng.

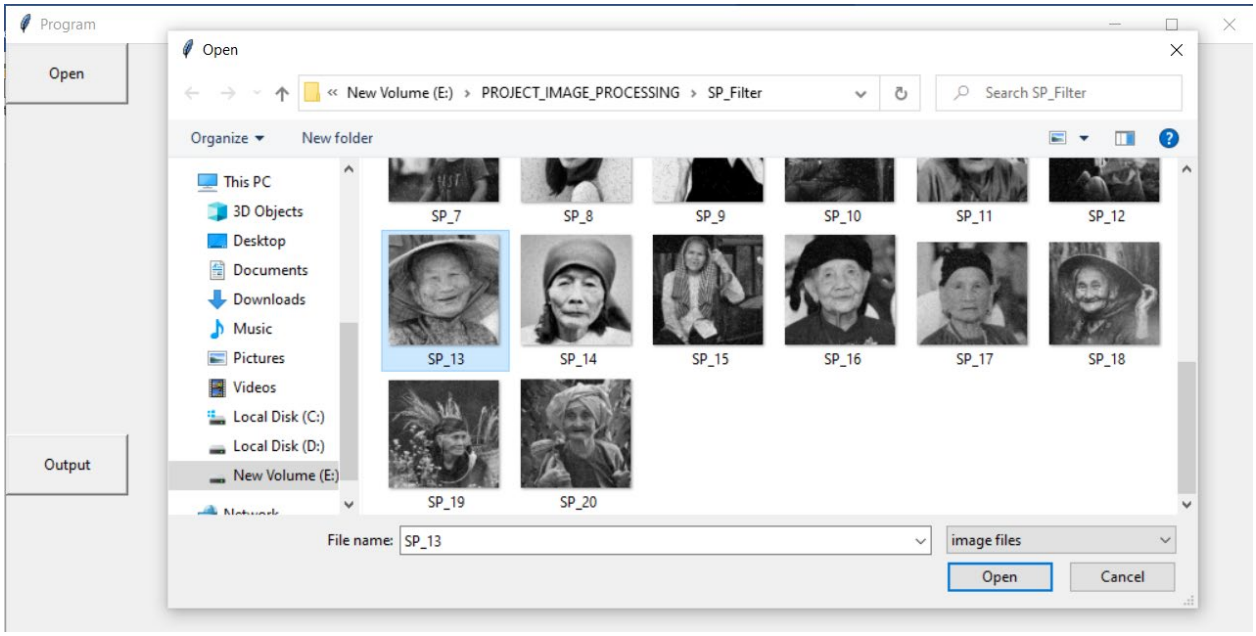
Nhược điểm của bộ lọc này là không thể quét hết được những chỗ bị nhiễu đen trên từng kênh màu, và tùy ảnh mà ta sẽ chỉnh sửa thông số của bộ chia và giá trị ngưỡng, và nó cũng không thể lọc hết được tất cả các nhiễu trong từng kênh, nó chỉ lọc được một số vị trí bị nhiễu quá đậm. Do nội suy tuyến tính chỉ từ 1 giá trị không nhiễu mà đối chiếu sang mà còn nhiều hạn chế trong việc xử lý pixel bị nhiễu.

(Những vùng màu cam nổi bật đã được xử lý nhưng những vùng nhiễu khác lại không thể xử lý được)

Chương 3. Chương trình hoạt động

Giao diện chương trình:

Chọn nút Open để chọn một ảnh bất kì:

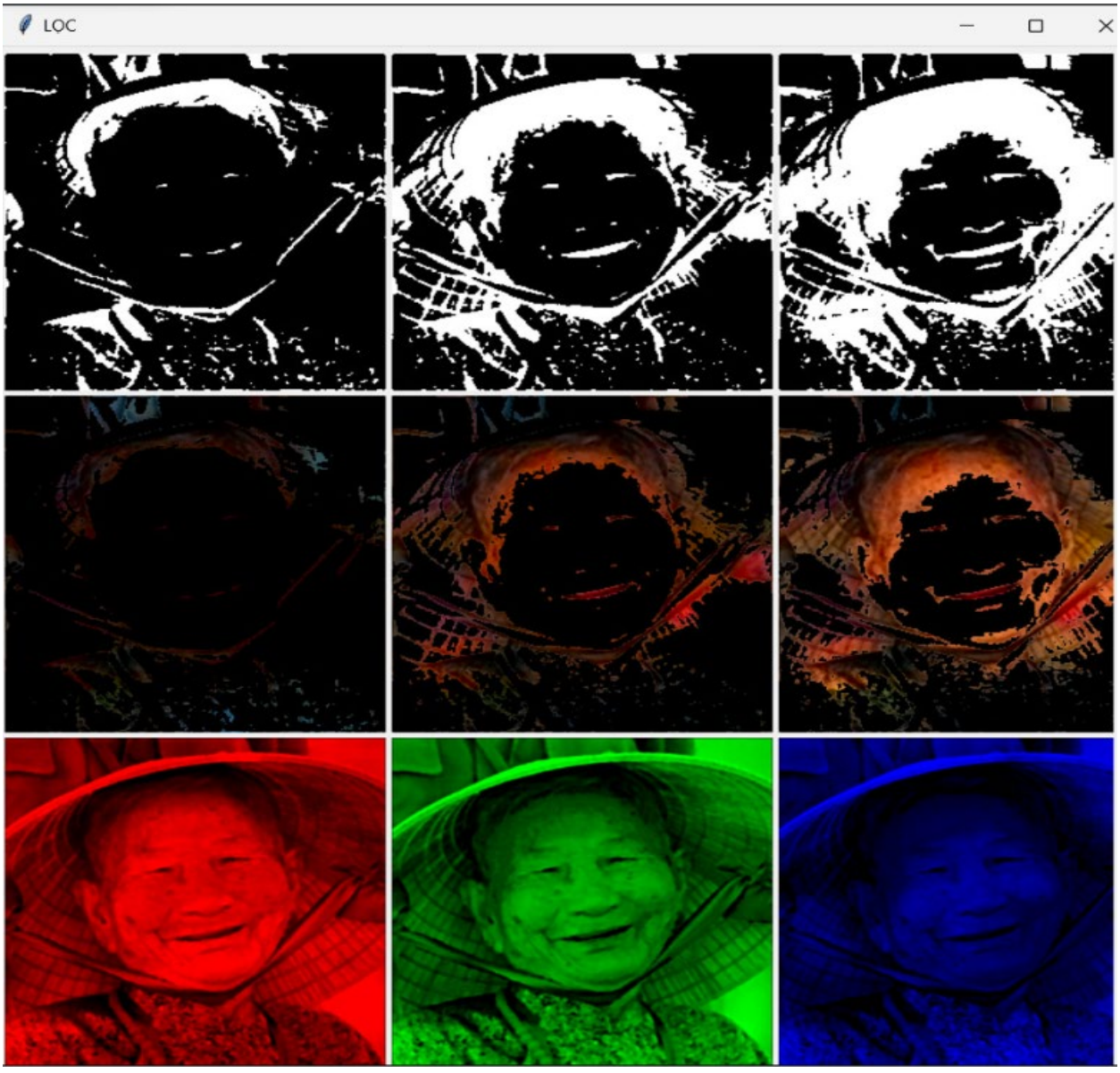


Sau khi lựa chọn hình ảnh:

Nhận thấy ảnh bị chấm trắng đen, ta chọn vào nút **“Lọc nhiễu”:**

Sau khi đã lọc nhiễu, cải thiện ảnh ta bắt đầu chuyển ảnh xám sang ảnh màu chọn vào nút Ouput để xử lý:

Để có thể xem được quá trình tiền xử lý ảnh, lọc nhiễu, khoanh vùng nhiễu ta có thể chọn vào nút **“LỌC”:**


LQC

Chương 4. NHẬN XÉT VÀ ĐÁNH GIÁ KẾT ĐỀ TÀI

4.1 Kết quả đạt được

- Xây dựng được một chương trình khôi phục ảnh xám và chuyển đổi sang ảnh màu bằng việc kết hợp xử lý ảnh và trí tuệ nhân tạo (AI), sử dụng các kiến thức môn học về xử lý ảnh và áp dụng các thuật toán nâng cao để xây dựng chương trình.
- Sau khi hoàn thành quá trình khôi phục ảnh xám, chương trình sử dụng trí tuệ nhân tạo để chuyển đổi ảnh từ xám sang màu. Qua việc học từ tập dữ liệu lớn chứa các cặp ảnh màu và ảnh xám tương ứng, chương trình có khả năng phân tích và đánh giá các mối quan hệ giữa màu sắc và thông tin trong ảnh xám. Điều này cho phép chương trình dự đoán màu sắc phù hợp cho các đối tượng và môi trường trong ảnh.
- Chương trình hoạt động khá ổn định, khôi phục ảnh xám và tạo ra ảnh màu chất lượng cao, với giao diện đơn giản dễ dàng sử dụng.

4.2 Hạn chế

Chương trình vẫn có một số hạn chế:

- Phụ thuộc vào tập dữ liệu huấn luyện: Chương trình cần một tập dữ liệu lớn để có thể huấn luyện mô hình học máy tốt hơn, nếu dữ liệu không đủ đa dạng thì sẽ gặp khó khăn trong việc khôi phục ảnh chính xác.
- Có những loại nhiễu, mất mát của tấm ảnh đầu vào quá phức tạp rất khó để cải thiện được chất lượng ảnh, mất rất nhiều thời gian để xử lý.

4.3 Kết luận

- Trong bài báo cáo này, chúng em đã tìm hiểu về quá trình khôi phục ảnh xám và chuyển đổi thành ảnh màu bằng cách kết hợp xử lý ảnh và trí tuệ nhân tạo (AI). Với những tấm ảnh đầu vào bị mất mát hư hại do thời gian gây ra, chúng em đã tìm hiểu những bộ lọc để cải thiện chất lượng hình ảnh. Tiếp theo, thu thập dữ liệu huấn luyện để xây dựng mô hình học máy có khả năng học và tái tạo thông tin màu sắc một cách chính xác, chuyển ảnh xám cũ thành ảnh màu sống động và rõ ràng.
- Phục hồi ảnh cũ bằng xử lý ảnh và trí tuệ nhân tạo không chỉ đơn thuần là một quá trình kỹ thuật, mà còn là một cách để chúng ta tôn vinh và tưởng nhớ những chiến sĩ và người mẹ Việt Nam Anh hùng cũng như những tấm hình ghi lại cuộc sống của ông cha ta trong quá khứ. Đồng thời, công nghệ này mang đến cho chúng ta cơ hội để gìn giữ và lan tỏa những giá trị quý giá từ quá khứ, nhằm đảm bảo rằng những khoảnh khắc kỷ niệm thời xa xưa không bao giờ bị lãng quên.

TÀI LIỆU THAM KHẢO

1. Hoạt động bên trong của cv2.rectangle () bằng NumPy (ichi.pro)
2. OpenCV - Grayscale, Binary Image, Adaptive Threshold | GMO- Z.com Vietnam Lab Center
3. Contour (phamduytung.com)
4. Rafael C. Gonzalez, Richard E. Woods - Digital Image Processing (2008, Prentice Hall)
5. Huấn luyên mô hình AI tự động tô màu cho hình ảnh xám (http://cs229.stanford.edu/proj2013/KabirzadehSousaBlaes-AutomaticColorizationOfGrayscaleImages.pdf)
6. Làm quen với Keras(https://viblo.asia/p/lam-quen-voi-keras-gGJ59mxJ5X2)
7. https://www.geeksforgeeks.org/place_forget-method-using- tkinter-in-python/
8. Hướng dẫn sử dụng thư viện PILLOW(viblo.com)
9. Lọc mịn ảnh, lọc nhiễu trong ảnh xử lý ảnh số (https://www.youtube.com/watch?v=2orKHgntBJ8&list=PLANUYCovX5nq28GO9el_fny-MSQ0Y_aaV)
10. Thử ứng dụng tô màu bằng deep learning (https://viblo.asia/p/thu-lam-ung-dung-to-mau-anh-voi-mang-deep-learning-1Je5EJQjKnL)