

21C16

by C 16

Submission date: 21-Jul-2021 10:20PM (UTC+0530)

File name: C16.docx
(3.83M)

Word count: 5686

Character count: 25190

A Project report on

**RESTORATION OF DEGRADED IMAGES
USING VAE AND GAN**

in partial fulfillment for the award of the degree of

**BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING**

Submitted by

ASHUTOSH MISHRA	(17B91A05P4)
AAYUSH KARNA	(17B91A05P2)
RAVI SHANKER KUMAR YADAV	(17B91A05P8)

Under the Guidance of

Sri BNV NARASIMHA RAJU
Assistant Professor



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRKR ENGINEERING COLLEGE (A)

Chinna Amiram, Bhimavaram, West Godavari Dist., A.P.

[2017–2021]

Appendix 2

SRKR ENGINEERING COLLEGE (A)

BONAFIDE CERTIFICATE

This is to certify that the project work entitled **RESTORATION OF DEGRADED IMAGES USING VAE AND GAN**, is the bonafide work of **ASHUTOSH MISHRA bearing 17B91A05P4, AAYUSH KARNA bearing 17B91A05P2, RAVI SHANKER KUMAR YADAV bearing 17B91A05P8** who carried out the project work under my supervision in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering.

HEAD OF THE DEPARTMENT

V. ChandraShekhar

SUPERVISOR

Sri BNV Narasimha Raju
Assistant Professor

SELF DECLARATION

We hereby declare that the project work entitled Restoration of Degraded Images Using VAE and GAN is a genuine work carried out by us in B.Tech., (Computer Science and Engineering) at SRKR Engineering College(A), Bhimavaram and has not been submitted either in part or full for the award of any other degree or diploma in any other institute or University.

1. Ashutosh Mishra	17B91A05P4	Ashutosh
2. Aayush Karna	17B91A05P2	Aayush
3. Ravi Shanker Kumar Yadav	17B91A05P8	Ravi

TABLE OF CONTENTS

ABSTRACT	i
LIST OF TABLES	ii
LIST OF FIGURES	iii
LIST OF SYMBOLS	iv
1. INTRODUCTION	1
2. PROBLEM STATEMENT	2
3. LITERATURE SURVEY	3
4. SOFTWARE REQUIREMENTS SPECIFICATION	4
4.1. PURPOSE	4
4.2. SCOPE	4
4.3. DEFINITIONS AND ACRONYMS	4
4.4. EXISTING SYSTEM	5
4.5. PROPOSED SYSTEM	5
4.6. REQUIREMENTS	5
4.6.1. FUNCTIONAL REQUIREMENTS	5
4.6.2. NON-FUNCTIONAL REQUIREMENTS	5
4.6.3. PSEUDO REQUIREMENTS	6
5. SYSTEM DESIGN	7
6. METHODOLOGY	13
7. TESTING & RESULT ANALYSIS	19
8. CONCLUSION	33
9. REFERENCES	34
APPENDIX I: SAMPLE CODE	35

ABSTRACT

Old and deteriorated photos suffer from severe degradation because of multiple reasons. This degradation is restored algorithmically through a deep learning approach. Contrary to conventional restoration tasks that can be solved through supervised learning techniques, the degradation in real photos is complicated and the domain gap between synthetic images and real old photos makes the network fail to generalize. Therefore, a customized generative adversarial network (GAN) is proposed by leveraging real photos along with numerous generated image pairs. Two variational autoencoders (VAEs) are trained to respectively transform old photos and legit photos into two distinct latent spaces. Further the translation between these two latent spaces is learned with synthetic paired data. This translation generalizes pretty much to real photos because the domain gap is limited in the compact latent space. Besides, to address multiple degradations mixed in a single photo, a global branch is designed with a partial nonlocal block targeting to the structured defects, such as scratches and dust spots, and a local branch targeting to the unstructured defects, such as noises and blurriness. Two branches are integrated in the latent space, leading to improved capability to restore old photos from multiple defects. Furthermore, another face refinement network is applied to recover fine details of faces in the old photos, thus ultimately generating photos with enhanced perceptual quality. With comprehensive experiments, the proposed pipeline demonstrates superior performance over state-of-the-art methods as well as existing commercial tools in terms of visual quality for old photos restoration.

LIST OF FIGURES

Disk Requirements	6
Use Case Diagram	8
Class Diagram	9
Sequence Diagram	10
Collaboration Diagram	11
Activity Diagram	12
Architecture of restoration network	14
Progressive Generator network of face enhancement	17
ROC Curve for Scratch Definition	18
Testing Levels	19
Screenshot of new notebook setup	20
Screenshot of requirements.txt	21
Screenshot of setting up the environment	21
Screenshot showing loading the images	22
Screenshot showing the restoration in normal mode	23
Screenshot showing a grid of restored images	24
Screenshot for setting up environment for scratches	25
Screenshot for restoring photos with scratches	26
Screenshot for restoring our own photos	28
Screenshot for determining internal working of the model	29
Screenshot for analysis of restored images against the original images	30
Qualitative analysis and comparison of degraded image	32

LIST OF TABLES

Test cases for Unit Testing	20
Test cases for Black Box Testing	29
Test cases for System Testing	30
Quantitative Comparison of the proposed model	30

Image Restoration is a procedure for reconstruction or recovery of the image affected by a certain phenomenon of degradation like bad atmosphere. Modeling of the degradation and executing the reverse process are primarily restored approaches to recover the initial image. The approaches of image restoration exist both for spatial domains and for frequencies.

Single degradation image restoration: Existing image degradation can be roughly categorized into two groups: unstructured degradation such as noise, blurriness, color fading, and low resolution, and structured degradation such as holes, scratches, and spots. For the unstructured degradations, traditional works often impose different image priors, including nonlocal self-similarity, sparsity and local smoothness. Recently, a lot of deep-learning based methods have also been proposed for different image degradation, like image denoising, super-resolution, and deblurring.

Mixed degradation image restoration: In the real world, a corrupted and degraded image may suffer from complicated defects mixed with a lot of degradations such as loss of resolution, scratches, film noises and color fading. However, a research solving mixed degradation is very less explored. The pioneer work RL-Restore proposed a toolbox that comprises multiple light-weight networks, and each of them responsible for a specific degradation. Then they learn a controller that dynamically selects the operator from the toolbox based on the requirement of the context. The model, built here, performs different convolutional operations in parallel and uses the attention mechanism to select the most suitable combination of operations. However, these methods are still based on supervised learning from synthetic data and hence cannot generalize to real photos. Besides, the focus is only on unstructured defects but do not support structured defects like image inpainting. On the other hand, DIP found that the deep neural network inherently resonates with low-level image statistics and thereby can be utilized as an image prior for blind image restoration without external training data. This method has the potential, to restore in-the-wild images corrupted by mixed factors. In comparison, our approach excels in both restoration performance and efficiency, as per the requirement of the design.

PROBLEM STATEMENT

Old and deteriorating photographic prints decay if they are kept in a poor atmosphere, thereby permanently damaging the precious photo content. Luckily, as mobile and scanning cameras are handy, now users can digitize the images and invite a qualified expert for recovery. However, the manual retouching is usually time consuming and difficult, and leaves stacks unable to recover old photographs. Therefore, automatic methods can be designed to repair old pictures as soon as possible. Degradations in images like optical blur, Black & White colorization, motion blur, quantization, additive noises, grains and low-resolution are to be corrected and restored. The proposed technique should increase efficiency of the algorithms, as well as make it possible to restore more severe degradations, which cannot be elegantly done by the existing approaches.

Prior to the deep learning era, there were some attempts [6] that restore photos by automatically detecting the localized defects such as scratches and blemishes, and filling in the damaged areas with inpainting techniques. Yet these methods [6] focus on completing the missing content and none of them can repair the spatially-uniform defects such as film grain, sepia effect, color fading, etc., so the photos after restoration still appear outdated compared to modern photographic images. With the emergence of deep learning, one can address a variety of low-level image restoration problems [5] by exploiting the powerful representation capability of convolutional neural networks, i.e., learning the mapping for a specific task from a large amount of synthetic images.

Image restoration in clinical imaging is significant and alluring, since it empowers imaging in more attractive conditions, for example imaging with quicker conventions Pruessmann, less expensive gadgets and lower radiation Naidich et al. (1990), and so forth. In few cases, clinical picture reclamation requires a harder assessment than reestablishing characteristic pictures. It does not just need more keen and outwardly practical rebuilding [3], though additionally requires precise picture fulfillment without adjusting any obsessive highlights or influencing any demonstrative characteristics/properties. Subsequently clinical picture rebuilding can be a benchmark task for related image restoration techniques [2]. Inside this decade, picture rebuilding strategy has been quickly developing by joining different earlier data into addressing the badly presented opposite imaging task. The earlier data develops from utilizing meager portrayal presumption Mairal [4], authorizing low-position examination Dong more as of late utilizing profound learning based priors Wang or models Zhang and Zuo [5]. Anyways there are yet a few difficulties and restrictions for existing calculations:

- 1) Generative Adversarial Network (GAN) based strategies fundamentally improve the outcomes to create outwardly practical restoration. Anyways GANs guarantee the consistency to a learned appropriation [1] but do not really ensure the arrangement of precise coordinates with the relating ground truth.
- 2) It is still obvious that latent space misplacement or mode-breakdowns may occur while the restoration takes place in improved GAN models [2].

SOFTWARE REQUIREMENTS AND SPECIFICATIONS

4.1 PURPOSE:

The main aim of the proposed to restore and correct all the degradations present in an image in the most convenient way such that the image does not lose its visual meaning. The degradation to be restored or enhanced are optical blur, black and white colorization, motion blur, quantization, additive noises, grains and low-resolution. The challenge also includes designing an efficient technique to remove all these apparently permanent deteriorations.

4.2 SCOPE:

The images can be of any kind like it might be related to medical imaging / astronomical imaging etc. Let us take astronomical imaging as an example, planetary observations are degraded by movement blur as the rapid spaceship movement but sluggish/slow camera's shutter speed. The degradation problems of astronomical imaging are generally marked by noise from Poisson, Gaussian noise etc. As these problems will be solved using deep learning it can be expected that this solution is having good scope in future.

4.3 DEFINITIONS AND ACCRONYMS:

- VAE: Variational Auto-Encoders
- GAN: Generative Adversarial Networks.
- JUPYTER NOTEBOOK: The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

4.4 EXISTING SYSTEM:

In current system, conventional restoration approaches are used that can be solved through complicated supervised learning algorithms. However, if the degradation in real photos is complex and difficult to visualize, the network can fail to generalize, and restore the image degradation.

4.5 PROPOSED SYSTEM:

The proposed technique is designed as to leverage real photos along with massive synthetic image pairs. It restores old photos that suffer from severe degradation through a deep learning approach, where predictions are done for the latent space content. Degradations in images like optical blur, Black & White colorization, motion blur, quantization, additive noises, grains and low-resolution are to be corrected and restored. The proposed technique can increase efficiency of the algorithms, as well as make it possible to restore more severe degradations, which cannot be elegantly done by the existing approached.

4.6 REQUIREMENTS ANALYSIS:

4.6.1 FUNCTIONAL REQUIREMENTS

- Receiving the provided dataset consisting of degraded images.
- Preprocessing the data by cleaning any noisy data.
- Transforming the data if required.
- Splitting the dataset into training, test and validation set.
- Train the dataset through different algorithms to make a model.
- Checking the accuracy for the models on testing set and contrasting it with the benchmark accuracy.
- Start predicting the latent space content, and additional visual properties through new images on the best model obtained.

4.6.2 NON-FUNCTIONAL REQUIREMENTS

- **Availability:** The software for detection of the degraded portion in the image and also the scratches present and be available in all the systems.
- **Performance:** Performance is calculated in terms of prediction of the model. Accuracy is used to measure the performance of the model.

- **SOFTWARE REQUIREMENTS:**

PYTHON

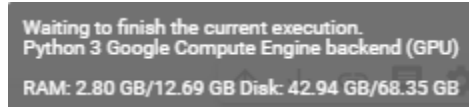
Python is a deciphered, significant level, broadly useful programming language which is used for different tasks like web development, data science.

GOOGLE COLABS

Google Colabs is a free cloud service provided by google and it includes GPU and helps to develop deep learning applications by providing libraries like Keras, TensorFlow, Pytorch and OpenCV.

- **HARDWARE REQUIREMENTS:**

1. RAM : 4GB
2. Processor : Intel core i3
3. Hard Disk : 50GB
4. OS : Windows 10
5. High speed internet



Waiting to finish the current execution.
Python 3 Google Compute Engine backend (GPU)
RAM: 2.80 GB/12.69 GB Disk: 42.94 GB/68.35 GB

This figure is a screenshot of the status of disk and GPU while executing the files in Google Colab.

Chapter 5

SYSTEM DESIGN

System Design is a process of organizing different elements of a system such as modules, interfaces and data that flows through the system. The entire spectrum of diagrams and explanations which depicts the system design of Image Restoration using VAE and GAN is shown below.

UML DIAGRAMS:

The Unified Modeling Language is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system.

Each Unified Modeling Language is designed to let developers and customers view software system from a different perspective and in varying degrees of abstraction.

UML diagrams commonly created in visual modeling tools include:

1. Use Case Diagram
2. Class Diagram
3. Interaction Diagram
 - i. Sequence Diagram
 - ii. Collaboration Diagram
4. State Chart Diagram
5. Activity Diagram

1. USE CASE DIAGRAM:

Use Case Diagram represents the interaction of use with the system which shows the relationship between user and use cases.

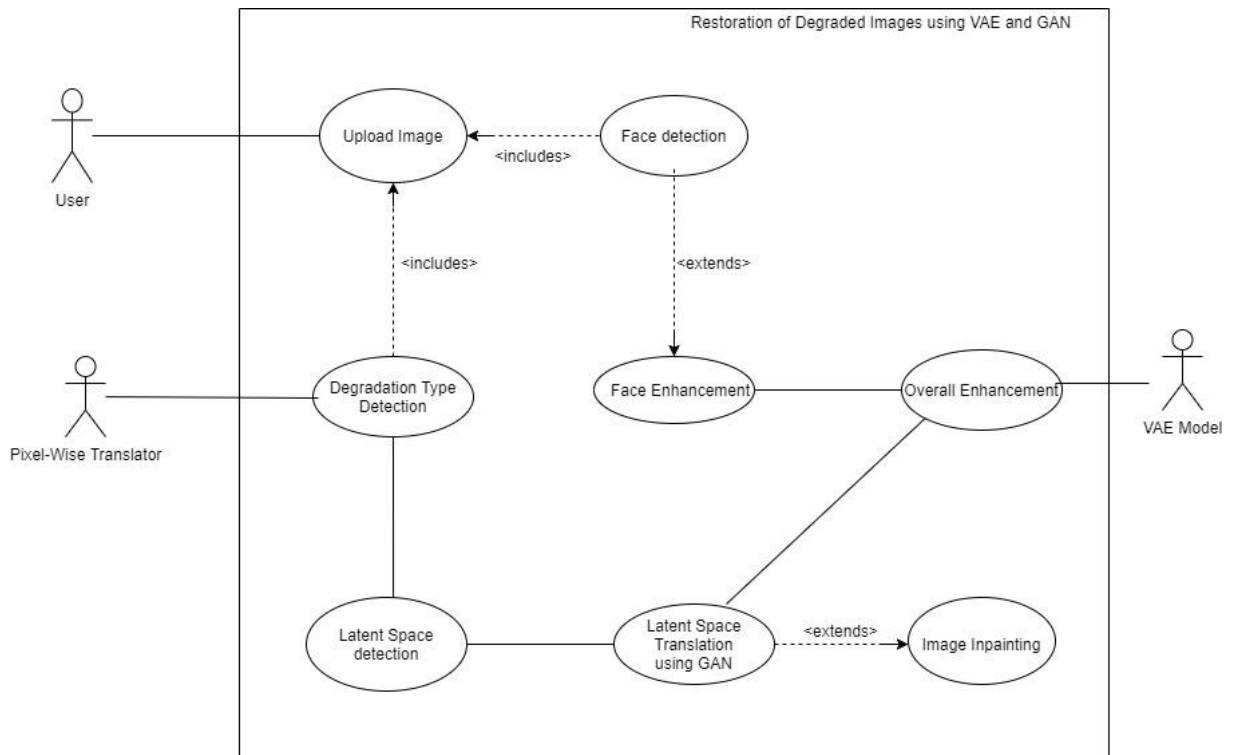


Fig: 5.1 Use-Case Diagram

2. CLASS DIAGRAM:

The class diagram is the diagram that which classifies the actor which is defined in the use case diagram into a set of inter-related classes.

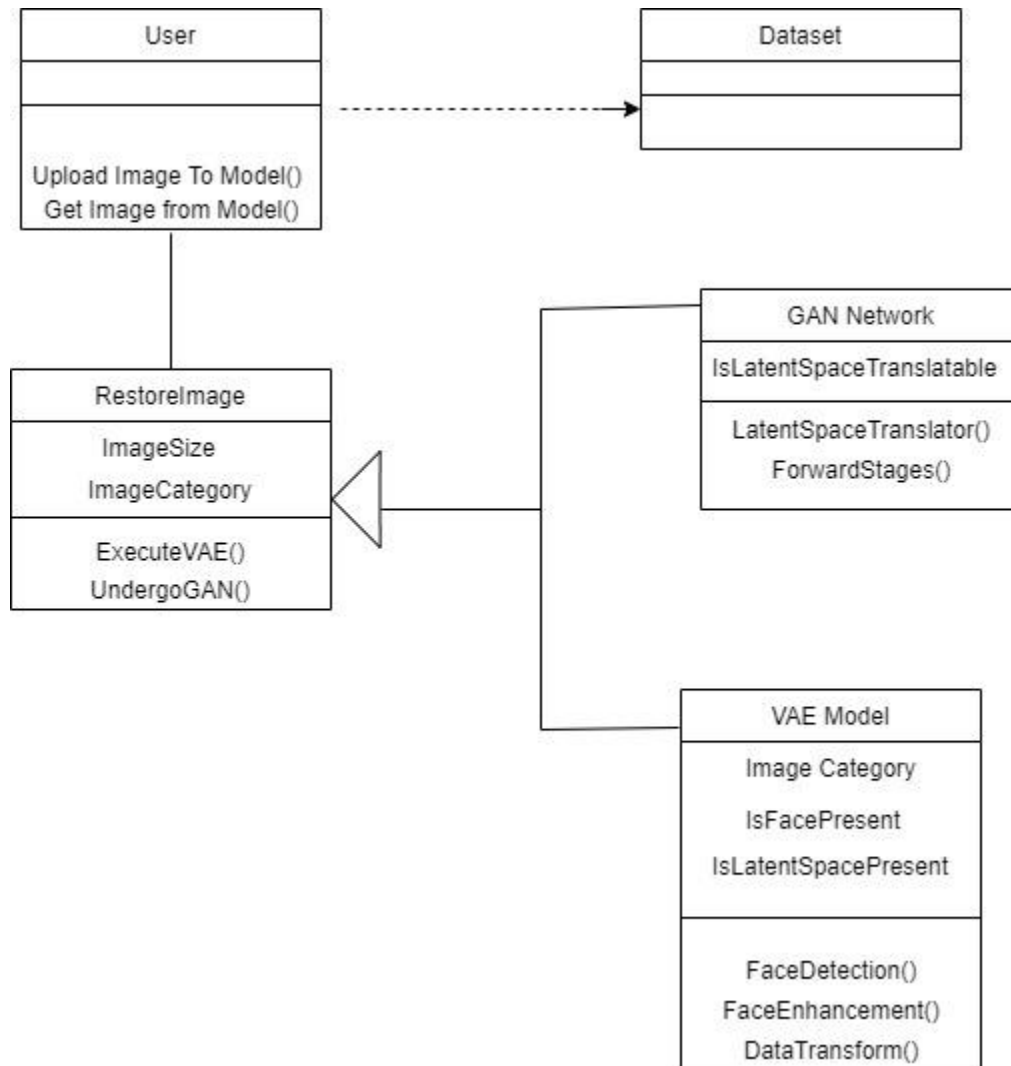


Fig: 5.2 Class Diagram

3. INTERACTION DIAGRAM:

The purpose of interaction diagram is to visualize the interactive behavior of the system. This interactive behavior is represented in UML by two diagrams known as Sequence diagram and collaboration diagram.

- Sequence Diagram
- Collaboration Diagram

3.1 SEQUENCE DIAGRAM

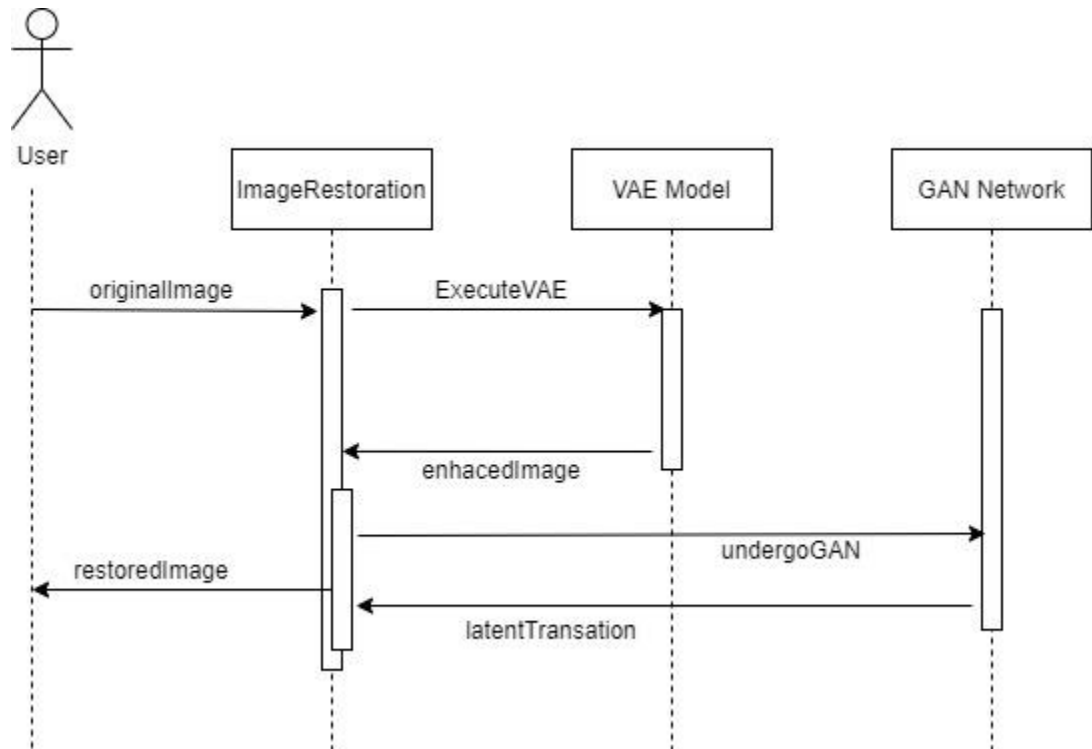


Fig: 5.3.1 Sequence Diagram

3.2 COLLABORATION DIAGRAM

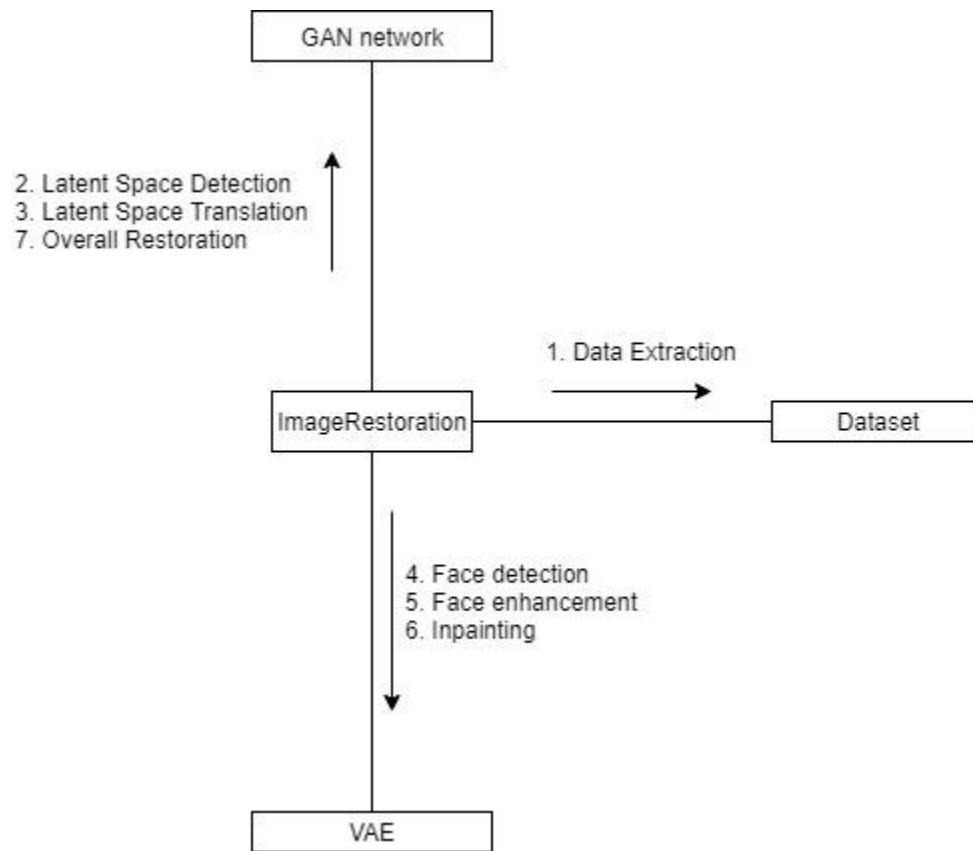


Fig: 5.3.2 Collaboration Diagram

4. ACTIVITY DIAGRAM:

Activity Diagram displays the flow from initial point to the very last point depicting all the decision paths when an activity gets executed.

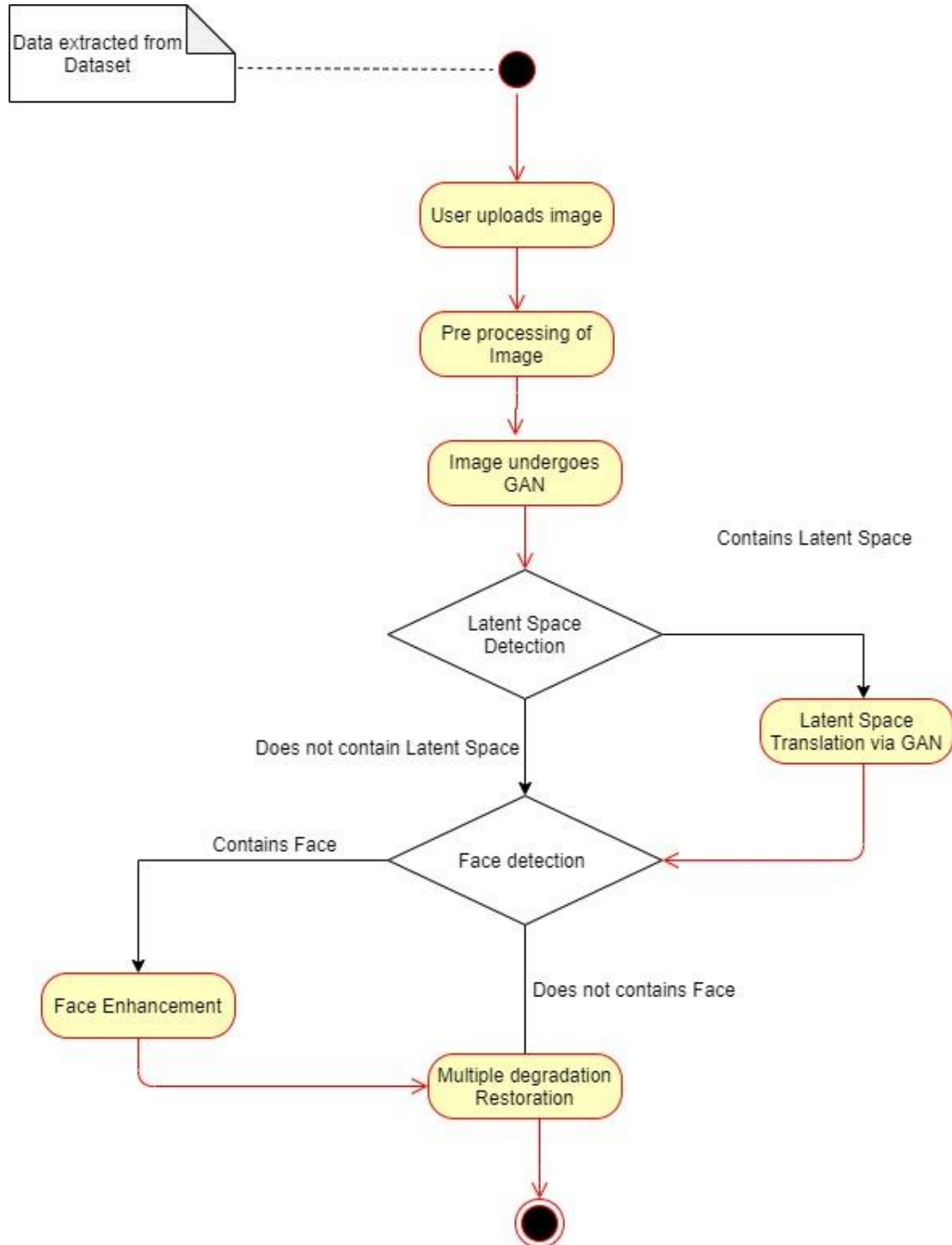


Fig: 5.4 Activity Diagram

In contrast to standard image restoration tasks, old photo restoration is tougher. The reasons are mentioned below:

- i. Old photos contain far more complex degradation that is hard to be modeled realistically and there always exists a domain gap between synthetic and real photos.
- ii. The defect of old photos is a compound of multiple degradations.
- iii. Unstructured defects such as film noise, blurriness and color fading, are to be restored with spatially homogeneous filters by using surrounding pixels within the local patch.
- iv. Structured defects such as scratches and blotches should be in-painted by considering the global context to ensure the structural consistency.

The proposed methodology has been organized into four different phases. Each phase is explained intimately below.

6.1 Restoration via latent space translation:

In order to destroy the domain gap, the old photo restoration is formulated as an image translation problem, where clean images and old photos are treated as images from distinct domains and the mapping in between is learned. However, as opposed to general image translation methods that connect two different domains, here, images across three domains are translated: the real photo domain R , the synthetic domain X where images suffer from artificial or intentional degradation, and the corresponding reality domain Y that comprises images without degradation. Such triplet domain translation is very important in this task as it supports the unlabeled real photos as well as a large amount of synthetic data associated with ground truth.

Images from three domains are denoted respectively with $r \in R$, $x \in X$ and $y \in Y$, where x and y are paired by data synthesis, i.e., x is degraded from y . Directly learning the mapping from real photos $\{r\}_{N_{i=1}}$ to clean images $\{y\}_{N_{i=1}}$ is hard since they are not paired and thus unsuitable for supervised learning. It is thereby proposed to decompose

the translation with two stages. First, it is proposed to map R, X, Y to corresponding latent spaces via $E_R : R \rightarrow Z_R$, $E_X : X \rightarrow Z_X$, and $E_Y : Y \rightarrow Z_Y$, respectively. Specifically, as synthetic images and real old photos are both corrupted, sharing similar looks, their latent space is aligned into the common domain by holding up some constraints. Thus we have $Z_R \approx Z_X$. This aligned latent space encodes features for all the corrupted images, either synthetic or real ones. Then it is proposed to analyze the image restoration method in the latent space. Specifically, by utilizing the synthetic data pairs $\{x, y\}_{i=1}^N$, the translation is learned from the latent space of corrupted images, Z_X , to the latent space of ground truth, Z_Y , through the mapping $T_Z : Z_X \rightarrow Z_Y$, where Z_Y can be further reversed to Y through generator $G_Y : Z_Y \rightarrow Y$.

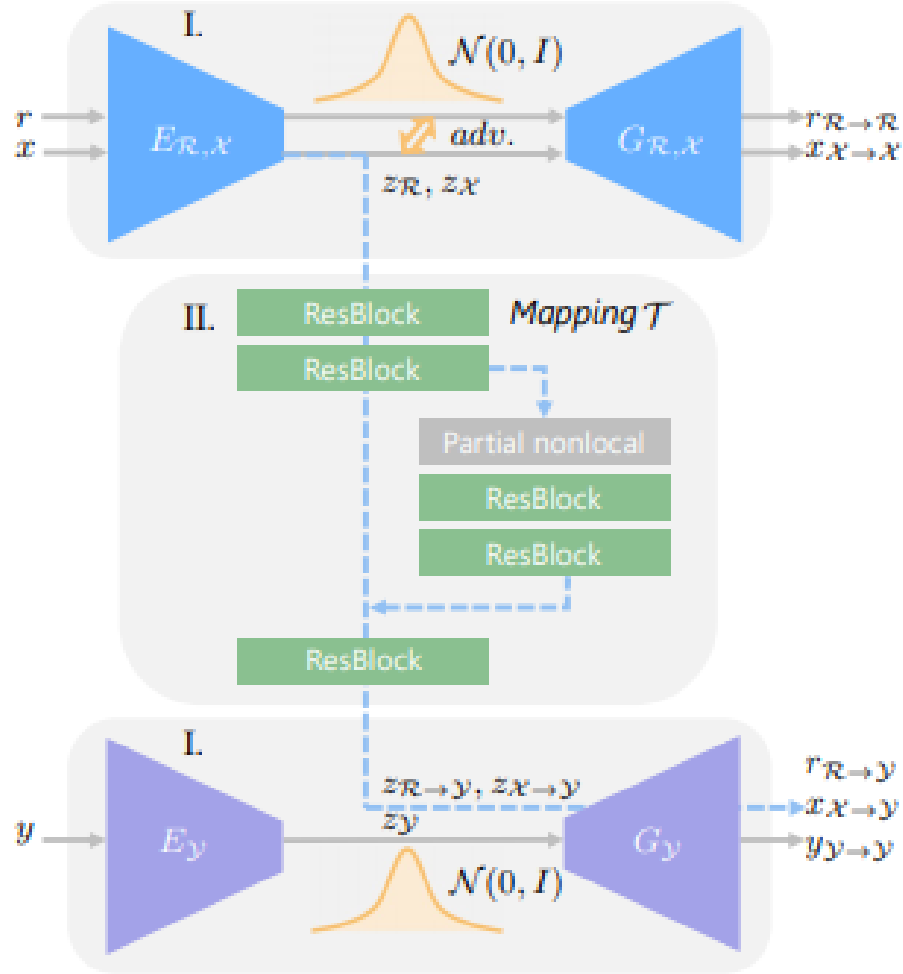


Fig: 6.1 Architecture of our restoration Network

- (I.) First, train two VAEs: VAE1 for images in real photos $r \in \mathcal{R}$ and synthetic images $x \in \mathcal{X}$, with their domain gap closed by jointly training an adversarial discriminator; VAE2 is trained for clean images $y \in \mathcal{Y}$. With VAEs, photos are transformed to dense latent space.
- (II.) Then, learn the mapping that restores the corrupted images to clean ones in the latent space.

Domain alignment in the VAE latent space

One gist of our method is to abide by the assumption that \mathcal{R} and \mathcal{X} are encoded into the same latent space. To this end, it is proposed to utilize variational autoencoder (VAE) to encode photos with compact representation, whose domain gap is further examined by an adversarial discriminator. We use the network architecture shown in Figure 6.1 to realize this concept. In the first stage, two VAEs are to be learned for the latent representation. Old photos $\{r\}$ and synthetic images $\{x\}$ share the first one termed VAE1, with the encoder $E_{\mathcal{R},\mathcal{X}}$ and generator $G_{\mathcal{R},\mathcal{X}}$, while the ground true images $\{y\}$ are fed into the second one, VAE2 with the encoder-generator pair $\{E_Y, G_Y\}$. VAE1 is common for both r and x in the aim that images from both corrupted domains can be mapped to a shared latent space. The VAEs assumes Gaussian prior for the distribution of latent codes, so that images can be reconstructed by sampling from the latent space. The re-parameterization trick is used to execute differentiable stochastic sampling and optimize VAE1 with data $\{r\}$ and $\{x\}$ respectively. The objective with $\{r\}$ is defined as:

$$\begin{aligned}\mathcal{L}_{\text{VAE}_1}(r) = & \text{KL}(E_{\mathcal{R},\mathcal{X}}(z_r | r) || \mathcal{N}(0, I)) \\ & + \alpha \mathbb{E}_{z_r \sim E_{\mathcal{R},\mathcal{X}}(z_r | r)} \left[\|G_{\mathcal{R},\mathcal{X}}(r_{\mathcal{R} \rightarrow \mathcal{R}} | z_r) - r\|_1 \right] \\ & + \mathcal{L}_{\text{VAE}_1, \text{GAN}}(r)\end{aligned}$$

Where definition for the expression is mentioned as follows:

$$\begin{aligned}\mathcal{L}_{\text{VAE}_1, \text{GAN}}^{\text{latent}}(r, x) = & \mathbb{E}_{x \sim \mathcal{X}} \left[D_{\mathcal{R},\mathcal{X}}(E_{\mathcal{R},\mathcal{X}}(x))^2 \right] \\ & + \mathbb{E}_{r \sim \mathcal{R}} \left[\left(1 - D_{\mathcal{R},\mathcal{X}}(E_{\mathcal{R},\mathcal{X}}(r)) \right)^2 \right]\end{aligned}$$

6.2 Multiple degradation restoration

The latent restoration using the residual blocks, only concentrates on local features due to the limited receptive field of each layer. However, the restoration of structured defects requires plausible inpainting, which has to consider long-range dependencies to ensure global structural consistency. Since legacy photos often have mixed degradations, the design of a restoration network that simultaneously supports the two mechanisms is necessary. Towards this goal, it is proposed to enhance the latent restoration network by incorporating a global branch as shown in Figure 6.1.

The affinity between i th location and j th location in F , denoted by $s_{i,j} \in \mathbb{R}^{HW \times HW}$, is calculated by the correlation of F_i and F_j modulated by the mask $(1 - m_j)$, i.e.,

$$s_{i,j} = (1 - m_j) f_{i,j} / \sum_k (1 - m_k) f_{i,k}$$

where, $f_{i,j} = \exp \left(\theta(F_i)^T \cdot \phi(F_j) \right)$ gives the pairwise affinity with embedded Gaussian.

The global branch is designed specifically for inpainting and hope the non-hole regions are left untouched, so the global branch is fused with the local branch under the guidance of the mask, i.e.,

$$F_{\text{fuse}} = (1 - m) \odot \rho_{\text{local}}(F) + m \odot \rho_{\text{global}}(O)$$

6.3 Defect Region Detection

Since the global branch of our restoration network requires a mask m as the guidance, in order to get the mask automatically, we train a scratch detection network in a supervised way by using a mixture of real scratched dataset and synthetic dataset. Specifically, let $\{s_i, y_i | s_i \in S, y_i \in Y\}$ denote the whole training pairs, where s_i and y_i are the scratched image and the corresponding binary scratch mask respectively, we use the cross-entropy loss to minimize the difference between the predicted mask \hat{y}_i and y_i .

6.4 Face Enhancement

The restoration network proposed above is general to all kinds of old photos. However, considering restoration quality on faces is most sensitive to human perception, it is

$$\mathcal{L}_{CE} = \mathbb{E}_{(s_i, y_i) \sim (\mathcal{S}, \mathcal{Y})} \left\{ \alpha \sum_{h=1}^H \sum_{w=1}^W -y_i^{(h,w)} \log \hat{y}_i^{(h,w)} - (1 - \alpha) \sum_{h=1}^H \sum_{w=1}^W (1 - y_i^{(h,w)}) \log (1 - \hat{y}_i^{(h,w)}) \right\}$$

further proposed to build a network for face enhancement. Given one real degraded photo r , the aim is to reconstruct degraded faces rf in r into a much detailed and clean version with proposed face enhancement network G_f . The conventional pixel-wise translation method could not solve such a blind restoration issue well because the degradation prior is totally unknown. Here, this problem is solved from the perspective of generative models.

As shown in Figure 6.2, we employ a coarse-to-fine generator to translate a low-dimensional code z into corresponding high-resolution and clean faces, where z is a down sampled patch of rf (8×8 in our implementation). At the same time of progressive generation, rf will be injected into the generator in each scale with a spatial adaptive manner.

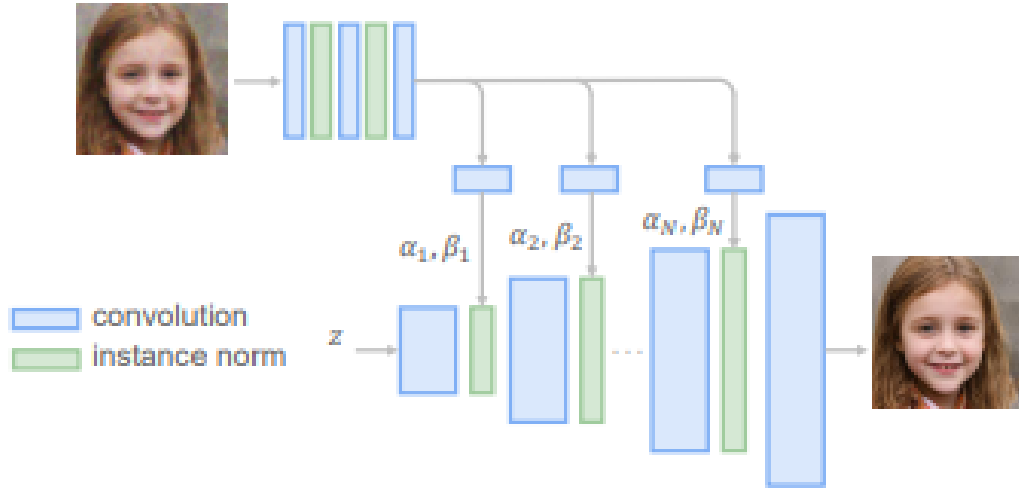


Fig: 6.2 The progressive generator network of face enhancement

Starting from a latent vector z , the network up-samples the feature map by deconvolution progressively. The degraded face will be injected into various resolutions in a spatial condition manner.

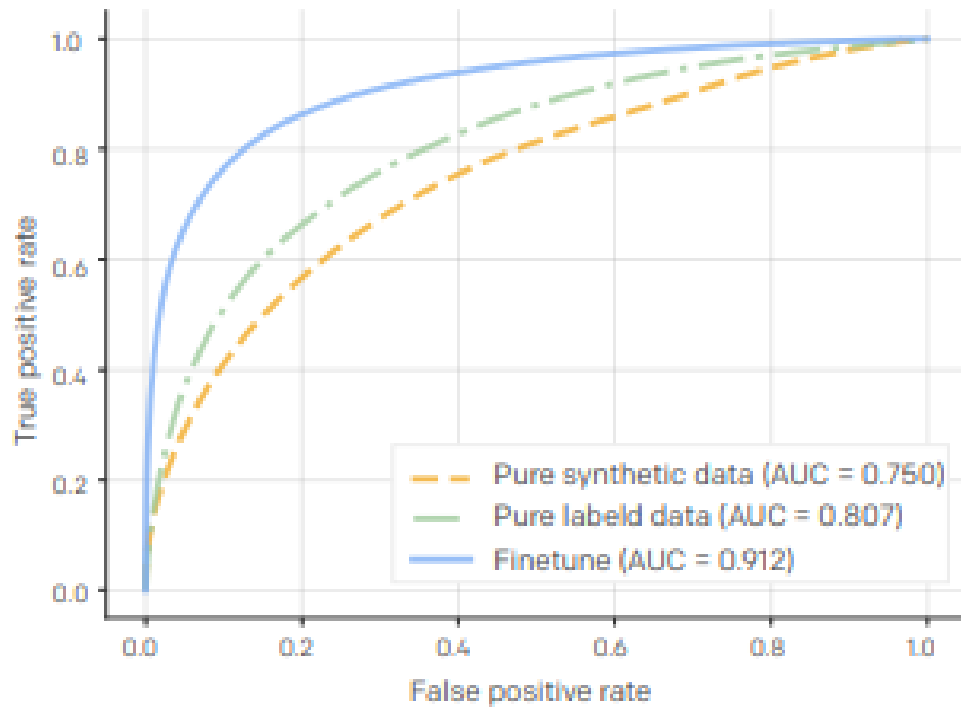


Fig: 6.3. ROC Curve for scratch definition of different data settings

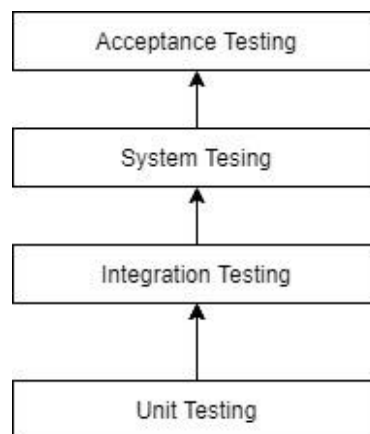
As the ROC Curve depicts, combining both synthetic structured degradations and a small amount of labeled data, the scratch detection network can achieve great results.

Testing:

Testing is the process of knowingly trying to cause failures and bugs in a system in order to detect any defects that might be present or possible. It is the time where we analyze whether all the user requirements are met or not by providing different testcases.

A test case in software engineering normally consists of a unique identifier, requirement references from a design specification, preconditions, events, a series of steps (also known as actions) to follow, input, output and it validates the system requirements and generates a pass or fail based on the check operation performed.

The different levels of testing strategies applied at various phases of software development are:



1. Unit Testing

Unit Testing is generally done on individual units/components of the software. It helps to verify every single unit of the software.

Tab: 7.1 Test cases for Unit Testing

S. No	Test Cases	Input Values	Expected Output	Obtained Output	Result
1.	Setting up a new notebook	Create a new notebook using google colab	Opening of new python notebook with all required specifications	Same	Success

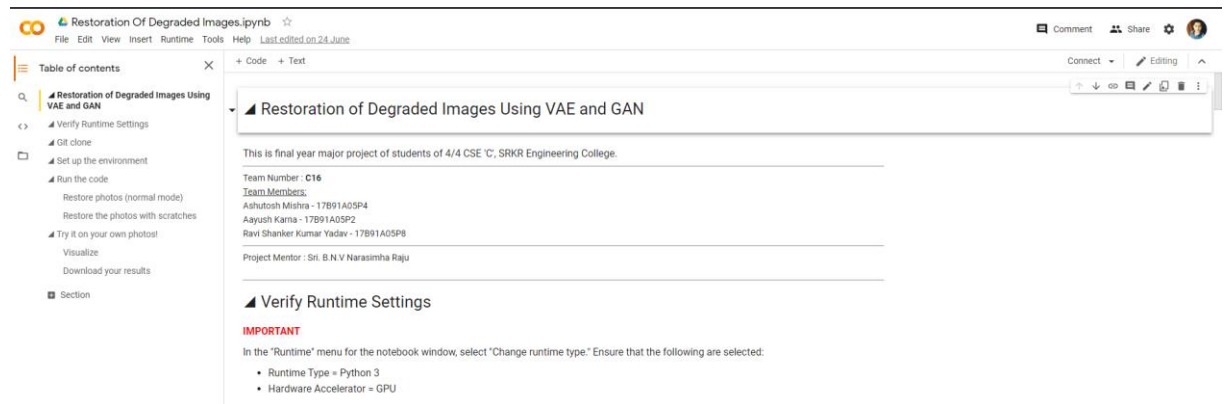


Fig: 7.1 Screenshot for new notebook setup

Tab: 7.2 Test cases for Unit Testing

S. No	Test Cases	Input Values	Expected Output	Obtained Output	Result
2.	Setting up the environment	List of requirements in a text file	Requirements satisfied	Requirements satisfied	Success

```

12 lines (12 sloc) | 113 Bytes

1 torch
2 torchvision
3 dlib
4 scikit-image
5 easydict
6 PyYAML
7 dominate>=2.3.1
8 dill
9 tensorboardX
10 scipy
11 opencv-python
12 einops

```

Fig: requirements.txt

```

+ Code + Text
Connect Editing

! pip install -r requirements.txt

Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 1)) (1.8.1+cu101)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 2)) (0.9.1+cu101)
Requirement already satisfied: dlib in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 3)) (19.10.0)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 4)) (0.16.2)
Requirement already satisfied: easydict in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 5)) (1.9)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 6)) (3.13)
Requirement already satisfied: dominate>=2.3.1 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 7)) (2.6.0)
Requirement already satisfied: dill in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 8)) (0.3.3)
Requirement already satisfied: tensorboardX in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 9)) (2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 10)) (1.4.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 11)) (4.1.2.30)
Requirement already satisfied: einops in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 12)) (0.3.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch->-r requirements.txt (line 1)) (3.7.4.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torch->-r requirements.txt (line 1)) (1.19.5)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.7/dist-packages (from torchvision->-r requirements.txt (line 2)) (7.1.2)
Requirement already satisfied: PyWavelets>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->-r requirements.txt (line 4)) (1.1.1)
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->-r requirements.txt (line 4)) (2.5.1)
Requirement already satisfied: matplotlib>=3.0.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->-r requirements.txt (line 4)) (3.2.2)
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->-r requirements.txt (line 4)) (2.4.1)
Requirement already satisfied: protobuf>=3.8.0 in /usr/local/lib/python3.7/dist-packages (from tensorboardX->-r requirements.txt (line 9)) (3.12.4)
Requirement already satisfied: decorator>=4.3 in /usr/local/lib/python3.7/dist-packages (from networkx>=2.0->-r requirements.txt (line 4)) (4.4.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0,>=2.0.0->-r requirements.txt (line 4)) (1.3.1)
Requirement already satisfied: pyparsing>=2.0.4,>=2.1.2,>=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0,>=2.0.0->-r requirements.txt (line 4)) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0,>=2.0.0->-r requirements.txt (line 4)) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0,>=2.0.0->-r requirements.txt (line 4)) (2.8.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from protobuf>=3.8.0->-r requirements.txt (line 9)) (57.0.0)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.7/dist-packages (from protobuf>=3.8.0->-r requirements.txt (line 9)) (1.15.0)

```

Fig: 7.2 Screenshot for Setting up the environment

S. No	Test Cases	Input Values	Expected Output	Obtained Output	Result
3.	Restoring the photos in normal mode	Blurred and granular Images	Clear images with removed grains	Same	Success

```

▼ 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
Warning: There is no face in f.png
1
Warning: There is no face in e.png
1
Warning: There is no face in d.png
Warning: There is no face in b.png
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]

```

Fig: 7.3 Screenshot showing loading the images and passing to corresponding functions

+ Code + Text

```
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[:,0: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[:,0: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
```

Fig: 7.4 Screenshot showing the restoration in normal mode, and building a grid of restored images

+ Code + Text

```
▶ 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))
```

□



b.png



Fig: 7.5 Screenshot showing the restoration in normal mode, and a grid of restored images

S. No	Test Cases	Input Values	Expected Output	Obtained Output	Result
4.	Restoring photos with scratches	Images with scratches	Images without scratches	Same	Success

```

+ Code + Text
Connect Editing ^
▼ Restore the photos with scratches

1 |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 .ipynb_checkpoints
Skipping non-file .ipynb_checkpoints
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/python3.7/dist-packages/torch/nn/functional.py:3458: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.0. Please specify align
"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
Warning: There is no face in d.png
1
Finish Stage 2 ...

Running Stage 3: Face Enhancement
The main GPU is
0
dataset [FaceTestDataset] of size 3 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 :)
Finish Stage 3 ...

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

```

Fig: 7.5 Screenshot for setting up environment for restoring photos with scratches

+ Code + Text

```
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:
    if( filename != '.ipynb_checkpoints'):
        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



Fig: 7.6 Screenshot for restoring photos with scratches

S. No	Test Cases	Input Values	Expected Output	Obtained Output	Result
5.	Uploaded Image has to be restored	Upload blurred image	Restored image	Same	Success

+ Code
+ Text

▼ ▲ Try it on your own photos!

```

[18] from google.colab import files
import shutil
import os

upload_path = os.path.join(basepath, "test_images", "upload")
upload_output_path = os.path.join(basepath, "upload_output")

if os.path.isdir(upload_output_path):
    shutil.rmtree(upload_output_path)

if os.path.isdir(upload_path):
    shutil.rmtree(upload_path)

os.mkdir(upload_output_path)
os.mkdir(upload_path)

uploaded = files.upload()
for filename in uploaded.keys():
    shutil.move(os.path.join(basepath, filename), os.path.join(upload_path, filename))

```

Choose Files
AshutoshImageDemo.jpg

- **AshutoshImageDemo.jpg**(image/jpeg) - 534106 bytes, last modified: 6/26/2021 - 100% done

Saving AshutoshImageDemo.jpg to AshutoshImageDemo.jpg

```

!python run.py --input_folder /content/photo_restoration/test_images/upload --output_folder /content/photo_restoration/upload_output --GPU 0

```

```

Running Stage 1: Overall restoration
Now you are processing AshutoshImageDemo.jpg
Finish Stage 1 ...

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

Running Stage 3: Face Enhancement
The main GPU is
0
dataset [FaceTestDataset] of size 2 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 :)
Finish Stage 3 ...

Running Stage 4: Blending
Finish Stage 4 ...

All the processing is done. Please check the results.

```

+ Code + Text

```
▶ filenames_upload = os.listdir(os.path.join(upload_path))
   filenames_upload.sort()

   filenames_upload_output = os.listdir(os.path.join(upload_output_path, "final_output"))
   filenames_upload_output.sort()

   for filename, filename_output in zip(filenames_upload, filenames_upload_output):
       image_original = PIL.Image.open(os.path.join(upload_path, filename))
       image_restore = PIL.Image.open(os.path.join(upload_output_path, "final_output", filename_output))

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



Fig: 7.7 Screenshot for restoring our own photos
(Left -> original, Right -> restored)

2. Integration Testing

Integration testing is performed by combining individual units and all these units are tested as group.

3. Black box Testing

In this testing, internal implementation of an item being tested is ignored by the tester. Tester only concentrates that for the given input, the output that was obtained is appropriate or not.

Tab: 7.2 Test cases for Black box Testing

S. No	Test Cases	Input Values	Expected Output	Obtained Output	Result
1.	Calculating Completion of all 4 steps asynchronously	Providing image for restoration	Analyze the internal process through diagnosis print statements	Same	Success

```
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/python3.7/dist-packages/torch/nn/functional.py:3613: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.0. Please specify
"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
2
1
1
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 :)
Finish Stage 3 ...

Running Stage 4: Blending
Finish Stage 4 ...

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

Fig: 7.8. Screenshot for Determining internal working of the model

4. System Testing

System testing is the level of testing that validates the complete and fully integrated software.

Tab: 7.3 Test cases for System Testing

S. No	Test Cases	Input Values	Expected Output	Obtained Output	Result
1.	Analyze the images after restoration	Provide degraded images	Obtain the Restoration analysis	Same	Success

RESULT ANALYSIS:

This method is compared against other standard methods and approaches of image restoration. For the sake of comparison, the same degraded photo is provided to all the approaches and then the results are analyzed using based on gross comparison. Further, degraded photos which include all types of degradations such as noise, blurred images, scratches, color disruption, not to mention all of them. The restorations done on all these aspects are then analyzed, based on factors such as efficiency, performance and result.

The methods against which our approach is compared are as follows:

- Pix2Pix: It is a supervised image translation method, which uses synthetic image pairs to learn the translation in pixel level.
- CycleGAN: It is a popular unsupervised image translation method that learns the translation using unpaired photos from different domains.
- Deep Image Prior (DIP): It learns the image restoration given a single degraded image and is powerful in denoising. It has super-resolution and uses blind inpainting.
- Sequential Method: It is a classical denoising method and provides a structured inpainting technique.

The result analysis is divided into two broad portions, namely Quantitative analysis and Qualitative analysis.

Qualitative analysis and comparison:

- i. Our method can restore both structured and unstructured degradations and our recovered results are significantly better than the others, based on the naked eye analysis. Below are the images for comparison.

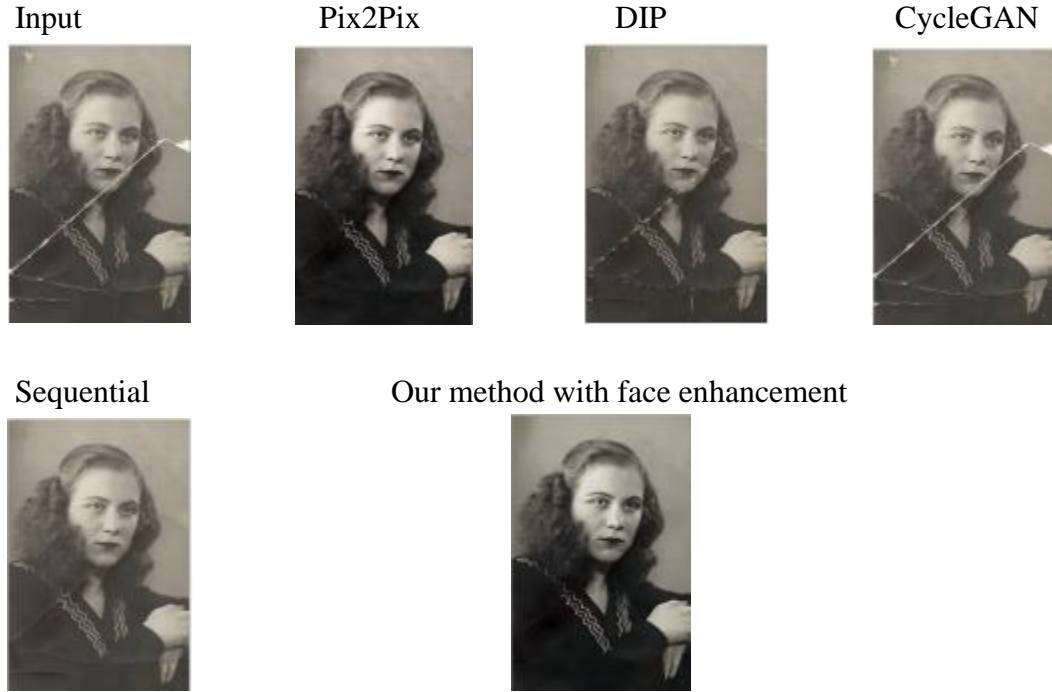


Fig 7.8: Qualitative analysis and comparison

- ii. However, our method has a limitation that scratches where the latent space cannot even be formulated using the global context may not be able to be synthesized, so it gets residual in the output image. Pix2Pix method might usually perform better than our method in such a concern.

Qualitative analysis and comparison

- i. Quantitatively, if we analyze our method, we take into consideration four parameters and metrics out of many. The concerned ones are as follow:
 - Peak Signal-to-Noise Ratio (PSNR): It is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation. It compares the low-level difference between the restored output and the ground truth of the image.
 - Structural Similarity Index (SSM): It is a perceptual metric that quantifies image quality degradation caused by processing such as data compression or by losses in data transmission. It is a full reference metric that needs two images from the same image capture— a reference and a processed one.

- Learned Perceptual image Patch Similarity (LPIPS): This metric calculates the distance of multilevel activations of a pretrained network and is deemed to better correlate with human perception.
- Frechet Inception Distance (FID): It calculates the distance between the feature distributions of the final outputs and the real images. It is a metric for evaluating the quality of generated images and specifically developed to evaluate the performance of generative adversarial networks.

The below table shows the quantitative comparison of all the considered methods against our method with respect to the above-mentioned metrics and parameters.

Tab: 7.4 Quantitative Comparison of the proposed model

Method	PSNR	SSIM	LPIPS	FID
Input	12.92	0.49	0.59	306.80
DIP	22.59	0.57	0.54	194.55
Pix2Pix	22.18	0.62	0.23	135.14
Sequential	22.71	0.60	0.49	191.98
Our method	23.33	0.69	0.25	135.4

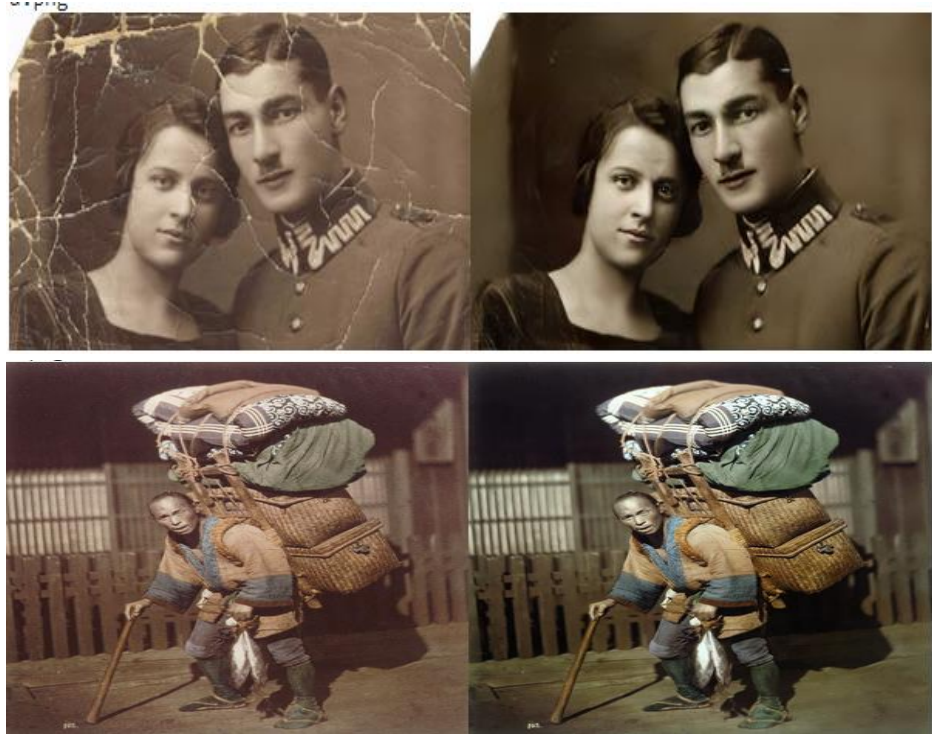


Fig: 7.6 Human Eye Analysis for restored image

CONCLUSION AND FUTURE WORK

A new triplet domain translation network is built that opens new avenue to restore the integrated degradation in the old photos. The domain gap is reduced between old photos and synthetic images, and the translation to clean images is analyzed in latent space. This method suffers less from generalization issue compared with prior methods. Besides, a partial nonlocal block is also made which restores the latent features by leveraging the global context, so the scratches can be in-painted with better structural consistency. Furthermore, a coarse-to-fine generator with spatial adaptive condition is also considered to reconstruct the face regions of old photos. Our method demonstrates good performance in restoring severely degraded old photos. However, our technique may not be able to handle more complex shading. This is because our dataset contains few old photos with complex shading defects. One could possibly address this limitation using our framework by explicitly considering the shading effects during synthesis or adding more such photos as training data.

REFERENCES

- [1] Microsoft Corporation – “Bringing Old Photos back to life, CVPR2020.”
- [2] Degraded Photo Restoration through Deep Latent Space Translation, PAMI – Research Paper
- [3] F. Stanco, G. Ramponi, and A. De Polo, “Towards the automated restoration of old photographic prints: a survey,” in The IEEE Region 8 EUROCON 2003. Computer as a Tool., vol. 2. IEEE, 2003, pp. 370–374.
- [4] V. Bruni and D. Vitulano, “A generalized model for scratch detection,” IEEE transactions on image processing, vol. 13, 2004.
- [5] K. Zhang, W. Zuo, S. Gu, and L. Zhang, “Learning deep cnn denoiser prior for image restoration,” in the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 3929–3938.
- [6] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, “Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising,” IEEE Transactions on Image Processing, vol. 26, no. 7, pp. 3142–3155, 2017.

APPENDIX – I (SAMPLE CODE)

- Git Clone

```
!git clone https://github.com/ashutoshm1771/Bringing-Old-Photos-Back-to-Life.git photo_restoration
```

- Setting up the environment

```
import argparse
```

```
import gc
```

```
import json
```

```
import os
```

```
import time
```

```
import warnings
```

```
import numpy as np
```

```
import torch
```

```
import torch.nn.functional as F
```

```
import torchvision as tv
```

```
from PIL import Image, ImageFile
```

```
from detection_models import networks
```

```
from detection_util.util import *
```

```
warnings.filterwarnings("ignore", category=UserWarning)
```

```
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

```
def data_transforms(img, full_size, method=Image.BICUBIC):
```

```
    if full_size == "full_size":
```

```
        ow, oh = img.size
```

```
        h = int(round(oh / 16) * 16)
```

```

        w = int(round(ow / 16) * 16)
        if (h == oh) and (w == ow):
            return img
        return img.resize((w, h), method)

    elif full_size == "scale_256":
        ow, oh = img.size
        pw, ph = ow, oh
        if ow < oh:
            ow = 256
            oh = ph / pw * 256
        else:
            oh = 256
            ow = pw / ph * 256

        h = int(round(oh / 16) * 16)
        w = int(round(ow / 16) * 16)
        if (h == ph) and (w == pw):
            return img
        return img.resize((w, h), method)

def scale_tensor(img_tensor, default_scale=256):
    _, _, w, h = img_tensor.shape
    if w < h:
        ow = default_scale
        oh = h / w * default_scale
    else:
        oh = default_scale
        ow = w / h * default_scale

```

```

oh = int(round(oh / 16) * 16)
ow = int(round(ow / 16) * 16)

return F.interpolate(img_tensor, [ow, oh], mode="bilinear")

def blend_mask(img, mask):

    np_img = np.array(img).astype("float")

    return Image.fromarray((np_img * (1 - mask) + mask *
255.0).astype("uint8")).convert("RGB")

def main(config):
    print("initializing the dataloader")

    model = networks.UNet(
        in_channels=1,
        out_channels=1,
        depth=4,
        conv_num=2,
        wf=6,
        padding=True,
        batch_norm=True,
        up_mode="upsample",
        with_tanh=False,
        sync_bn=True,
        antialiasing=True,
    )

```

```

## load model
checkpoint_path = os.path.join(os.path.dirname(__file__),
"checkpoints/detection/FT_Epoch_latest.pt")
checkpoint = torch.load(checkpoint_path, map_location="cpu")
model.load_state_dict(checkpoint["model_state"])
print("model weights loaded")

if config.GPU >= 0:
    model.to(config.GPU)
model.eval()

## dataloader and transformation
print("directory of testing image: " + config.test_path)
imagelist = os.listdir(config.test_path)
imagelist.sort()
total_iter = 0

P_matrix = {}
save_url = os.path.join(config.output_dir)
mkdir_if_not(save_url)

input_dir = os.path.join(save_url, "input")
output_dir = os.path.join(save_url, "mask")
# blend_output_dir=os.path.join(save_url, 'blend_output')
mkdir_if_not(input_dir)
mkdir_if_not(output_dir)
# mkdir_if_not(blend_output_dir)

idx = 0

results = []

```

```
for image_name in imagelist:
```

```
    idx += 1
```

```
    print("processing", image_name)
```

```
    scratch_file = os.path.join(config.test_path, image_name)
```

```
    if not os.path.isfile(scratch_file):
```

```
        print("Skipping non-file %s" % image_name)
```

```
        continue
```

```
    scratch_image = Image.open(scratch_file).convert("RGB")
```

```
    w, h = scratch_image.size
```

```
    transformed_image_PIL = data_transforms(scratch_image, config.input_size)
```

```
    scratch_image = transformed_image_PIL.convert("L")
```

```
    scratch_image = tv.transforms.ToTensor()(scratch_image)
```

```
    scratch_image = tv.transforms.Normalize([0.5], [0.5])(scratch_image)
```

```
    scratch_image = torch.unsqueeze(scratch_image, 0)
```

```
    _, _, ow, oh = scratch_image.shape
```

```
    scratch_image_scale = scale_tensor(scratch_image)
```

```
    if config.GPU >= 0:
```

```
        scratch_image_scale = scratch_image_scale.to(config.GPU)
```

```
    with torch.no_grad():
```

```
        P = torch.sigmoid(model(scratch_image_scale))
```

```
    P = P.data.cpu()
```

```
    P = F.interpolate(P, [ow, oh], mode="nearest")
```

```
    tv.utils.save_image(
```

```
        (P >= 0.4).float(),
```

```

        os.path.join(
            output_dir,
            image_name[:-4] + ".png",
        ),
        nrow=1,
        padding=0,
        normalize=True,
    )
    transformed_image_PIL.save(os.path.join(input_dir, image_name[:-4] + ".png"))
    gc.collect()
    torch.cuda.empty_cache()

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    # parser.add_argument('--checkpoint_name', type=str, default="FT_Epoch_latest.pt",
    help='Checkpoint Name')

    parser.add_argument("--GPU", type=int, default=0)
    parser.add_argument("--test_path", type=str, default=".")
    parser.add_argument("--output_dir", type=str, default=".")
    parser.add_argument("--input_size", type=str, default="scale_256",
    help="resize_256|full_size|scale_256")
    config = parser.parse_args()

    main(config)

```

! pip install -r requirements.txt

- 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
```

```
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[:,0:W,:] = I1[:,0:3]
I_combine[:,W:,:] = I2[:,0:3]
else:
    I2 = np.asarray(I2.resize((W,H)).convert('L'))
    I_combine = np.zeros((H,W*2))
    I_combine[:,0:W] = I1[:,0:]
    I_combine[:,W:] = I2[:,0:]
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))

```

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

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:
    if( filename != '.ipynb_checkpoints'):
        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))

```

- Try it on your photos

```

from google.colab import files
import shutil
import os

```

```

upload_path = os.path.join(basepath, "test_images", "upload")
upload_output_path = os.path.join(basepath, "upload_output")

```

```

if os.path.isdir(upload_output_path):
    shutil.rmtree(upload_output_path)

```

```

if os.path.isdir(upload_path):
    shutil.rmtree(upload_path)

```

```

os.mkdir(upload_output_path)
os.mkdir(upload_path)

```

```

uploaded = files.upload()
for filename in uploaded.keys():
    shutil.move(os.path.join(basepath, filename), os.path.join(upload_path, filename))

```

```

!python run.py --input_folder /content/photo_restoration/test_images/upload --
output_folder /content/photo_restoration/upload_output --GPU 0

```

```

filenames_upload = os.listdir(os.path.join(upload_path))
filenames_upload.sort()

```

```

filenames_upload_output = os.listdir(os.path.join(upload_output_path, "final_output"))
filenames_upload_output.sort()

```

```

for filename, filename_output in zip(filenames_upload, filenames_upload_output):
    image_original = PIL.Image.open(os.path.join(upload_path, filename))
    image_restore = PIL.Image.open(os.path.join(upload_output_path, "final_output", filename_output))

```

Download the restored images

```
output_folder = os.path.join(upload_output_path, "final_output")
print(output_folder)
os.system(f"zip -r -j download.zip {output_folder}/*")
files.download("download.zip")
```