Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

FIIT-100241-116700

Artur Kozubov

Photo Restoration using Artificial Intelligence

Bachelor's Thesis

Thesis Supervisor: doc. Ing. Giang Nguyen Thu, PhD.

May 2025

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

FIIT-100241-116700

Artur Kozubov

Photo Restoration using Artificial Intelligence

Bachelor's Thesis

Study Programme:	Informatics
Study Field:	Computer Science
Training Workplace:	Bratislava, Slovakia
Thesis Supervisor:	doc. Ing. Giang Nguyen Thu, PhD.

May 2025

I declare in my honor that I have prepared this work independently, on the basis of consultations and using the mentioned literature.

In Bratislava, May 2025 Artur Kozubov

Annotation

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Study Programme: Informatics
Study Field: Computer Science

Author: Artur Kozubov
Master's Thesis: Photo Restoration using Artificial Intelligence
Thesis Supervisor: doc. Ing. Giang Nguyen Thu, PhD.
May 2025

Photo restoration, as a computer vision field, has always required intensive manual labor in traditional restoration tasks, never satisfying scalability in modern applications. In recent years, with the rapid development of AI, especially the invention of deep learning methods such as CNNs and GANs, the quality of the whole restoration process has gone to a new level. This project aims to devise with solution in photo restoration, by developing a system that utilizes the advantages of AI models to enhance images and remove visible defects, such as noise, blur, and compression artifacts. And also presents dockerized solution with API and web-based application to demonstrate the capabilities of the model.

Keywords: Photo Restoration, Deep Learning, Neural Networks, Image defects

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: Informatika
Študijný odbor: Informatika

Autor: Artur Kozubov
Bakalárska práca: Photo Restoration using Artificial Intelligence
Vedúci diplomového projektu: doc. Ing. Giang Nguyen Thu, PhD.
Máj 2025

Reštaurovanie fotografií, ako oblasť počítačového videnia, si vždy vyžadovalo intenzívnu manuálnu prácu pri tradičných úlohách reštaurovania a nikdy nespĺňalo požiadavky na škálovateľnosť v moderných aplikáciách. V posledných rokoch sa s rýchlym rozvojom umelej inteligencie, najmä s vynálezom metód hlbokého učenia, ako sú CNN a GAN, kvalita celého procesu reštaurovania posunula na novú úroveň. Cieľom tohto projektu je navrhnúť riešenie v oblasti obnovy fotografií, a to prostredníctvom vývoja systému, ktorý využíva výhody modelov umelej inteligencie na vylepšenie obrázkov a odstránenie viditeľných chýb, ako sú šum, rozmazanie a kompresné artefakty. A tiež predstavuje dockerizované riešenie s API a webovou aplikáciou na demonštráciu schopností modelu.

Oblasť problematiky: Obnova fotografií, Hlboké učenie, Neurónové siete, Defekty obrazu

Slovak University of Technology in Bratislava
Institute of Informatics, Information Systems and Software Engineering

Faculty of Informatics and Information Technologies
Academic year: 2024/2025
Reg. No.: FIIT-100241-116700

BACHELOR THESIS TOPIC

Student: **Artur Kozubov**
Student's ID: 116700
Study programme: Informatics
Study field: Computer Science
Thesis supervisor: doc. Ing. Giang Nguyen Thu, PhD.
Head of department: doc. Ing. Ján Lang, PhD.

Topic: **Photo Restoration using Artificial Intelligence**

Language of thesis: English

Specification of Assignment:

Reštaurovanie obrazov je úloha počítačového videnia, ktorá zahŕňa vrátenie poškodených obrazov do pôvodného stavu. Umelá inteligencia, konkrétne neurónové siete v poslednom čase priniesli revolúciu v tejto oblasti tým, že poskytujú moderné výsledky pri rôznych úlohách obnovy, napríklad denoizáciu obrazu založenú na hlbokom učení alebo vysokom rozlíšení. Tieto metódy majú schopnosť, ktorá sa dá využiť v mnohých aplikáciách s tým, že sú vytrénované na súbore údajov alebo bez nich. Analyzujte súčasný stav reštaurovania obrazov (angl. photo restoration) smerom k praktickému používaniu a v kontexte umelej inteligencie. Navrhnite a implementujte základný modul ako jadro inteligentnej aplikácie na spracovanie a obnovenie obrazov. Ako praktickú ukážku vývoja softvéru, vytvorte docker kontajner pre Vašu aplikáciu. Zvoľte vhodné metriky na meranie výsledkov a ich vyhodnoťte.

Length of thesis: 40

Deadline for submission of Bachelor thesis: 12. 05. 2025
Approval of assignment of Bachelor thesis: 15. 04. 2025
Assignment of Bachelor thesis approved by: doc. Ing. Ján Lang, PhD. – Study programme supervisor

Contents

Chapter 1

Introduction

Photo restoration has always been a difficult task that demanded considerable efforts from experts to restore damaged or lost image details. These days, after neural networks became commonly used with deep learning techniques [1, 2, 3, 4], new horizons have opened for this area, and a great advance has been achieved. Modern deep learning-based approaches [5, 6] can be fully automated, and they therefore present high-quality results and huge savings on processing time.

Today, such approaches are already in use or are planned to be deployed in many areas where the demands on image reconstruction accuracy and speed are high.

Medical imaging is one of the areas where AI-based image restoration methods are applied to clean medical images, allowing for better diagnosis and treatment of patients [7, 8].

For space and aerial images, superior algorithms result in sharper and more detailed views of the Earth's surface for mapping, environmental monitoring, and even military applications [9].

Image restoration technologies in the forensic and security areas help improve facial recognition [10, 11] and evidence analysis, increasing the efficiency of investigations.

Enhanced AI will be able to be used to create more powerful visual systems for autonomous vehicles [12, 13, 14], which will provide more reliable recognition of objects [15] and traffic conditions [16].

In this manner, the development of artificial intelligence-based photo restoration methods has solved not only the problem of image restoration but also opened wide perspectives.

Chapter 2

Overview of Image Restoration

Before proceeding further, we need to clarify what is an exactly an image restoration. Photo restoration simply aims to remove all possible defects to enhance the image appearance and quality, primarily for human perspective.

Defects, in fact, might be of various types, such as noise, blur, compression artifacts, color distortion, etc. However, each of these defects can differ in type, including: Gaussian, Poisson, Masked, High-frequency, Impulse, Quantization noise, and many many more. Furthermore, images may combine these defects, making the problem even more challenging. As the result, we face the problem of solving an equation with thousands of unknown variables and multiple constraints.

But, in contrast to these vague facts, most of these defects does not appear in real-world life. Therefore, following the studies [17],research [18, 19] and statistics, we can identify the most common defects and focus on them. Moreover, given the significant challenges faced by current deep learning models [20] in correcting color distortions outside the highly specialized image domain, we also exclude this category from further consideration. Then, we can summarize those defects as follows:

- Noise
  - Gaussian Noise, Poisson Noise, Speckle Noise, Salt-and-Pepper Noise
- Blur
- Compression Artifacts
- Low Resolution

Technically speaking, photo restoration represents a collection of mathematical and computational methods to cope with issues like noise reduction, artifact removal, color correction, enhancement of resolution, and other defects. The basic problem can be modeled as an inverse problem, recovering the original image HQ (High-Quality) from its degraded version, LQ (Low-Quality):

$$LQ = H(HQ) + N \tag{2.1}$$

where:

- LQ is the degraded image,
- H is the degradation operator (which can include blurring, compression artifacts, etc.),
- N represents noise or other forms of corruption.

The restoration process aims to estimate HQ given LQ, often by applying techniques that can inverse or somehow reverse the effects of H and N.

2.1 Traditional Digital Image Restoration Methods

In the past, before applying neural networks, the most common methods for image restoration were based on traditional signal processing techniques, which can be broadly summarized under two major methods: **spatial domain** and **frequency domain** methods.

2.1.1 Spatial Domain Methods

Spatial domain techniques work directly on pixels. The given image can be visualized as a 2D matrix of pixel values, whereby operations have to be performed directly on the values.

Filtering Noise and other unwanted artifacts could be removed using linear or non-linear filters. A very common example is the Gaussian filter [21], which, though smoothing the image, reduces high-frequency noise through simple averaging of pixel values with their neighbors.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

where $G(x, y)$ is the Gaussian kernel, and σ controls the amount of smoothing [21].

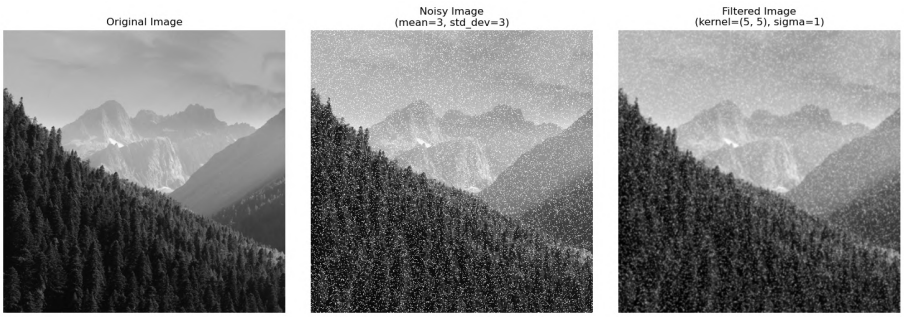

Figure 2.1: Applying a Gaussian filter to an image.

As seen in Figure 2.1 above, the Gaussian filter effectively removes high-frequency noise but unfortunately also significantly blurs the image, which is not always desirable and causes a loss of important and many details of the image.

Interpolation In other words, interpolation using the estimation of unknown pixel values in an image based on the values of surrounding pixels is essential in image restoration tasks, such as resizing, repairing damaged areas, and filling in missing or corrupted pixel data.

Bilinear Interpolation Bilinear interpolation [21] uses the four closest pixel values to calculate a weighted average. The formula for bilinear interpolation is:

$$I(x, y) = (1 - u)(1 - v)I_{00} + u(1 - v)I_{10} + (1 - u)vI_{01} + uvI_{11} \tag{2.3}$$

where $I(x, y)$ is the interpolated pixel value, $I_{00}, I_{10}, I_{01}, I_{11}$ are the intensities of the surrounding pixels and $u = x - \lfloor x \rfloor$, $v = y - \lfloor y \rfloor$ are the fractional parts of the coordinates.

Bicubic Interpolation Bicubic interpolation [22] uses 16 neighboring pixels and uses cubic polynomials for smoother results. The interpolated value is given by:

$$I(x, y) = \sum_{i=-1}^{2} \sum_{j=-1}^{2} w(i, u)w(j, v)I_{i,j} \tag{2.4}$$

where $w(k, t)$ is a cubic weight function, often defined as [22]:

$$w(k,t) = \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1, & |t| \leq 1 \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a, & 1 < |t| \leq 2 \\ 0, & |t| > 2 \end{cases} \tag{2.5}$$

where a is a constant, typically -0.5 [22].

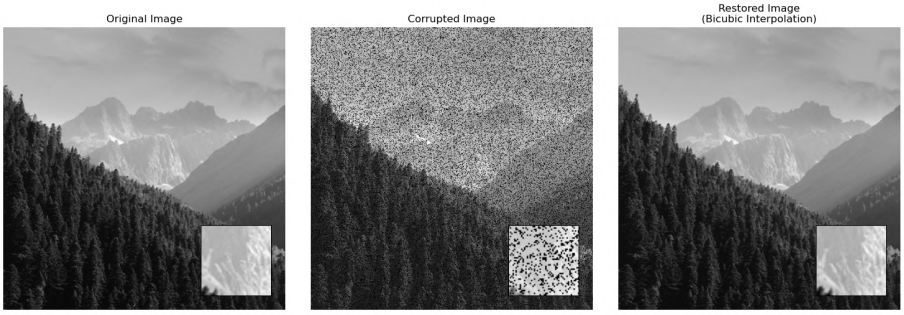

Figure 2.2: Applying a Bicubic interpolation to an image with missing pixels.

As seen in Figure 2.2 above, bicubic interpolation provides a smooth and more aesthetically pleasing result, but in some cases, it may introduce artifacts or distortions or lead to a loss of image details.

2.1.2 Frequency Domain Methods

Frequency domain methods [21] transfer the image into the frequency space by making use of tools such as the Fourier transform. These methods are very good at bringing out and then manipulating specific frequency components corresponding to various noises and/or artifacts.

Fourier Transform Fourier transform [23] selectively perform attenuation or amplification on some frequencies, firstly transform the image from the spatial to the frequency domain.

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \tag{2.6}$$

where $F(u,v)$ is the Fourier transform of the image $f(x,y)$, and M and N are the dimensions of the image [23].

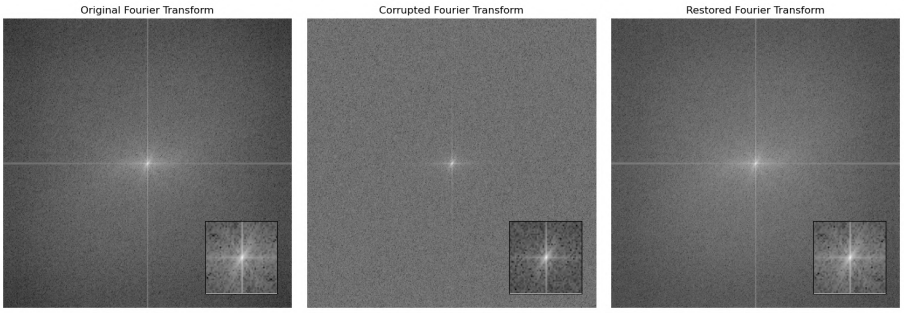

Figure 2.3: Using Fourier transform to convert an image to frequency domain.

As seen in Figure 2.3 above, the Fourier transform helps to visualize the frequency components of the image, which can be selectively manipulated to remove noise or other artifacts, but not as effective as bilinear or bicubic interpolation.

Wiener Filtering The statistical approach is to minimize the mean squared error between the estimated and true images wiener1949extrapolation, which shows a trade between the balance of noise suppression and the preservation of the signal,

$$H_{\text{Wiener}}(u,v) = \frac{|H(u,v)|^2}{|H(u,v)|^2 + \frac{S_N(u,v)}{S_I(u,v)}} \tag{2.7}$$

where $H(u,v)$ is the frequency domain degradation function, $S_N(u,v)$ denotes the power spectral density of noise, and $S_I(u,v)$ shows the power spectral density of the original image [24].

2.2 Neural Networks for Image Restoration

While this introduction of machine learning [25], and more precisely deep learning [26], has been able to enable restorers to learn complex patterns [27, 28] from large datasets in much better quality and enhanced effectiveness relative to conventional approaches.

2.2.1 Autoencoders

Autoencoders [29] are a type of neural network that can learn to compress data and then reconstruct it. Autoencoder includes two parts: an encoder E that maps the input x to a latent space representation z and a decoder D that reconstructs $\hat{x}$ from z:

$$z = E(x), \quad \hat{x} = D(z) \tag{2.8}$$

The loss function used for reconstruction is commonly the Mean Squared Error (MSE) [30]:

$$\mathcal{L}_{\text{reconstruction}} = \frac{1}{n} \sum_{i=1}^{n} \|x_i - \hat{x}_i\|^2 \tag{2.9}$$

They are excellent starting points for image restoration tasks, as they can learn about a wide range of image defects and can be used for various tasks. However, they are limited in terms of the quality of the restored images, as they tend to produce blurry results.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [31] is a more advanced type of neural network that is widely used for spatially aware tasks in image restoration, leveraging convolutional layers to extract hierarchical features. This convolution operation may be described by:

$$y[i, j] = \sum_{m} \sum_{n} x[i + m, j + n] \cdot k[m, n] \tag{2.10}$$

where x is the input, k is the kernel, and y is the output [31]. CNNs excel in tasks such as super-resolution [32], where the network learns to map low-resolution inputs to high-resolution outputs [33], but can also be used to denoise, de-blur and other restoration tasks.

2.2.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [34] is a type of neural network that consists of two models: a generator G that generates samples, and a discriminator D that distinguishes between real and generated samples. The objective function of GANs is:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))] \tag{2.11}$$

GANs are highly effective for image restoration tasks, as the generator learns to create visually plausible images from corrupted input, while the discriminator ensures

realism.

2.2.4 Transformers for Image Restoration

Transformers [35], that were originally developed for natural language processing, have been adapted for image restoration due to their ability to model long-range dependencies, rendering them appropriate for tasks such as inpainting and super-resolution [36]. The attention mechanism on the transformers allows the model to focus on the relevant parts of the image, allowing for high-quality restoration. And this core mechanism is the self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{2.12}$$

where Q, K, and V are the query, key, and value matrices, respectively. Transformers excel in handling global context, making them effective for high-level restoration tasks [35].

2.2.5 Complex and Combined Networks

Different architectures have been combined and proved to be effective for image restoration. For example, auto-encoders can serve as pre-processors for GAN inputs [37], while CNNs may work as feature extractors for transformers. The joint loss function may look like the sum of reconstruction, adversarial, and perceptual terms:

$$\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{reconstruction}} + \lambda_2 \mathcal{L}_{\text{adversarial}} + \lambda_3 \mathcal{L}_{\text{perceptual}} \tag{2.13}$$

where λ_1, λ_2, and λ_3 control the weight of each term.

These advanced architectures and their combinations have enabled recognizable progress in the field of image restoration, making it possible to recover details even from severely degraded images [38].

2.3 Related Work

The field of image restoration has seen significant advancements in recent years, with a focus on developing deep learning models that can effectively and qualitatively restore images. Here are some notable works that have contributed to the state-of-the-art in image restoration:

2.3.1 Flexible Blind Convolutional NN (FBCNN)

Flexible Blind Convolutional Neural Network (FBCNN) [39] represents a convolutional neural network for the removal of blind JPEG artifacts. It predicts variable quality factors for balancing the elimination of artifacts and the retention of details. This network comprises a decoupler module aimed at decomposition of the input JPEG image to extract its quality factor, and a reconstructor module that uses this factor for restoration.

Unfortunately, FBCNN is specialized for JPEG artifacts removal and may not be suitable for other types of image degradations. Also, the model's performance may vary depending on the quality of the input image, making it less robust in real-world scenarios.

2.3.2 Hybrid CNN-Transformer for LDCT Image Denoising

Hybrid CNN-Transformer for LDCT Image Denoising (HCformer) [40] is a hybrid Transformer-CNN codec network for low-dose CT (LDCT) image denoising that aims to reduce the radiation dose without compromising on diagnosis detail. HCformer introduces neighborhood pixel context through a Neighborhood Feature Enhancement (NEF) module into transformer self-attention modules. To mitigate the prohibitively high global attention cost, HCformer alternatively switches between Windows Multi-head Self-Attention (W-MSA) and Shifted Window MSA (SW-MSA) across transformer layers with lower computational overhead supporting cross-window information sharing.

Global transformer, that integrated into modeling with local CNN priors, achieves better noise removal and detail preservation than other state-of-the-art CNN-only denoisers. Even with applied windowing, it still require more FLOPs and memory than straightforward CNN architectures, which may hinder real-time deployment on edge devices or low-resource scanners.

2.3.3 Deep Image Prior

Deep Image Prior (DIP) [41] considers a convolutional neural network for image restoration tasks without the use of training data. The basic idea of DIP is that the structure of the network itself acts as a prior inherent in the structure of convolutional neural networks (2.2.2), thus enabling them to learn the underlying image distribution with a single corrupted input image and concentrate on minimizing some form of reconstruction loss, for example, Mean Squared Error (2.2.1), in between the output and the corrupted image. In other words, by optimizing the weights of the network to reduce

the reconstruction error, DIP is effective in the denoising, deblurring, and inpainting of images when no ground-truth data are available.

This unsupervised strategy has opened new avenues for image restoration tasks where labeled training data is not available or the collection of labels is not possible, such as medical imaging or satellite imagery.

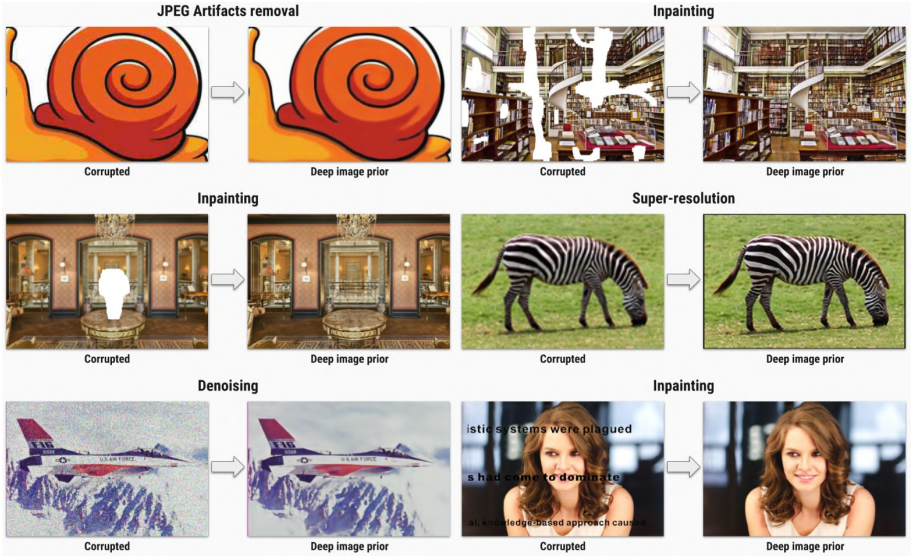

Figure 2.4: Example of image restoration using Deep Image Prior.

[41]

Based on these, the DIP could be considered as a very flexible and powerful approach to image restoration, especially - image inpainting, as in Space & Aerial Imaging [42], by showcasing great performance and semantic metrics.

On the other hand, DIP has several limitations in addition to all the benefits. For example, DIP cannot deal with complex image degradation and produce good quality results. Also, the optimization procedure may be very time-consuming and computationally expensive; therefore, in several real-time applications, it may not be practical.

2.3.4 ESRGAN and Real-ESRGAN

Enhanced Super-Resolution GAN (ESRGAN) [43] is a well-known model for image super-resolution, which aims to enhance the resolution of low-quality images. It uses a residual-in-residual dense block (RRDB) architecture, which allows for better feature extraction and reconstruction. The model is trained using a combination of percep-

tual loss and adversarial loss, enabling it to generate high-quality images with fine details.

Currently, ESRGAN is widely used for image super-resolution tasks, but is not specifically designed for other restoration tasks such as denoising or deblurring. This limitation of applicability specialization in scenarios with multiple types of degradation is trying to be solved by new Real-ESRGAN [44] model, which also incorporates a noise-aware, blur and jpeg compression module to handle various types of artifacts.

Its better version - Real-ESRGAN [44] that is focused on restoring real-world images, which are often more complex and diverse than the synthetic datasets used for training by containing various types of artifacts, such as noise, blur, and compression artifacts.

Figure 2.5: Example of image restoration using Real-ESRGAN.

[44]

Based on those, ESRGAN and Real-ESRGAN are very effective for super-resolution tasks, for example in the entertainment and gaming industries, by up-scaling textures [45], animations, and other visual content with employing high-quality details in resulting images.

2.3.5 Blind Image Restoration with Instant Gen Reference

Blind Image Restoration with Instant Gen Reference (InstantIR) [46] is a new single image restoration model designed to restore degraded images using a diffusion-based approach with instant generative references. It has a pre-trained vision encoder to get image features into one representation, capturing structure and semantic information. In a processing pipeline, a Previewer module generates restoration previews at each

step, serving as instant generative references. These references guide the Aggregator in refining the image, which excellently addresses unknown degradations faced during inference.

Figure 2.6: Example of image restoration using InstantIR.

[46]

Approaching diffusion, which involves iterative processing, computations of this model are intensive. Also, the model is not capable of handling multiple restoration tasks simultaneously, which limits its flexibility. But might be used in security and automotive industries, by blind inpainting of partial faces. license plates or enchanting low-light and adverse weather, for improving downstream object detection reliability.

2.4 Summary

Image restoration is a traditional and well-known problem that has been solved and is still being solved by many different methods, approaches, and ideas, including traditional methods of filtering and interpolation and advanced deep learning models. During the time when those approaches have been developed and improved, and still are, by new proposals and ideas such as the latest InstantIR model (2.3.5).

The models and methods proposed so far excel in many restoration tasks, though

some others suffer from one problem or another. Therefore, developing efficient, high-performance, all-in-one models to cope with several restoration tasks for combating various kinds of image degradations remains one major tendency of image restoration [36, 47, 39].

There is also room for such improvements to develop increasingly efficient and effective models of image restoration tasks.

Among the recent advances in image restoration, in the models of different type and range, the most noticeable gaps might be summarized to:

- **Task Specialization** Many existing models and methods (2.1,2.3.1) are optimized for specific restoration tasks and lack the flexibility to handle multiple types of image defects simultaneously.
- **Computational Efficiency** Complex models that address multiple defects tend to be computationally intensive (2.3.3), limiting their practicality for real-time or resource-constrained applications.
- **Data Dependency** Advanced models often require large paired data sets for training, such as diffusion data sets (2.6), which may not be readily available for all types of image degradation.
- **Scalability** Existing models may struggle to scale effectively to high-resolution images or adapt to diverse and unpredictable degradation patterns encountered in real-world scenarios [48].

Chapter 3

Objectives and Methodology

3.1 Objectives

The project targets to train a model with developing graphical user interface to be accessible for external users. Following the summary section (2.4) the goals of this thesis are focused on selected objectives to try to solve the following challenges:

Create a Dataset Workflow

- Build a strong workflow capable of loading images from different sources, such as online repositories, local directories, and cloud storage.
- Use mapping and preprocessing steps like augmentation to prepare the data for training.
- Handle image defects by applying different types of distortion 2 and be able to control the power of distortion for gradually increasing the difficulty of the task duration the training process.

Identify a Solution for Enhancing Image Quality and Train the Model

- Attempt to design — or leverage an existing - architecture that will be able to handle common image defects encountered in real-world scenarios.
- Focus primarily on image up-scaling while also incorporating noise reduction, blur removal, and artifact correction, such as in jpeg compressions.
- Train the model on the prepared dataset and evaluate its performance and results with metrics, including PSNR, SSIM, and visual inspection.

Pack the Solution with API and Web-based Application

- Pack solution to be run on multiple devices using Docker [49] containers technology.
- Develop the light-weight API to host the model and offer a simple-to-use interface for interacting with the system.
- Develop a web-based application to showcase the functionality of the model.

3.2 Methodology

The following steps have been chosen for the design and development of the proposed solution:

- Step 1. Create a dataset workflow to load, preprocess, and augment images from diverse sources.
- Step 2. Identify an architecture for enhancing image quality while maintaining a wide range of image defects.
- Step 3. Train or Finetune the model on the prepared dataset and evaluate its performance and results with various metrics, including PSNR [50], SSIM [51], and visual inspection.
- Step 4. Create a lightweight API to serve the model and provide a user-accessible interface for interacting with the system. Including a web-based application to demonstrate the capabilities of the system.
- Step 5. Pack the solution using Docker containers technology.

3.3 Evaluation metrics

The following metrics will be used to evaluate the model: Peak signal-to-noise ratio (PSNR), structural similarity index measure (SSIM), learned perceptual image patch similarity (LPIPS) and inference time, as well as Blind Image Quality Assessment (BIQA) metrics.

3.3.1 Peak Signal-to-Noise Ratio

Peak Signal-to-Noise Ratio (PSNR) [50] was one of the existing objective metrics for measuring the quality of reconstructed images in relation to the originals. The PSNR has been defined as the ratio between the maximum possible power of a signal and the power of corrupting noise influencing its representation fidelity.

$$\text{PSNR} = 20 \cdot \log_{10} \left(\frac{\text{MAX}_I}{\sqrt{\text{MSE}}} \right) \tag{3.1}$$

where:

- MAX_I is the maximum possible pixel value of the image (e.g., 255 for an 8-bit image).
- MSE is the Mean Squared Error between the restored image I_{restored} and the ground truth image I_{original}:

$$\text{MSE} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(I_{\text{original}}(i,j) - I_{\text{restored}}(i,j) \right)^2$$

PSNR gives the reconstruction accuracy in quantitative terms. Its larger value accounts for better fidelity to the original. However, PSNR does not sometimes show good correspondence with perceived visual quality sometimes; hence, it is complemented by other metrics.

3.3.2 Structural Similarity Index Measure

Structural Similarity Index Measure (SSIM) [51] has been designed to outperform traditional metrics such as PSNR. It is sensitive to changes with respect to structural information, luminance, and contrast.

The SSIM between two images x and y is calculated as:

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.2}$$

where:

- μ_x and μ_y are the mean luminance of x and y, respectively.
- σ_x^2 and σ_y^2 are the variance of x and y.
- σ_{xy} is the covariance of x and y.
- C_1 and C_2 are stabilization constants to avoid division by zero.

Actually, SSIM was proposed to measure the similarity between two given images based on these factors important for perception. Second, SSIM is in concert with human perception of vision, since it puts more weight on structural information. This metric

provides a more precise visual quality assessment than PSNR. So, SSIM is the metric relevant for the image restoration performance test.

3.3.3 Blind Image Quality Assessment

In fact, it's assessment metrics for any given image without the availability of a reference or ground truth image. Indeed, the underlying approaches reviewed in this paper are of particular importance when the original is not available, for instance, in real-world applications where images may suffer from a host of unknown degradations.

Common Blind Image Quality Assessment (BIQA) metrics:

- **NIQE (Naturalness Image Quality Evaluator)** [52]: Measures the deviation of image statistics from natural scene statistics.
- **BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator)** [53]: Assesses image quality based on spatial statistics of the natural environment.

BIQA can solve the image quality assessment problem with more flexibility and practically, especially in real-world applications where the availability of reference images is not possible.

Chapter 4

AI Image Restoration Design

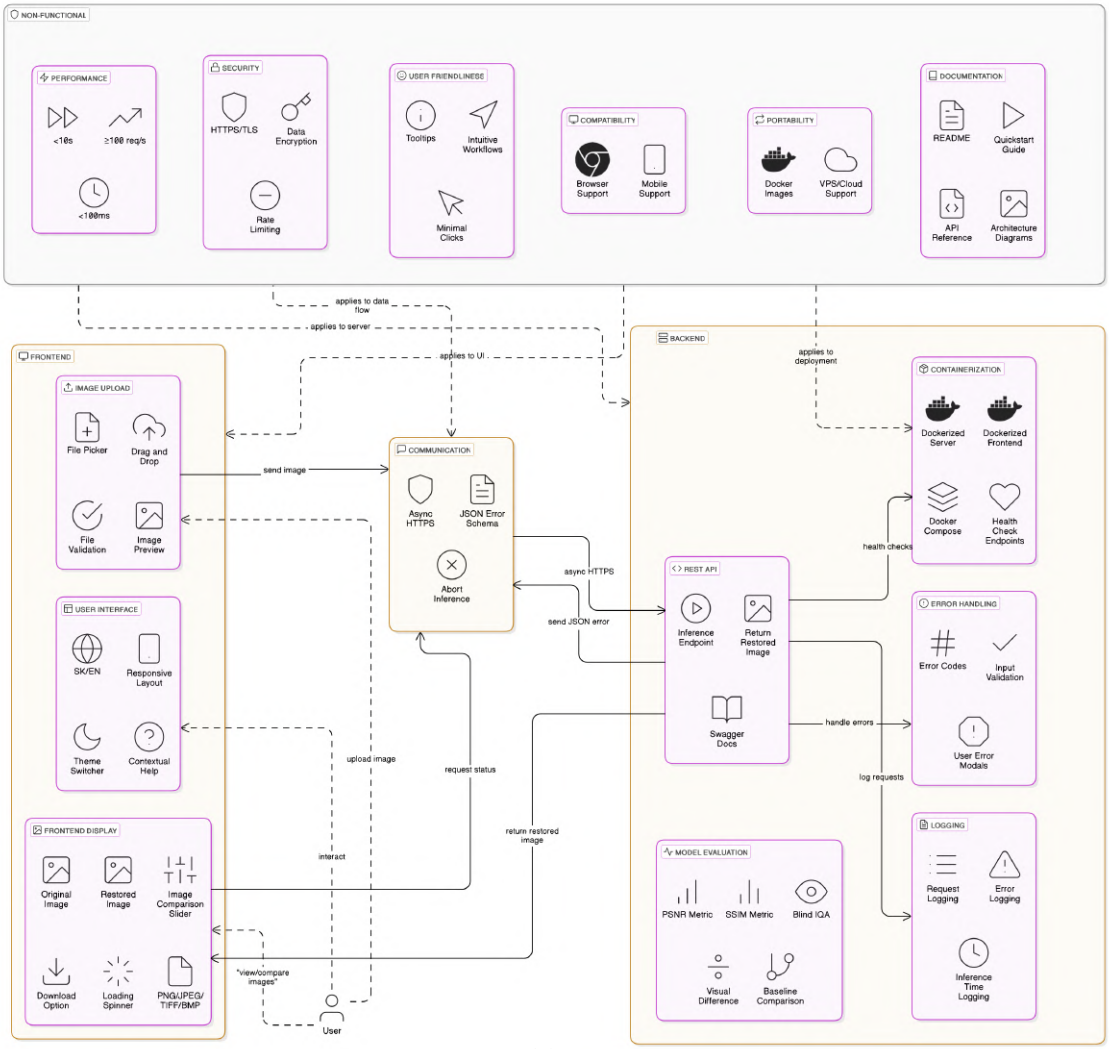

Figure 4.1: Expectations diagram

Expectations When designing a complex full-stack image restoration solution, we need to understand what the goal is and what it should do. For this purpose, we have come up with the following expectations, which can be seen in the Figure 4.1 below. All of these elements cover all parts of our designed application.

4.1 System Architecture Overview

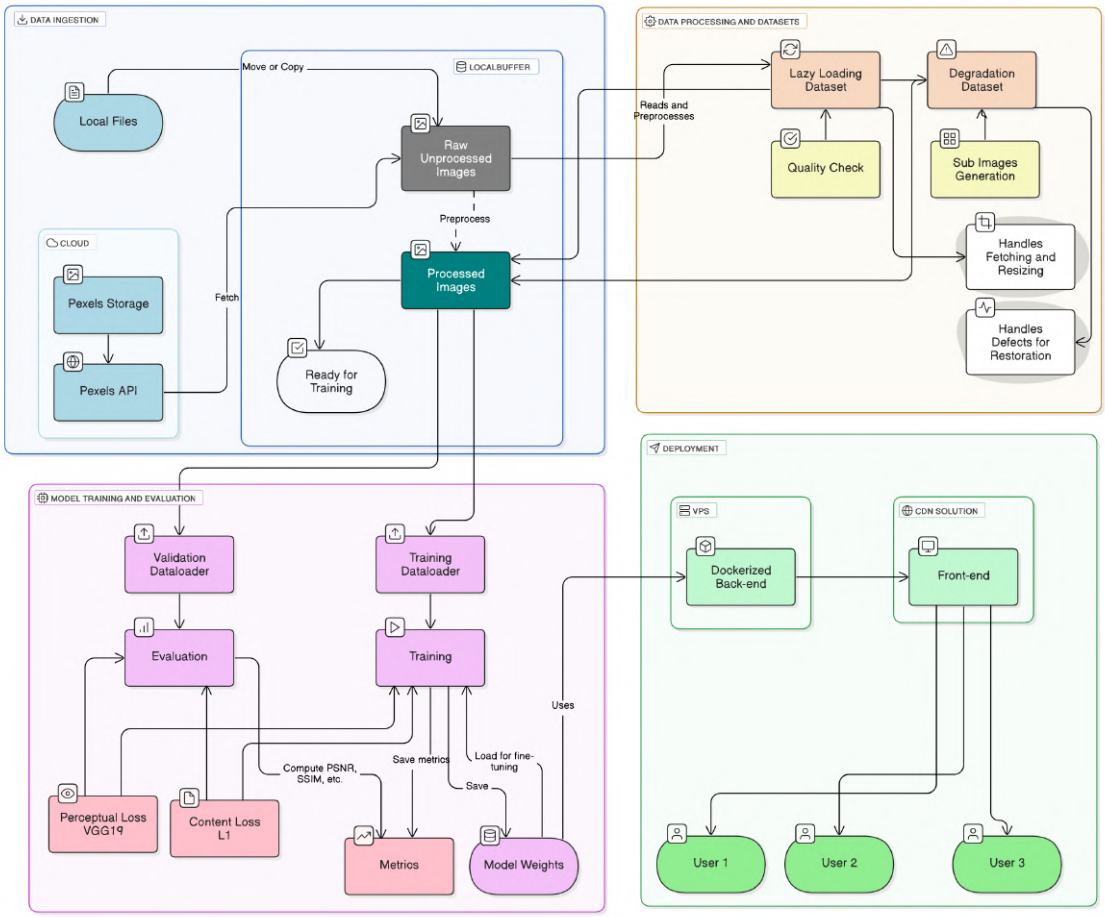

Figure 4.2: Architecture of the whole designed workflow of application

Based on the Expectations (4) and Objectives (3) outlined in the previous sections, we have come up with next comprehensive architecture for the photo restoration application, as illustrated on the Figure (4.2).

The proposed architecture includes several stages, phases, that are divided between

main parts of application. Stages are divided accordion to life cycle of the image restoration process, starting from data ingestion and preprocessing, through model training and evaluation, to deployment and serving the model to end-users. Each phase is made to deal with certain responsibilities and tasks, thereby facilitating a smooth flow of information and operations across the system:

1. **Data Ingestion**: Handles the collection and initial preprocessing of raw images from various sources.

2. **Datasets & Data Loaders**: Manages the organization and loading of datasets for training and validation.

3. **Model Training & Evaluation**: Encompasses the training routines, loss computations, and performance evaluations of the restoration model.

4. **Deployment**: Focuses on serving the trained model to end users through a web application.

4.1.1 Stages of the Architecture

4.1.1.1 Data Ingestion

Data might be sourced from multiple channels, including online source, local directories, and cloud storage, or uploaded from web GUI in training provider session. Then, raw images are ingested and stored in a local buffer, as caching mechanism, where they undergo initial preprocessing steps such as cropping and resizing also.

4.1.1.2 Datasets & Data Loaders

Two data sets (data loaders) were designed. First, that also partially integrates into Data Ingestion (4.1.1.1) stage, handles lazy loading and caching images by storing them in buffer from different, configured, and integrated with some online source, also, making initial preparation like cropping, resizing to one size.

Secondly, it generates low-resolution (LR) and high-resolution (HR) requested image pairs required for training or fine-tuning the model with applied several typified distortions, such as was described in the Objectives (3.1) section. Data loaders are also used to efficiently batch and shuffle data during the training and validation phases.

The final data set is then split into training and validation sets, ensuring that the model is evaluated on unseen data to gauge its performance accurately.

4.1.1.3 Model Training & Evaluation

To handle that critical part of the architecture, the core of application, for training and fine-tuning we propose to use Jupyter Notebook [54] with Python [55], which is the only way to train model in several available for public cloud services, that provides access to GPU resources. The implemented training pipeline and its runtime have be able to support local files I/O operations.

Metrics During the training and evaluation phases, various metrics are might be collected and processed with graphs for better understanding and visualization of the model's performance. The metrics proposed and applied are accessed in Evaluation Metrics (3.3) section.

4.1.1.4 Serving and Deployment

To provide end-users with access to the trained photo restoration model, we propose a robust and scalable deployment strategy. This strategy assure that the model is accessible and efficient, aiding seamless interaction between the back-end restoration processes and the front-end user interface.

Model Deployment The trained models is deployed in a production environment that ensures high availability and reliability. The deployment process involves the following.

- **Model Hosting**: The restoration model is hosted on a scalable server infrastructure capable of handling multiple concurrent requests. This setup ensures that the model can process numerous image restoration tasks simultaneously without significant delays.
- **API Integration**: An application programming interface (API) is developed to facilitate communication between the front-end user interface and the back-end model. The API handles image upload requests, processes them through the restoration model, and returns the restored images to the user.
- **Scalability and Load Balancing**: To accommodate varying loads and ensure consistent performance, the deployment architecture incorporates scalability measures and load balancing mechanisms. This approach allows the system to dynamically allocate resources based on demand, maintaining optimal performance during peak usage times.

4.1.2 Graphical User Interface

As mentioned the application serves in three different parts: Front-end (Graphical User Interface (GUI)), Back-end (API Server), and DevOps (Deployment and Serving both front-end and back-end on one server).

For the front-end side, we should build a solution based on defined Expectations (4), which leading us to one cross-platform solution that will be able to serve on any machine, including different operating systems, such as Windows, Linux, and MacOS, and mobile devices. So we decided to propose the web-based solution, which will be able to serve on any device with a web browser. With the help of different frameworks, such as React-Native [56], Ionic [57], and others, we could build a lightweight and responsive solution that will able to use native features of the device, such a camera, GPS, and others.

Also, by having a web-based solution, we could easily solve problem with updating, fixing, and shipping new features to the production environment. This will eliminate the need to wait for version approval from application server providers.

4.1.3 Server Architecture

The server architecture is only restricted by non-functional Expectations (4), and frontend-implemented solution. Corresponding to the chosen model, or models in experiments and implementation the server might be write with the same technologies as the model, such as Python or JavaScript [58] in a pair with TypeScript [59], or any other language that will be able to serve the model or models.

Microservice Solution However, a better option would be to divide the server architecture into microservices [60], where each model would be in a separate running process, with a lightweight API endpoint to use it and one main API server, that will be responsible for routing the requests to the corresponding model, and handling the rest of the requests, including frontend part.

4.2 Deployment Strategy

The deployment strategy is a critical aspect of the architecture of the whole application, ensuring that it is accessible and efficient. To resolve the offered solution with different technologies and services that would be used in the application architecture, we propose to integrate Docker containers into this strategy.

4.2.1 Docker Containers Technology

Docker is a tool that is used to automate the deployment of applications in lightweight containers so that applications can work efficiently in different environments in isolation.

Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. Because all of the containers share the services of a single operating system kernel, they use fewer resources than machines [61].

Chapter 5

Implementation of the Full-Stack Restoration Solution

Based on the Design chapter (4), we have implemented the full-stack restoration solution based on the proposed Stages of architecture (4.1.1):

5.1 Data Collection & Handling

As described in the sections about Data Ingestion (4.1.1.1) we have implemented lazy loading and defects handling datasets.

The first one is used to obtain the quality data from the stock service with high-quality images. For this purpose, we have used the Pexels API [62], due to their free-to-use and license-free policy that applies to images. The second dataset is described below (5.1.2).

The dataset handles and preserves the metadata file, which is used to store categorized image URLs by search and sizes to obtain those corresponding images.

To handle various dataset size requirements, we have embedded a lazy-loading mechanism that is on-demand requests as image metadata, and images itself. All images are stored in a specified directory, in a tree-like structure built upon query, sizes, and root directory.

To maintain the diversity of the dataset we have crawled and fetched images from several categories, listed below (5.1). The categories were selected to ensure a diverse dataset, which should contain images with different textures, colors, patterns, objects, and so on. Additionally, a percentage was assigned to each category, indicating the percentage

of those images in relation to the whole amount of images.

Category	Description	Percentage
Faces & Portraits	Human skin tones, fine textures	10%
Urban & Architecture	Straight lines, text, signs	20%
Nature & Landscapes	Organic textures, foliage, sky	20%
Texture	Fabrics, surfaces, food close-ups	15%
Indoor Scenes	Low-contrast areas, mixed lighting	15%
Brands & Text	Sharp edges, fonts	5%
Low-Light / Night	Heavy noise, color casts	5%
Misc (objects, plants, food)	Varied textures	10%

Table 5.1: Dataset categories and their respective percentages

The complete process of data collection and handling is shown in the next Figure 5.1:

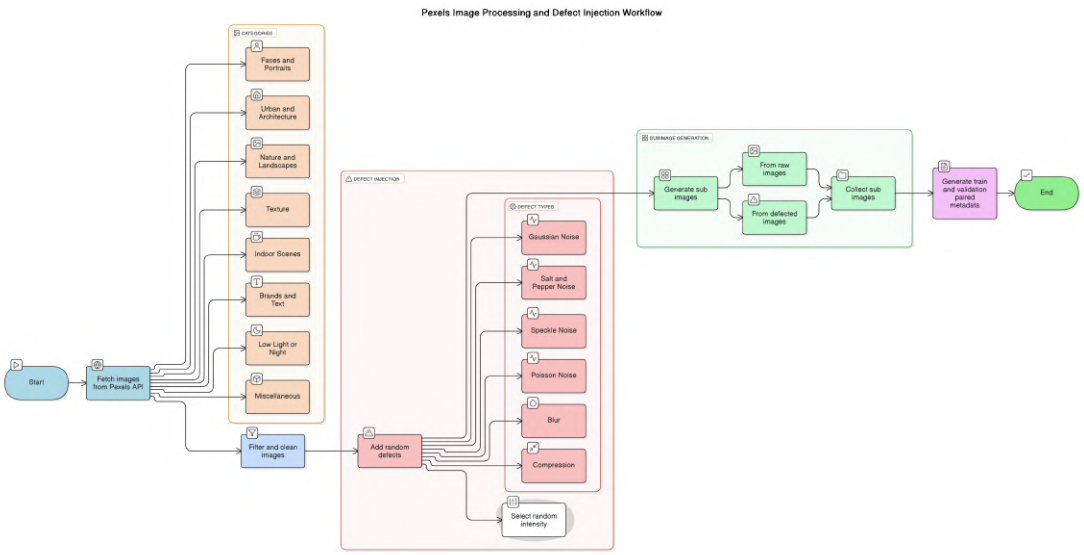

Figure 5.1: Data handling process

Data saving To store the whole amount of those images, we decided to use Google Drive, corresponding to Google Colab usage, which has integration with it. In different experiments, we have used different amounts of images and sub-images with applied or not defects to train.

5.1.1 Quality assessing

To improve our dataset and minimize the amount of data, images that could affect the model's result, we have additionally implemented a quality checker function. By non-quality images, we mean images that we don't want to see as the output of the model, images that already contain some artifacts, such as blur, noise, or not enough or too much detail. Those checking parameters we have used are:

- **Blurness** - the image should not be blurry; otherwise it will not be useful for our model. The blurriness is calculated using the variance of Laplacian method [21]. The threshold is set to 100.

- **Exposure/Contrast** - the image should not be too dark or too bright. The exposure is calculated by the mean of the pixel values. The threshold is set to 0.5.

- **No-reference IQA metrics** - As was mentioned in the objectives (3.1) we have used NIQE [52] and BRISQUE [63] metrics to check the image quality. The threshold is set to 0.5 for both metrics.

By filtering fetched images based on the above parameters, we have eliminated more than **70%** of the images. This fact means that such a procedure has very helped us with clearing our training data. On the next Figure 5.2, could be seen examples of images that passed the quality check:

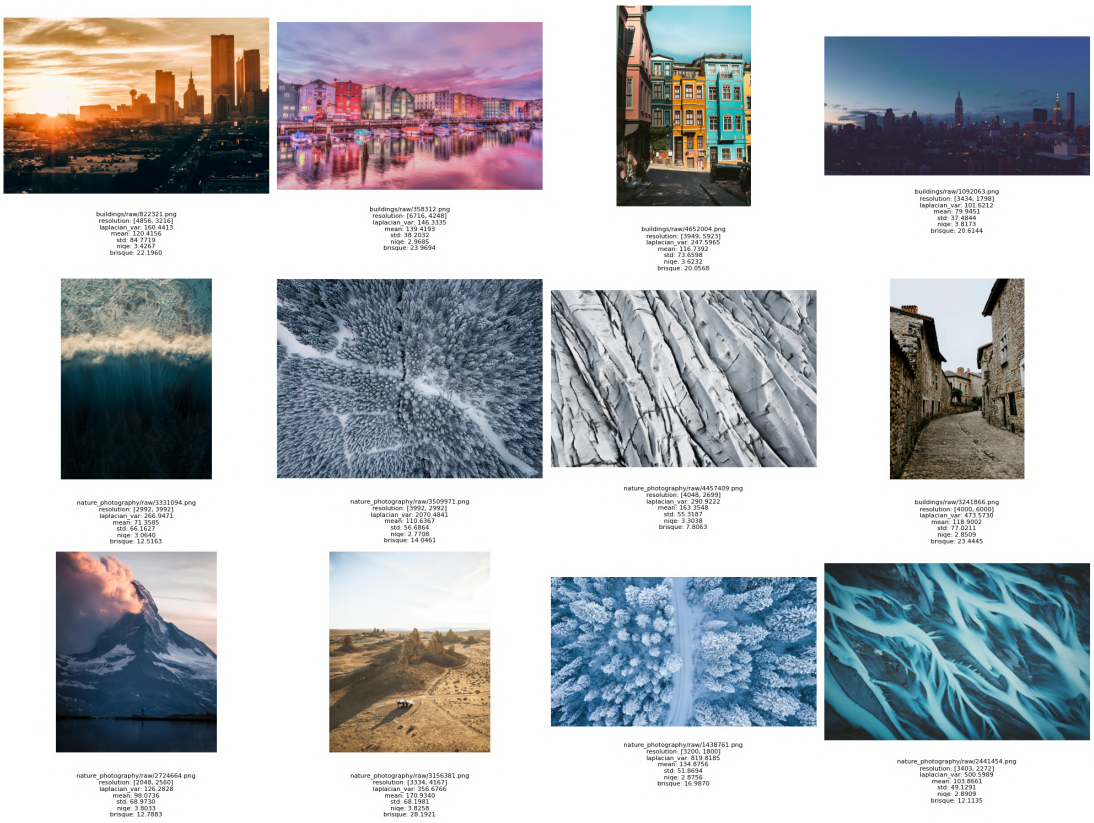

Figure 5.2: Fetched images after quality check

Although our dataset contains fewer unique images than commonly employed baselines like DIV2K [64], this estimate understates the actual volume of training data available. The reason is that each of our source images is photographed at significantly higher resolutions — up to 4K — thereby enabling the fine-tuning process by dividing them into a series of sub-images, tiles, which number of patches thus garnered is more than twice that of DIV2K.

5.1.2 Defects handling

Defects handling is a crucial part of our project, as we are trying to solve the problem of defects removal. So, the second dataset is responsible for the defects applying. That function's ability can be described by its ability to apply the following defects:

- **Gaussian noise** - statistical noise having a probability density function equal to that of the normal distribution, which is also known as the Gaussian distribution.

- **Salt & Pepper noise** - also known as impulse noise, is a form of noise that presents itself as sparsely occurring white and black pixels.

- **Speckle Noise** - a granular 'noise' that inherently exists in and degrades the quality of the active radar, synthetic aperture radar, medical ultrasound, and optical coherence tomography images.

- **Poisson noise** - a type of noise that can be modeled by a Poisson process. In electronics, shot noise originates from the discrete nature of electric charge.

- **Blur** - a reduction in the sharpness of an image. The blurriness is calculated by the variance of Laplacian method.

- **JPEG compression** - a lossy compression method for digital images, particularly for those images produced by digital photography.

- **Low resolution** - downscaling the image to a lower resolution.

Those defects are concurrently with the defects in Overview (2) chapter.

Also, to maintain a gradual training process, we have added the ability to apply defects with certain intensivity, which can be described in the Figure 5.3 below.

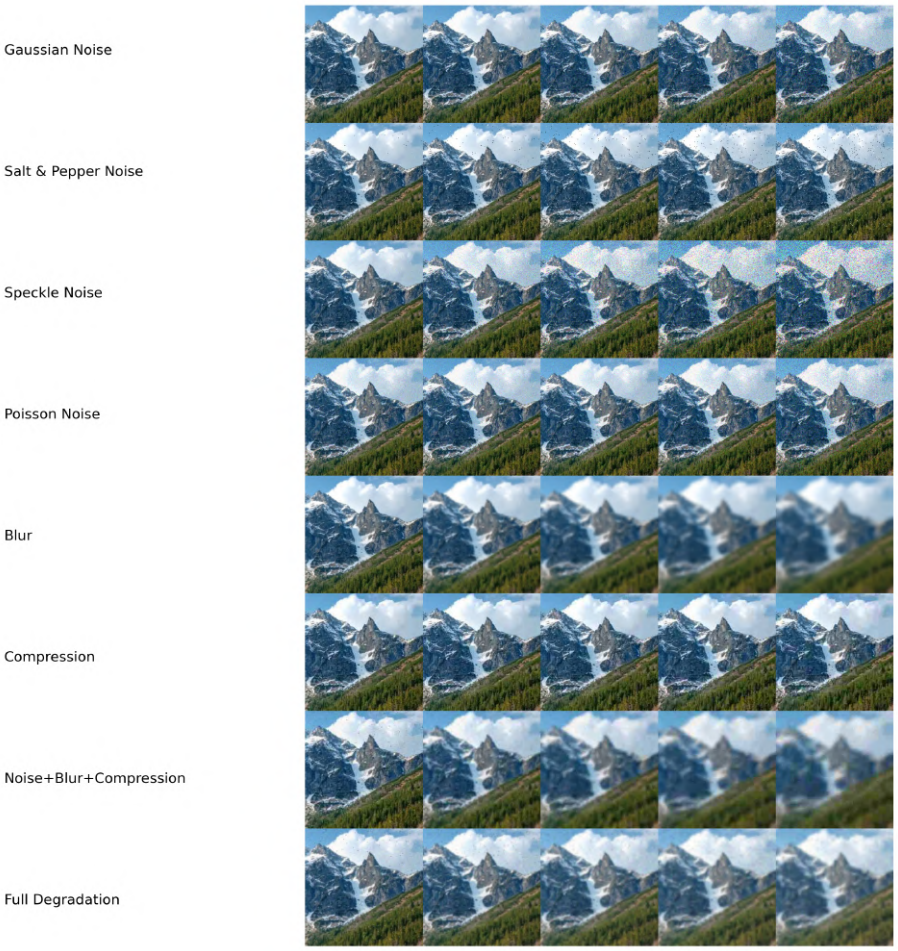

Figure 5.3: Showcase of applying different defects with intensivity from 0 to 50%

In the final fine-tuning process, we have applied those defects by random selection and with random intensity from 0 to 15% to the image.

5.2 Model Implementation

During the implementation, we have created two custom model architectures, inspiring by the latest great works in the field of image restoration with the use of CNN (2.2.2), like Residual [65], Up-sample [66], and Dense Blocks [67].

5.2.1 Super Resolution Model

The first architecture leverages CNN (2.2.2) with Residual (5.2.1.1) and Up-sampling (5.2.1.2) blocks to achieve high-quality restorations. Primary, at the first, model was aimed only at up-scaling tasks. The following tables provide a detailed overview of each component within the model.

5.2.1.1 Residual Block

The `residual block` is a fundamental building block of the model, allowing the network to learn residual mappings. This facilitates the training of deeper networks by mitigating the vanishing gradient problem.

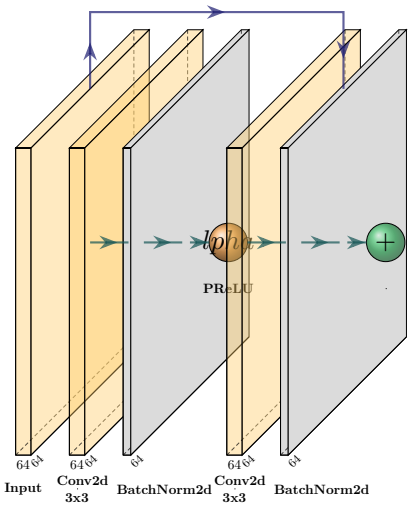

Figure 5.4: Architecture of Residual Block

This "skip-connection" design allows the block to learn only the residual mapping $\mathcal{F}(x)$—that is, the difference between the desired underlying mapping and the identity:

$$y = \mathcal{F}(x, \{W_i\}) + x \tag{5.1}$$

By adding the input x back into the output, gradients can bypass the convolutional layers during backpropagation, mitigating vanishing-gradient issues and enabling the training of very deep networks without degradation of performance.

5.2.1.2 Up-sample Block

The `Up-sample Block`s are responsible for increasing the spatial resolution of feature maps, enabling the model to generate high0resolution outputs from low-resolution inputs.

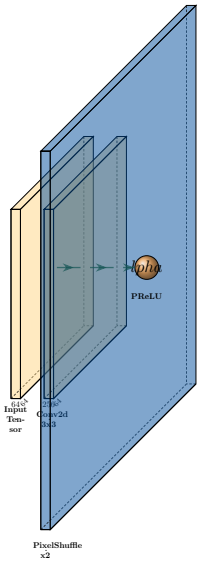

Figure 5.5: Architecture of Up-sample Block

The architecture is based on the following formula:

$$y = \text{PReLU}\big(\text{PixelShuffle}\big(W * x + b;\ s\big)\big) \tag{5.2}$$

Where, by first using a convolution to expand channel dimension to $C\,s^2$, the PixelShuffle operation then efficiently converts channel information into higher spatial resolution without introducing checkerboard artifacts common to transposed convolutions. The subsequent PReLU adds learnable non-linearity, improving the model's capacity to reconstruct fine details in the up-sampled image.

5.2.1.3 Super Resolution Model Architecture

The model integrates multiple Residual (5.4) blocks and Up-sample (5.5) blocks to perform a comprehensive image restoration. The model architecture is designed to capture both local and global image features, ensuring detailed and accurate restorations.

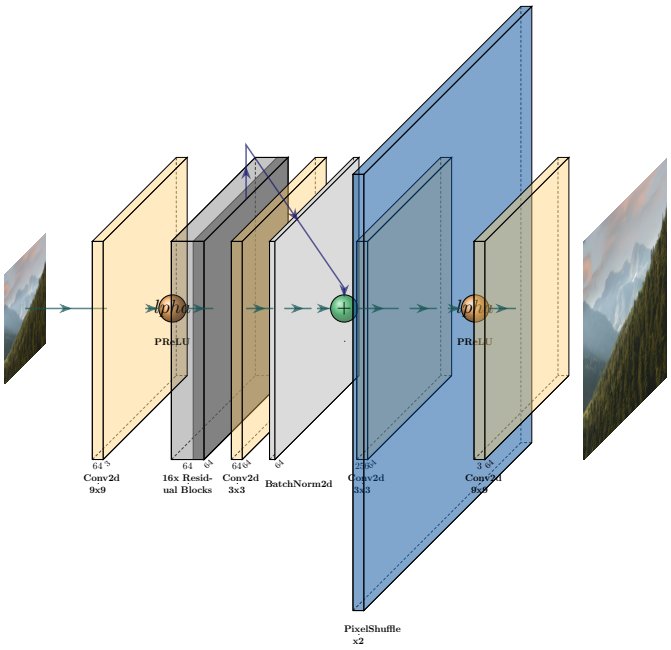

Figure 5.6: Architecture of whole Super Resolution Model

5.2.1.4 Perceptual Loss

To enhance the perceptual quality of the restored images, the `Perceptual Loss` class computes the loss based on feature maps extracted from a pre-trained VGG19[68] network. This approach ensures that the restored images maintain high-level semantic features and textures similar to the ground-truth images.

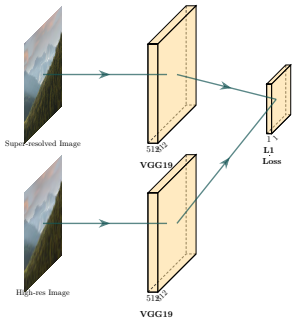

Figure 5.7: Architecture of whole Super Resolution Model

5.2.2 Enhanced Super Resolution Model

With having promise results and identifying problems from the first experiment (6.2.3), we decided to use the same model with applying some improvements.

The first improvement was to use the Dense Blocks (5.2.2.1) with a pair of **Residual Blocks** in the model architecture. Since dense blocks encourage thorough feature reuse and reinforce local representations by adding direct connections between all layers in a block, leading to more informative hierarchical features and mitigating gradient-vanishing problems [69, 70], they are particularly effective in restoration of image details.

Additionally, residual connections offer identity mappings that stabilize training of deep networks and ease gradient flow, thus speeding up convergence and enhancing reconstruction fidelity [69]. By combining both mechanisms within Residual Dense Blocks, we exploit their complementary merits to create a unified memory structure that can effectively encode both local and global information. This approach yields improved super-resolution performance and demonstrates a strong capability in addressing challenging defect patterns [71].

5.2.2.1 Dense Block

The `Dense blocks` are a type of neural network architecture that consists of multiple layers, where each layer is connected to all previous layers:

$$x_\ell \;=\; H_\ell\big(\big[x_0, x_1, \ldots, x_{\ell-1}\big]\big) \tag{5.3}$$

This means that the output of each layer is fed as input to all subsequent layers. The key idea behind dense blocks is to encourage feature reuse and improve gradient flow during training.

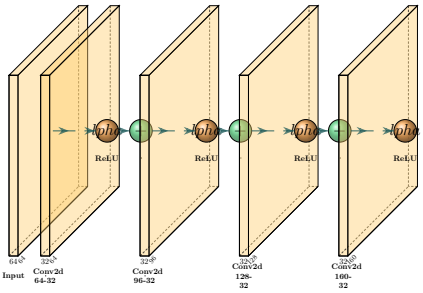

Figure 5.8: Architecture of whole Super Resolution Model

5.2.2.2 Enhanced Super Resolution Model

The final model architecture with applying new improvements is shown in the Figure 5.9:

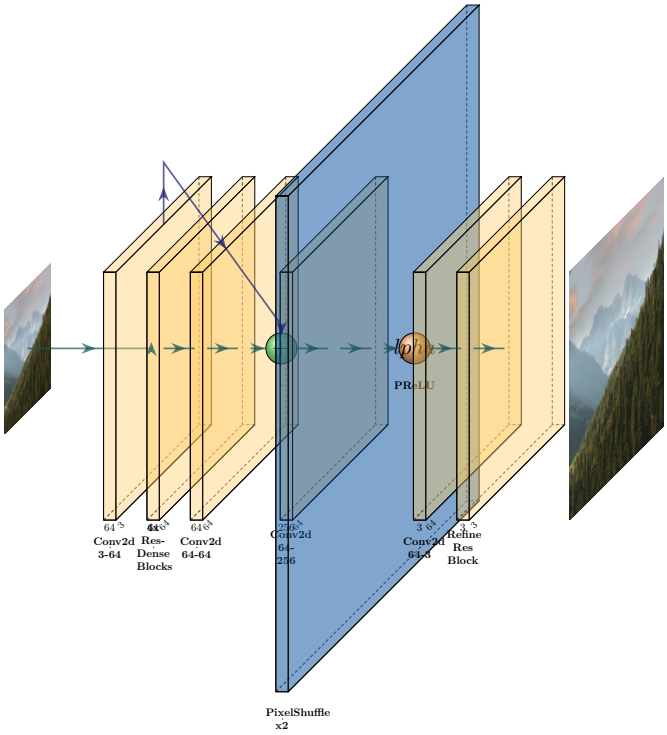

Figure 5.9: Architecture of whole Super Resolution Model

5.2.3 Fine-tuning Real-ESRGAN

After several experiments and attempts at creating a custom model architecture and training it from scratch, by defining limitations (6.5), we have moved to using a pre-trained model and fine-tuning it. To our selection went one of the most popular open-source models - Real-ESRGAN [44], which was briefly referred to Related Work (2.3.4) section, and which is based on ESRGAN [43](5.10) architecture:

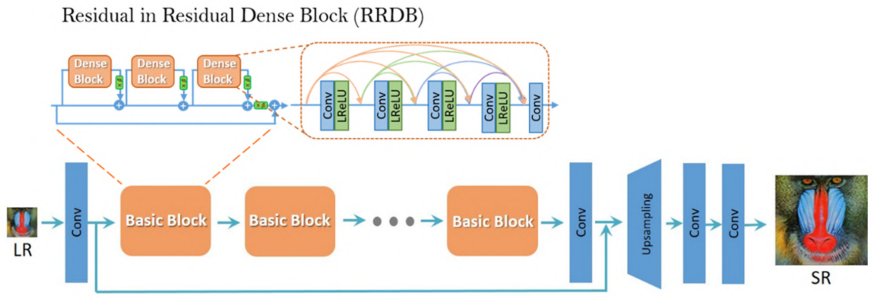

Figure 5.10: ESRGAN generator architecture [44].

Real-ESRGAN is a state-of-the-art model for image super-resolution, which is capable of generating high-quality images from low-resolution inputs. The model is based on the ESRGAN architecture. The main point of the model is to solve the problem of one-for-all super-resolution and defects removal solution, which aligns with our objectives (3.1).

The model was also selected due to its open-source code and pre-trained weights, which are available on GitHub [72]. The model is trained on a large dataset of high-resolution images, including DIV2K [64], Flickr2K [73] and OST [74].

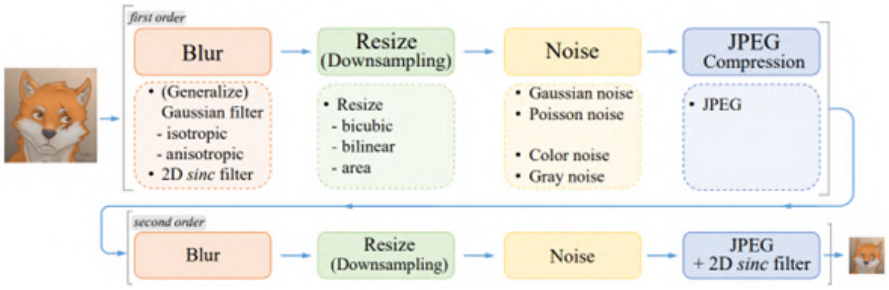

Figure 5.11: Real-ESRGAN degradation process [44].

Their solution also includes a degradation module that incorporates noise, blur, and compression artifacts (5.11). However, in our case, we have used our own implemented datasets (5.1) with the added ability to download, filter, and preprocess all images at once, and to cover the Real-ESRGAN code requirements [72] for fine-tuning.

5.2.4 Model Evaluation

Corresponding to the proposed evaluation metrics solution (3.3), we have evaluated the model output by metrics that was already discussed above (2). The process of evaluation involved selecting images, applying the model to downscaled and optionally defective images, and finally calculating metrics. The evaluation results for each model can be found in the corresponding experiments (6).

5.3 Application & Server API

As we discussed in the previous chapter, we have chosen to use two technology stacks in our implementation. The first one is related to model development and training, including data collection, preprocessing, evaluation of the model, and all other things related to model, and it is Python-based.

5.3.1 Back-end

Python was chosen because it is the most popular and widely used language for data science and machine learning. Other options were not considered due to the limitation in other languages, due to the fact of missing or not well-supported libraries were available. Additionally all models (2.3) mentioned in this paper are also implemented in Python, so not using Python would be very difficult in such an implementation.

To use the model is production, we have needed only one endpoint that would work as input and output of the model. The endpoint hosting was implemented in a Flask [75, 76] server, which is a micro web framework for Python. Flask was chosen because it is widely used in the industry.

5.3.2 Defects applying server

In addition to the main server, which is for hosting and using the binary NCNN model 5.4.1, in the final implementation, we have one more server, to apply defects to the image.

As we aim to implement a model that has to remove image defects we want to show in the production environment how the model handles defects removal. For that purpose, we have created a server that is applying our implemented degradation functionality (5.1.2).

As the model hosting server, one also has only one API endpoint with supported query parameters to adjust the defects applying parameters. The server is

5.3.3 Front-end

For the second part (4.1), the Front-end (GUI) (4.1.2), selected language is JavaScript in a pair with TypeScript. To maintain quality and performance of the front-end due to facing this part to the external client, we need to ensure and chose a proper framework that will help us to achieve that. In order for our choices to be valid, we must justify them with the help of statistics and metrics. Based on which [77, 78, 79, 80] we have selected to use Next.js [81] with React.js [82] under the hood.

By selecting Next.js, we have eliminated several issues, including performance ones, like server-side rendering, caching, browser-based optimization techniques, SEO and core-web-vitals, and the need to use different process for back-end of our user-faced application. With Next.js we were able to create a full-stack application that is still able to use external servers with our deployed models.

5.4 Deployment

As proposed in the design chapter (4.2), we have used Docker containers to minimize deployment and installation issues. The Docker containers are used to pack the whole solution, including the model with API endpoint and GUI. The Docker containers are used to ensure that the application will run in any environment, regardless of the underlying operating system or hardware.

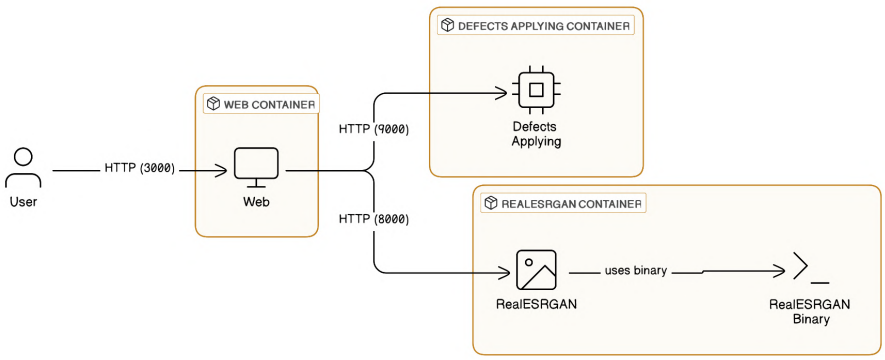

Figure 5.12: Deploying docker containers

The whole architecture of the containers is shown in the Figure 5.12 above. In diagram, we could also see the usage of a binary file in the model running container. This binary is an implementation of Real-ESRGAN model in NCNN [83] format, what is make it able to run model efficiently on any device and platform, corresponding to NCNN documentation [44, 72]. The NCNN is a high-performance neural network inference

framework optimized for all platforms; it does not have any dependencies, and it is very lightweight [83].

5.4.1 Converting to NCNN

For better performance and to be able to showcase the model usage, we have decided to inspire and use the NCNN framework. With the help of already implemented Real-ESRGAN model in the NCNN format, we had to convert our PyTorch [84] result model weights to NCNN format ourselves.

By following the guide [85], we have installed the chaiNNer [86] application, with its dependencies, constructed the pipeline, where we are loading our PyTorch model, converting it to NCNN via a special node, and finally saving dot **param** and dot **bin** files. The pipeline is shown in the Figure 5.13 below. The resulting weights can then be easily used in the NCNN compiled binary.

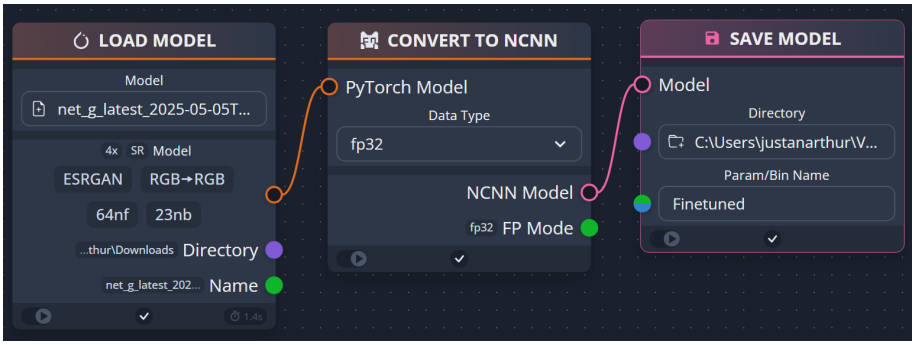

Figure 5.13: ChaiNNer pipeline for converting PyTorch model to NCNN

Issue with GPU The main issue in our deployment of the AI model is using the GPU computation unit in model inference. Due to having large model weights, the fallback to CPU is making the output very slow, up to several minutes, and this also adds the decreased efficiency due to Docker container overhead.

Solution to this problem is to use NVIDIA CUDA [87] runtime, with which Docker has already an integration. Unfortunately, in our work, we did not have access to CUDA-supported GPU, so we could not present a solution that would work with a GPU inside a container. In contrast, the NCNN-implemented model with custom-loaded weights was tested on bare metal, where the Vulkan API was used to run the model on an integrated GPU, which decreased the inference time to several seconds.

5.5 Project Structure & Management

To maintain the whole project, we were using the version control system - Git [88], with GitHub as a remote repository hosting provider. The GitHub [89] hosting was additionally used for project management, including issues, code review by mentors, and other.

Also, to aim for the best practices, we have used the possibilities of git modules, where each certain module is stored in a separate Git instance, repository, and then included into one bigger. This approach helped us with clean separation and a development process, where each member could work only on those modules that he or she needed.

Chapter 6

Experiments and Evaluation

6.1 Setup

All experiments were carried out using Google Colab [90], a cloud-based platform that provides access to computational resources. The specific hardware configurations used in experiments are:

- **Processor:** Intel Xeon CPU @ 2.30 GHz (a multicore processor with 2-4 cores).
- **Graphics Processing Unit (GPU):** NVIDIA A100, a high-performance GPU designed for deep learning and AI workloads, offering 40 GB of VRAM.
- **Memory:** 52 GB of RAM, sufficient for handling large datasets.
- **Storage:** Provided storage with a capacity of 100 GB, which is utilized to store data sets, model checkpoints, and intermediate results during training and evaluation.

6.2 Case Study - Experiment 01

The first experiment involved into the training of our first implemented model, which is mention in the Implementation chapter (5.2.1). The details about the architecture and data processing are mentioned in the section about the first model implementation (5.6).

6.2.1 Data Obtaining

To obtain data, we used our lazy dataset (5.1). A total of 5000 images were lazily downloaded during training from the platform. Due to limitations, at the fist, we specifically limited the search to the `nature` category to focus on a specific domain for

getting better results in limited scope. This decision was made due to the uncertain results of the first model results.

The dataset options were set to resize all images to a resolution of 512 pixels High-Quality (HQ) and to 256x256 Low-Quality (LQ). The images were then split into training and validation sets, with **80%** of the images used for training and the remaining **20%** used for validation.

Additionally, for the first model training, we did not apply either quality filtering or image artifacts, only downscaling. That first experiment was made to prove the model concept as a working one.

6.2.2 Training Process

The initial training process was made with the following parameters:

Parameter	**Value**
Number of Images	5000
Batch Size	16
Number of Epochs	20
Loss Functions	L1 Content Loss, Perceptual Loss (VGG19)
Optimizer	Adam (learning rate: 1e-4)
Learning Rate Scheduler	StepLR (step size: 20, gamma: 0.5)
Device	CUDA A-100 GPU

Table 6.1: Training parameters

6.2.3 Results

Extensive experiments were conducted to validate the performance of the proposed photo restoration model. During the training process, the training loss and validation loss consistently decreased within 20 epochs using 5,000 images, reflecting good learning and generalization ability. PSNR and SSIM increased steadily during this process, indicating that the quality and structure of images would be gradually improved during training.

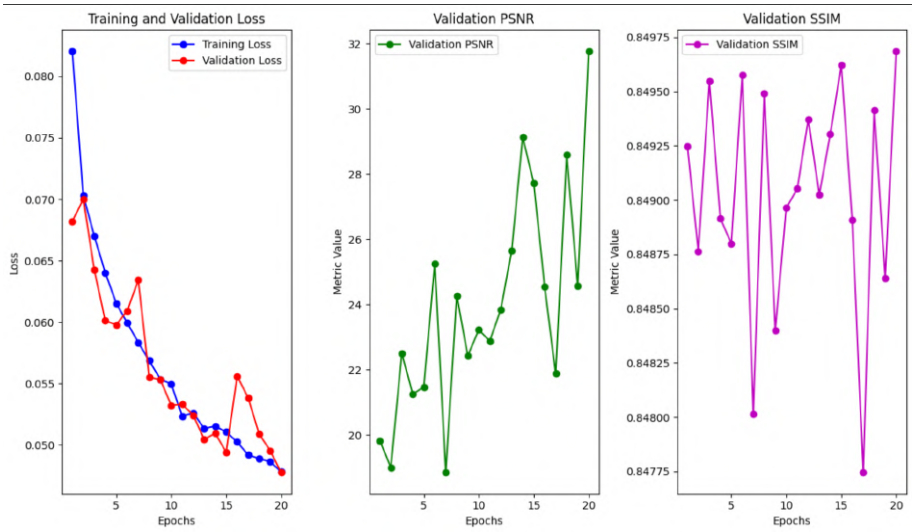

Figure 6.1: Loss and metric values through epochs

Figure 6.1 illustrates the increasing metric values for the PSNR and SSIM metrics over the course of the training process. The model's performance improved significantly after the first **13** epochs, with the PSNR and SSIM values reaching **25.5** dB and **0.85**, respectively. The model's performance continued to improve until the **20**th epoch, with the PSNR and SSIM values reaching approximately **31.8** dB and **0.86**, respectively.

In addition to PSNR and SSIM, the Blind Image Quality Assessment (NIQE) scores further highlight the model's effectiveness. Sample #1 recorded a NIQE of **4.13**, while Samples #2 and #3 achieved higher scores of **5.44** and **7.41**, respectively. These NIQE scores suggest that the restored images possess enhanced perceptual quality, with higher values indicating better naturalness and reduced artifacts. Furthermore, the inference times for all samples were exceptionally low, ranging from **0.45** to **0.51** milliseconds, underscoring the model's computational efficiency and suitability for real-time applications.

Figure 6.2 showcases the evaluation results for three sample images. Each row displays one image in several variants: Low-Quality (LQ), Super-Quality (SQ) (Restored), and High-Quality (HQ).

Figure 6.2: Evaluation results for a few samples

6.2.4 Discussion

The first model output showcased encouraging outcomes, demonstrating that the selected CNN architecture has the ability to solve photo-restoration issues effectively. Although the result image contains certain defects, it is a drastic improvement compared to the degraded input.

Among the notable limitations is the fact that the network exclusively operates on 256 × 256 pixel patches, which may cause minor artifacts to appear in certain outputs.

Is should also be noted that the model was trained on a specific domain with uncurated

data - numerous images that are not relevant to the target domain or already contain some artifacts. Additionally, the relatively short training duration of merely 20 epochs is insufficient for the model to fully learn the underlying patterns and features of the data.

Despite these limitations, the model still works well. This is evident in the high PSNR and SSIM metric values and low inference time, which show that it has learned to restore images effectively, even when presented with noisy and varying inputs.

6.3 Case Study - Experiment 02

The second experiment is related to the implemented second model, that is described in section about Enhanced Super Resolution Model (5.2.2.2).

6.3.1 Data Preprocessing

The data obtaining process was the same as in the first experiment (6.2.1).

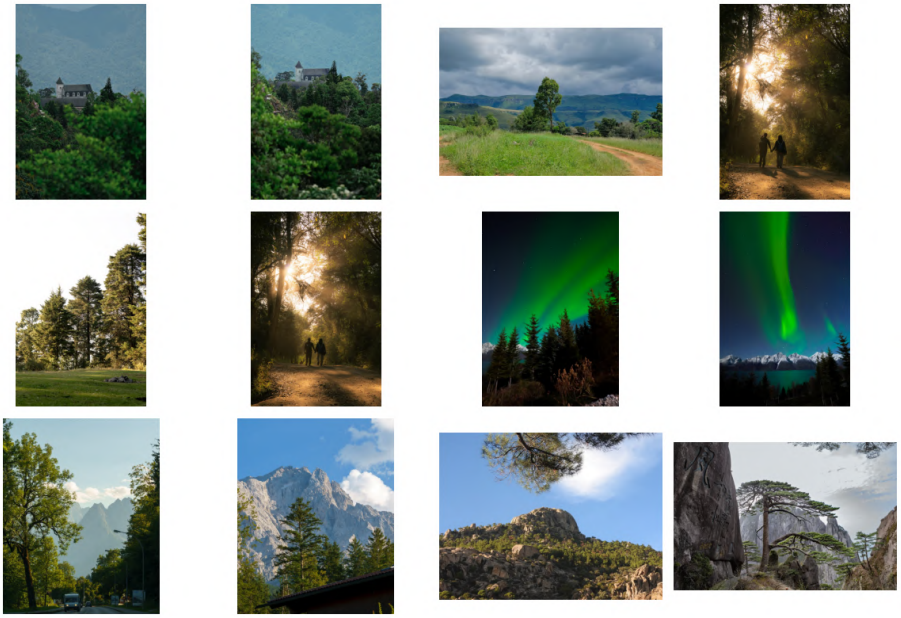

Figure 6.3: Example of high-quality (HQ) images used for training

Figure 6.3 shows the high-quality (HQ) images used for training. The images are obtained from the Pexels [62].

However, in this experiment, we modified the data workflow to also augment the image data by applying the following transformations:

- Apply several defects to one low-quality (LQ) image, including downscaling, noise, blur, and JPEG compression. The defect intensity is randomly selected.
- Apply random rotation and flipping to the LQ image.
- Crop and mask images to the required sizes.

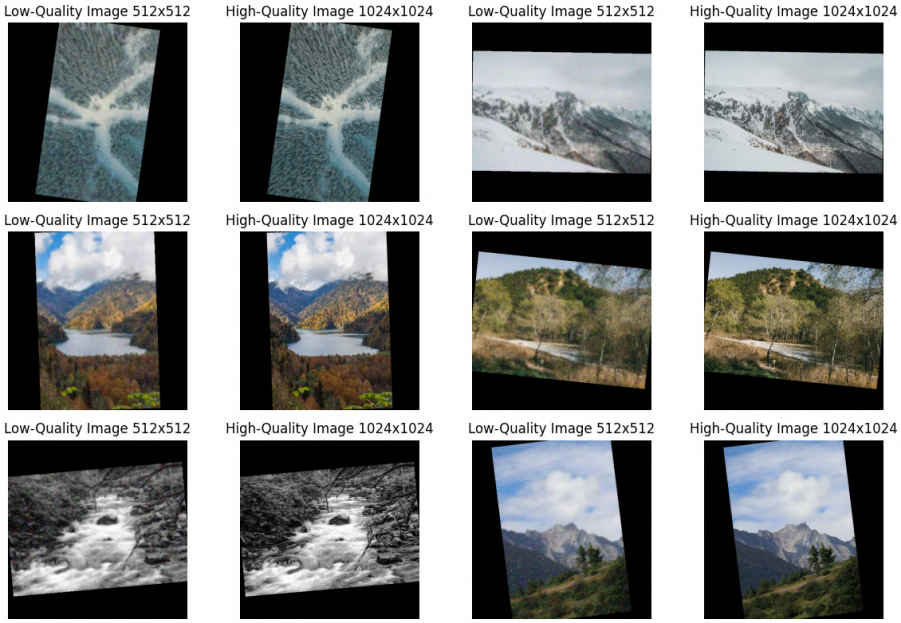

Figure 6.4: Simulating low-quality image with defects and cropping/masking

Figure 6.4 showcases the simulation of applying defects to simulate low-quality (LQ) image. Apply random rotation, cropping and masking to the LQ image.

The main part of this experiment is applying defects to our training images and restore then, which is the main aim in our work. The transformations were applied to make the dataset more diverse, even when trained on a single domain of images.

6.3.2 Results

During the training process with various parameters, the model was unable to achieve the expected results.

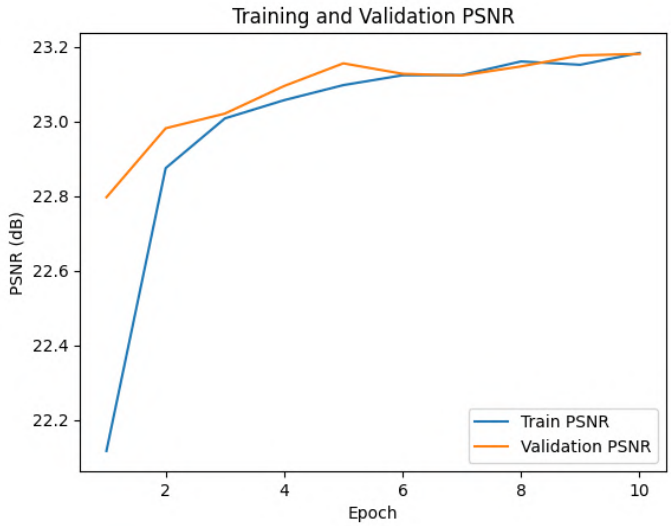

Figure 6.5: PSNR through epochs

Figure 6.5 shows the loss and PSNR through epochs. The model was trained for only 10 epochs, but the loss did not decrease significantly, and the PSNR remained low, so we stopped the training process.

6.3.3 Conclusion

The second experiment focused on improving the model architecture and data preprocessing based on the first one (6.2.3). Corresponding to several references, we added Dense Blocks into our architecture, which migh help to improve model detail tracking and reconstruction.

However the results were not as expected, and the model was unable to achieve the desired results. After several iterations, the metrics remained low, and the loss did not decrease significantly. We stopped the training process after 10 epochs, as the model was not able to learn the defects and reconstruct the images properly.

6.4 Case Study - Experiment 03

After narrowing several limitations and problems with trying to train a custom model, we have decided to move to fine-tuning the model.

6.4.1 Model selection

After several tests with different models, we have decided to use the Real-ESRGAN model. It is also one of a few super-resolution models that have open-source code with pre-trained weights. The model is based on the ESRGAN architecture and has been trained on a large dataset of high-resolution images, including DIV2K[64], Flickr2K[73], and OST[74].

The model was selected by comparing several latest state-of-the-art models in different domains. The final choice was grounded due to the similarity of the model's aim, which is incorporating with our goal of image restoration. Additionally, as mentioned, the model has open-source code and pre-trained weights, which is very important due to our limitations (6.5).

Testing defects handling on native model Before fine-tuning we also tested the plain Real-ESRGAN model on defects. The aim was to check how the model is handling defects and if it is capable of removing them. We have used the sample image with applying gradual defects, that was present in the implementation section (5.1.2).

Based on results - the model was already capable of removing multiple types of artifacts. This capability stems directly from its prior training, which specifically focused on handling various image defects. However, the model is still far from perfection due to a lack of image details and sharpness.

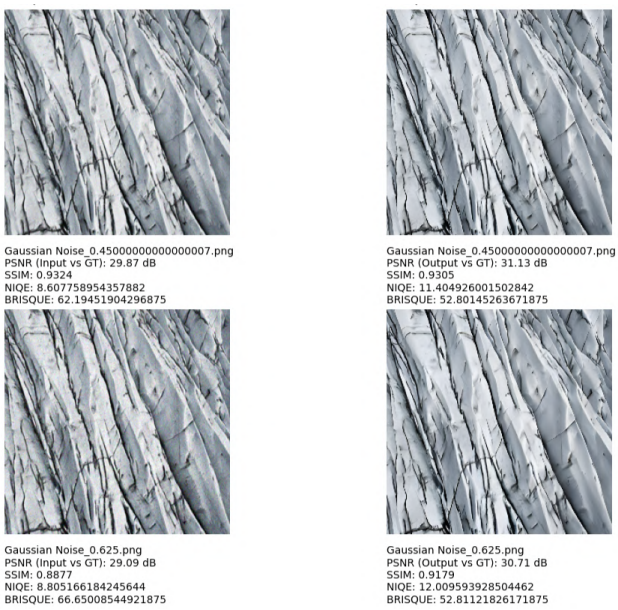

Figure 6.6: The part of the tested images on model

On the Figure 6.6, we can see only a part (two) paired images, where the left one is the original image with applied defects, and the right one is the output of the model. The whole image could be found in the codes of the digital submission (A).

6.4.2 Data preparation

After handling downloading good images and generating degraded ones, we started to create sub-images by dividing the images into patches. The patch size is **480x480** pixels, and the stride is **256** pixels, which refers to the HQ image. The LQ image is **4x** downscaled, into **120x120** pixels. The patches are created by using the sliding window approach. Finally, by having more than **69,000** images of high and low-quality images, we have created a metadata file with the paths to the images.

6.4.3 Training

To fine-tune the Real-ESRGAN, we have included the use of BasicSR [91], an open-source image and video restoration toolbox, which was also used in Real-ESRGAN training.

Final setup of the training includes the following parameters:

Parameter	**Value**
Number of Images	5000
Batch Size	2
Number of Epochs	5000
Loss Functions	L1 Loss, Perceptual Loss, GAN Loss, Contextual Loss
Optimizer	Adam (learning rate: 1e-5, betas: [0.9, 0.99])
Learning Rate Scheduler	MultiStepLR (milestones: [5000], gamma: 0.5)
Device	CUDA A-100 GPU

Table 6.2: Fine-tuning parameters

6.4.4 Results of fine-tuning

The final result was achieved after **5000** epochs of training. The evaluation PSNR and SSIM metrics increased from an initial **28.57** dB and **0.82** to **30.05** dB and **0.87** respectively. The whole fine-tuning metrics over iterations can be seen in the Figure 6.4.4 below.

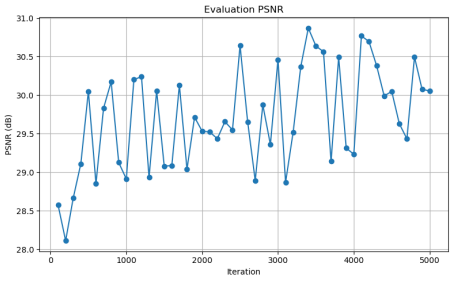

Figure 6.7: Evaluation PSNR

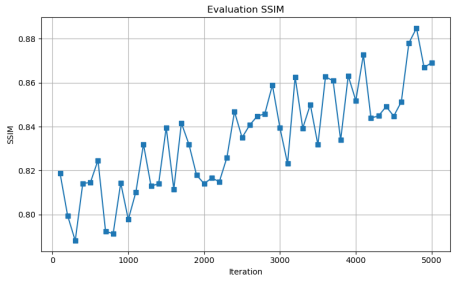

Figure 6.8: Evaluation SSIM

In the practical evaluation with the test images, even with increasing PSNR and SSIM metrics, the actual quality mean of image, what could be say by applying blind image quality assessment (BIQA) metrics, and human evaluation, decreased (6.10).

However, in some specific examples, it outperformed the native Real-ESRGAN model in removing effects, which can be seed in the Figure 6.9. Additionally, it can be said that the higher PSNR metric that the fine-tuned model outputs more realistic images than the native one, because the native model has very clean images, more like synthetic ones, whereas our model was directly trained on real stock images.

Figure 6.9: Advantageous fine-tuned results

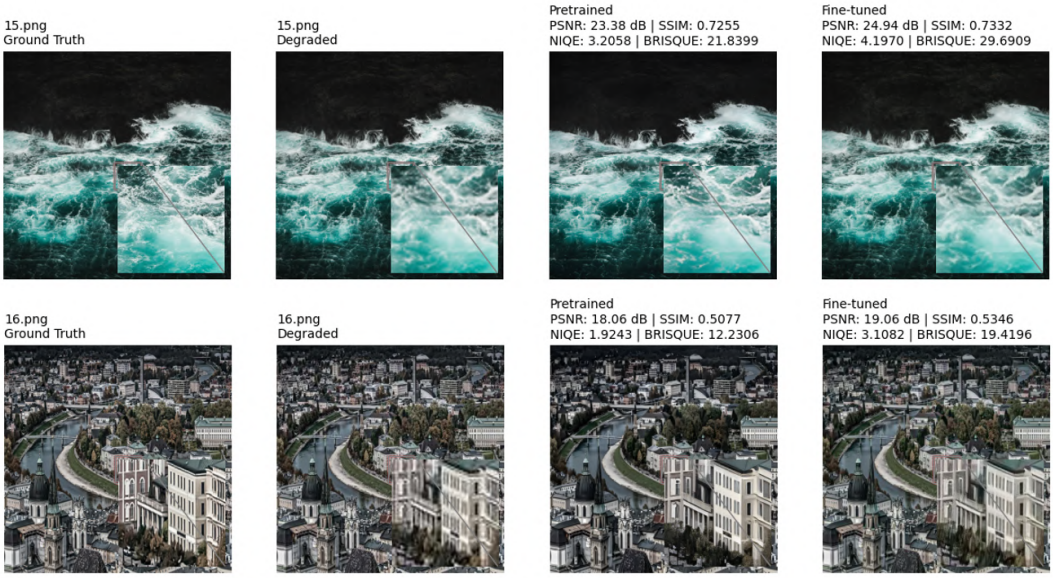

Figure 6.10: Poor fine-tuned results

6.4.5 Conclusion

Unfortunately, the fine-tuning results did not lead us to the expected outcomes. Even with increasing metrics, the actual mean output was worse than that of the native Real-ESRGAN model. After fine-tuning, the model it was slightly lost the details of the images, and some artifacts were still present.

These mean poor results could be attributed to the low quality of the dataset that was used for training. Even with the diversity of the dataset, by populating several image categories (5.2) and filtering them by blind image quality assessment metrics, the dataset was still, in our opinion, not of sufficient quality. This factor, from our point of view, along with a small batch size, could also negatively affect the results.

In the end, we think that the fine-tuning process was successful; we were able to create a robust pipeline: dataset crawling, defects applying, and make it work with the existing model within limited space, budget, and time.

6.5 Summary

Main *key findings* based on Section (6) results are:

- **Trade-off Between Quality and Efficiency**. Deeper and more complex architectures consistently improve restoration quality, but at the cost of longer training times and higher GPU memory footprints, which is a significant limitation.
- **Benefit of Transfer Learning**. Leveraging a pre-trained GAN model (Real-ESRGAN) enabled us to reach state-of-the-art perceptual results with fewer epochs than training from scratch.
- **Hardware Constraints Still Matter**. Even with a high-end A100 GPU, Colab's session limits and single-GPU memory caps forced compromises (e.g. smaller batches, image down-sampling or tiling) that may have slightly undercut the model's true potential.

Across all three progressively challenging case studies, we observed the same trend: as both model complexity and prior knowledge increased, so did restoration quality — at growing computational cost. In the first Experiment (6.2), our plain CNN, trained from scratch, produced good PSNR and SSIM values but struggled to restore fine detail on highly degraded inputs. Representative of this, the second Experiment (6.3) incorporated a defects module feature and a more complex architecture, which resulted in worse PSNR and SSIM scores even in the first epochs. Finally, the third Experiment (6.4) trained a more sophisticated pre-trained Real-ESRGAN model on our own dataset with improved blindness and perceptual metrics.

In spite of the promising outcomes seen, our work is narrowed by several critical *limitations*, significantly due to specialist knowledge and computing power constraints:

- Knowledge in Developing Tailor-Made Models While we tried to design an optimal AI architecture for image restoration, the inexperience of our team in creating and optimizing novel deep-learning models likely limited the extent of architectural explorations. Standard benchmarks and pre-configured frameworks took over much of the effort, possibly barring the discovery of more effective or efficient network architectures.
- Computation Time Limitations The whole training of the models was done on Google Colab, which imposes a stringent limitation of 12 hours per session with additional rules to keep this time active in the tab. This time constraint made it difficult for us to fully train deeper or more complex variants, especially those that require longer convergence times. This meant that some network configurations were stopped halfway through, perhaps before they could reach their best

performance levels.

- Single-GPU and Memory Limitations Our experiments were restricted to a single GPU instance (we have used the maximum allowable - NVIDIA A100) with restricted VRAM capacity. Input images at high resolution — important for the capacity to discern subtle structural details — could not be processed in batches of more than a few without exceeding memory, and we were forced to downsample or decrease batch sizes in training and fine-tuning. These compromises may have limited the model from learning more fine-grained restoration features and limited the generalizability of our findings to truly large-scale, high-quality images.
- Lack of Quality Dataset In our implementation, we decided to create and use a custom dataset for training and fine-tuning the models. Even though the images were selected from high-quality sources and filtered with metrics like blind image quality assessment (BIQA) to increase the quality of the dataset by reducing the number of deficient images.

As always, the reinventing of the wheel is a double-edged sword. While we have shown that it is possible to achieve state-of-the-art results with a custom architecture, the time and effort required to do so may not be worth it for all applications. In many cases, it may be more efficient to use an existing model and fine-tune it for specific tasks, with our own dataset, which is more valuable than the model. As was said, reinventing, or creating something totally new is a hard task, but if the resources, both intellectual and computational, are available, it is worth trying.

Chapter 7

Conclusion

Our goals were to design and implement dockerized full-stack solution in problem domain of image restoration. Through whole study we did a strong work on the each part of the solution:

Reviewing the current status of image restoration, by going through as traditional restoration process (spatial domain and frequency domain methods), and as with applying as plain and a deep networks, with variance of architectures, including CNN (2.2.2), GAN (2.2.3), and more complex solutions.

Our goals and objectives, by defining strong expectations from our result work, which covers all parts and stages of our system (4.1).

We designed and implemented a full-stack solution architecture, which going along with image restoration process, from dataset handling (5.1) to the final training and fine-tuning the model (4.1.1.3). In addition, based on our expectations, the Web GUI and Python server was added to source-code, where each part is also is Dockerized, which allows to run it in any environment.

In several tries we have been developing new solutions to cover new incoming issues. Here, we got the first model, with promising results (6.2.3), the second with more complex architecture and defects handling (6.3.2), and finally, the third model, which was based on the Real-ESRGAN model, which was fine-tuned on our dataset (6.4.4). Even though the results were not as expected, we were able to create a robust pipeline for crawling the dataset, applying defects, and making it work with the model.

In the end, in our opinion, the study outcomes might be scored as successful, by addressing all expectations and our defined goals. Those results could be used as a base for further research in the field of image restoration, and also as a base for the future

work:

Future Work Addressing limitations and issues we had duration the study, we can suggest several future work directions:

- **Dataset Quality**. The dataset quality was one of the main issues we faced. We suggest to use a more diverse and high-quality dataset, which could be crawled from different sources, and filtered by more advanced metrics.
- **Hardware Limitations**. The hardware limitations we faced were significant. We suggest to handle the training and create more robust deploy strategy, that would allow also to use model directly in the browser, or on the mobile device, and without facing Docker limitations.
- **Front-end Improvements**. The front-end part of our solution is quite simple. There are many ways to improve it, such as adding more features, as authorization, real-time image processing time, image saving, sharing and more.

Resumé

Chapter je bez cislovania, pouziva sa len **boldovanie textu** na zyraznenie useku textu. Bez Figure, Table, atd. ak treba tak odvolava sa na originalne Figure a Table.

10% v slovenčine pre tých, ktorí majú prácu v EN: preklad

EN	SK
Introduction	primerane strucne
SOTA	primerane strucne
SOTA - cast Summary and Starting Points	cele
Objective and Methodology	cele
Design, Implementation	primerane strucne
Experiments and Evaluation	primerane strucne
Experiments and Evaluation - cast Summary	cele
Conclusion and Future Work	cele

Table 7.1: Resumé

7.1 Úvod

Restaurácia fotografií bola kedysi náročná a časovo i finančne nákladná, no vďaka neurónovým sieťam a hlbokému učeniu sa stala plne automatizovanou, rýchlejšou a presnejšou. Moderné metódy sa využívajú v medicínskom zobrazovaní, spracovaní satelitných snímok, kriminalistike aj autonómnych vozidlách, čo výrazne zlepšuje kvalitu, znižuje náklady a skracuje spracovací čas.

7.2 Prehľad o reštaurácii obrazu

Pred pokračovaním je potrebné upresniť pojem obnova obrazu. Ide o odstránenie rôznych typov defektov – šumu, rozmazania, kompresných artefaktov a nízkeho rozlíšenia – aby sme dosiahli ostrejší a kvalitnejší vzhľad. Šum môže mať podobu Gaussovho,

Poissonovho, speckle alebo soľ-pepper šumu. Často sa však vyskytuje kombinácia týchto javov, čo zvyšuje náročnosť riešenia. Popri týchto štyroch najbežnejších defektoch sa niekedy objavujú aj ďalšie, ako kvantizačný či vysokofrekvenčný šum, ktoré tu pre zameranie vylučujeme.

Technicky je obnova obrazu inverzným problémom: z nízkokvalitnej (LQ) alebo poškodeného záberu sa pomocou matematických a výpočtových metód obnovuje pôvodný vysokokvalitný (HQ) obraz. Pred rozšírením neurónových sietí sa používali klasické priestorové (napr. filtrácia v pixeloch) a frekvenčné (napr. Fourierova transformácia) prístupy.

Dnešné metódy hlbokého učenia (FBCNN, HCformer, DIP, ESRGAN, InstantIR) dokážu učiť z veľkých datasetov a automatizovať obnovu so zlepšenou presnosťou a rýchlosťou. Tento prístup zároveň umožňuje lepšie prispôsobenie sa špecifikám jednotlivých typov obrázkov a znižuje potrebu ručnej intervencie.

7.2.1 Zhrnutie

Obnova obrazu je tradičný a dobre známy problém, ktorý riešili a stále riešia rôzne metódy – od filtrovania a interpolácie až po pokročilé modely hlbokého učenia. Počas obdobia vývoja a vylepšovania týchto prístupov, vrátane najnovšieho modelu InstantIR, navrhnuté metódy excelujú v mnohých úlohách obnovy, hoci niektoré majú svoje obmedzenia. Preto je hlavným trendom vývoj efektívnych, výkonných univerzálnych modelov na riešenie rôznych degradácií obrazu. Medzi nedávnymi pokrokmi možno zhrnúť nasledujúce medzery:

- Špecializácia úloh: existujúce modely sú často optimalizované na konkrétne úlohy a postrádajú flexibilitu riešiť viac typov defektov naraz.

- Výpočtová náročnosť: komplexné modely, ktoré riešia viaceré defekty, sú výpočtovo intenzívne, čo obmedzuje ich použiteľnosť v reálnom čase alebo v prostrediach s obmedzenými zdrojmi.

- Závislosť od dát: pokročilé modely vyžadujú veľké párové datasety na tréning, ktoré nie sú vždy dostupné pre všetky typy degradácií.

- Škálovateľnosť: modely môžu mať problémy so škálovaním na vysoké rozlíšenie alebo adaptáciou na rôzne a nepredvídateľné degradácie v reálnych podmienkach.

7.3 Ciele a metodológia

7.3.1 Ciele

Projekt si kladie za cieľ natrénovať model a vyvinúť grafické užívateľské rozhranie prístupné externým používateľom. Podľa zhrnutia sú ciele tejto práce zamerané na vybrané úlohy na riešenie nasledujúcich výziev:

- Vytvorenie pracovného postupu pre dátovú sadu
  - Vytvoriť robustný pracovný postup schopný načítavať obrázky z online repozitárov, lokálnych adresárov a cloudového úložiska.
  - Použiť mapovanie a predspracovacie kroky, ako je augmentácia, na prípravu dát na trénovanie.
  - Riešiť defekty obrázkov aplikáciou rôznych typov skreslenia a umožniť riadenie intenzity skreslenia pre postupné zvyšovanie náročnosti počas trénovania.
- Identifikovať riešenie pre zlepšenie kvality obrazu a natrénovať model
  - Navrhnúť alebo využiť existujúcu architektúru schopnú riešiť bežné defekty obrázkov v reálnych scénach.
  - Zamerať sa na zväčšovanie obrazu s integráciou redukcie šumu, odstránenia rozmazania a korekcie artefaktov (napr. pri JPEG kompresii).
  - Trénovať model na pripravenej dátovej sade a hodnotiť jeho výkon pomocou metrík PSNR, SSIM a vizuálneho posúdenia.
- Zabaliť riešenie do API a webovej aplikácie
  - Zabaliť riešenie do Docker kontajnerov pre použitie na viacerých zariadeniach.
  - Vyvinúť ľahké API na hosťovanie modelu a poskytnutie jednoducho použiteľného rozhrania.
  - Vyvinúť webovú aplikáciu na predvedenie funkčnosti modelu.

7.3.2 Metodológia

Pre návrh a vývoj navrhovaného riešenia boli zvolené tieto kroky:

- Krok 1. Vytvorenie pracovného postupu na načítanie, predspracovanie a augmentáciu obrázkov z rôznych zdrojov.

- Krok 2. Určenie architektúry na zlepšenie kvality obrazu pri zachovaní širokého spektra defektov.
- Krok 3. Trénovať alebo dolaďovať model na pripravenej dátovej sade a hodnotiť jeho výkon metrikami PSNR, SSIM a vizuálnym posúdením.
- Krok 4. Vytvorenie ľahkého API na poskytovanie modelu a zabezpečenie užívateľsky prístupného rozhrania vrátane webovej aplikácie na demonštráciu schopností systému.
- Krok 5. Zabaliť riešenie pomocou technológie Docker kontajnerov.

7.3.3 Hodnotiace metriky

Na hodnotenie modelu sa budú používať tieto metriky: PSNR, SSIM, LPIPS, čas inferencie a metriky BIQA.

7.3.3.1 Pomerný pomer signálu k šumu (PSNR)

PSNR je objektívna metrika na meranie kvality rekonštruovaných obrázkov vzhľadom na originály. Definuje sa ako. Vyššia hodnota PSNR znamená lepšiu vernosť originálu, avšak často nekoreluje presne s vnímanou vizuálnou kvalitou, preto sa dopĺňa ďalšími metrikami.

7.3.3.2 Index štrukturálnej podobnosti (SSIM)

SSIM je citlivý na štrukturálne informácie, jas a kontrast. SSIM kladie vyšší dôraz na štruktúru a lepšie koreluje s ľudským vnímaním než PSNR.

7.3.3.3 Hodnotenie kvality obrazu bez referencie (BIQA)

BIQA metriky hodnotia kvalitu obrazu bez referenčného obrazového originálu:

- NIQE (Naturalness Image Quality Evaluator): meria odchýlku štatistík od prirodzených scén.
- BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator): hodnotí kvalitu na základe priestorových štatistík.

BIQA poskytuje flexibilné a praktické hodnotenie kvality obrazu v reálnych aplikáciách, kde nie je k dispozícii referenčný obraz.

7.4 Návrh AI obnovy obrazu

Na základe (4) a (3) navrhujeme architektúru rozdelenú do fáz:

- **Ingest dát**: načítanie a predspracovanie surových obrázkov.
- **Dáta & Dataloadery**: organizácia datasetov, lazy loading, generovanie LR–HR párov so skresleniami, rozdelenie na tréning/validáciu.
- **Tréning & hodnotenie**: Jupyter Notebook & Python; trénovanie modelu, výpočet metrík (PSNR, SSIM) a ich vizualizácia.
- **Nasadenie**: hostovanie modelu cez API, škálovateľná webová aplikácia, load balancing.

Grafické používateľské rozhranie Webová aplikácia pre všetky zariadenia s prehliadačom (React-Native/Ionic), prístup k natívnym funkciám (kamera, GPS), jednoduché aktualizácie.

Architektúra servera Možnosti implementácie v Python/JavaScript/TypeScript, mikroservisná štruktúra podľa (4.1.3): každý model v samostatnom procese s vlastným API a hlavným routovacím serverom.

Stratégia nasadenia Využitie Docker kontajnerov pre izoláciu, prenosnosť a spoľahlivé nasadenie podľa (4.2.1).

7.5 Implementácia riešenia full-stack obnovy

Na základe návrhu (4) sme implementovali riešenie v týchto fázach:

Zber a spracovanie dát Lazy loading a spracovanie metadát z Pexels API a vlastných datasetov (5.1.2). Obrázky ukladané v stromovej štruktúre, filtrované kvalitou (rozostrenie, expozícia, BIQA).

Aplikácia defektov Simulácia Gaussovho, Poissonovho, speckle a soľ–pepper šumu, rozmazania, JPEG kompresie a zníženého rozlíšenia s náhodnou intenzitou.

Modely Vytvorili sme dve CNN architektúry so zvyškovými a up-sample blokmi a Dense Residual blokmi (5.2.2.1). Modely natrénované v Jupyter Notebooku. Plus, fine-tune Real-ESRGAN (5.2.3).

API a aplikácia Backend v Pythone (Flask), jednoduché endpointy pre obnovu a aplikáciu defektov (5.3.1, 5.3.2). Frontend s Next.js a React pre plnohodnotnú webovú aplikáciu (5.3.3).

Nasadenie Kontajnery Docker pre izoláciu a prenosnosť (4.2), model konvertovaný do NCNN pre rýchlu inferenciu pomocou chaiNNer (5.4.1).

Manažment projektu Git/GitHub pre verzovanie, issue tracking a modulárne repozitáre.

7.6 Experimenty a hodnotenie

7.6.1 Nastavenie

Všetky experimenty prebiehali v prostredí Google Colab [90], ktoré poskytuje výpočtové prostriedky.

7.6.1.1 Prípadová štúdia – Experiment 01

Zber dát Naša lazy pipeline načítala 5000 obrázkov z kategórie `nature`, zmenšených na 512×512 HQ a 256×256 LQ. Dáta sme rozdelili 80% pre tréning a 20% pre validáciu, bez ďalšieho filtrovania či pridávania artefaktov.

Proces tréningu Tréning prebehol na GPU A100 s batch 16, 20 epoch. Použili sme L1 obsahovú stratu + VGG19 perceptuálnu, optimizer Adam (lr=1e-4) a StepLR (=0.5 po 20 epochách).

Výsledky Straty klesali, metriky PSNR a SSIM rástli z 25.5 dB/.85 pri 13. epoche na 31.8 dB/.86 pri 20. epoche. NIQE pre tri vzorky boli 4.13, 5.44 a 7.41. Inferenčný čas sa pohyboval medzi 0.45–0.51 ms. Obnovené obrázky výrazne prevyšujú LQ vstupy.

Diskusia Navrhnutá CNN architektúra dokáže efektívne obnovovať detaily, no je limitovaná prácou na 256×256 výrezoch, špecifickou doménou a krátkym trvaním tréningu. Napriek tomu dosahuje vysoké kvalitatívne výsledky a nízku latenciu vhodnú pre realtime aplikácie.

7.6.1.2 Prípadová štúdia – Experiment 02

Experiment využil rozšírenú architektúru (experiment 01 5.2.2.2).

Predspracovanie dát Rovnaká lazy pipeline, HQ obrázky z Pexels. Pridali sme náhodné defekty (downscale, šum, rozmazanie, JPEG), rotácie, flip a orez/maskovanie.

Výsledky Po 10 epochách strata neklesala a PSNR zostalo nízke (pozri 6.5), tréning zastavený.

Záver Napriek Dense blokom model nedosiahol očakávané obnovenie defektov, metriky sa nezlepšili.

7.6.1.3 Prípadová štúdia – Experiment 03

Experiment zameraný na fine-tuning Real-ESRGAN (6.4.1).

Výber modelu Real-ESRGAN (ESRGAN, DIV2K, Flickr2K, OST) pre otvorený kód a predtrénované váhy.

Testovanie pred fine-tuningom Pôvodný model odstraňoval defekty, no postrádal ostré detaily (6.4.1).

Príprava dát HQ patch 480×480, LQ 120×120, stride 256, sliding window; >69 000 párov (6.4.2).

Tréning BasicSR finetuning, batch 2, 5000 epôch, L1+percep+GAN+context loss, Adam lr=1e-5, MultiStepLR (6.4.3).

Výsledky PSNR/SSIM vzrástli z 28.57/0.82 na 30.05/0.87 po 5000 epóhach, no BIQA a vnímaná kvalita sa zhoršili, so zmiešanými príkladmi lepších i horších výstupov (6.4.4).

Záver Pipeline fine-tuningu fungovala, no detaily ustúpili a výsledky boli nekonzistentné, pravdepodobne kvôli datasetu a malej veľkosti batchu (6.4.5).

7.7 Zhrnutie

Hlavné *kľúčové zistenia* na základe výsledkov časti (6) sú:

- **Kompromis medzi kvalitou a efektivitou**. Hlbšie a zložitejšie architektúry konzistentne zlepšujú kvalitu obnovy, no za cenu dlhšieho času trénovania a vyššej pamäťovej náročnosti na GPU, čo predstavuje významné obmedzenie.

- **Výhoda prenosového učenia**. Využitie predtrénovaného GAN modelu (Real-ESRGAN) nám umožnilo dosiahnuť špičkové percepčné výsledky s menším počtom epoch než pri trénovaní od nuly.
- **Hardvérové obmedzenia majú stále význam**. Aj pri výkonnom GPU A100 nás limity relácií v Colabe a maximálna kapacita pamäte na jednom GPU nútili robiť kompromisy (napr. menšie batch-e, zmenšovanie rozlíšenia obrázkov alebo delenie na dlaždice), ktoré mohli mierne znížiť skutočný potenciál modelu.

Naprieč všetkými tromi postupne náročnejšími prípadovými štúdiami sme pozorovali rovnaký trend: s rastúcou zložitosťou modelu a množstvom predchádzajúcich znalostí sa zvyšovala aj kvalita obnovy — za cenu rastúcich výpočtových nákladov. V prvom Experimente (6.2) náš jednoduchý CNN, trénovaný od nuly, dosiahol dobré hodnoty PSNR a SSIM, ale mal problém obnoviť jemné detaily na silne degradovaných vstupoch. Príkladom toho je druhý Experiment (6.3), ktorý zahŕňal modul pre defekty a zložitejšiu architektúru, čo viedlo k horším skóre PSNR a SSIM už v prvých epochách. Nakoniec tretí Experiment (6.4) využil sofistikovanejší predtrénovaný model Real-ESRGAN na našom vlastnom dátovom súbore s vylepšenými metrikami slepoty a percepcie.

Napriek sľubným výsledkom je naša práca obmedzená niekoľkými kľúčovými *limitáciami*, najmä vzhľadom na nedostatok špecializovaných znalostí a výpočtovej kapacity:

- **Znalosti pri vývoji na mieru šitých modelov**. Hoci sme sa snažili navrhnúť optimálnu AI architektúru pre obnovu obrázkov, neskúsenosť nášho tímu v tvorbe a optimalizácii nových hlbokých modelov pravdepodobne obmedzila rozsah architektonických prieskumov. Bežné benchmarky a predkonfigurované rámce prevzali väčšinu práce, čo mohlo zabrániť objaveniu účinnejších alebo efektívnejších sieťových architektúr.
- **Obmedzenia času výpočtu**. Celé trénovanie modelov sme realizovali na Google Colab, ktorý uplatňuje prísny limit 12 hodín na reláciu s ďalšími pravidlami na udržanie aktivity v karte. Tento časový limit nám sťažoval úplné trénovanie hlbších alebo zložitejších variant, najmä tých, ktoré vyžadujú dlhší čas na konvergenciu. To znamenalo, že niektoré konfigurácie sietí boli prerušené v polovici, pravdepodobne ešte pred dosiahnutím ich optimálneho výkonu.
- **Obmedzenia jedného GPU a pamäte**. Naše experimenty boli obmedzené na jeden GPU (použili sme maximálne povolený - NVIDIA A100) s limitovanou kapacitou VRAM. Vstupné obrázky vo vysokom rozlíšení – dôležité pre schopnosť rozlišovať jemné štrukturálne detaily – sa nedali spracovať v dávkach väčších ako len niekoľko bez prekročenia pamäte, a boli sme nútení zmenšovať rozlíšenie alebo znižovať veľkosť batch-ov pri trénovaní a dolaďovaní. Tieto kompromisy mohli obmedziť

model v učení jemnejších funkcií obnovy a znížiť generalizovateľnosť našich zistení na skutočne veľkoformátové, vysoko kvalitné obrázky.

- **Nedostatok kvalitného datasetu**. V našej implementácii sme sa rozhodli vytvoriť a použiť vlastný dataset pre trénovanie a dolaďovanie modelov. Aj keď boli obrázky vybrané z vysoko kvalitných zdrojov a filtrované pomocou metrík ako blind image quality assessment (BIQA) na zvýšenie kvality datasetu znížením počtu nekvalitných obrázkov.

Ako vždy, znovuobjavovanie kolesa je dvojsečný meč. Hoci sme ukázali, že je možné dosiahnuť špičkové výsledky s vlastnou architektúrou, čas a úsilie potrebné na to nemusia stáť za to vo všetkých prípadoch. V mnohých prípadoch môže byť efektívnejšie použiť existujúci model a doladiť ho na konkrétne úlohy pomocou nášho vlastného datasetu, ktorý je cennejší než samotný model. Ako už bolo povedané, znovuobjavovanie alebo tvorba niečoho úplne nového je náročná úloha, ale ak sú k dispozícii zdroje – intelektuálne aj výpočtové – stojí za to to skúsiť.

7.8 Záver

Naším cieľom bolo navrhnúť a implementovať dockerizované full-stack riešenie v problematike obnovy obrázkov. V priebehu celej štúdie sme intenzívne pracovali na každej časti riešenia:

Preskúmali sme súčasný stav obnovy obrázkov, a to tradičnými procesmi obnovy (metódy v priestorovej a frekvenčnej doméne), ako aj aplikáciou jednoduchých a hlbokých sietí s rôznorodými architektúrami, vrátane CNN (2.2.2), GAN (2.2.3) a ďalších komplexnejších riešení.

Definovali sme naše ciele a úlohy, s jasnými očakávaniami od výslednej práce, ktoré pokrývajú všetky časti a etapy nášho systému (4.1).

Navrhli a implementovali sme architektúru full-stack riešenia, ktorá sprevádza proces obnovy obrázkov od spracovania datasetu (5.1) cez finálne trénovanie až po dolaďovanie modelu (4.1.1.3). Na základe našich očakávaní sme do zdrojového kódu pridali webové GUI a Python server, pričom každá časť je kontajnerizovaná pomocou Dockeru, čo umožňuje spustenie v ľubovoľnom prostredí.

Po viacerých iteráciách vývoja sme vyvinuli nové riešenia pre riešenie vznikajúcich problémov. Získali sme prvý model so sľubnými výsledkami (6.2.3), druhý model s komplexnejšou architektúrou a spracovaním defektov (6.3.2), a nakoniec tretí model založený na Real-ESRGAN, jemne dolaďovaný na našom datasete (6.4.4). Hoci výsledky neboli úplne podľa očakávaní, podarilo sa nám vytvoriť robustný pipeline pre získavanie

datasetu, aplikáciu defektov a jeho integráciu s modelom.

Na záver, podľa nášho názoru, možno výsledky štúdie považovať za úspešné, keďže boli splnené všetky očakávania a definované ciele. Tieto výsledky môžu slúžiť ako základ pre ďalší výskum v oblasti obnovy obrázkov a ako východisko pre budúcu prácu:

Budúca práca Riešenie obmedzení a problémov, s ktorými sme sa stretli počas štúdie, navrhujeme niekoľko smerov budúcej práce:

- **Kvalita datasetu**. Kvalita datasetu bola jedným z hlavných problémov. Odporúčame použiť rozmanitejší a kvalitnejší dataset, ktorý možno získať z rôznych zdrojov a filtrovať pomocou pokročilých metrík.
- **Hardvérové obmedzenia**. Hardvérové obmedzenia boli významné. Odporúčame optimalizovať proces trénovania a vytvoriť robustnejšiu stratégiu nasadenia, ktorá by umožnila používanie modelu priamo v prehliadači alebo na mobilnom zariadení bez stretávania sa s obmedzeniami Dockeru.
- **Vylepšenia front-endu**. Front-end našej aplikácie je pomerne jednoduchý. Existuje množstvo možností na jeho vylepšenie, napríklad pridanie autorizácie, zobrazovanie času spracovania obrázku v reálnom čase, možnosť ukladania a zdieľania obrázkov a ďalšie.

List of Figures

List of Tables

References

[1] Yunfan Lu et al. *Priors in Deep Image Restoration and Enhancement: A Survey*. 2023. arXiv: 2206.02070 [cs.CV]. URL: https://arxiv.org/abs/2206.02070.

[2] Junjun Jiang et al. *A Survey on All-in-One Image Restoration: Taxonomy, Evaluation and Future Trends*. 2024. arXiv: 2410.15067 [cs.CV]. URL: https://arxiv.org/abs/2410.15067.

[3] Amanturdieva Akmaral and Muhammad Hamza Zafar. *Efficient Transformer for High Resolution Image Motion Deblurring*. 2025. arXiv: 2501.18403 [cs.CV]. URL: https://arxiv.org/abs/2501.18403.

[4] Boyun Li et al. *MaIR: A Locality- and Continuity-Preserving Mamba for Image Restoration*. 2025. arXiv: 2412.20066 [cs.CV]. URL: https://arxiv.org/abs/2412.20066.

[5] Jingwen Su, Boyan Xu, and Hujun Yin. "A survey of deep learning approaches to image restoration". In: *Neurocomputing* 487 (2022), pp. 46–65. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2022.02.046.

[6] Lujun Zhai et al. "A Comprehensive Review of Deep Learning-Based Real-World Image Restoration". In: *IEEE Access* 11 (2023), pp. 21049–21067. DOI: 10.1109/ACCESS.2023.3250616.

[7] Zhiwen Yang et al. "All-In-One Medical Image Restoration via Task-Adaptive Routing". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2024*. Ed. by Marius George Linguraru et al. Cham: Springer Nature Switzerland, 2024, pp. 67–77. ISBN: 978-3-031-72104-5. URL: https://arxiv.org/abs/2405.19769.

[8] K. A. Saneera Hemantha Kulathilake et al. "A review on Deep Learning approaches for low-dose Computed Tomography restoration". In: *Complex & Intelligent Systems* 9.3 (June 2023), pp. 2713–2745. ISSN: 2198-6053. DOI: 10.1007/s40747-021-00405-x.

[9] Xi Yang et al. "SRDN: A Unified Super-Resolution and Motion Deblurring Network for Space Image Restoration". In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022), pp. 1–11. DOI: 10.1109/TGRS.2021.3131264.

[10] Xiaoguang Tu et al. "Joint Face Image Restoration and Frontalization for Recognition". In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.3 (2022), pp. 1285–1298. DOI: 10.1109/TCSVT.2021.3078517.

[11] Haichao Zhang et al. "Close the loop: Joint blind image restoration and recognition with sparse representation prior". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 770–777. DOI: 10.1609/aaai.v36i2.20033.

[12] Xuan Di and Rongye Shi. "A survey on autonomous vehicle control in the era of mixed-autonomy: From physics-based to AI-guided driving policy learning". In: *Transportation Research Part C: Emerging Technologies* 125 (2021), p. 103008. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2021.103008.

[13] A. V. Shreyas Madhav and Amit Kumar Tyagi. "Explainable Artificial Intelligence (XAI): Connecting Artificial Decision-Making and Human Trust in Autonomous Vehicles". In: *Proceedings of Third International Conference on Computing, Communications, and Cyber-Security*. Ed. by Pradeep Kumar Singh et al. Singapore: Springer Nature Singapore, 2023, pp. 123–136. ISBN: 978-981-19-1142-2. DOI: 10.1007/978-981-19-1142-2_10.

[14] Kunpeng Zhang et al. "AI-TP: Attention-Based Interaction-Aware Trajectory Prediction for Autonomous Driving". In: *IEEE Transactions on Intelligent Vehicles* 8.1 (2023), pp. 73–83. DOI: 10.1109/TIV.2022.3155236.

[15] Hong-yu Zhang et al. "Deep Learning of Color Constancy Based on Object Recognition". In: *2023 15th International Conference on Computer Research and Development (ICCRD)*. 2023, pp. 215–219. DOI: 10.1109/ICCRD56364.2023.10080343.

[16] Weidong Min et al. "Traffic Sign Recognition Based on Semantic Scene Understanding and Structural Traffic Sign Location". In: *IEEE Transactions on Intelligent Transportation Systems* 23.9 (2022), pp. 15794–15807. DOI: 10.1109/TITS.2022.3145467.

[17] Nikolay Ponomarenko et al. "Image database TID2013: Peculiarities, results and perspectives". In: *Signal Processing: Image Communication* 30 (2015), pp. 57–77. ISSN: 0923-5965. DOI: https://doi.org/10.1016/j.image.2014.10.009.

[18] Tai-Yin Chiu, Yinan Zhao, and Danna Gurari. "Assessing Image Quality Issues for Real-World Problems". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 3643–3653. DOI: 10.1109/CVPR42600.2020.00370.

[19] Rémi Cogranne and Florent Retraint. "Statistical detection of defects in radiographic images using an adaptive parametric model". In: *Signal Processing* 96 (2014), pp. 173–189. ISSN: 0165-1684. DOI: https://doi.org/10.1016/j.sigpro.2013.09.016.

[20] Yiyu Li et al. *Color Shift Estimation-and-Correction for Image Enhancement*. 2024. arXiv: 2405.17725 [cs.CV]. URL: https://arxiv.org/abs/2405.17725.

[21] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2002. ISBN: 0-13-168728-X. URL: https://www.pearson.com/en-us/subject-catalog/p/digital-image-processing/P200000003002/9780131687288.

[22] Robert G. Keys. "Cubic Convolution Interpolation for Digital Image Processing". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29.6 (1981), pp. 1153–1160. DOI: 10.1109/TASSP.1981.1163711.

[23] Ronald N. Bracewell. *The Fourier Transform and Its Applications*. 3rd ed. New York, NY: McGraw-Hill, 2000. ISBN: 0-07-118927-7. URL: https://www.mheducation.com.

[24] Norbert Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. Cambridge, MA: MIT Press, 1949. ISBN: 9780262191928. URL: https://mitpress.mit.edu/9780262191928/extrapolation-interpolation-and-smoothing-of-stationary-time-series/.

[25] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. ISBN: 9780070428072. URL: https://www.cs.cmu.edu/~tom/mlbook.html.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN: 9780262035611. URL: https://www.deeplearningbook.org.

[27] Dick De Ridder et al. "Nonlinear image processing using artificial neural networks". In: ed. by Peter W. Hawkes. Vol. 126. Advances in Imaging and Electron Physics. Elsevier, 2003, pp. 351–450. DOI: https://doi.org/10.1016/S1076-5670(03)80019-8.

[28] Alex S. Palmer, Moe Razaz, and Danilo P. Mandic. "Spatially Adaptive Image Restoration by Neural Network Filtering". In: *University of East Anglia, School of Information Systems* (2002). DOI: 10.48550/arXiv.2108.08617.

[29] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647.

[30] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 9780387310732. URL: https://link.springer.com/book/10.1007/978-0-387-45528-0.

[31] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[32] Yan Lv and Hui Ma. "Improved SRCNN for super-resolution reconstruction of retinal images". In: *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*. 2021, pp. 595–598. DOI: 10.1109/ICSP51882.2021.9408850.

[33] Yuning Cui et al. "Revitalizing Convolutional Network for Image Restoration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.12 (2024), pp. 9423–9438. DOI: 10.1109/TPAMI.2024.3419007.

[34] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 27. 2014. URL: https://papers.nips.cc/paper_files/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

[35] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. 2017. URL: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[36] Jingyun Liang et al. "SwinIR: Image Restoration Using Swin Transformer". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*. 2021, pp. 1833–1844. DOI: 10.48550/arXiv.2108.10257.

[37] Christian Ledig et al. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network". In: *CVPR* (2017), pp. 4681–4690. DOI: 10.1109/CVPR.2017.19. URL: https://ieeexplore.ieee.org/document/8099502.

[38] Ruyu Liu et al. "A Transformer-Based Network for Multi-Stage Progressive Image Restoration". In: *2024 6th International Conference on Data-driven Optimization of Complex Systems (DOCS)*. 2024, pp. 393–401. DOI: 10.1109/DOCS63458.2024.10704475.

[39] Jiaxi Jiang, Kai Zhang, and Radu Timofte. "Towards flexible blind JPEG artifacts removal". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 4997–5006. DOI: 10.48550/arXiv.2109.14573.

[40] Jinli Yuan et al. "HCformer: Hybrid CNN-Transformer for LDCT Image Denoising". In: *Journal of Digital Imaging* 36.5 (2023). Epub 2023 Jun 29, pp. 2290–2305. ISSN: 1618-727X. DOI: 10.1007/s10278-023-00842-9.

[41] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Deep Image Prior". In: *International Journal of Computer Vision* 128.7 (Mar. 2020), 1867–1888. ISSN: 1573-1405. DOI: 10.1007/s11263-020-01303-4.

[42] "Deep Image Prior Amplitude SAR Image Anonymization". In: *Remote Sensing* 15.15 (2023). ISSN: 2072-4292. DOI: 10.3390/rs15153750.

[43] Xintao Wang et al. "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks". In: *Computer Vision – ECCV 2018 Workshops*. Ed. by Laura Leal-Taixé and Stefan Roth. Cham: Springer International Publishing, 2019, pp. 63–79. ISBN: 978-3-030-11021-5.

[44] Xintao Wang et al. "Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data". In: 2021. arXiv: 2107.10833 [eess.IV]. URL: https://arxiv.org/abs/2107.10833.

[45] VRNord. *ESRGAN is one of the best tools for texture enhancing*. Reddit: r/skyrimmods. 2019. URL: https://www.reddit.com/r/skyrimmods/comments/go6bls/esrgan_is_one_of_the_best_tools_for_texture/.

[46] Jen-Yuan Huang et al. *InstantIR: Blind Image Restoration with Instant Generative Reference*. 2024. arXiv: 2410.06551 [cs.CV]. URL: https://arxiv.org/abs/2410.06551.

[47] Le Chang et al. "miRNet 2.0: network-based visual analytics for miRNA functional analysis and systems biology". In: *Nucleic acids research* 48.W1 (2020), W244–W251. DOI: 10.1093/nar/gkaa467.

[48] Lujun Zhai et al. "A Comprehensive Review of Deep Learning-Based Real-World Image Restoration". In: *IEEE Access* 11 (2023), pp. 21049–21067. DOI: 10.1109/ACCESS.2023.3250616.

[49] Docker Inc. *Docker: Empowering App Development for Developers*. https://www.docker.com. Accessed: 2025-05-12. 2024.

[50] National Instruments. "Peak Signal-to-Noise Ratio as an Image Quality Metric". In: *NI* (2024). URL: https://www.ni.com/en/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/peak-signal-to-noise-ratio-as-an-image-quality-metric.html.

[51] Zhou Wang et al. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[52] Anish Mittal, Rajiv Soundararajan, and Alan C. Bovik. "Making a "Completely Blind" Image Quality Analyzer". In: *IEEE Signal Processing Letters* 20.3 (2013), pp. 209–212. DOI: 10.1109/LSP.2012.2227726.

[53] Anish Mittal, Anush Krishna Moorthy, and Alan Conrad Bovik. "Blind/Referenceless Image Spatial Quality Evaluator". In: *Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*. 2011, pp. 723–727. DOI: 10.1109/ACSSC.2011.6190099.

[54] Thomas Kluyver et al. "Jupyter Notebooks—a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87.

[55] Python Software Foundation. *Python Programming Language*. Version 3.x. 2024. URL: https://www.python.org.

[56] Inc. Meta Platforms. *React Native*. 2024. URL: https://reactnative.dev.

[57] Ionic Team. *Ionic Framework*. 2024. URL: https://ionicframework.com.

[58] Ecma International. *ECMAScript Language Specification*. https://www.ecma-international.org/publications-and-standards/standards/ecma-262/. ECMA-262, 13th Edition. 2023. URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/.

[59] Microsoft Corporation. *TypeScript: JavaScript with Syntax for Types*. Version 5.x. 2024. URL: https://www.typescriptlang.org.

[60] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015. ISBN: 9781491950357. URL: https://www.oreilly.com/library/view/building-microservices/9781491950340/.

[61] Wikipedia contributors. *Docker (software)*. https://en.wikipedia.org/wiki/Docker_(software). Accessed: 2025-05-12. 2024.

[62] Pexels. *Pexels License Information*. https://www.pexels.com/license/. License details for using free stock photos and videos from Pexels, including rights and restrictions for personal and commercial use. Accessed: 2025-01-03. 2025.

[63] Anish Mittal, Anush Krishna Moorthy, and Alan C. Bovik. "No-reference Image Quality Assessment in the Spatial Domain". In: *IEEE Transactions on Image Processing* 21.12 (2012), pp. 4695–4708. DOI: 10.1109/TIP.2012.2214050.

[64] Eirikur Agustsson and Radu Timofte. "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2017, pp. 1124–1133. DOI: 10.1109/CVPRW.2017.150.

[65] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[66] Wenzhe Shi et al. "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 1874–1883. DOI: 10.1109/CVPR.2016.207.

[67] Gao Huang et al. "Densely Connected Convolutional Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4700–4708. DOI: 10.1109/CVPR.2017.243.

[68] R. Nandhini Abirami et al. "Deep CNN and Deep GAN in Computational Visual Perception-Driven Image Analysis". In: *Complexity* 2021.1 (2021), p. 5541134. DOI: https://doi.org/10.1155/2021/5541134. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/5541134.

[69] Yulun Zhang et al. "Residual Dense Network for Image Super-Resolution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018, pp. 2472–2481.

[70] Tong Tong et al. "Image Super-Resolution Using Dense Skip Connections". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 4809–4817.

[71] Kuldeep Purohit, Srimanta Mandal, and A. N. Rajagopalan. "Scale-Recurrent Multi-Residual Dense Network for Image Super-Resolution". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. Sept. 2018.

[72] xinntao. *Real-ESRGAN ncnn Vulkan*. https://github.com/xinntao/Real-ESRGAN-ncnn-vulkan. Acesssed: 2025-05-12. 2025. URL: https://github.com/xinntao/Real-ESRGAN-ncnn-vulkan.

[73] Bee Lim et al. *Flickr2K dataset*. 2024. DOI: 10.57702/gqmt7i52.

[74] Yuxin Wang et al. *From Two to One: A New Scene Text Recognizer with Visual Language Modeling Network*. 2021. arXiv: 2108.09661 [cs.CV]. URL: https://arxiv.org/abs/2108.09661.

[75] Miguel Grinberg. *Flask: Web Development, One Drop at a Time*. https://flask.palletsprojects.com/. Accessed: 2025-05-12. 2018.

[76] Armin Ronacher and the Pallets team. *Flask*. https://pypi.org/project/Flask/. Accessed: 2025-05-12. 2024.

[77] InfoWorld Editorial Team. *Next.js 13.5 brings faster startups and refreshes*. Credit: Jamesboy Nuchaikong / Shutterstock; accessed 01 May 2025. Sept. 2023. URL: https://www.infoworld.com/article/2334928/nextjs-135-brings-faster-startups-and-refreshes.html.

[78] Pau Sánchez. *Frontend SSR frameworks benchmarked: Angular, Nuxt, NextJs and SvelteKit*. June 2024. URL: https://www.pausanchez.com/en/articles/frontend-ssr-frameworks-benchmarked-angular-nuxt-nextjs-and-sveltekit/.

[79] TechEmpower Framework Benchmarks Project. *Framework Benchmarks – Round 23 (TechEmpower)*. GitHub; accessed 01 May 2025. Comprehensive, language-agnostic performance comparisons across dozens of backend and full-stack frameworks, including raw plaintext and JSON serialization tests. 2025. URL: https://github.com/TechEmpower/FrameworkBenchmarks.

[80] Sonu Jangra. "React vs Svelte: A Performance Benchmarking". In: *DEV Community* (Jan. 2024). Compares load time, bundle size, render/update speed, and memory usage between React and Svelte. URL: https://dev.to/im_sonujangra/react-vs-svelte-a-performance-benchmarking-33n4.

[81] Vercel. *Next.js Documentation*. https://nextjs.org/docs. Accessed: 2025-05-12. 2025.

[82] Meta and the React Community. *React Documentation*. https://react.dev/. Accessed: 2025-05-12. 2025.

[83] Hui Ni and the ncnn contributors. *ncnn: High-Performance Neural Network Inference Framework*. Accessed: 2025-05-12. 2017. URL: https://github.com/Tencent/ncnn.

[84] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019, pp. 8024–8035. URL: https://papers.nips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[85] Aaron Liu and the Upscayl Contributors. *Model Conversion Guide*. https://github.com/upscayl/upscayl/wiki/Model-Conversion-Guide. Accessed: 2025-05-12. 2024.

[86] chaiNNer Contributors. *chaiNNer: Node-Based Image Processing GUI*. https://github.com/chaiNNer-org/chaiNNer. Accessed: 2025-05-12. 2025.

[87] NVIDIA Corporation. *CUDA Toolkit Documentation*. https://docs.nvidia.com/cuda/. Accessed: 2025-05-12. 2024.

[88] Linus Torvalds and the Git Community. *Git - Fast Version Control System*. https://git-scm.com/. Accessed: 2025-05-12. 2024.

[89] Inc. GitHub. *GitHub: Where the World Builds Software*. https://github.com/. Accessed: 2025-05-12. 2024.

[90] Google Research. *Google Colaboratory*. https://colab.research.google.com/. Accessed: 2025-05-12. 2024.

[91] Xintao Wang et al. *BasicSR: Open Source Image and Video Restoration Toolbox*. https://github.com/XPixelGroup/BasicSR. 2022.

Appendix A

Description of Digital Submission

Thesis Evidence Number:
FIIT-100241-116700

The content of the electronic medium is larger than 1GB (dataset) and is therefore handed over to the thesis supervisor (doc. Ing. Giang Nguyen Thu, PhD.).

Name of submitted archive: `BP_ArturKozubov.zip`

The digital part included in the thesis contains the following files:

```
├── application/                              <-- Full-stack application source code folder, divided into modules
│   ├── apps/
│   │   ├── defects-applying/                 <-- Python module for applying defects to images
│   │   └── web/                              <-- Next.js front-end application
│   ├── models/
│   │   └── Real-ESRGAN_x4plus-finetuned/     <-- Python module for hoisting the fine-tuned model
│   │       ├── models/
│   │       │   ├ net_g_latest_*.pth*         <-- Fine-tuned model weights in PyTorch format
│   │       │   └ net_g_latest_*.bin*/.param* <-- Fine-tuned model weights in NCNN format
│   │       ├── realesrgan-ncnn-vulkan*       <-- Linux executable for running the model
│   │       └── realesrgan-ncnn-vulkan.exe*   <-- Windows executable for running the model
│   ├── docker-compose.yml*                   <-- Docker Compose file for running the application
├── latex/                                    <-- LaTeX files for the paper with all assets and scripts for draws
├── projects/
│   ├── denoising/                            <-- Several projects for learning and practice in image restoration
│   ├── digit-classification-scratch/         <-- .
│   ├── mnist-denoising/                      <-- .
│   ├── mri-brain-denoising-nn/               <-- .
│   ├── simple-autoencoder/                   <-- .
│   └── super-resolution-nn/                  <-- Main codes for the image restoration model
│       ├── degrade_image.py*                 <-- Python script for degrading images
│       ├── degraded_dataset.py*              <-- Dataset class that uses degradation function
│       ├── pexels_lazy_dataset.py*           <-- Handles lazy loading of images from Pexels
│       ├── quality_filter.py*                <-- Python script for filtering images based on several metrics
│       ├── real-esrgan/                      <-- Real-ESRGAN Git submodule, fork of open-source repository
│       ├── requirements.txt*                 <-- Python requirements file
│       ├── v1.main.ipynb*                    <-- Jupyter notebook that was used to train the first model
│       ├── v2.main.ipynb*                    <-- Jupyter notebook that was used to train the second model
│       └── v3.main.ipynb*                    <-- Jupyter notebook that was used to fine-tune the final model

├── datasets/
│   ├── fine-tune/                            <-- The final dataset used for fine-tuning the Real-ESRGAN model
```

Figure A.1: The structure of the ZIP file

Appendix B

Technical Documentation

B.1 Installation Manual

1. Clone the project from the Git repository (URL) with populating all submodules OR unzip the ZIP file.

2. **Application**:

   - Install Docker and Docker Compose.

- Run the build command `docker-compose build` and `docker-compose up` in the `./application` directory. To support the GPU acceleration - follow the instructions in corresponding to your system. Or - run on bare metal with installing Python 3.11 and all dependencies from the `requirements.txt` files in each sub-directories: `./application/models/Real-ESRGAN`$_x$4$plus-finetuned$, `./application/apps/defects-apply` $installfrontenddependencies\ via$ `npm ci` $in\ the$ `./application/apps/wep`$directory(copy\ endapplicationwith$ `npm start` $andpythonmodulesas$ `python server.py` .

3. **Training & Fine-tuning Jupiter Notebook Pipeline**:

   - Install Jupyter Notebook with Python.

   - Corresponding to each training model - install Python dependencies from the `./projects/super-resolution-nn/requirements.txt` file. For fine-tuning the Real-ESRGAN - refer to their installing instructions or go through `v3.main.ipynb` notebook.

B.2 User Manual

1. To start the application - refer to the Installation Manual in Section (B.1).
2. The application is designed to run in a web browser. The default URL is `http://localhost:3000`.
3. Adjust the language, theme switchers, controls, as to degrade image or keep original input (B.1)

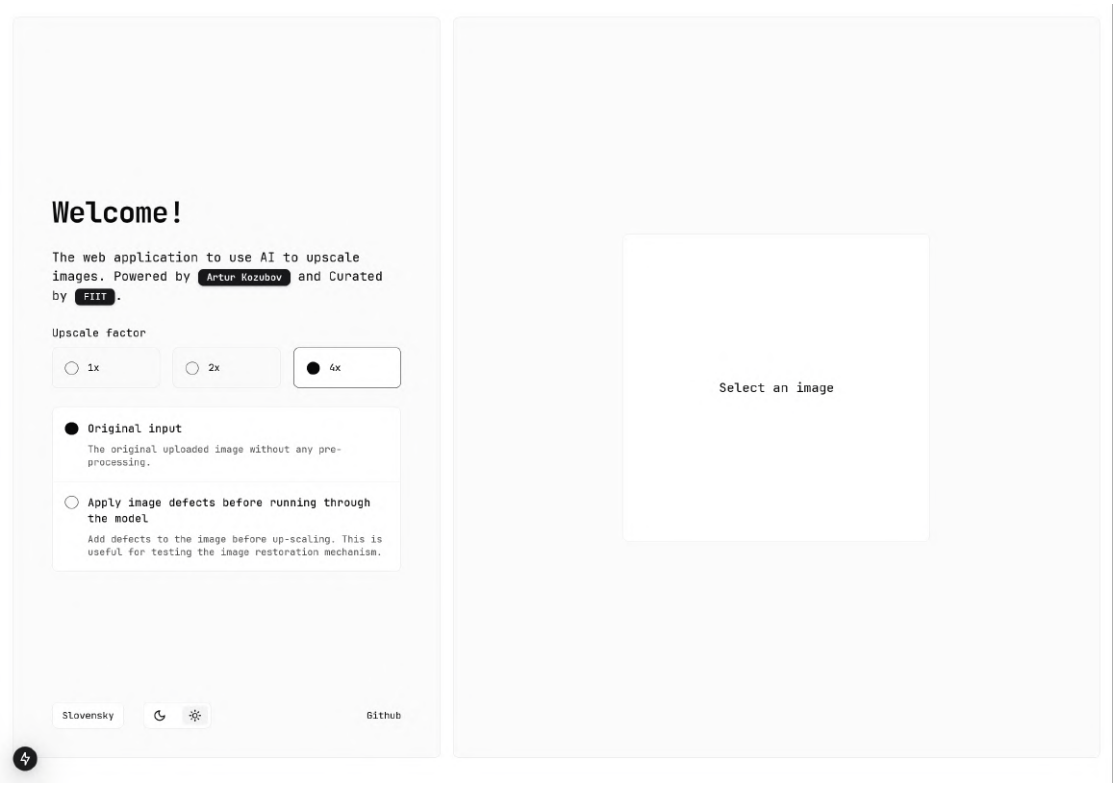

Figure B.1: Screenshot of the application

Appendix C

Work Schedule

C.1 Work Schedule for BP1

1. **Weeks 1-2: Research and Familiarization**
    - Understand the basics of photo restoration and AI techniques.
    - Explore the current trends and challenges in the field.
2. **Weeks 3-4: Literature Review**
    - Conduct a comprehensive review of related work.
    - Identify key models and methodologies used in photo restoration.
3. **Weeks 5-6: Problem Definition and Goal Setting**
    - Define specific problems to address within photo restoration.
    - Set clear and measurable project goals.
4. **Weeks 7-8: Data Collection and Preparation**
    - Gather diverse datasets relevant to photo restoration.
    - Make up scraping additional images if necessary and preprocess the data.
5. **Weeks 9-10: Model Development**
    - Design and implement the CNN-based restoration model.
    - Implement to be possible to handle continuos super-resolution.
    - Optionally explore GAN-based approaches for comparison.

6. **Weeks 11-12: Training and Optimization**
    - Train and evaluate the restoration model.
7. **Weeks 13-14: Experimentation and Evaluation**
    - Conduct experiments to evaluate the model's performance.

Self-Evaluation of Work Schedule for BP1: Due to the enormous problems and the novelty of the topic to the stident there were delays in the planned plan, but by the end it turned out to be successful to stick to the plan and overall outcomes.

C.2 Work Schedule for BP2

1. **Weeks 1-2: FIIT IAU Course | Research and investigate the current state of course**
    - Review the IAU course materials and code.
    - Discuss with with the supervisor and students about code and materials challenges.
2. **Weeks 3-4: Model Training with wider dateset, metrics, defects, and improvements**
    - Modify the dataset loading to prepare for training model to remove defects, such as noise, blur, jpeg artifacts.
    - Increase the dataset size, number of epochs.
    - Utilize more metrics to evaluate the model. Compare results, with other models.
3. **Weeks 5-6: Model Optimization**
    - Optimize the model to improve performance and efficiency.
4. **Weeks 7-8: FIIT IAU Course | Enchace the course materials**
    - Enhance the course materials based on students feedback.
    - Prepare the final version of the course materials.
5. **Weeks 9-10: Model packing and API development**
    - Compile the model and create.
    - Create RESTful API that exposes the model's functionality.
6. **Weeks 11-12: Web Application Development**
    - Create a lightweight web application that integrates with the model back-end API.
7. **Weeks 13-14: Final Testing and Deployment**
    - Conduct final testing and debugging of the web application.
    - Deploy the application to a cloud platform.
    - Prepare the final report.

Self-Evaluation of Work Schedule for BP1: As was tell in Conclusion chapter (7) our work is considered satisfied and successful. Even with small difficult to stay with plan - we achived our goals.