Jewelry Image Enhancement via Real-ESRGAN Baseline and CNN Refinement

Tejas Thakare
tejas.c.thakare@gmail.com
GitHub: github.com/TejasCThakare/jewelry-enhancement

November 9, 2025

Contents

1 Introduction

This report is for explainability of enhancement in images of jewellery and the methods used for it. The key observation of mine was that there are some low-quality images of jewellery on the internet, but majorly they are for object detection or classification tasks. Whereas what I wanted was a proper jewellery-dedicated dataset where we can see that this image is meant just for jewellery showcase or as a commercial jewellery shop.

Keeping this in mind, in this study I used the Tanishq Jewellery Dataset from Kaggle (https://www.kaggle.com/datasets/sapnilpatel/tanishq-jewellery-dataset) containing 490 images, out of which 301 are of necklaces and 189 are of rings. These images are 262×262 and 8-bit.

In this study, I have implemented the degradation module to simulate real-world artifacts in jewellery images like blur, noise, compression, etc. I have used Real-ESRGAN as a baseline (pretrained models/weights), and on top of it, I have architected my own approach. The reason why I architected my own approach despite very good results with Real-ESRGAN is that I wanted to explore and experiment with my thoughts and intuitions.

2 Methodology

2.1 System Architecture

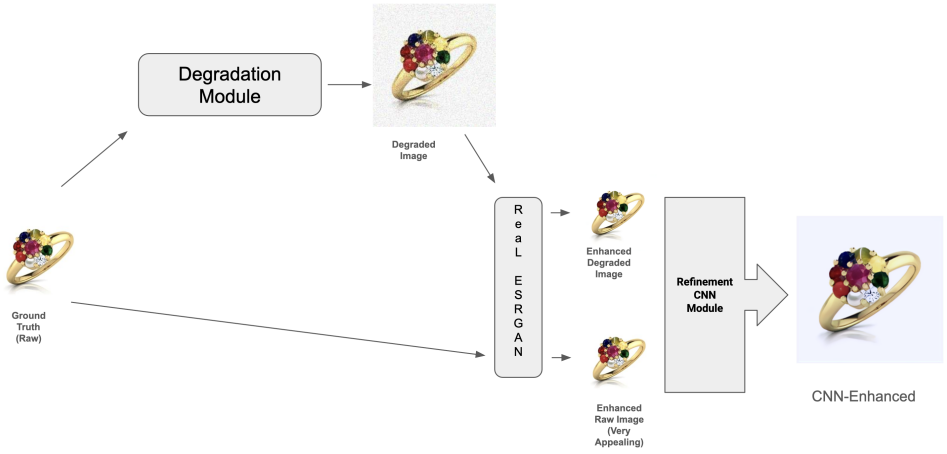

Figure 1: The raw image/ground truth is given to the degradation module, which simulates blur, noise, compression, etc., and then it is passed to Real-ESRGAN to get ESRGAN(degraded). Similarly, the raw image is also passed through Real-ESRGAN to get ESRGAN(raw). Then both enhanced images are passed on to train the RefinementCNN module to get the final CNN-enhanced output.(Though ESRGAN(raw) is the most visually appealing but still the CNN-Enhanced is better than the degraded image)

The pipeline consists of five stages:

1. **Data acquisition**: I used Kaggle Tanisque jwellery dataset. The pipeline flow downloaded the dataset using kaggle API and organize it in `data/raw`.
2. **Degradation**: Since the dataset is not low quality, (it is not also High Quality) so to get the low quality dataset i implemented the configurable degradations (blur, noise, compression, color shift, downscale) to synthesize realistic low-quality images and pipeline organizes it in `data/degraded/<level>`.
3. **Real-ESRGAN baseline**: The Real-ESRGAN is a model architecture developed by Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. It is a powerful model to restore real-world blind blurry, compressed, shifted, etc., images. Though it is trained on a synthetic dataset, its applicability is robust, and it is open-source. I used this model as a baseline, and preliminary results directly on raw data show that the enhanced image from this Real-ESRGAN is better than the original itself. I ran Real-ESRGAN on raw and degraded images to obtain input pairs for the next module on top of ESRGAN, which is RefinementCNN. Run Real-ESRGAN on raw and degraded images to obtain ESRGAN(raw) and ESRGAN(degraded).
4. **Pair generation**: This step created the (X, y) for RefinementCNN. It creates supervised pairs as `_input.npy` = ESRGAN(degraded) and `_target.npy` = ESRGAN(raw), both normalized to $[-1, 1]$.
5. **Refinement training**:This is what I implemented on top of Real-ESRGAN. It is U-Net style Encoder-Decoder based architecure, What it does is train CNN models on 512×512 tensors, validate (PSNR/SSIM), checkpoint the best model.

2.2 Degradation Pipeline

A YAML configuration defines level-wise parameter ranges and application probabilities for:

- **Blur**: Gaussian (sigma range) and motion blur (length, angle).
- **Noise**: Gaussian noise (sigma range), salt-and-pepper (configurable).
- **Compression**: JPEG quality ranges.
- **Color shifts**: LAB blue-yellow channel shift (temperature).
- **Downscale**: Implemented downscaling.

A seed can be set for reproducibility.

2.3 Pair Construction

We construct supervised pairs using the same baseline enhancer for both input and target to align scale and statistics. Specifically,

$$I_R = \text{ESRGAN(degraded)}, \qquad I_{HQ} = \text{ESRGAN(raw)}.$$

Both outputs are saved as float32 arrays, normalized to $[-1, 1]$, and written as paired files (`_input.npy`, `_target.npy`) with consistent indices. The dataset is split into train/-val/test (80/10/10) via a fixed-seed splitter to preserve reproducibility and pair integrity.

2.4 Refinement Model

RefinementCNN A lightweight U-Net style residual CNN designed to apply small, precise corrections on top of the ESRGAN baseline. The architecture is:

- **Encoder**: 3 blocks, each with two 3×3 Conv2d layers (padding 1) + BatchNorm2d + ReLU, followed by MaxPool2d(2) when downsampling is enabled.
- **Bottleneck**: two 3×3 Conv2d + BatchNorm2d + ReLU layers.
- **Decoder**: 3 stages, each with bilinear upsampling (factor 2) followed by two 3×3 Conv2d + BatchNorm2d + ReLU layers (no explicit skip connections in this implementation).
- **Output head**: 1×1 Conv2d + Tanh predicting a residual in $[-1, 1]$, scaled by 0.1 and added to the input: $y = x + 0.1 \cdot r$.

2.5 Losses

A weighted sum of pixel losses:

$$\mathcal{L}_{\text{CNN}} = \lambda_1 \left\|\hat{y} - y\right\|_1 \; + \; \lambda_2 \left\|\hat{y} - y\right\|_2^2,$$

implemented as L1Loss + MSELoss with configurable weights.

2.6 Implementation Details

- **Inputs**: CHW tensors, 3×512×512, normalized to $[-1, 1]$ (paired `.npy`).
- **Optimization**: Adam with β=(0.5, 0.999); lr_G=10^{-4}.
- **Validation**: PSNR and SSIM (torchmetrics) on the validation split; best checkpoint selected by PSNR; periodic checkpoints saved; TensorBoard logs for losses/-metrics.

Listing 1: Training loop

```
for epoch in range(num_epochs):
model.train()
for inputs, targets in train_loader:
inputs, targets = inputs.to(dev), targets.to(dev)
outputs = model(inputs)
loss_l1 = l1(outputs, targets)
loss_mse = mse(outputs, targets)
loss = w1 * loss_l1 + w2 * loss_mse
opt_g.zero_grad(); loss.backward(); opt_g.step()
# validation (PSNR/SSIM), checkpoint best
```

3 Results

For this entire pipeline run i used google colab T4 GPU.

Raw/Ground Truth | Enhanced Raw Image through Real-ESRGAN | Degraded Image | Enhanced Degraded image through Real-ESRGAN | CNN-Enhanced (Ours)

Figure 2: Comparison of Raw Ground Truth, Enhanced Raw, Degraded, Degraded-Enhanced, and CNN-Enhanced images.

3.1 Results from Real-ESRGAN and RefinmentCNN

What I observed is that the raw images also have inbuilt defects, like they are not sharp and clean, but when I pass them through the pipeline, Real-ESRGAN produces very impressive results, even better than the original raw image. Also, the degradation module takes raw images and simulates degradation as discussed. This degradation is applied alternatively to each image, but I have set the probability of applying the respective degradation, like x% of the time applying blur or y% of the time applying color shift, etc. (all parameters are configurable). So it gives varying and diverse degraded data for RefinementCNN.

Figure 3: The first image in the row is the raw image, the second is simulated degraded, the third is degraded enhanced via Real-ESRGAN, the fourth is our CNN-Enhanced, and the last one is raw enhanced image via Real-ESRGAN.

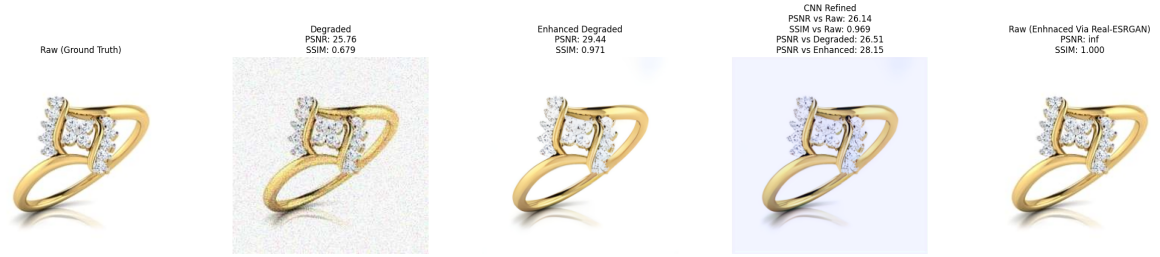

Figure 4: The first image in the row is the raw image, the second is simulated degraded, the third is degraded enhanced via Real-ESRGAN, the fourth is our CNN-Enhanced, and the last one is raw enhanced image via Real-ESRGAN.

Figure 5: The first image in the row is the raw image, the second is simulated degraded, the third is degraded enhanced via Real-ESRGAN, the fourth is our CNN-Enhanced, and the last one is raw enhanced image via Real-ESRGAN.

What I observed again, as discussed, is that the direct Real-ESRGAN output is very visually appealing and enhanced compared to the original itself. However, the results from our RefinementCNN also show promising enhancement, though they are not at the level of Real-ESRGAN. The PSNR value between CNN-Enhanced and ground truth raw image is lower than the PSNR value between degraded enhanced; the reason being, as I observed, if I change the degradation module configured parameters, this PSNR fluctuates—sometimes it increases and sometimes it decreases. However, rerunning training again and again costs time (I trained for 20 epochs, with a batch size of 4, so it takes almost 40–50 seconds for each epoch, though the time again depends on the config parameters setting). So, for some config settings, I observed an increase in PSNR value, for example:

Figure 6: The first image is the raw input, the second image is the enhanced degraded image, the third is our CNN-refined, and the last one is the raw enhanced image via Real-ESRGAN. Here the PSNR and SSIM of our CNN-Enhanced increased by +0.39 and +0.001, respectively, compared to the enhanced degraded image.

3.2 Gradio Demo

This pipeline has Gradio Demo too. Below is the Screenshot of Demo:

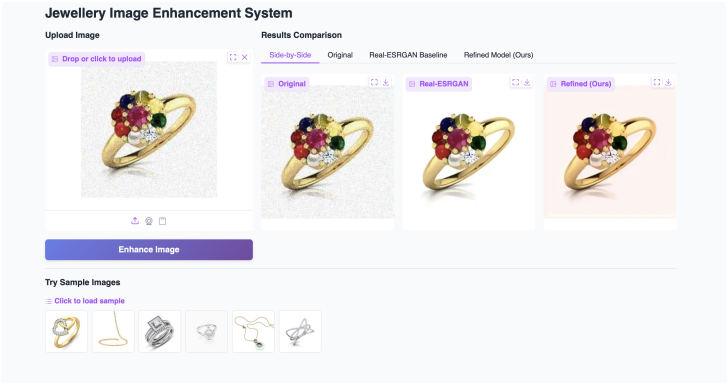

Figure 7: You can upload the image to be enhanced, then we can see the output by Real-ESRGAN and RefinementCNN.

For the demo video, please click here: Demo Video

Note: The background appears reddish most likely because the model is overfitting, or the uploaded image was directly passed to the RefinementCNN instead of first being degraded and then processed by the RefinementCNN. It requires improvement in hyper-parameter tuning and an increase in dataset size. However, when I trained the same model for fewer epochs, the reddishness disappeared, and the results were significantly better!

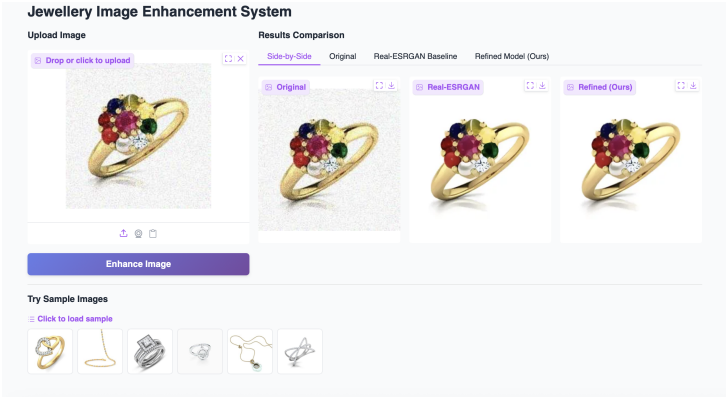

Figure 8: **Note:** The result from RefinementCNN improved as I reduced the number of epochs.

4 Discussion

4.1 Key Findings

Real-ESRGAN Baseline Efficacy On the Tanishq jewelry dataset, the pre-trained Real-ESRGAN model outperforms the original 262 ×262 raw photos in terms of visual quality. Despite being trained on synthetic data rather than jewelry-specific pictures, this confirms Real-ESRGAN's robustness to domain transfer.

Degradation Module Effectiveness The configurable degradation pipeline successfully simulated realistic quality degradations—including Gaussian blur , additive Gaussian noise , and JPEG compression. To avoid overfitting to a particular degradation pattern, a probabilistic application of each form of degradation produced a variety of training pairs..

RefinementCNN Performance In certain configurations, the lightweight U-Net style refinement network demonstrated measurable gains over the degraded-enhanced baseline; for particular degradation parameter values, PSNR increases of up to +0.39 dB and SSIM improvements of +0.001 were noted (Figure 6).

4.2 Limitations & Improvments

To accomplish production-level image improvement, this work needs a lot of processing power, particularly GPUs. Unusually high PSNR values (over 60 dB) on validation during RefinementCNN training indicate overfitting caused by a lack of training data, which results in poor generalization. Although GAN-based refinement was investigated for improved perceptual quality, the tiny dataset size caused instability and hallucinogenic artifacts. Consequently, CNN refining is the main focus of the work. In order to increase GAN improvements and model robustness, future work should focus on expanding datasets, incorporating real-world degradations. Also, end-to-end fine-tuning with Real-ESRGAN, color constancy modules, segmentation-guided refinement for metals/gems, and broader real-world data. To account for other artifacts, we should try other base models or a hybrid approach.

5 Code Structure

Listing 2: High-level directory structure

```
jewelry-enhancement/
          config/           # YAML configuration files
          data/             # Dataset storage (raw, degraded,
   enhanced, training)
          demo/             # Gradio interactive demo
          docs/             # Documentation and reports
          results/          # Experiment outputs (metrics, logs,
   visualizations)
          scripts/          # End-to-end automation scripts
          src/              # Core source modules
          tests/            # Unit and integration tests
          requirements.txt  # Python dependencies
          setup.py          # Package installation script
```

The project is organized into a modular directory structure. The main components include:

- **config/:** Contains YAML configuration files defining parameters for degradation simulation, model architecture, and training settings:
  - `degradation_config.yaml`
  - `model_config.yaml`
  - `training_config.yaml`
- **data/:** Structured dataset folders for different pipeline stages and splits:
  - `raw/`: Original Tanishq jewelry images.
  - `degraded/`: Synthetic low-quality images organized into severity levels:
    - `level1_mild` (used for training in this study)
    - `level2_moderate` (found too unnatural; not used)
    - `level3_severe` (found too harsh; not used)
  - `enhanced/`: Real-ESRGAN outputs of raw and degraded images.
  - `training/`: Split into `train`, `val`, and `test` subsets containing paired training tensors.
- **src/:** Core source code modules categorized by functionality:
  - `degradation/`: Implements image degradations like blur, noise, compression, and color shifts.
  - `enhancement/`: Wraps the Real-ESRGAN inference pipeline and handles preprocessing/postprocessing. (this pre/post processing i have removed)
  - `training/`: Contains model definitions for RefinementCNN and GAN, dataset splitting, loss functions, and training routines (GAN training was explored but focused refinement used CNN due to GAN instability).

  - `evaluation/`: Metrics calculation including PSNR, SSIM, and LPIPS.
  - `utils/`: Helper utilities for image I/O and visualization.
- **scripts/:** Automation scripts enabling sequential execution of pipeline stages such as dataset download, degradation creation, enhancement, pair generation, training, evaluation, and launching the interactive Gradio demo.
- **demo/:** Contains the Gradio web app `app.py` for showcase and real-time enhancement with side-by-side comparisons.
- **tests/:** Unit and integration tests verifying correctness of degradation modules, model interfaces, metric computations, and training process.
- **results/:** Contains outputs generated by runs:
  - `comparisons`: Visual comparison images.
  - `metrics`: Quantitative evaluation files.
  - `logs`: TensorBoard logs.
- **Other files:** `requirements.txt` listing Python dependencies for package installation.

References

[1] Xintao Wang, Liangbin Xie, Chao Dong, Ying Shan (2021). *Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data*. arXiv:2107.10833.

[2] Xintao Wang, et al. (2018). *ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks*. ECCVW.

[3] Isola, Zhu, Zhou, Efros (2017). *Image-to-Image Translation with Conditional Adversarial Networks*. CVPR.

[4] Simonyan and Zisserman (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556.