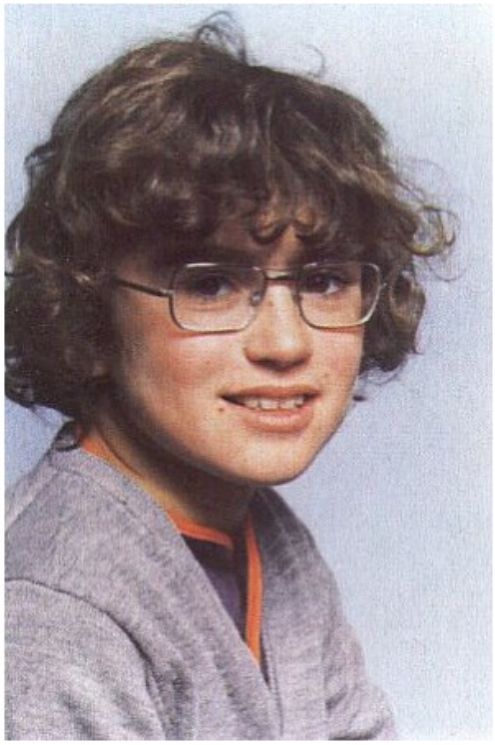
```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
import os

# Step 1: List of input image paths
image_paths = ['a.png','c.png','b.png','d.png','e.png'  # Add paths to
multiple images here
]

# Step 2: Directory to save enhanced images
output_dir = '/mnt/data/enhanced_images'
os.makedirs(output_dir, exist_ok=True)

# Step 3: Adjust brightness using gamma correction (natural
adjustment)
def gamma_correction(img, gamma=1.1):  # Lower gamma for subtle
correction
    inv_gamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** inv_gamma) * 255 for i in
range(256)]).astype("uint8")
    return cv2.LUT(img, table)

# Process each image
for i, image_path in enumerate(image_paths):
    # Read the image
    image = cv2.imread(image_path)

    # Check if the image is loaded properly
    if image is None:
        print(f"Error: Could not load image at {image_path}")
        continue

    # Step 4: Apply gamma correction
    gamma_corrected = gamma_correction(image, gamma=1.1)

    # Step 5: Enhance colors using a soft boost in saturation
    hsv = cv2.cvtColor(gamma_corrected, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)
    s = cv2.add(s, 20)  # Moderate saturation boost for natural tones
    enhanced_hsv = cv2.merge((h, s, v))
    color_enhanced = cv2.cvtColor(enhanced_hsv, cv2.COLOR_HSV2BGR)

    # Step 6: Skin smoothing (minimal smoothing for natural look)
    hsv_skin = cv2.cvtColor(color_enhanced, cv2.COLOR_BGR2HSV)
    lower_skin = np.array([0, 30, 60])
    upper_skin = np.array([20, 150, 255])
    skin_mask = cv2.inRange(hsv_skin, lower_skin, upper_skin)
    smoothed = color_enhanced.copy()
    smoothed_skin = cv2.bilateralFilter(smoothed, d=7, sigmaColor=20,
```

```
sigmaSpace=20)
    smoothed[skin_mask > 0] = smoothed_skin[skin_mask > 0]

    # Step 7: Color grading for DSLR-like effect
    lab = cv2.cvtColor(smoothed, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    l = clahe.apply(l)
    lab = cv2.merge((l, a, b))
    graded = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)

    # Step 8: Subtle sharpening
    kernel = np.array([[0, -0.5, 0],
                       [-0.5, 3, -0.5],
                       [0, -0.5, 0]])
    sharpened_image = cv2.filter2D(graded, -1, kernel)

    # Step 9: Noise reduction
    denoised = cv2.fastNlMeansDenoisingColored(sharpened_image, None,
10, 10, 7, 21)

    # Save the enhanced image
    enhanced_image_path = os.path.join(output_dir, f"enhanced_image_{i
+ 1}.png")
    cv2.imwrite(enhanced_image_path, denoised)

    # Display the original and enhanced images side by side
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title(f"Original Image {i + 1}")
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.title(f"Enhanced Image {i + 1}")
    plt.imshow(cv2.cvtColor(denoised, cv2.COLOR_BGR2RGB))
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    # Log the output
    print(f"Enhanced image saved at: {enhanced_image_path}")
```

Original Image 1



Enhanced Image 1

```
Enhanced image saved at:
/mnt/data/enhanced_images/enhanced_image_1.png
```

Original Image 2

Enhanced Image 2

```
Enhanced image saved at:
/mnt/data/enhanced_images/enhanced_image_2.png
```

Original Image 3

Enhanced Image 3

```
Enhanced image saved at:
/mnt/data/enhanced_images/enhanced_image_3.png
```

Original Image 4

Enhanced Image 4


```
Enhanced image saved at:
/mnt/data/enhanced_images/enhanced_image_4.png
```

Original Image 5

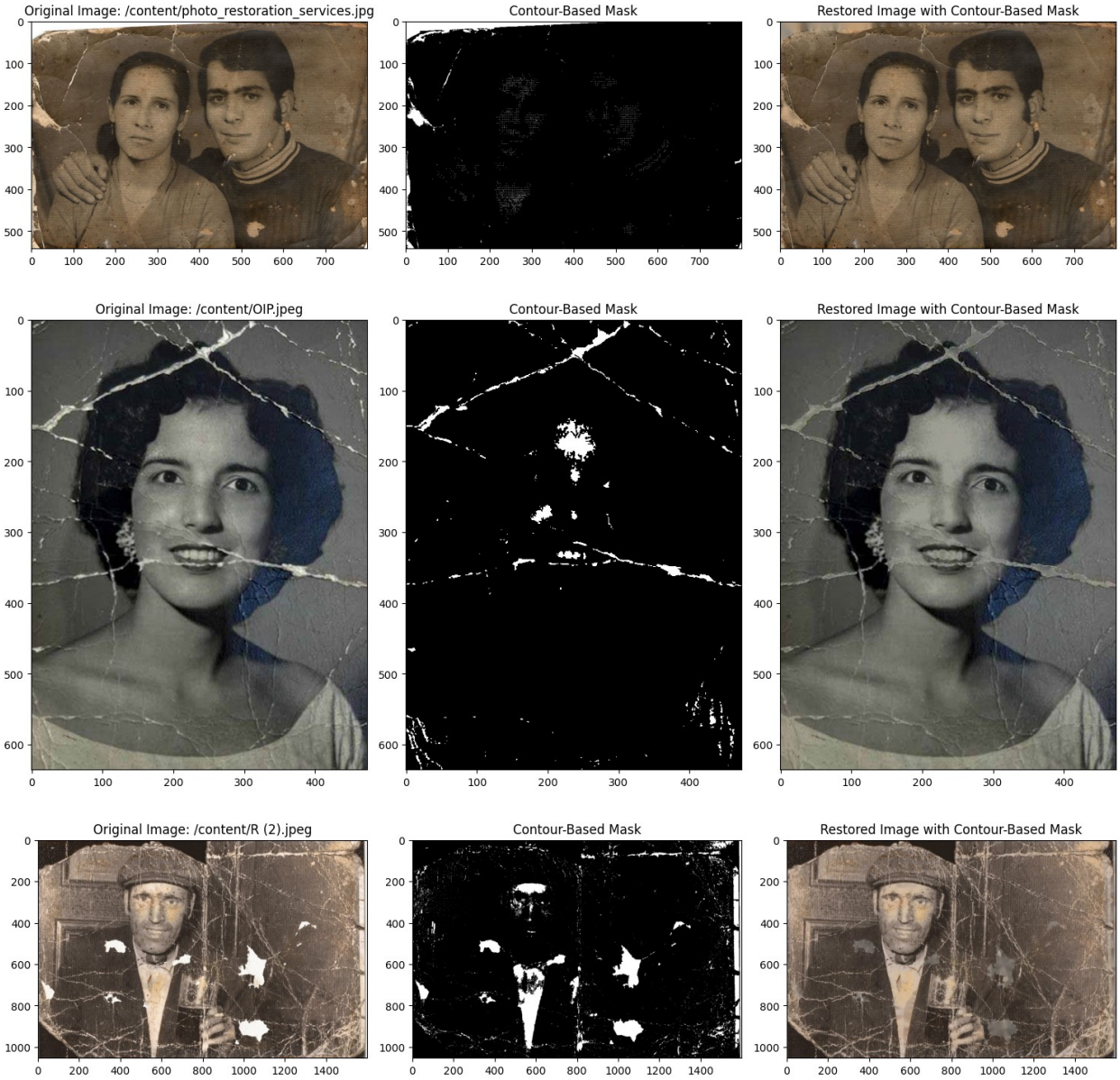
Enhanced Image 5

```
Enhanced image saved at:
/mnt/data/enhanced_images/enhanced_image_5.png
```

shivangi - CONTOUR MASK

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# List of image file names and their corresponding threshold values
images = ['/content/photo_restoration_services.jpg',
'/content/OIP.jpeg', '/content/R (2).jpeg']
thresholds = [180, 160, 200]

# Loop through each image and process
for img_file, threshold in zip(images, thresholds):
    # Read the original image
    original_image = cv2.imread(img_file)

    # Read the grayscale version
    marked = cv2.imread(img_file, 0)

    # Apply thresholding
    ret, thresh1 = cv2.threshold(marked, threshold, 255,
cv2.THRESH_BINARY)

    # Find contours of the thresholded image
    contours, _ = cv2.findContours(thresh1, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Create a new mask using contours
    mask_contour = np.zeros_like(marked)
    cv2.drawContours(mask_contour, contours, -1, (255),
thickness=cv2.FILLED)

    # Apply inpainting with the new mask
    restore_contour = cv2.inpaint(original_image, mask_contour, 7,
cv2.INPAINT_TELEA)

    # Convert images to RGB for Matplotlib
    original_image_rgb = cv2.cvtColor(original_image,
cv2.COLOR_BGR2RGB)
    restore_contour_rgb = cv2.cvtColor(restore_contour,
cv2.COLOR_BGR2RGB)

    # Display the results
    plt.figure(figsize=(15, 10))
```

```
    plt.subplot(1, 3, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title(f"Original Image: {img_file}")

    plt.subplot(1, 3, 2)
    plt.imshow(mask_contour, cmap='gray')
    plt.title("Contour-Based Mask")

    plt.subplot(1, 3, 3)
    plt.imshow(restore_contour_rgb)
    plt.title("Restored Image with Contour-Based Mask")

    plt.tight_layout()
    plt.show()
```



This code is designed for **restoring images with scratches or other unwanted artifacts** by leveraging thresholding and contour-based inpainting. Below is an explanation of the key steps:

---

1. Importing Required Libraries

- **`cv2`**: OpenCV for image processing operations like reading images, thresholding, and inpainting.
- **`matplotlib.pyplot`**: For visualizing images as subplots.
- **`numpy`**: For numerical operations and creating masks.

---

2. Input Images and Thresholds

```
images = ['/content/photo_restoration_services.jpg',
'/content/OIP.jpeg', '/content/R (2).jpeg']
thresholds = [180, 160, 200]
```

- **`images`**: A list containing the file paths of the images to process.
- **`thresholds`**: A list of threshold values corresponding to each image. These values are used for creating binary masks to identify regions requiring restoration.

---

3. Loop Through Images

The code iterates through the list of images and their respective thresholds to process them individually.

---

4. Read Images

```
original_image = cv2.imread(img_file)
marked = cv2.imread(img_file, 0)
```

- **`original_image`**: Reads the color image in BGR format.
- **`marked`**: Reads a grayscale version of the image, simplifying the processing for thresholding.

---

5. Thresholding

```
ret, thresh1 = cv2.threshold(marked, threshold, 255,
cv2.THRESH_BINARY)
```

- **Purpose**: Identifies bright areas (e.g., scratches) by setting pixel values above the threshold to `255` (white) and below to `0` (black).
- **`threshold`**: Specific to each image, determines the sensitivity of detecting scratches or unwanted marks.

- **Output**: A binary image (`thresh1`) highlighting the areas to be restored.

---

6. Contour Detection

```
contours, _ = cv2.findContours(thresh1, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

- **`cv2.findContours`**: Finds the boundaries of the white regions in the binary mask (`thresh1`).
- **`cv2.RETR_EXTERNAL`**: Retrieves only the outermost contours, ignoring nested ones.
- **`cv2.CHAIN_APPROX_SIMPLE`**: Compresses the contour points to save memory.

---

7. Create Mask Using Contours

```
mask_contour = np.zeros_like(marked)
cv2.drawContours(mask_contour, contours, -1, (255),
thickness=cv2.FILLED)
```

- **`np.zeros_like`**: Creates a black mask of the same size as the grayscale image.
- **`cv2.drawContours`**: Draws the detected contours as filled white regions (`255`) on the mask.

---

8. Inpainting

```
restore_contour = cv2.inpaint(original_image, mask_contour, 7,
cv2.INPAINT_TELEA)
```

- **Purpose**: Fills the white regions in the mask with content from the surrounding pixels.
- **`inpaintRadius=7`**: Determines the pixel neighborhood size for inpainting.
- **`cv2.INPAINT_TELEA`**: Uses a fast-marching method for inpainting, creating a natural restoration.

---

9. Convert Images to RGB

```
original_image_rgb = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
restore_contour_rgb = cv2.cvtColor(restore_contour, cv2.COLOR_BGR2RGB)
```

- OpenCV uses BGR format by default, but Matplotlib expects RGB. The images are converted for proper visualization.

---

10. Visualization

```
plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
plt.title(f"Original Image: {img_file}")

plt.subplot(1, 3, 2)
plt.imshow(mask_contour, cmap='gray')
plt.title("Contour-Based Mask")

plt.subplot(1, 3, 3)
plt.imshow(restore_contour_rgb)
plt.title("Restored Image with Contour-Based Mask")
```

- **Subplot 1**: Displays the original image.
- **Subplot 2**: Displays the mask highlighting regions to restore.
- **Subplot 3**: Displays the restored image after inpainting.
- **`plt.tight_layout()`**: Ensures proper spacing between subplots.
- **`plt.show()`**: Renders the plots.

Summary of Workflow

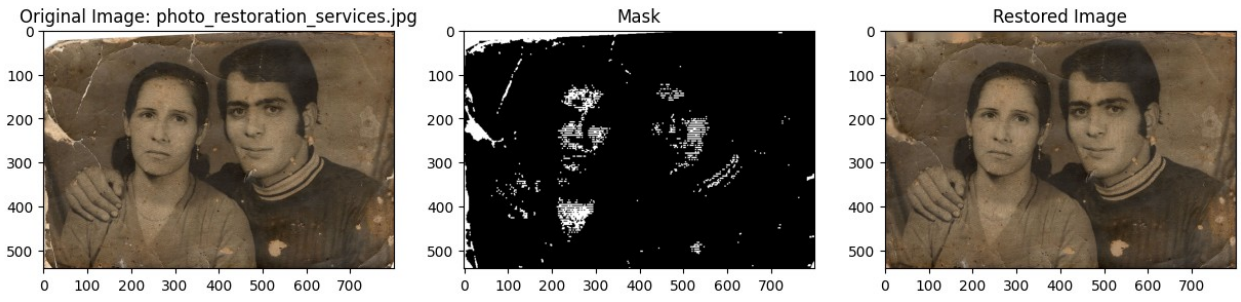
1. **Thresholding**: Identifies regions requiring restoration (e.g., scratches or marks).
2. **Contours**: Extracts the boundaries of the identified regions.
3. **Mask Creation**: Generates a binary mask based on the contours.
4. **Inpainting**: Restores the marked regions using pixel data from surrounding areas.
5. **Visualization**: Displays the original image, mask, and restored image for comparison.

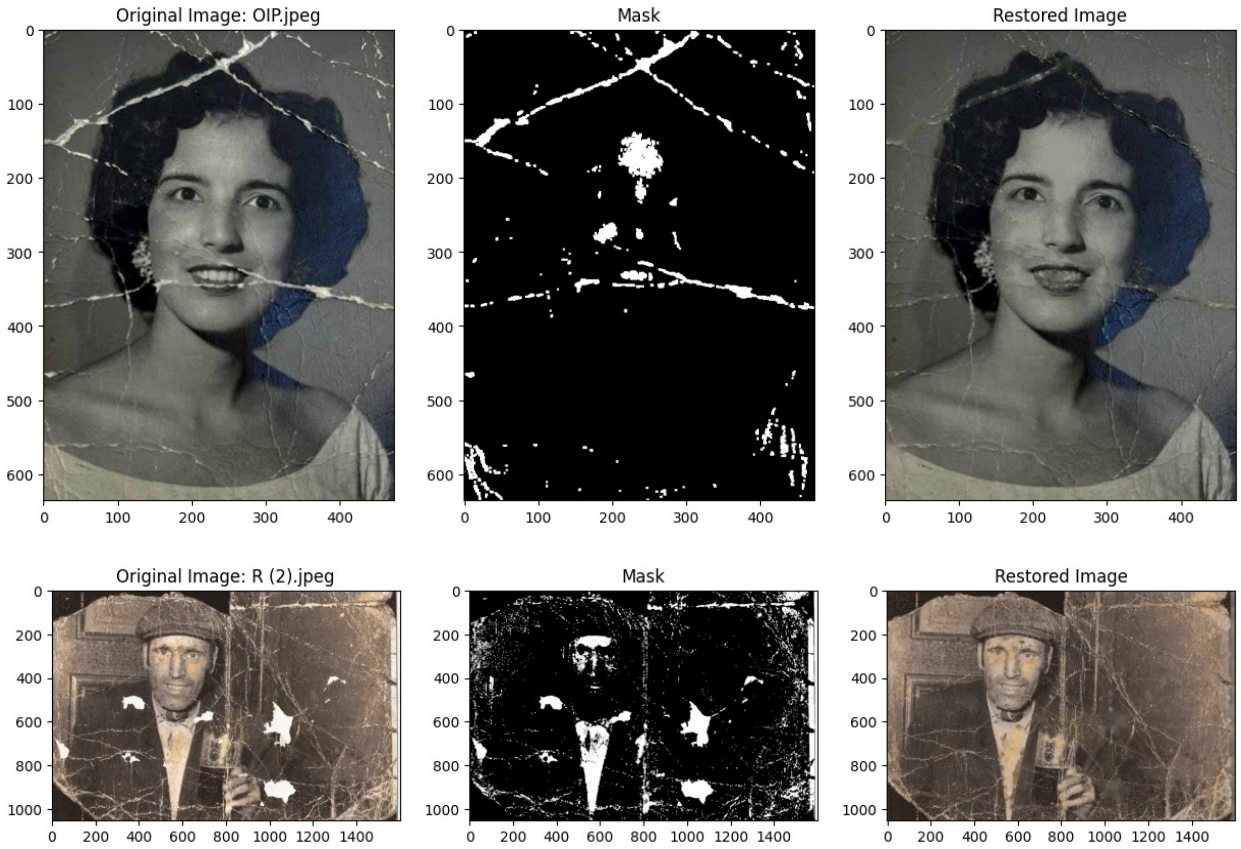
This approach is particularly effective for images with noticeable scratches or bright artifacts and offers a clear before-and-after comparison.

yashi - THRESHOLD MASK

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# List of image file names and their corresponding threshold values
images = ['photo_restoration_services.jpg', 'OIP.jpeg', 'R (2).jpeg']
thresholds = [180, 160, 200]

# Loop through each image and process
for img_file, threshold in zip(images, thresholds):
    # Read the original image
    original_image = cv2.imread(img_file)

    # Read the grayscale version
```

```
    marked = cv2.imread(img_file, 0)

    # Apply thresholding
    ret, thresh1 = cv2.threshold(marked, threshold, 255,
cv2.THRESH_BINARY)

    # Create a kernel and apply dilation
    kernel = np.ones((3, 3), np.uint8)
    mask = cv2.dilate(thresh1, kernel, iterations=1)


    # Apply inpainting
    restore = cv2.inpaint(original_image, mask, 7, cv2.INPAINT_TELEA)

    # Convert images to RGB for Matplotlib
    original_image_rgb = cv2.cvtColor(original_image,
cv2.COLOR_BGR2RGB)
    restore_rgb = cv2.cvtColor(restore, cv2.COLOR_BGR2RGB)

    # Plot the original and restored images side by side
    figure = plt.figure(figsize=(15, 15))
    plt.subplot(1, 3, 1)
    plt.imshow(original_image_rgb)
    plt.title(f"Original Image: {img_file}")

    plt.subplot(1, 3, 2)
    plt.imshow(mask, cmap='gray')
    plt.title("Mask")

    plt.subplot(1, 3, 3)
    plt.imshow(restore_rgb)
    plt.title("Restored Image")

    plt.show()
```




Original Image: OIP.jpeg
Mask
Restored Image
Original Image: R (2).jpeg
Mask
Restored Image

Bringing Old Photos Back to Life - Modelling

```
!git clone https://github.com/microsoft/Bringing-Old-Photos-Back-to-Life.git photo_restoration
```

```
Cloning into 'photo_restoration'...
remote: Enumerating objects: 183, done.ote: Counting objects: 100% (183/183), done.ote: Compressing objects: 100% (166/166), done.ote: Total 183 (delta 38), reused 136 (delta 13), pack-reused 0
```

Setting up the environment

```
# pulling the syncBN repo
%cd photo_restoration/Face_Enhancement/models/networks
!git clone https://github.com/vacancy/Synchronized-BatchNorm-PyTorch
!cp -rf Synchronized-BatchNorm-PyTorch/sync_batchnorm .
%cd ../../../

%cd Global/detection_models
!git clone https://github.com/vacancy/Synchronized-BatchNorm-PyTorch
!cp -rf Synchronized-BatchNorm-PyTorch/sync_batchnorm .
%cd ../../

# downloading the landmark detection model
%cd Face_Detection/
!wget http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2
!bzip2 -d shape_predictor_68_face_landmarks.dat.bz2
%cd ../

# downloading the pretrained model
%cd Face_Enhancement/
!wget https://facevc.blob.core.windows.net/zhanbo/old_photo/pretrain/Face_Enhancement/checkpoints.zip
!unzip checkpoints.zip
%cd ../

%cd Global/
!wget https://facevc.blob.core.windows.net/zhanbo/old_photo/pretrain/Global/checkpoints.zip
!unzip checkpoints.zip
%cd ../
```

```
/content/photo_restoration/Face_Enhancement/models/networks
Cloning into 'Synchronized-BatchNorm-PyTorch'...
```

```
remote: Enumerating objects: 16, done.ote: Counting objects: 100%
(16/16), done.ote: Compressing objects: 100% (12/12), done.ote: Total
177 (delta 8), reused 9 (delta 4), pack-reused 161odels
Cloning into 'Synchronized-BatchNorm-PyTorch'...
remote: Enumerating objects: 16, done.ote: Counting objects: 100%
(16/16), done.ote: Compressing objects: 100% (12/12), done.ote: Total
177 (delta 8), reused 9 (delta 4), pack-reused 161arks.dat.bz2
Resolving dlib.net (dlib.net)... 107.180.26.78
Connecting to dlib.net (dlib.net)|107.180.26.78|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 64040097 (61M)
Saving to: ‘shape_predictor_68_face_landmarks.dat.bz2’

shape_predictor_68_ 100%[===================>]  61.07M  8.21MB/s    in
46s

2020-10-07 04:24:16 (1.33 MB/s) -
‘shape_predictor_68_face_landmarks.dat.bz2’ saved [64040097/64040097]

/content/photo_restoration
/content/photo_restoration/Face_Enhancement
--2020-10-07 04:24:23--
https://facevc.blob.core.windows.net/zhanbo/old_photo/pretrain/Face_En
hancement/checkpoints.zip
Resolving facevc.blob.core.windows.net
(facevc.blob.core.windows.net)... 52.239.237.4
Connecting to facevc.blob.core.windows.net
(facevc.blob.core.windows.net)|52.239.237.4|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 342496657 (327M) [application/x-zip-compressed]
Saving to: ‘checkpoints.zip’

checkpoints.zip     100%[===================>] 326.63M  13.3MB/s    in
27s

2020-10-07 04:24:51 (12.1 MB/s) - ‘checkpoints.zip’ saved
[342496657/342496657]

Archive:  checkpoints.zip
   creating: checkpoints/
   creating: checkpoints/Setting_9_epoch_100/
  inflating: checkpoints/Setting_9_epoch_100/latest_net_G.pth
/content/photo_restoration
/content/photo_restoration/Global
--2020-10-07 04:24:55--
https://facevc.blob.core.windows.net/zhanbo/old_photo/pretrain/Global/
checkpoints.zip
Resolving facevc.blob.core.windows.net
(facevc.blob.core.windows.net)... 52.239.237.4
Connecting to facevc.blob.core.windows.net
```

```
(facevc.blob.core.windows.net)|52.239.237.4|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1739076350 (1.6G) [application/x-zip-compressed]
Saving to: ‘checkpoints.zip’

checkpoints.zip     100%[===================>]   1.62G  13.5MB/s    in
2m 6s

2020-10-07 04:27:02 (13.1 MB/s) - ‘checkpoints.zip’ saved
[1739076350/1739076350]

Archive:  checkpoints.zip
   creating: checkpoints/
   creating: checkpoints/detection/
  inflating: checkpoints/detection/FT_Epoch_latest.pt
   creating: checkpoints/restoration/
   creating: checkpoints/restoration/mapping_quality/
  inflating: checkpoints/restoration/mapping_quality/latest_net_D.pth

  inflating:
checkpoints/restoration/mapping_quality/latest_net_mapping_net.pth
  inflating:
checkpoints/restoration/mapping_quality/latest_optimizer_D.pth
  inflating:
checkpoints/restoration/mapping_quality/latest_optimizer_mapping_net.p
th
   creating: checkpoints/restoration/mapping_scratch/
 extracting: checkpoints/restoration/mapping_scratch/iter.txt
  inflating: checkpoints/restoration/mapping_scratch/latest_net_D.pth

  inflating:
checkpoints/restoration/mapping_scratch/latest_net_mapping_net.pth
  inflating:
checkpoints/restoration/mapping_scratch/latest_optimizer_D.pth
  inflating:
checkpoints/restoration/mapping_scratch/latest_optimizer_mapping_net.p
th
  inflating: checkpoints/restoration/mapping_scratch/loss_log.txt
  inflating: checkpoints/restoration/mapping_scratch/model.txt
   creating: checkpoints/restoration/VAE_A_quality/
  inflating: checkpoints/restoration/VAE_A_quality/latest_net_D.pth
  inflating:
checkpoints/restoration/VAE_A_quality/latest_net_featD.pth
  inflating: checkpoints/restoration/VAE_A_quality/latest_net_G.pth
  inflating:
checkpoints/restoration/VAE_A_quality/latest_optimizer_D.pth
  inflating:
checkpoints/restoration/VAE_A_quality/latest_optimizer_featD.pth
  inflating:
checkpoints/restoration/VAE_A_quality/latest_optimizer_G.pth
```

```
   creating: checkpoints/restoration/VAE_B_quality/
  inflating: checkpoints/restoration/VAE_B_quality/latest_net_D.pth
  inflating: checkpoints/restoration/VAE_B_quality/latest_net_G.pth
  inflating:
checkpoints/restoration/VAE_B_quality/latest_optimizer_D.pth
  inflating:
checkpoints/restoration/VAE_B_quality/latest_optimizer_G.pth
   creating: checkpoints/restoration/VAE_B_scratch/
  inflating: checkpoints/restoration/VAE_B_scratch/latest_net_D.pth
  inflating: checkpoints/restoration/VAE_B_scratch/latest_net_G.pth
  inflating:
checkpoints/restoration/VAE_B_scratch/latest_optimizer_D.pth
  inflating:
checkpoints/restoration/VAE_B_scratch/latest_optimizer_G.pth
/content/photo_restoration
```

```
! pip install -r requirements.txt
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-
packages (from -r requirements.txt (line 1)) (1.6.0+cu101)
Requirement already satisfied: dlib in /usr/local/lib/python3.6/dist-
packages (from -r requirements.txt (line 2)) (19.18.0)
Requirement already satisfied: scikit-image in
/usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line
3)) (0.16.2)
Requirement already satisfied: easydict in
/usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line
4)) (1.9)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line
5)) (3.13)
Collecting dominate>=2.3.1
  Downloading
https://files.pythonhosted.org/packages/c0/03/1ba70425be63f2aab42fbc98
894fe5d90cdadd41f79bdc778b3e404cfd8f/dominate-2.5.2-py2.py3-none-
any.whl
Requirement already satisfied: dill in /usr/local/lib/python3.6/dist-
packages (from -r requirements.txt (line 7)) (0.3.2)
Collecting tensorboardX
ent already satisfied: scipy in /usr/local/lib/python3.6/dist-packages
(from -r requirements.txt (line 9)) (1.4.1)
Requirement already satisfied: opencv-python in
/usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line
10)) (4.1.2.30)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-
packages (from torch->-r requirements.txt (line 1)) (1.18.5)
Requirement already satisfied: future in
/usr/local/lib/python3.6/dist-packages (from torch->-r
requirements.txt (line 1)) (0.16.0)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.0 in
```

```
/usr/local/lib/python3.6/dist-packages (from scikit-image->-r
requirements.txt (line 3)) (3.2.2)
Requirement already satisfied: imageio>=2.3.0 in
/usr/local/lib/python3.6/dist-packages (from scikit-image->-r
requirements.txt (line 3)) (2.4.1)
Requirement already satisfied: networkx>=2.0 in
/usr/local/lib/python3.6/dist-packages (from scikit-image->-r
requirements.txt (line 3)) (2.5)
Requirement already satisfied: pillow>=4.3.0 in
/usr/local/lib/python3.6/dist-packages (from scikit-image->-r
requirements.txt (line 3)) (7.0.0)
Requirement already satisfied: PyWavelets>=0.4.0 in
/usr/local/lib/python3.6/dist-packages (from scikit-image->-r
requirements.txt (line 3)) (1.1.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-
packages (from tensorboardX->-r requirements.txt (line 8)) (1.15.0)
Requirement already satisfied: protobuf>=3.8.0 in
/usr/local/lib/python3.6/dist-packages (from tensorboardX->-r
requirements.txt (line 8)) (3.12.4)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib!
=3.0.0,>=2.0.0->scikit-image->-r requirements.txt (line 3)) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib!
=3.0.0,>=2.0.0->scikit-image->-r requirements.txt (line 3)) (1.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!
=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from
matplotlib!=3.0.0,>=2.0.0->scikit-image->-r requirements.txt (line 3))
(2.4.7)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.6/dist-packages (from matplotlib!
=3.0.0,>=2.0.0->scikit-image->-r requirements.txt (line 3)) (0.10.0)
Requirement already satisfied: decorator>=4.3.0 in
/usr/local/lib/python3.6/dist-packages (from networkx>=2.0->scikit-
image->-r requirements.txt (line 3)) (4.4.2)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.6/dist-packages (from protobuf>=3.8.0-
>tensorboardX->-r requirements.txt (line 8)) (50.3.0)
Installing collected packages: dominate, tensorboardX
Successfully installed dominate-2.5.2 tensorboardX-2.1
```

#◢ Run the code

Restore photos (normal mode)

```
%cd /content/photo_restoration/
input_folder = "test_images/old"
output_folder = "output"

import os
```

```
basepath = os.getcwd()
input_path = os.path.join(basepath, input_folder)
output_path = os.path.join(basepath, output_folder)
os.mkdir(output_path)

!python run.py --input_folder
/content/photo_restoration/test_images/old --output_folder
/content/photo_restoration/output/ --GPU 0
```

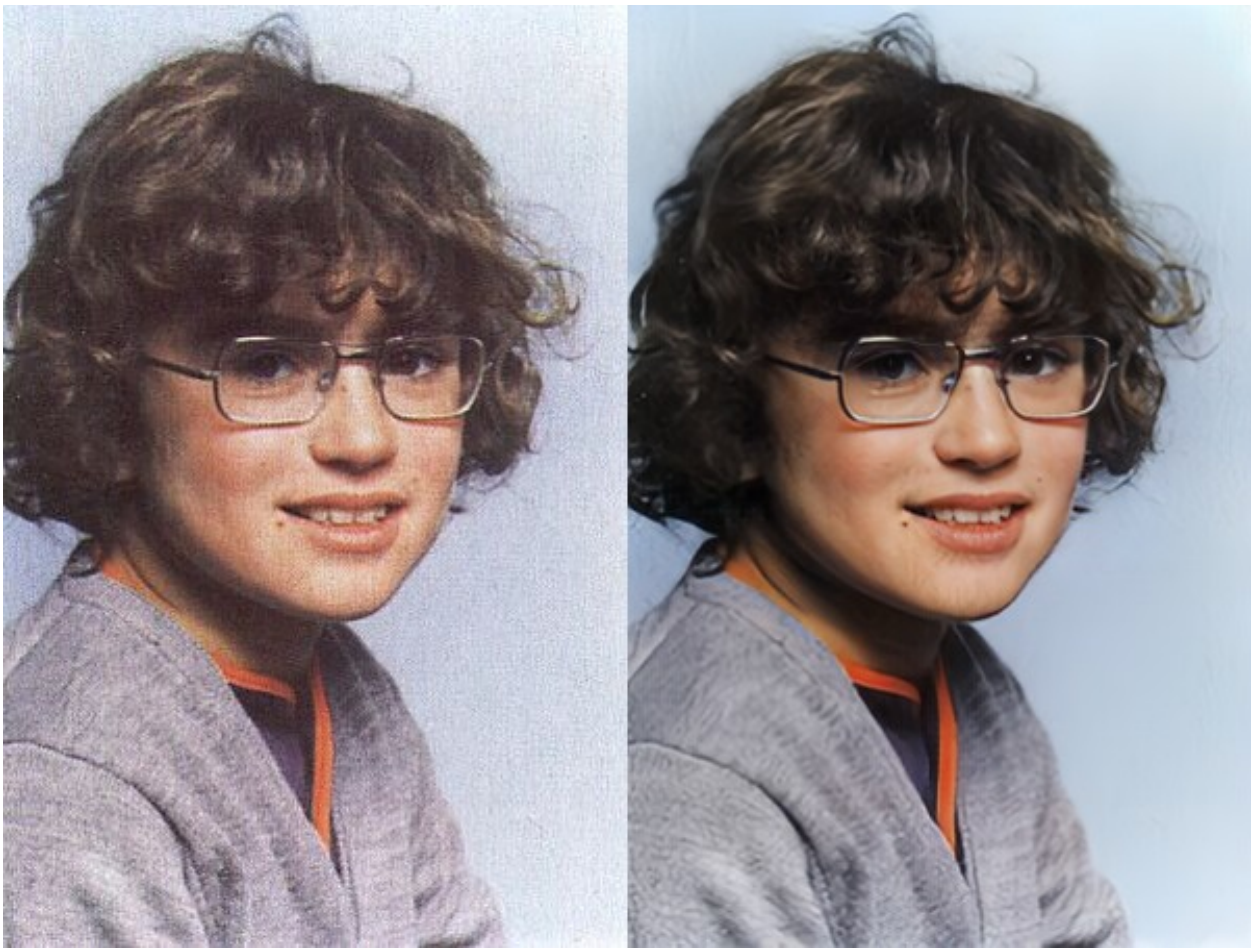
```
/content/photo_restoration
Running Stage 1: Overall restoration
Now you are processing a.png
Now you are processing b.png
Now you are processing c.png
Now you are processing d.png
Now you are processing e.png
Now you are processing f.png
Now you are processing g.png
Now you are processing h.png
Finish Stage 1 ...


Running Stage 2: Face Detection
1
1
1
Warning: There is no face in f.png
Warning: There is no face in d.png
Warning: There is no face in e.png
Warning: There is no face in b.png
1
Finish Stage 2 ...


Running Stage 3: Face Enhancement
The main GPU is
0
dataset [FaceTestDataset] of size 4 was created
The size of the latent vector size is [8,8]
Network [SPADEGenerator] was created. Total number of parameters: 92.1
million. To see the architecture, do print(network).
hi :)
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:3121:
UserWarning: Default upsampling behavior when mode=bilinear is changed
to align_corners=False since 0.4.0. Please specify align_corners=True
if the old behavior is desired. See the documentation of nn.Upsample
for details.
  "See the documentation of nn.Upsample for details.".format(mode))
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:1614:
UserWarning: nn.functional.tanh is deprecated. Use torch.tanh instead.
```

```
  warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh
instead.")
Finish Stage 3 ...


Running Stage 4: Blending
Warning: There is no face in f.png
Warning: There is no face in d.png
Warning: There is no face in e.png
Warning: There is no face in b.png
Finish Stage 4 ...


All the processing is done. Please check the results.
```

```
import io
import IPython.display
import numpy as np
import PIL.Image

def imshow(a, format='png', jpeg_fallback=True):
    a = np.asarray(a, dtype=np.uint8)
    data = io.BytesIO()
    PIL.Image.fromarray(a).save(data, format)
    im_data = data.getvalue()
    try:
      disp = IPython.display.display(IPython.display.Image(im_data))
    except IOError:
      if jpeg_fallback and format != 'jpeg':
        print(('Warning: image was too large to display in format
"{}"; '
               'trying jpeg instead.').format(format))
        return imshow(a, format='jpeg')
      else:
        raise
    return disp

def make_grid(I1, I2, resize=True):
    I1 = np.asarray(I1)
    H, W = I1.shape[0], I1.shape[1]

    if I1.ndim >= 3:
        I2 = np.asarray(I2.resize((W,H)))
        I_combine = np.zeros((H,W*2,3))
        I_combine[:,:W,:] = I1[:,:,:3]
        I_combine[:,W:,:] = I2[:,:,:3]
    else:
        I2 = np.asarray(I2.resize((W,H)).convert('L'))
        I_combine = np.zeros((H,W*2))
        I_combine[:,:W] = I1[:,:]
```

```
        I_combine[:,W:] = I2[:,:]
    I_combine = PIL.Image.fromarray(np.uint8(I_combine))

    W_base = 600
    if resize:
      ratio = W_base / (W*2)
      H_new = int(H * ratio)
      I_combine = I_combine.resize((W_base, H_new), PIL.Image.LANCZOS)

    return I_combine

filenames = os.listdir(os.path.join(input_path))
filenames.sort()

for filename in filenames:
    print(filename)
    image_original = PIL.Image.open(os.path.join(input_path,
filename))
    image_restore = PIL.Image.open(os.path.join(output_path,
'final_output', filename))

    display(make_grid(image_original, image_restore))
```

a.png



```
b.png
```

```
c.png
```

```
d.png
```


```
e.png
```

```
f.png
```

```
g.png
```

```
h.png
```

Restore the photos with scratches

```
!rm -rf /content/photo_restoration/output/*
!python run.py --input_folder
/content/photo_restoration/test_images/old_w_scratch/ --output_folder
/content/photo_restoration/output/ --GPU 0 --with_scratch

Running Stage 1: Overall restoration
initializing the dataloader
model weights loaded
directory of testing image:
/content/photo_restoration/test_images/old_w_scratch/
processing a.png
processing b.png
processing c.png
processing d.png
You are using NL + Res
Now you are processing a.png
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:3121:
UserWarning: Default upsampling behavior when mode=bilinear is changed
to align_corners=False since 0.4.0. Please specify align_corners=True
```

```
if the old behavior is desired. See the documentation of nn.Upsample
for details.
  "See the documentation of nn.Upsample for details.".format(mode))
Now you are processing b.png
Now you are processing c.png
Now you are processing d.png
Finish Stage 1 ...


Running Stage 2: Face Detection
1
1
2
1
Finish Stage 2 ...


Running Stage 3: Face Enhancement
The main GPU is
0
dataset [FaceTestDataset] of size 5 was created
The size of the latent vector size is [8,8]
Network [SPADEGenerator] was created. Total number of parameters: 92.1
million. To see the architecture, do print(network).
hi :)
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:3121:
UserWarning: Default upsampling behavior when mode=bilinear is changed
to align_corners=False since 0.4.0. Please specify align_corners=True
if the old behavior is desired. See the documentation of nn.Upsample
for details.
  "See the documentation of nn.Upsample for details.".format(mode))
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:1614:
UserWarning: nn.functional.tanh is deprecated. Use torch.tanh instead.
  warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh
instead.")
Finish Stage 3 ...


Running Stage 4: Blending
Finish Stage 4 ...


All the processing is done. Please check the results.
```

```
input_folder = "test_images/old_w_scratch"
output_folder = "output"
input_path = os.path.join(basepath, input_folder)
output_path = os.path.join(basepath, output_folder)

filenames = os.listdir(os.path.join(input_path))
```

```
filenames.sort()

for filename in filenames:
    print(filename)
    image_original = PIL.Image.open(os.path.join(input_path,
filename))
    image_restore = PIL.Image.open(os.path.join(output_path,
'final_output', filename))

    display(make_grid(image_original, image_restore))
```

a.png

b.png

```
c.png
```

```
d.png
```