Unsupervised Image Segmentation and Denoising

Project Report Submitted in Partial Fulfilment of the Requirements for the Degree of

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

SATHWIK VINTHA: 2110110469
HRISHIK PYLA: 2110110405
NALLAM MANI SHAKAR: 2110110341

Under the Supervision of

Instructor: Dr Saurabh Shigwan
Associate Professor, Department of Computer Science and Engineering

SHIV NADAR UNIVERSITY

Department of Computer Science and Engineering

(October 2025)

DECLARATION

I/We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)
SATHWIK VINTHA (2110110469)

(Signature)
HRISHIK PYLA
(2110110405)

(Signature)
NALLAM MANI SHANKAR (2110110341)

ACKNOWLEDGEMENTS

I wish to extend my deepest gratitude to Mr. Saurabh Shigwan, Associate Professor, Department of Computer Science and Engineering, of Shiv Nadar Institute of Eminence for their contributions to the completion of my project titled Python-based Vulnerability Scanner.
I confirm that this project was completed solely by me and is not the work of someone else.

(Signature)
SATHWIK VINTHA
2110110469

ACKNOWLEDGEMENTS

I wish to extend my deepest gratitude to Mr. Saurabh Shigwan, Associate Professor, Department of Computer Science and Engineering, of Shiv Nadar Institute of Eminence for their contributions to the completion of my project titled Python-based Vulnerability Scanner.
I confirm that this project was completed solely by me and is not the work of someone else.

(Signature)
HRISHIK PYLA
2110110405

ACKNOWLEDGEMENTS

I wish to extend my deepest gratitude to Mr. Saurabh Shigwan, Associate Professor, Department of Computer Science and Engineering, of Shiv Nadar Institute of Eminence for their contributions to the completion of my project titled Python-based Vulnerability Scanner.
I confirm that this project was completed solely by me and is not the work of someone else.

(Signature)
NALLAM MANI SHANKAR
2010110341

Contents

ABSTRACT

This project introduces Python-based unsupervised methods for image segmentation and denoising, focusing on overcoming the limitations of traditional techniques that depend heavily on labelled data. Labelling data is not only expensive but also a time-consuming process, making it a bottleneck in many image processing applications. By employing unsupervised methods, we aim to develop more scalable and efficient solutions that can detect inherent patterns in the data without requiring manual annotation.

Image segmentation, which involves dividing an image into meaningful parts, and denoising, which aims to remove noise while preserving important details, are essential tasks in numerous fields such as medical imaging, autonomous driving, and satellite image analysis. However, conventional approaches typically rely on labelled data for training, which can be impractical to obtain for large datasets or specialized domains.

Our approach leverages the flexibility and power of Python to implement these unsupervised methods, enabling automatic pattern recognition and feature extraction from unlabelled data. This makes the methods more adaptable and less dependent on costly resources, enhancing the overall efficiency and robustness of the image processing workflow. Ultimately, this project seeks to contribute to advancements in image segmentation and denoising by providing a cost-effective and scalable alternative to traditional, data-intensive methods.

Chapter 1

Introduction

1.1 BACKGROUND:

Traditional supervised approaches to image processing require many labelled images, which are time-consuming and expensive to generate, particularly in massive or application-specific datasets. Important labelled information is required to generate such datasets. The limitations of these methods have become evident with the necessity of efficient image segmentation and denoising techniques. Scalable alternatives are offered by unsupervised approaches that directly extract information from raw data by detecting patterns.

Recent advancements in machine learning, especially with Python, enabled designing unsupervised approaches that perform segmentation and denoising with automated methods. This results in non-linearity processing images that are not labelled, and the said techniques can particularly be utilized effectively in applications such as medical imaging and autonomous driving. This project uses Python to design unsupervised methods as a contribution toward more economical and scalable solutions for image processing.

1.2 MOTIVATION:

Owing to the rapid sprawl of challenges in image processing, effective yet scalable solutions are needed, as well as demands for contributing unsupervised methods aiding automatic means of segmenting and denoising images. Labelled-data reliance bears high time and cost on it, making it impractical for large-scale or special datasets, particularly in such crucial realms as medical diagnostics, autonomous driving, and satellite imaging; this circumstance needs more versatile approaches.

This project was oriented toward the discovery of structures and specific features within data so as to facilitate logics beyond traditional methods in order to assist in deploying the full capabilities of unsupervised algorithms in automatic discrimination of the features without needing any manual labelling. This trend aims not only at efficiency but also toward enacting a simpler, more versatile, and resource-efficient image processing task. Thus, with the aid of Python and machine learning, the project tries to take state-of-the-art image segmentation and denoising borders to a new level in so far as it provides people with a scalable solution to issues appearing in modern applications. In a much broader sense, it represents an exciting state in a journey that should inspire forays into the realm of data science that will more and more yield twists of relatively economical solutions to process vast quantities of complex image data.

1.3 PROBLEM STATEMENT:

The core issue being tackled here is how traditional image segmentation and denoising methods, based on labelled data, are often criticized for being slow and expensive. In numerous critical fields, such as medical imaging, autonomous driving, and satellite analysis, obtaining labelled data for the purposes of training supervised models is very often impractical due to the time-consuming and costly natures of these proven methods. ""Because of the dramatically growing volume and evolving complexity of input imaging data, there is a pressing need to find scalable solutions that can be flexible enough to handle future troubles."" This failure of existing methods signifies the great necessity for an approach that can be efficient when reliant on other information without manual data-labelling.

This project targets conceivably this complication dwelling alongside developing unsupervised methods for image segmentation and denoising in Python. These methods detect patterns and features in raw data automatically and may serve as more efficient, cost-effective, and more scalable alternatives to conventional data-hungry methods.

Chapter 2

Literature Review

2.1 INTRODUCTION TO IMAGE DENOISING:

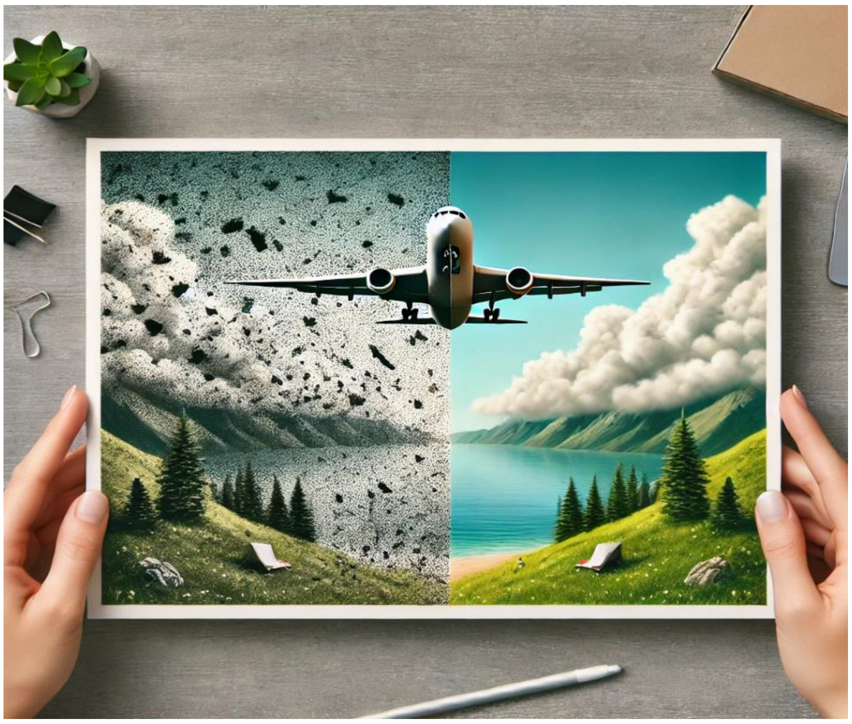
Image denoising is one of the primary tasks in image processing, which deals with noise elimination, maintaining important details. Noise may enter the image-acquisition process due to several factors, such as inadequate lighting, poor-quality sensors, or surrounding influences. The quality of the image is compromised with a presence of noise, affecting later-stage tasks such as segmentation, object detection, and even medical diagnosis. Traditional denoising methods are characterized, statistically or by rule-based ones, such as Gaussian filtering, median filtering, or wavelet-based techniques. However, these methods usually smooth out not only the noise but also the fine details of the image, so they can also lead to loss of useful information. Deep learning has emerged in recent times as a powerful software for the task of denoising, and its design provides a more adaptive data-driven approach to solve the problem of noise reduction.

This project applies unsupervised deep learning techniques, using a method known as Deep Image Prior, which can achieve denoising without relying on labelled training data. In other words, the model implemented through neural networks will learn to reconstruct noise-free images from a noisy input in a pure way, of course, without the necessity for external datasets.

2.2 Unsupervised Denoising Approach:

The unsupervised denoising method employed in this project relies on the concept of Deep Image Prior, which is a strategy for training a neural network to fit a noisy image until the denoised image is learned. The model works directly on the noise image patch, unlike the traditional supervised learning paradigm in which a large, labelled dataset is required.

Unlike supervised learning requiring large amounts of labelled data, wherein much exertion would go toward acquiring these labels, this method naturally favours the denoised outputs through the intrinsic architecture of the network acting as a regularizer. In this study, we employ a deep convolutional neural network (CNN) with skip connections to refine an image stepwise through multiple layers thus enabling the network focus on salient details while recursively filtering noise. This makes this method particularly useful where there is no prior knowledge available on the image or the noise characteristics and finds utility in varied applications ranging across medical imagining, satellite images to casually clicked photographs.



So, the image on this page shows the denoising process and compares it with a method that uses some already existing denoising techniques. A pair-wise comparison is shown on the noisy version and the cleaned or denoised one. The left side of an image shows random noise and distortions, while the right side shows a clean-noise-free image after denoising methods have been applied. The use of these images highlights some such advancements in respect to the other techniques considered in this project, and it provides an excellent visual presentation of how unsupervised denoising enhances image clarity without relying on labelled data.

2.3 NEURAL NETWORK ARCHITECUTRE

In this project, the neural network architecture is based on U-Net or a model with skip connections. U-Net is one widely used architecture for image-to-image translation tasks, notably in segmentation and denoising applications. The U-Net architecture possesses a contracting path (encoder) which reduces the spatial size of the input and then extracts high-level features, and an expanding path (decoder) which restores the original size by progressively up sampling the feature maps. It usually contains skip connections between corresponding layers of the encoder and decoder, which help preserve spatial information that may be lost during down sampling.

In this work, critical in providing structural organization to images by denoising without losing fine details, skip connection models add marks of spatial features or comparisons based on identical structures in an image originating from U-Net's multi-scale working ability in affording themselves room for both the global context and local features making way for better noise removal.

2.3.1 Explanation of Skip Connection-based Networks (U-Net):

Skip connections represent an important development in the U-Net architecture allowing it to merge high-level semantic information from the deeper layers with low-level spatial information from the earlier layers, this effectively avoids the loss of certain unique design information during the encoding process, extremely unique for cases such as denoising of images where edge and fine texture features must be preserved. Skip connections allow better learning representations of features and hence overall improvement in restoring the denoised image.

2.3.2 Multi-scale Processing for Effective Denoising:

Multi-scale processing is the analysis of an image through several resolutions, essential for handling noise appearing at different scales within an image. The neural network that performs this denoising operation is designed to operate on multi-scales, allowing it to effectively remove noise irrespective of its magnitude or complexity.

According to the U-Net architecture made of skip connections, multiscale processing is done in repetitive down sample and up-sample operations. This provides the network with the ability to fine-tune image representations by collapsing and refining the image in a progressive way, aiding with better-denoised images.

2.4 Optimization Process

The approach of optimisation is quite critical in training the neural network to denoise the image where, in this particular case, the image clean-up output by the neural network as opposed to the input-the noisy image-endures several iterations till the difference is well minimized. For this project, the loss function used is **Mean Squared Error (MSE)** and optimization is done using the **Adam optimizer**.

The optimizer adjusts the network weights such that loss can be minimized. The objective of optimization becomes of prime importance in assisting the network in learning the features that help differentiate between noise and the actual content of the image over time, which improves denoising performance.

2.4.1 Role of MSE Loss Function and Adam Optimizer:

Mean Squared Error (MSE) loss is based on the average squared difference between the predicted image and the ground truth. For outputting the last-layer result, it checks how effectively the network has been in reducing noise for image denoising. Using **Adam optimizer**, then, comes minimizing the MSE loss. Adam is an adaptive learning rate optimization algorithm that is designed to combine the strengths of two other extensions of stochastic gradient descent, AdaGrad and RMSProp. It adjusts both the first and second moments of the gradient in every update, which makes it a very efficient, trustworthy way of updating the network's coefficients.

2.4.2 Noise Regularization to Prevent Overfitting:

To prevent the training of a model that overfits to the noisy input, **regularization** against noise is used. The technique introduces small amounts of random noise to the network input during optimization. It thereby keeps the model from specializing on the noise patterns in the input, i.e., it allows the network to generalize better to unseen data.

Such regularization techniques are central to unsupervised denoising because they attempt to find a reasonably good compromise between efforts in noise removal versus the preservation of important features in the image. In this case, the noise regularization helps the network not to overtrain on the specific noise distribution of the training image, thereby producing denoising that is more robust and generalizable.

2.5 Evaluation Metrics

The primary measure of output image quality is assumed to be the **Peak Signal-to-Noise Ratio**. PSNR is the ratio between the maximum possible value of a signal-the image-and the power of corrupting noise, with higher values being the indicators of the better quality of the image. For achieving the PSNR, images are compared differently-between noisy input and denoised output, and PSNR from the perspective of denoised image and ground truth. Thus, a clearer insight is obtained regarding what amounts of noise have been removed and how well the reproduced image fits the original.

2.5.1 Use of PSNR to Evaluate Denoising Quality:

The most well-known method for estimating the quality of reconstruction in image reconstruction tasks, including denoising tasks, is the **Peak Signal-to-Noise Ratio (PSNR)**. Indeed, higher values for PSNR indicate better quality reconstruction, which further means that the denoised image is closest to the original/ground truth image, therefore more noise has been effectively removed. In this project, through each iteration, the PSNR is calculated to monitor the progress of the denoising process. This metric is objective, providing a basis upon which to compare the improvements in image quality and to establish the effectiveness of the network in denoising. The PSNR is one of the most important metrics when assessing and comparing different denoising approaches.

2.5.2 Early Stopping Mechanism to Ensure Optimal Results:

Optimization tracks the network performance through early stopping: it prevents overfitting, so it ensures the best denoised image. With early stopping, track either PSNR or loss function and stop the optimization whenever the performance drops significantly, a sign that this is the point where the model probably is going to overfit. Then it just goes back to some earlier checkpoint with better PSNR between iterations. This mechanism, therefore, supports the prevalence of the balance between arresting noise and leaving crucial details intact. With this, it performs denoising at the desirable level; indeed, free from fitting to noise patterns.

Chapter 3

Work Done

3.1 Project Design and Planning:

In this phase, great attention was paid to outlining the project definitively - its architecture, features, and the development road map, around which critical decisions had to be made setting the goal of the project and outlining the range of functionality that segmentation and denoising methods were to have. It therefore considered parameters like specified timeline, algorithmic complexity, available resources, and the extent of the project.

To make the process development smooth and on time, we drafted a structured roadmap that indicated the sequence of work to be done and some important milestone activities. This was the planning phase devoted to defining what technologies to be used through careful choices of programming language, frameworks, and libraries. Python was the best choice of language due to its rich ecosystems of libraries with machine learning capabilities and flexibility related to image processing requirements.

The design stage then followed by perfecting the technical implementation and friendliness of the algorithms in the segmentation and denoising. Several approaches towards unsupervised learning types, screening techniques for noise removal, and methods of optimizing models toward scalability would be worked on. There was much special focus on structuring the image processing pipeline to be modularly efficient to handle various types of images and also varying degrees of noise. The design work gave a basis of flexible extensible architecture that was easily changed according to future developments and could be integrated with other machine learning frameworks.

3.2 Technology Stack

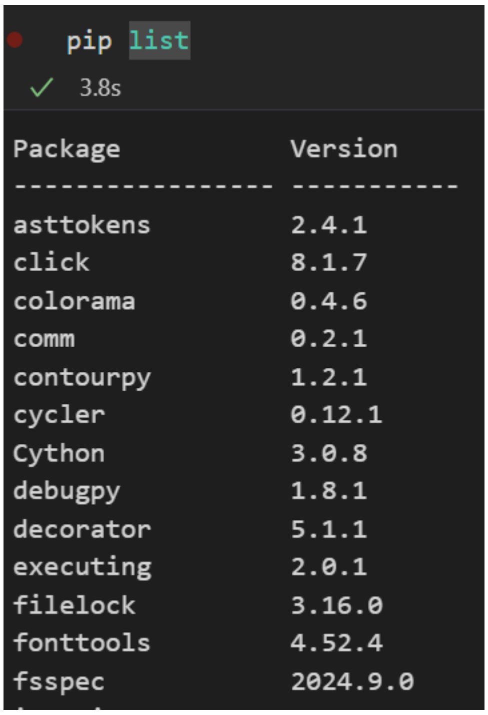
The technology stack for the Unsupervised Image Segmentation and Denoising project is designed to leverage modern tools and libraries that facilitate efficient image processing and machine learning. Python was selected as the primary programming language due to its rich ecosystem of scientific computing libraries and its widespread use in machine learning and data science applications.

Key components of the technology stack include:

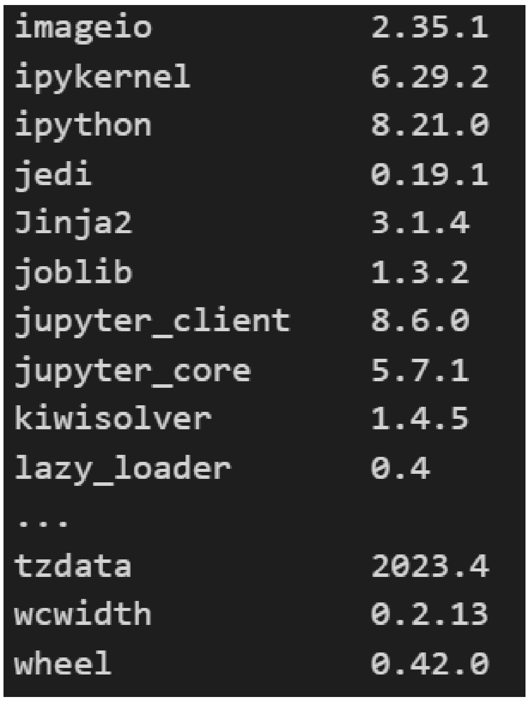
1. **Python**: In this project, Python will be taken as a base since it is applicable and easy to be coding. The cumbersome libraries of image processing and deep learning make it the best for development and validation of unsupervised methods.

2. **PyTorch**: PyTorch is a deep learning framework used to develop neural network models in the denoising and segmentation process. Dynamic computation the graph and flexibility allow rapid experimentations of new architectures to which skip connection-based networks, like U-Net especially, belong.

3. **Scikit-Image**: This library is designed to be used for some variety of image-processing tasks, such as adding synthetic noise. Image Handling Many formats of images and general operations. Scikit-Image also offers Utilities for the measuring of the quality of a picture with metrics like Peak Signal-to-Noise Ratio (PSNR).
4. **Matplotlib**: Plot using Matplotlib to represent what really makes the difference in segmentation and denoised images to be able to compare noisy and denoised images and between explicitly differentiated outputs.
5. **NumPy**: NumPy supports the concept of an array and a matrix as base data types. Extremely important: computationally efficient enough to calculate numbers in images and tensors.
6. **Denoising and Segmentation Utilities**: Custom utility functions are used for managing noise addition, image manipulation, and other preprocessing steps, enhancing the overall flexibility and reusability of the code.



```
pip list
✓ 3.8s

Package            Version
------------------ -----------
asttokens          2.4.1
click              8.1.7
colorama           0.4.6
comm               0.2.1
contourpy          1.2.1
cycler             0.12.1
Cython             3.0.8
debugpy            1.8.1
decorator          5.1.1
executing          2.0.1
filelock           3.16.0
fonttools          4.52.4
fsspec             2024.9.0
```



```
imageio            2.35.1
ipykernel          6.29.2
ipython            8.21.0
jedi               0.19.1
Jinja2             3.1.4
joblib             1.3.2
jupyter_client     8.6.0
jupyter_core       5.7.1
kiwisolver         1.4.5
lazy_loader        0.4
...
tzdata             2023.4
wcwidth            0.2.13
wheel              0.42.0
```

3.3 Feature Development

In this project, Unsupervised Image Segmentation and Denoising, a robust feature set is designed to enhance the quality of images and perform unusually complex image processing operations without the use of labelled data. The objectives of developing numerous core features have so far been directed toward solving the problems associated with image restoration and segmentation, thus enabling the solutions to be useful in many applications.

1. **Unsupervised Denoising Algorithm**:
   It is anticipated to develop an unsupervised denoising algorithm using Python, which will remove noise from images without the need for labelled training data.
   This is achieved through an algorithm which, based on techniques such as Deep Image Prior (DIP) and using a skip connection-based architecture, the neural network U-Net automatically identifies and erases noise from images while keeping all relevant details intact. Such automatic de-noising enables image clarity by merely scaling the noisy inputs into the system.

2. **Unsupervised Image Segmentation**:
   Another important feature is unsupervised image segmentation, which segments an image into semantically meaningful regions defined by the patterns that come along with data itself. This lends itself very well to applications in medical imaging, object detection, and scene understanding, as it allows the detection of structural differences based on image content rather than relying on labelled segmentation masks. Essentially, multi-scale neural networks are applied to perform this segmentation and very accurately do the extraction of objects and regions through the images.

3. **Image Inpainting**:
   The project is based on image inpainting, including algorithms capable of reconstructing portions of an image that are not visible, or replacing existing content. It is thus an application engaged in manipulating an image in various ways: removing text, unwanted objects, or adding new elements in such a way that those merge well with the present scene. The feature acquires contextual information from the neighbouring pixels through neural networks so that inpainting goes on smoothly and is biologically adequate.

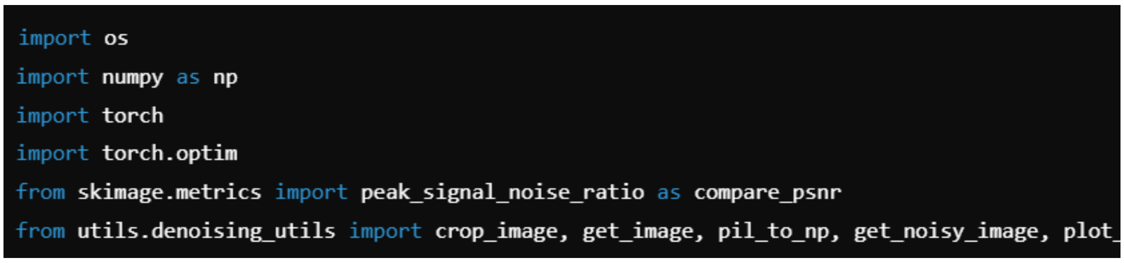
4. **Adaptive Noise Removal**:
   Features of adaptive noise removal help to remove noise based on varying degrees of background noise with careful observation at denoised areas so that finer structural details are preserved in relatively less-noisy regions. Such methods are automatic in the detection of noise distribution and their localized application of denoising algorithm.

5. **Edge-Preserving Denoising**:
   Edge-preserving denoising preserves sharpness at visible object boundaries but applies smoothing and denoising elsewhere. Its strength lies in applications where its ability to preserve edges ensures effective communication and analyses of images, as is the case with various medical and satellite imaging.

3.4 Denoising (Breakdown of the Code)

1. Importing Necessary Libraries

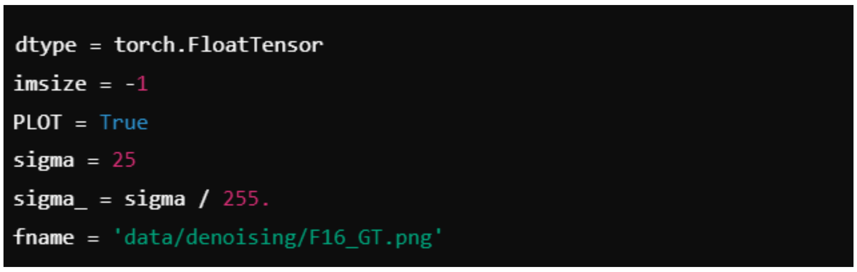
- Installing the **Scikit-image** for the image processing tasks, that are predominantly used for computation of image quality metrics such as Peak Signal-to-Noise Ratio.
- Image display with **Matplotlib**; explanation of the algorithm for fine-tuning a neural network in PyTorch.



```
import os
import numpy as np
import torch
import torch.optim
from skimage.metrics import peak_signal_noise_ratio as compare_psnr
from utils.denoising_utils import crop_image, get_image, pil_to_np, get_noisy_image, plot_
```

2. Image and Noise Setup

- In F16_GT.png image, add the synthetic Gaussian noise; This results in an over-noise image There is an implementation of it in **get_noisy_image**. Sigma is the parameter for the noise intensity, in terms of standard deviation of the noise.
- Two image paths are managed:
  - **Snail Image**: Uses a pre-cropped noisy image.
  - **F16_GT Image**: Apart from the artificial noise, it also presents more noisy copies of the image.



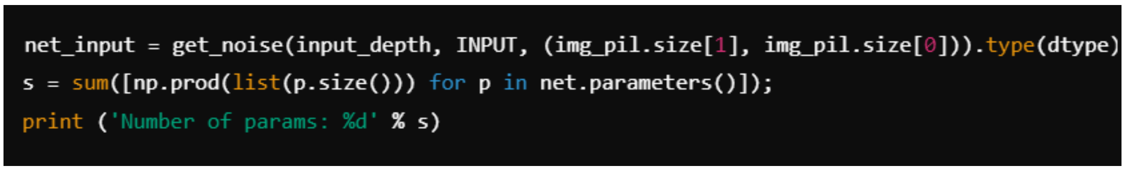
```
dtype = torch.FloatTensor
imsize = -1
PLOT = True
sigma = 25
sigma_ = sigma / 255.
fname = 'data/denoising/F16_GT.png'
```

3. Defining Neural Network Architecture

- A **Skip network** is declared privately as follows using the skip function for snail.jpg and get_net for the image F16_GT.png.
- **Skip Networks** confirm the information details retained in the images during the down-sampling and up-sampling procedure.
- Number of layers is determined by the source image and the number of channels; it initializes a couple of layers with activation functions that use **LeakyReLU** and bilinear up sampling at start up.

4. Defining Input Noise

- Initialize the network's input as a random noise and let get_noise be the initialization for what to learn in order to clean the image. Have the input size follow the size of the original image.



```
net_input = get_noise(input_depth, INPUT, (img_pil.size[1], img_pil.size[0])).type(dtype)
s = sum([np.prod(list(p.size())) for p in net.parameters()]);
print ('Number of params: %d' % s)
```

5. Loss Function (MSE)

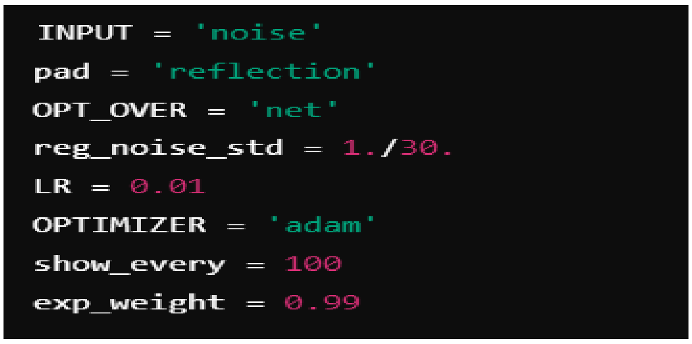
- **MSE (Mean Squared Error)** is a cost function that measures how much difference exists between the network-degenerated image and the noisy input image. It basically measures how good fits the output of the network is towards the noisy target image.

6. Quality Metric (PSNR)

- For each iteration, **PSNR (Peak Signal-to-Noise Ratio)** is calculated so that it is known how much the network is reducing the image's noise. Psnr of both the noising and the denoising outputs are computed. The higher the PSNR value the better is the quality of the image.

7. Optimization (Adam)

- This uses an **Adam optimizer** to have optimal parameters towards the network by allowing the weights for the MSE loss to be at their lowest possible extent. It also has an LR controlling as fast as the optimizer decides to make those changes for minimising the loss.



```
INPUT = 'noise'
pad = 'reflection'
OPT_OVER = 'net'
reg_noise_std = 1./30.
LR = 0.01
OPTIMIZER = 'adam'
show_every = 100
exp_weight = 0.99
```

8. Regularization and Noise

- **reg_noise_std:** Added small quantities of regularization noise at training, so the model should not memorize the noised input image and generalizes very well without overfitting.

9. Training Loop

- The network needs to traverse multiple times, say 3000 times for an image, such as F16_GT.png:
    - Feeding this noisy input through the network will give us a denoised output.

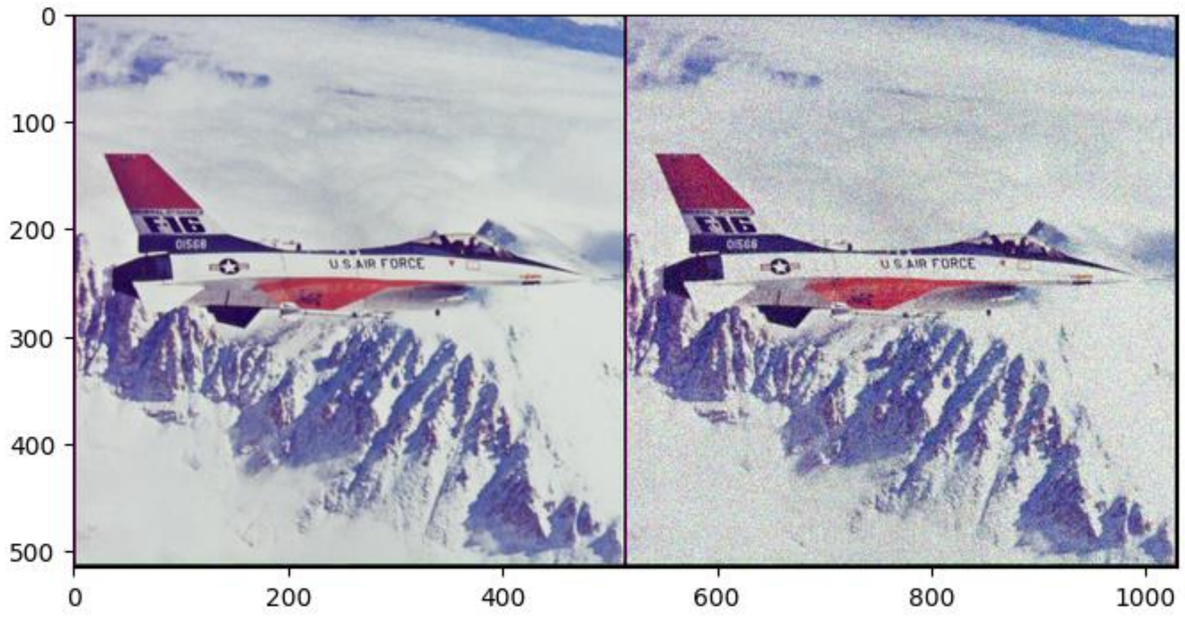
- This thus calculates the PSNR at each step and measures how well it does on this scale.
- Checkpoints are made at periodic intervals so that the performance does not degrade drastically (if PSNR degrades too much, then the model reverts to the last checkpoint).

10. Visualization

- It computes at intervals of intervals of regular frequency plot and visualizes whether there is an original noisy image denoised image, and ground truth image if it is present to visually indicate that process.
- Allows observing the performance of the network in real-time as it is training.

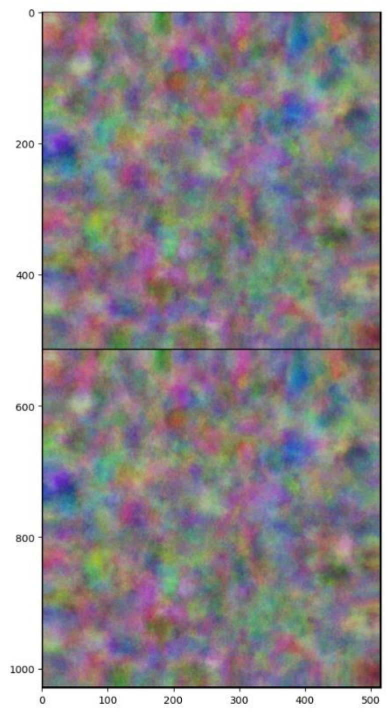
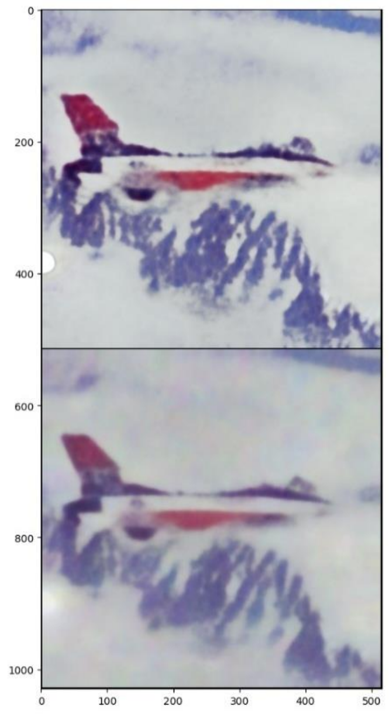
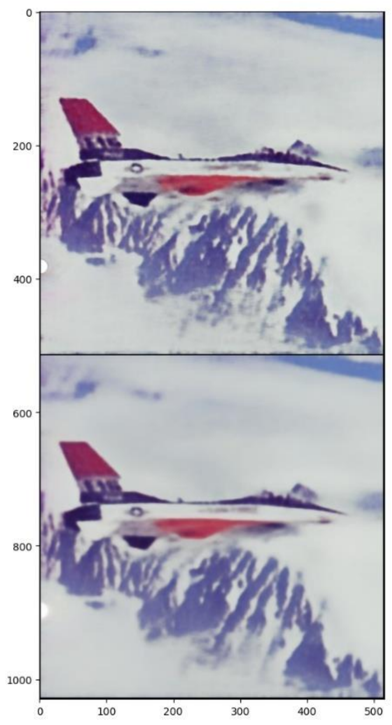
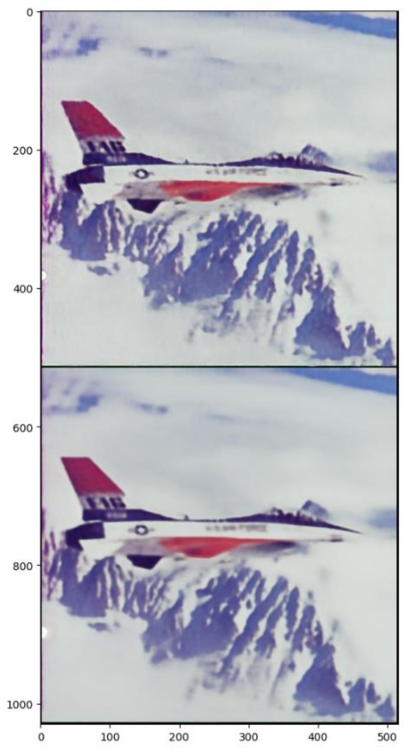
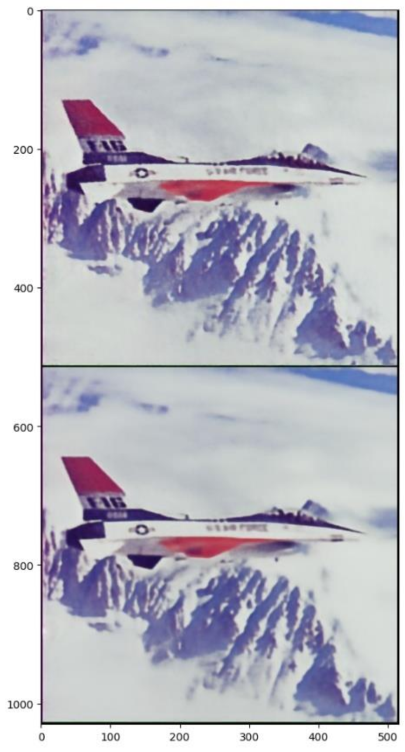
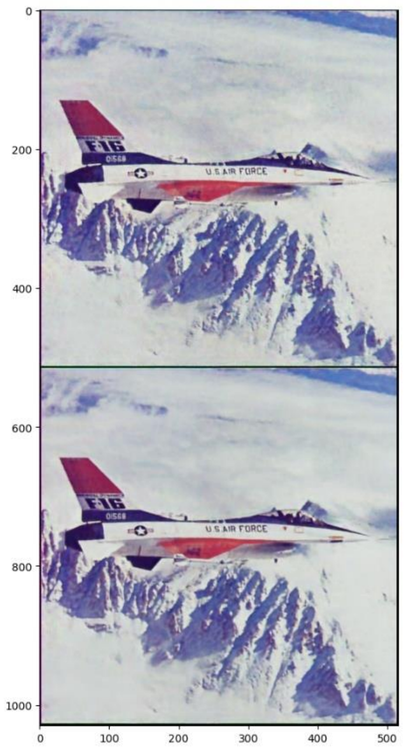
11. Final Output

- The network produces an output after reaching a specific number of iteration loops by giving out a final denoised image to be compared with the originally corrupted version. PSNR was used to measure the output and check whether the quality of the denoised output was appropriate.



The left image shows output from the denoising process, a situation where the algorithm has cleared noise from the image and restored its clarity. The right image is the original input that was most corrupted by noise and was, therefore, entirely unclear. This denoised image is significantly improved in visual quality in that features can be remarked on, along with substantial reduction in noise-caused artifacts. This makes a good case study for examining the denoising technique used in the model.

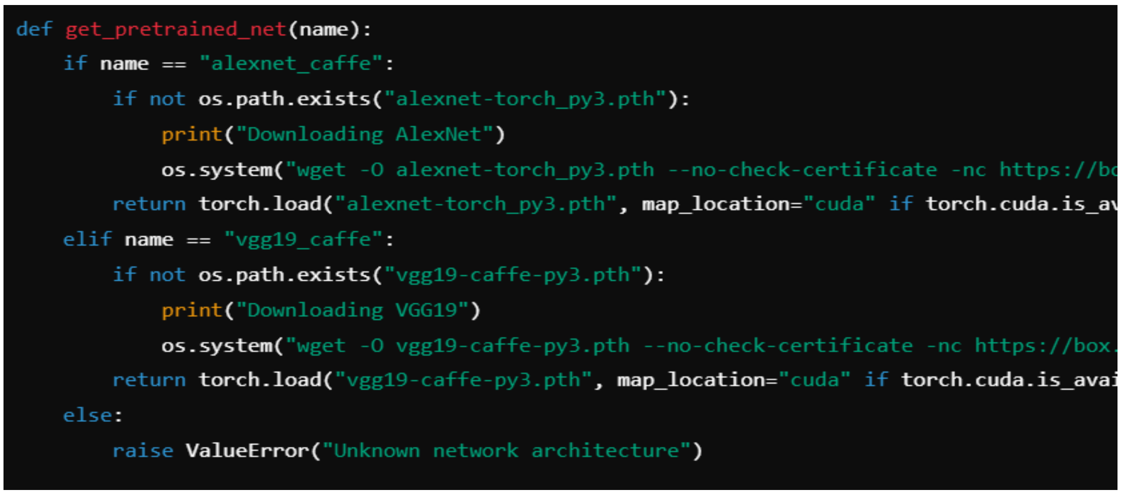
Working➔

3.5 Feature-Inversion (Breakdown of the Code)

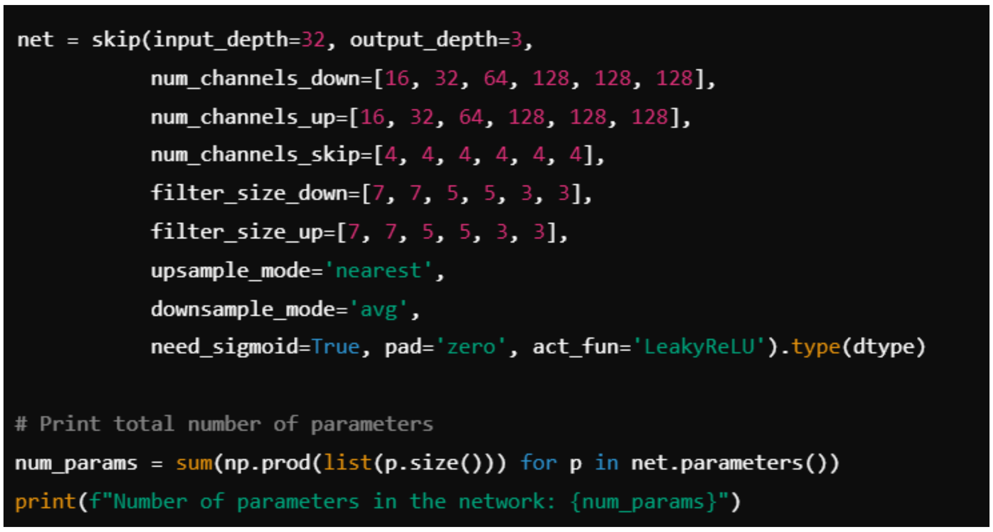
1) **Importing Necessary Libraries**
    - Libraries are Matplotlib These libraries are required for visualization and PyTorch for model building and optimization.
    - **Utilities for Inversion and Feature**:
        - Provide helper functions for feature inversion, storage, and preprocessing.
        - Download pre-trained models like AlexNet or VGG19 that can be used within feature extraction.
2) **Image Setup**
    - Load the target image building.jpg and then resize it to suit the dimension provided, probably 224x224 ideal since that is also the size that would work with AlexNet/VGG19.
    - get_image performs resizing with the image converted into a numpy array to continue other processing.

3) **Pre-trained Network for Feature Extraction**
    - The model will use AlexNet or VGG19 pre-trained on ImageNet as feature extractor.
    - Define which layer to use in feature inversion (e.g., fc6).
    - These feature sets are stored by the matcher object for use at a later time in optimization.



```
def get_pretrained_net(name):
    if name == "alexnet_caffe":
        if not os.path.exists("alexnet-torch_py3.pth"):
            print("Downloading AlexNet")
            os.system("wget -O alexnet-torch_py3.pth --no-check-certificate -nc https://bo
        return torch.load("alexnet-torch_py3.pth", map_location="cuda" if torch.cuda.is_av
    elif name == "vgg19_caffe":
        if not os.path.exists("vgg19-caffe-py3.pth"):
            print("Downloading VGG19")
            os.system("wget -O vgg19-caffe-py3.pth --no-check-certificate -nc https://box.
        return torch.load("vgg19-caffe-py3.pth", map_location="cuda" if torch.cuda.is_avai
    else:
        raise ValueError("Unknown network architecture")
```

4) **Generative Network Architecture**
    - A pass connection-based network is used to create the output image.
    - **The architecture:**
        - Has down-sampling and up-sampling layers.
        - Uses skip connections to save detail during transformation.
        - Uses Non-linear transformation with LeakyReLU activation.
        - It has a customizable configuration for filter sizes to up sampling and down sampling modes.



```
net = skip(input_depth=32, output_depth=3,
           num_channels_down=[16, 32, 64, 128, 128, 128],
           num_channels_up=[16, 32, 64, 128, 128, 128],
           num_channels_skip=[4, 4, 4, 4, 4, 4],
           filter_size_down=[7, 7, 5, 5, 3, 3],
           filter_size_up=[7, 7, 5, 5, 3, 3],
           upsample_mode='nearest',
           downsample_mode='avg',
           need_sigmoid=True, pad='zero', act_fun='LeakyReLU').type(dtype)

# Print total number of parameters
num_params = sum(np.prod(list(p.size())) for p in net.parameters())
print(f"Number of parameters in the network: {num_params}")
```

5) Input Noise Initialization

- Initializes the network input as random noise through the get_noise function.
- Dimensions of noise are in line with those expected by the input size of the generative network.
- This noise acts as a foundation to reconstruct the required features.

6) Loss Function

- The loss is computed as the difference between the features of the generated image and the target features extracted from the pre-trained model.
- Matcher object contains and computes those losses while optimization.

7) Optimization Process

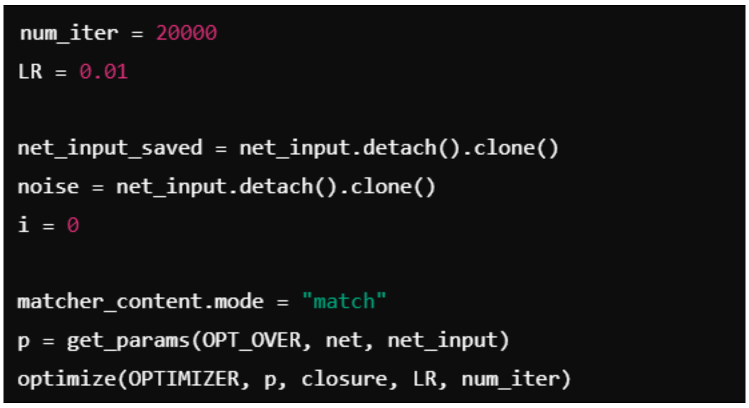
- The optimizer used to minimize the feature loss is Adam.
- Learning rate and other hyperparameters (e.g., num_iter) control the optimization process.
- Iteratively updates the weights of the generative network to match the target features.

8) Regularization and Perturbations

- Regularization noise contributes to the network weights during training for overfitting prevention and robustness development.
- Perturbations to input noise are modified as iterations advance for improvement of generalization.

9) Training Loop

- **The optimization loop performs the following tasks:**
  - Passes random noise through the generative network.
  - Computes and minimizes feature loss with respect to their target features stored previously.
  - Keeps driving the reconstruction process and visualizes some intermediate outputs at intervals.



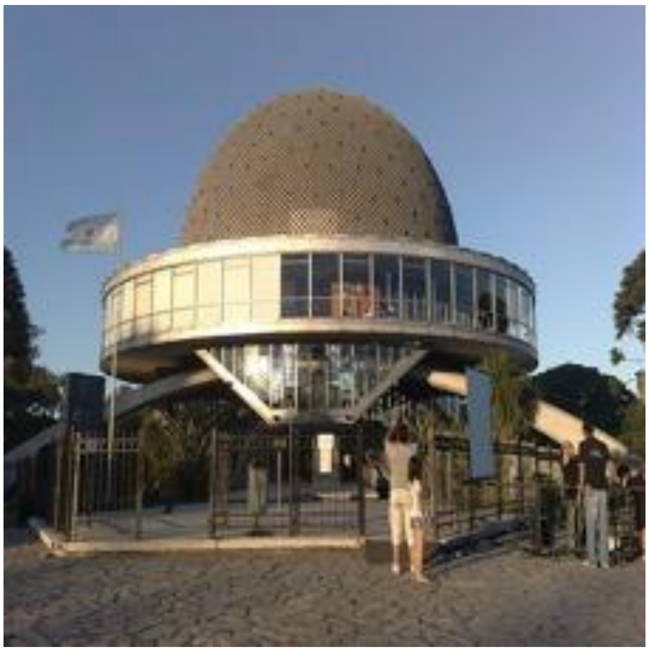
```
num_iter = 20000
LR = 0.01

net_input_saved = net_input.detach().clone()
noise = net_input.detach().clone()
i = 0

matcher_content.mode = "match"
p = get_params(OPT_OVER, net, net_input)
optimize(OPTIMIZER, p, closure, LR, num_iter)
```

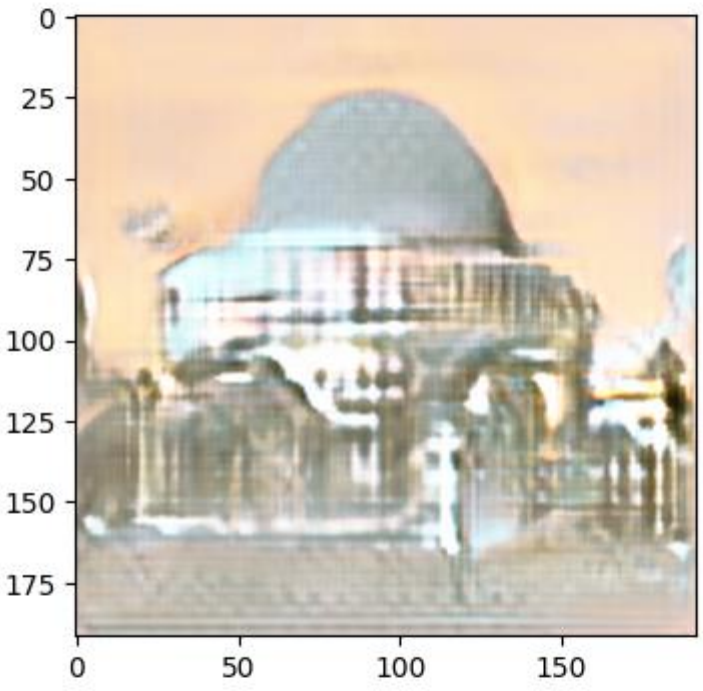
10) Visualization

- The intermediate and final outputs will be visualized regularly via plot-image grid.
- There are comparisons between the generated image, the target image, and the noise input for progress tracking.

11) Final Output

- After the prescribed number of iterations, the generative network outputs a reconstructed image.
- The reconstructed image resembles the target image concerning features extracted from it.
- **Visualization highlights:**
    - It is very similar to the original image's feature space.
    - Noise turned into a visually coherent representation resembling the target image.





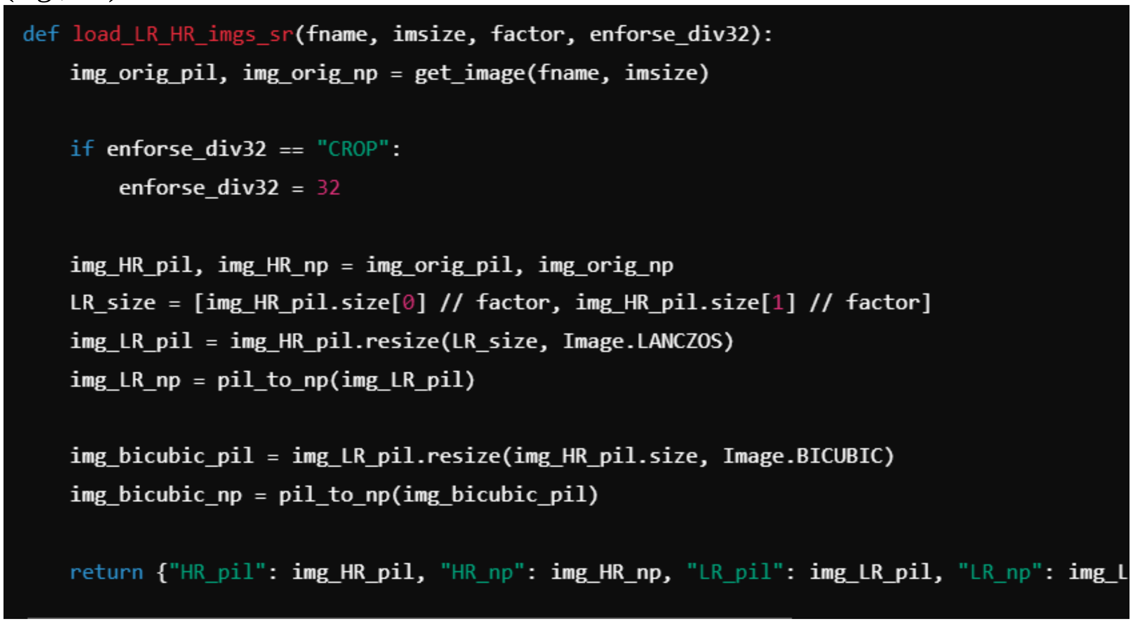
3.6 Super-Resolution (Breakdown of the Code)

1. **Importing Necessary Libraries**

- Thus, Scikit-image is a library that contains image functionality for computing quality metrics such as Peak Signal-to-Noise Ratio (PSNR).
- Matplotlib is suitable for image display and monitoring the training's process.
- PyTorch, on the other hand, is responsible for the deployment, optimization, and execution of networks in training processes.

2. **Image Preprocessing and Setup**

- Load a high-resolution image and generate a low-resolution (LR) image through downscaling by a given factor, say, 4 or 8.
- Then the LR image should be up-scaled into the original format through bicubic interpolation to perform comparison purposes.
- Crop or create images to dismantle them into dimensions defined by the power of 2 (e.g., 32).



```
def load_LR_HR_imgs_sr(fname, imsize, factor, enforse_div32):
    img_orig_pil, img_orig_np = get_image(fname, imsize)

    if enforse_div32 == "CROP":
        enforse_div32 = 32

    img_HR_pil, img_HR_np = img_orig_pil, img_orig_np
    LR_size = [img_HR_pil.size[0] // factor, img_HR_pil.size[1] // factor]
    img_LR_pil = img_HR_pil.resize(LR_size, Image.LANCZOS)
    img_LR_np = pil_to_np(img_LR_pil)

    img_bicubic_pil = img_LR_pil.resize(img_HR_pil.size, Image.BICUBIC)
    img_bicubic_np = pil_to_np(img_bicubic_pil)

    return {"HR_pil": img_HR_pil, "HR_np": img_HR_np, "LR_pil": img_LR_pil, "LR_np": img_L
```

3. **Neural Network Architecture**

- Skip networks are indeed known to maintain image detail at a fine level between both the down-sampling and the up-sampling processes.
- **The network's architecture includes:**
    - Skip connections for spatial information retention.
    - Activation functions such as LeakyReLU.
    - Reconstruction of images resolution restoration by bi-linear up-sampling.
- The parameters such as Scales, Channels, and Layers can be scaled down according to the project.

4. Defining Input Noise

- Input to the network is initialized as a random noise by generative function get_noise.
- The noise matches the size of a high-resolution image.
- This gives a basis for the refinement and reconstruction processes of the network.

5. Loss Function (MSE)

- **Loss function is ideally mean squared error (MSE).**
- It holds the difference of reconstruction image to their respective low-resolution input.
- Minimized loss value tends to bring the approximated image more closer to original high-resolution image.

6. Quality Metric (PSNR)

- PSNR (Peak signal-to-noise ratio), is computing from both the outputs: LR and HR for every single iteration during evaluation.
- This was quantitatively expressed and imparted the image quality discrepancy terrains in direct proportionality with the reconstruction superiority.

7. Optimization (Adam)

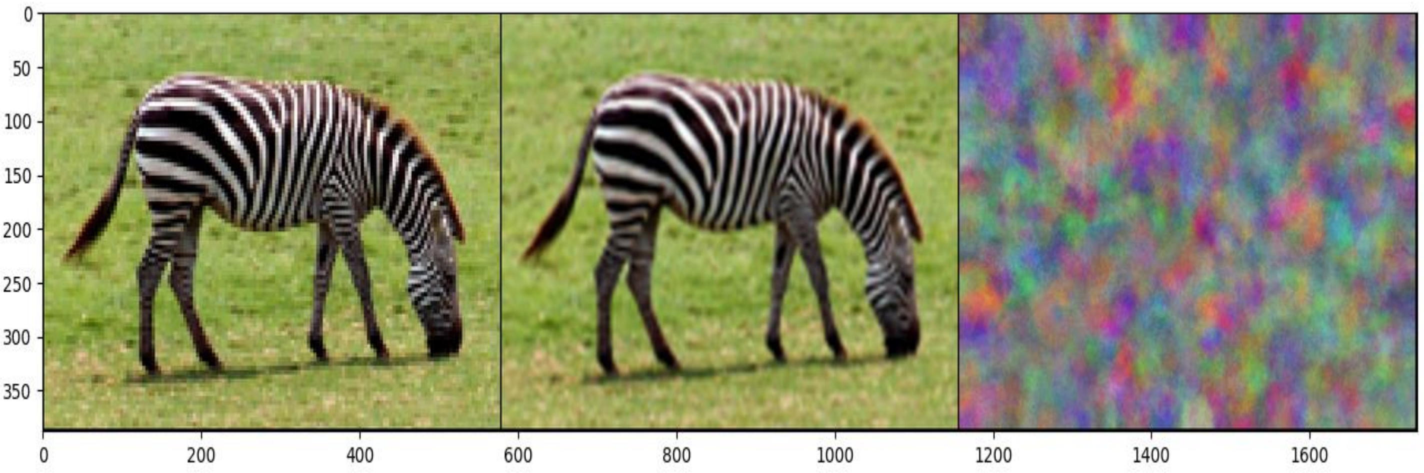
- The parameters of the network are optimized by Adam optimizer for optimization.
- Learning rate (LR) and also regularization settings under control of optimizer behavior for stable and effective training.

8. Training Loop

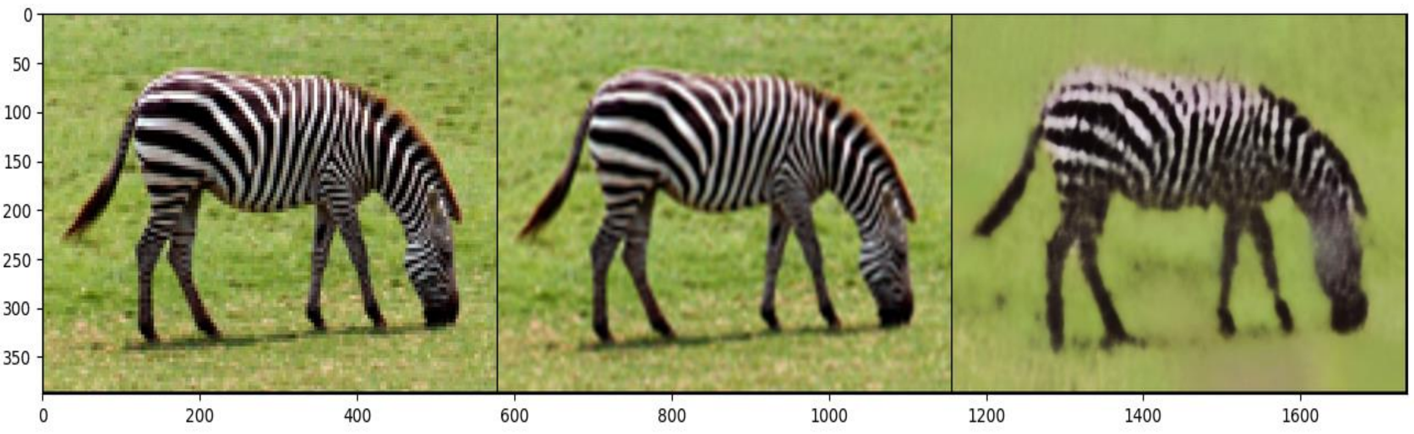
- It would go on to many iterations, such as 2000 or 4000, depending on the scaling factor.
- **Key steps in each iteration:**
  - Network passes the noisy input so that high- and low-resolution output can be produced.
  - Loss calculates backward through.
- Model performance degradation is checked through snapshots and visualizations in between.

9. **Final Output**

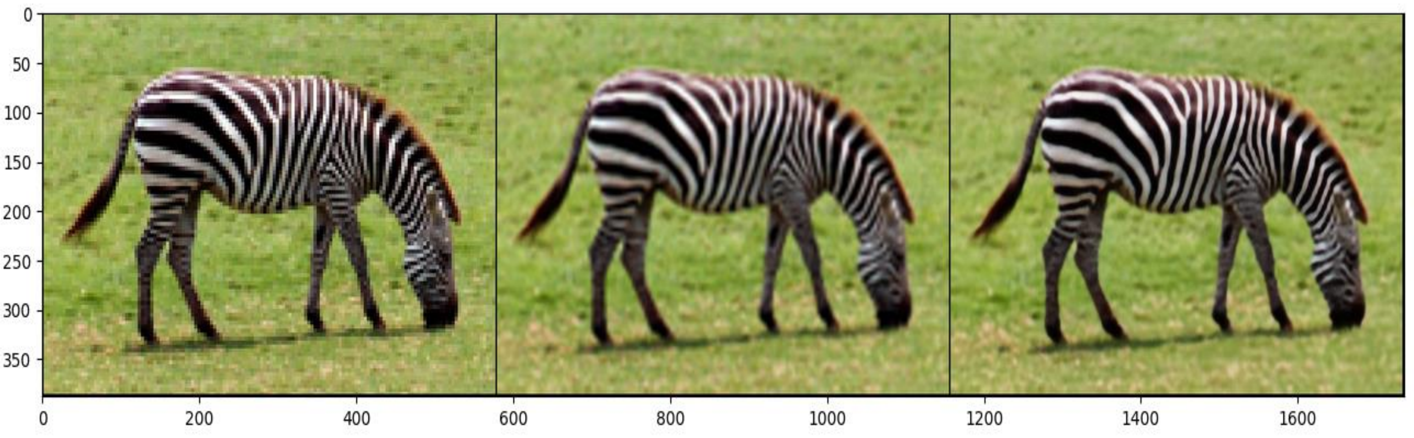
- Once training is done, the high-resolution final image is produced by the network
- **Key observations:**
  - The denoised and upscaled image by network is compared with original noisy input.
  - The reconstructed image shows significantly more clarity and less noise artifact.
  - PSNR quantifies overall value of the final image and proves the success of the Deep Image Prior technique in performing super-resolution.



0
50
100
150
200
250
300
350
0
200
400
600
800
1000
1200
1400
1600



0
50
100
150
200
250
300
350
0
200
400
600
800
1000
1200
1400
1600



0
50
100
150
200
250
300
350
0
200
400
600
800
1000
1200
1400
1600

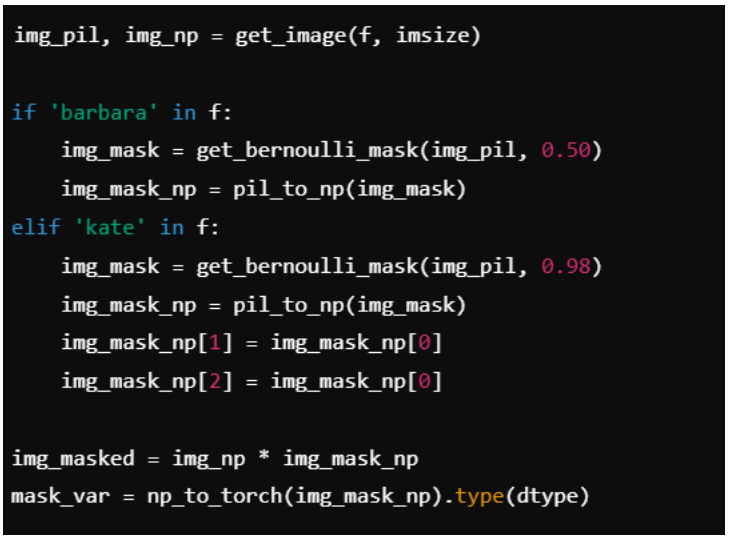
3.7 Restoration (Breakdown of the Code)

1. **Importing Necessary Libraries**

- **Libraries and Frameworks:**
  Such libraries as PyTorch are used for deep learning model construction and training, while Matplotlib helps visualize outcomes.
  - torch: For model creation and training (ResNet, UNet, Skip).
  - matplotlib.pyplot: To display images and results during training.
  - skimage.metrics: For computing image quality indexes such as Peak Signal-to-Noise Ratio (PSNR).
- **Hardware Utilization**:
  It can confirm CUDA compatibility and avail GPU acceleration for rapid computation speed.

2. **Image and Mask Setup**

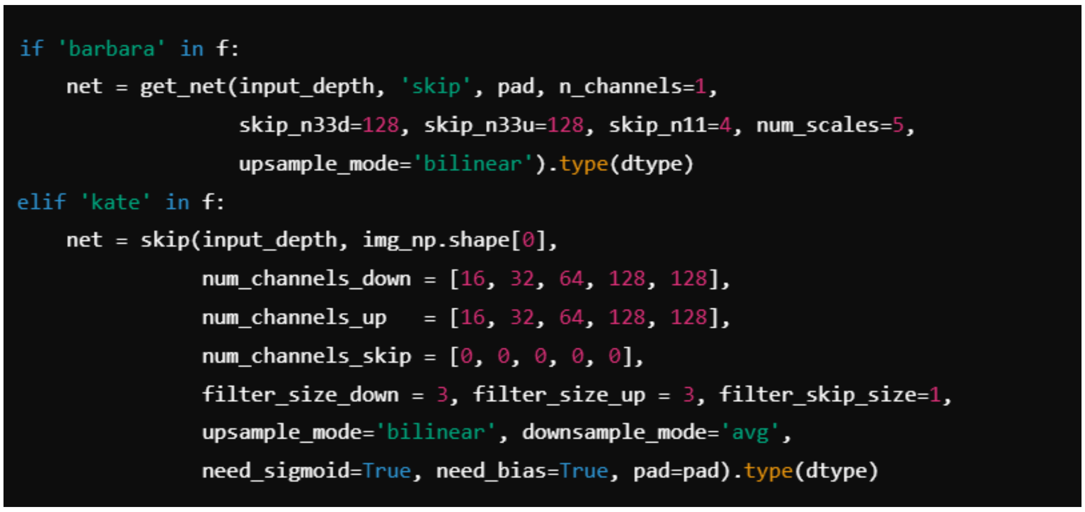
- **Input Images**:
  It loads both the barbara.png and kate.png images to perform restoration experiments on.
- **Mask Generation**:
  - Random binary Bernoulli mask of 50% sparsity created for "barbara.png".
  - Highly sparse mask (98%) was created for "kate.png" with extra pre-processing that removes the difference across channels in image.
- **Masked Image Creation**:
  Multiplying the input images with the above masks simulates partially corrupted data.



```
img_pil, img_np = get_image(f, imsize)

if 'barbara' in f:
    img_mask = get_bernoulli_mask(img_pil, 0.50)
    img_mask_np = pil_to_np(img_mask)
elif 'kate' in f:
    img_mask = get_bernoulli_mask(img_pil, 0.98)
    img_mask_np = pil_to_np(img_mask)
    img_mask_np[1] = img_mask_np[0]
    img_mask_np[2] = img_mask_np[0]

img_masked = img_np * img_mask_np
mask_var = np_to_torch(img_mask_np).type(dtype)
```

3. **Defining Neural Network Architecture**

- **Model Selection:**
    - The "Skip" shape was used with barbara.png.
    - A tailor-made Skip architecture being initialized with user-defined down-sampling and up-sampling layers is being crafted for kate.png.
- **Key Features:**
    - Important details would be preserved by Skip Networks during up-sampling and down-sampling in images.
    - The architecture incorporates bilinear up-sampling, reflection padding and allows the configuring of channel depths to adjust to the nature of the images in terms of size and complexity.



```
if 'barbara' in f:
    net = get_net(input_depth, 'skip', pad, n_channels=1,
                  skip_n33d=128, skip_n33u=128, skip_n11=4, num_scales=5,
                  upsample_mode='bilinear').type(dtype)
elif 'kate' in f:
    net = skip(input_depth, img_np.shape[0],
               num_channels_down = [16, 32, 64, 128, 128],
               num_channels_up   = [16, 32, 64, 128, 128],
               num_channels_skip = [0, 0, 0, 0, 0],
               filter_size_down = 3, filter_size_up = 3, filter_skip_size=1,
               upsample_mode='bilinear', downsample_mode='avg',
               need_sigmoid=True, need_bias=True, pad=pad).type(dtype)
```

4. **Loss Function (MSE)**

- **Mean Squared Error**:
  MSE is calculated by the difference between a pixel at output to the network and an associated pixel in the masked input. Reconstructing the corrupted regions of the image is the expected assignment for the network.

5. **Regularization and Noise**

- **Regularization Noise:**
  A little noise addition (reg_noise_std) to the input during training would prevent from overfitting while enhancing generalization.
    - The values for reg_noise_std are 0.03 for barbara.png; and 0.00 for kate.png.

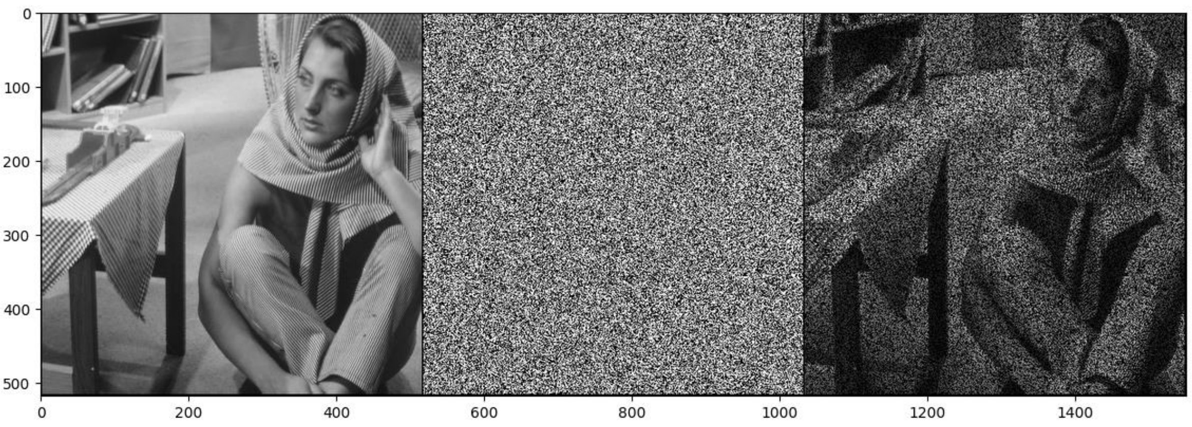
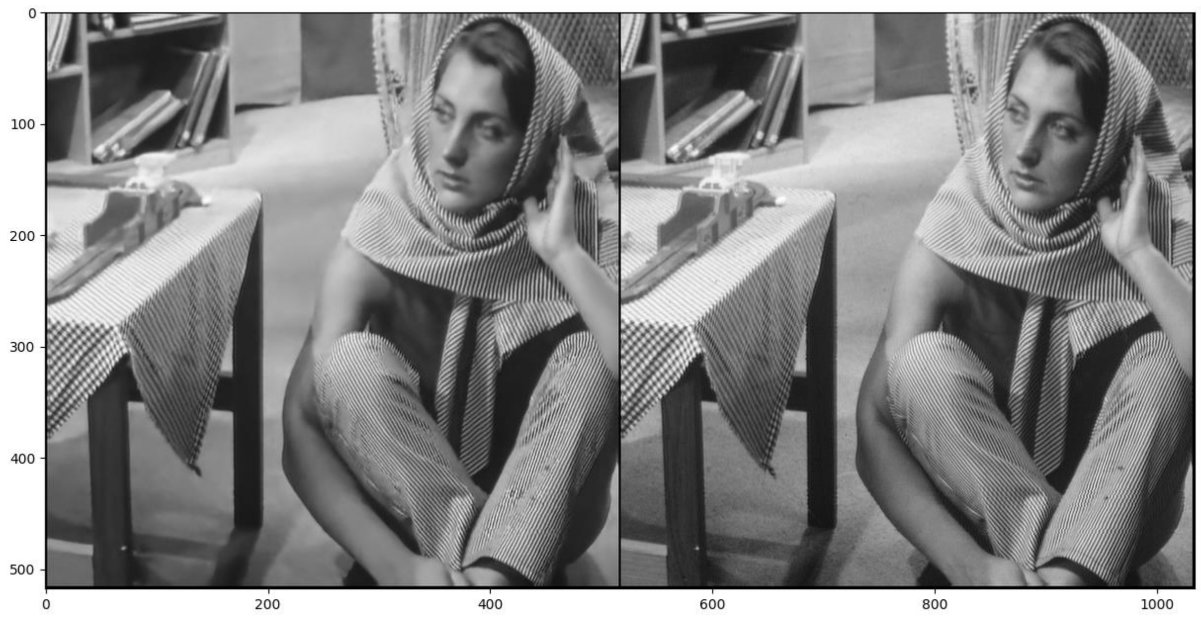
6. **Training Loop**

- **Iteration Setup**:
  The training loop is executed at an interval of 5000 iterations for barbara.png, and it runs 1000 iterations for kate.png.

- **PSNR Monitoring**:
  The regular readings of the measured PSNR values will be kept in record and also checked whether the quality restoration is carried out or not.
- **Checkpointing**:
  When the model is returned to the last checkpoint in cases where PSNR declines appreciably during the training, performance should be expected to maintain so.

7. **Final Output**

- **Restored Image:**
  After a minimal number of iterations, the output generated by the network will be something like this:
  - They are showing reduced noise and sharper image features.
  - Compared to the original masked image, this gives a very good restoration.
- **Evaluation**:
  The PSNR of the output image will be a numeric measure for success in the restoration.

3.8 Sr_Prior_Effect (Breakdown of the Code)

1. Importing Necessary Libraries

- The script imports the necessary libraries required for image processing, machine learning, and visualization.
  - **Matplotlib** is capable of visualizing images and drawings of results.
  - **Scikit-image** computes image quality metrics like Peak Signal-to-Noise Ratio (PSNR).
  - **PyTorch** brings accelerative features to neural network implementation and optimization using GPUs.
  - **NumPy** contains numerical computations.
- Calls CUDA for performing computations in a GPU so that it can speed up training when available.

2. Image and Down sampling Setup

- Load the high-resolution image and downsample it into a low resolution (LR) image using bicubic interpolation (LANCZOS).
- **Function load_LR_HR_imgs_sr() preprocesses these images:**
  - The HR image is resized to specific dimensions where applicable.
  - The generation of the LR image occurs by downscaling the HR image by specific ratios such as 4.
- Base LR images through bicubic, nearest-neighbour and sharpening methods for comparisons.
- Resolves and displays image resolutions while also visualizing the original HR, down sampled LR, and baselines images.

3. Defining Neural Network Architecture

- A trivial identity network for experimentation.
- A skip connection based network using the skip() function.
  - Down-sampling having up-sampling with skip connections along with image detail preservation.
  - Use LeakyReLU as the activation function; bilinear interpolation is adopted for up-sampling.
  - Ensures that high-res details are retained during the reproduction.
- The construction of the network is dynamic concerning source image size along with the channels.
- Assign parameters such as the number of layers to restrict the balance between computational cost and quality.

4. Quality Metric (PSNR)

- The PSNR measurements will for every training iteration compare how things are:
    - PSNR_LR: Images produced by the network imitate the original LR image.
    - PSNR_HR: Very much the same with the last one but referring to the network output HR actual HR image.
- Higher PSNR values indicate better reconstruction quality.

5. Regularization Techniques

- Introduces tiny random noise in the input when training (for instance, set to' ref_noise_std').
    - Results to a less overfitting model on noisy input image.
    - Supports the generalization ability of the resultant network.
- **Total Variation (TV) Prior**:
    - Having smoothness within the HR output while maintaining sharp edges.
    - Controlled by a regularization weight (tv_weight).

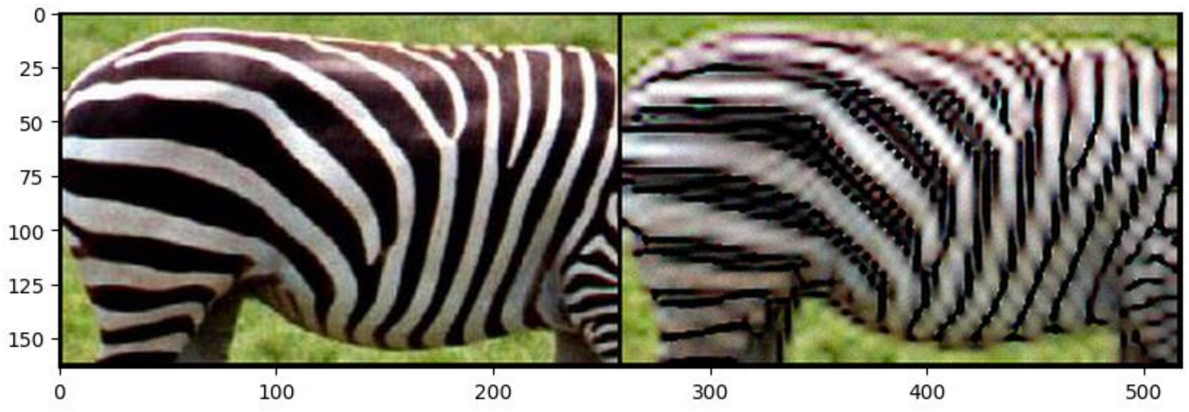
6. Visualization

- **Plots key results:**
    - HR ground truth image.
    - An LR image generated from different baselines (bicubic, sharp, nearest neighbour).
    - Final reconstructed HR images using varied techniques for regularization:
        - **Without Prior**
        - **With TV Prior**
        - **With Deep Image Prior**

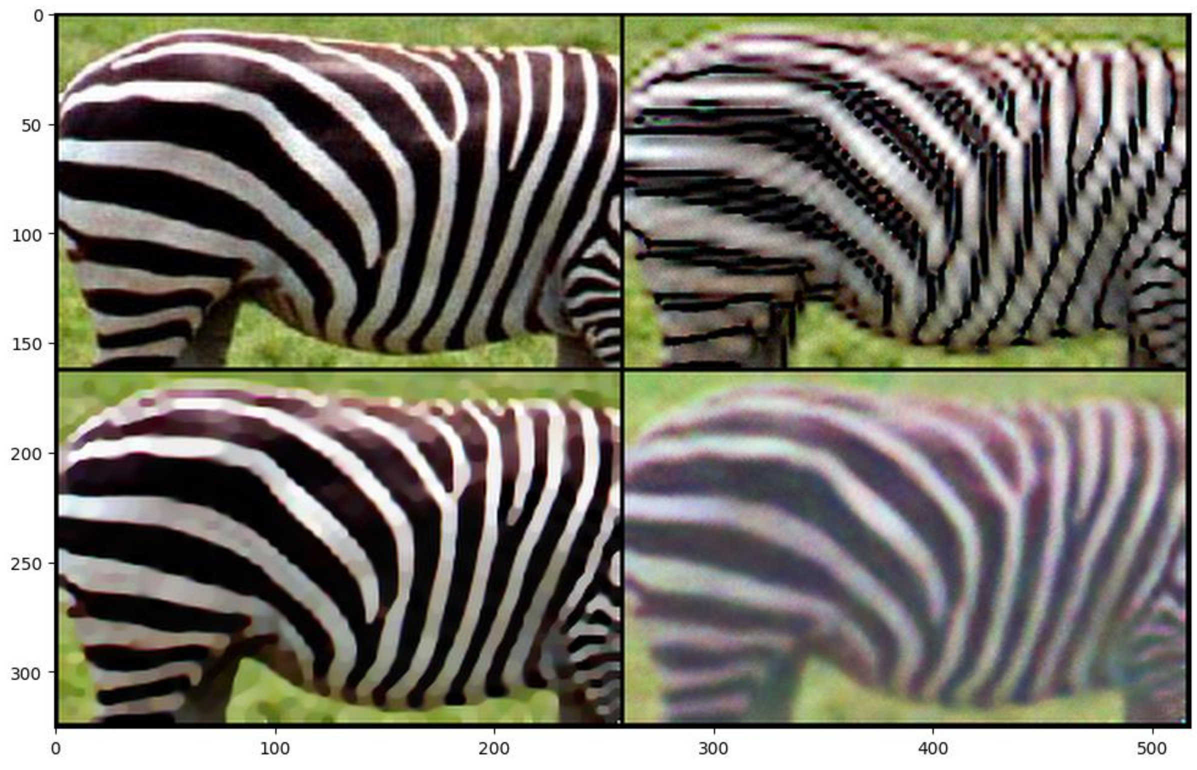
7. Final Output

- Three high-resolution reconstructions:
    - **No Regularization**: The network is directly optimized.
    - **TV Prior**: Smoothness Regularization in HR output.
    - **Deep Image Prior**: Media induction of the network bias is used for reconstruction.
- PSNR and visual inspection data are evaluated.
- Outputs that show significant noise reduction and detail restoration in HR images.

0
25
50
75
100
125
150
0
200
400
600
800
1000



0
25
50
75
100
125
150
0
100
200
300
400
500



0
50
100
150
200
250
300
0
100
200
300
400
500

3.9 Challenges and Solutions

Challenges:

1. **Preserving Image Details**: Removing noise without losing important image features is practically the greatest obstacle in denoising; if overly smooth, the edges and textures will be blurred, resulting in the loss of information values in the image.
2. **Balancing Noise Removal and Overfitting**: In the case of unsupervised denoising, with dearth of ground truth data, making it challenging to appraise performance during training. As there is no ground truth, the network tends to overfit the noise and may more learning noise removal by memorization instead.
3. **Choosing the Right Network Architecture**: Choose an appropriate architecture which differs in complexity of the model, and performance is quite a challenging task. Deeper networks may easily overfit whereas unsophisticated networks face significant challenges in comparatively complicated noise observation.
4. **Handling Different Noise Levels**: In addition, levels of noise affect model performance. By the level of noise present in the image. Higher severity of noise more would impede the quantity of valuable, useful information Required for denoising to be extracted.
5. **Computational Complexity**: Training a quite deep neural network would consume time and memory having incompatible resources throughout its iterative run while handling large enough images. This is a computationally expensive task.

Solutions:

1. **Skip Connections in Network Architecture**: In layman's words, skip connections, helping neural networks preserve fine details in images behind the scenes either during the down-sampling or up-sampling process, of course prevents the loss of critical information that may have gone missing due basically to convoluted intermediary layers the information passes through.
2. **Noise Regularization**: Generally, noise is a variation added to the input in order not to memorize the input image and as a distance-regularization method offering a range of properties against overfitting. The network learns more generally from one denoised image to another.
3. **PSNR as a Performance Metric**: Having no actual ground truth, the PSNR will tell how well the model has denoised the image through a qualified measure
4. **Fine-Tuning Hyperparameters**: In this case, we fine-tuned the strengths of the regularizer and the learning rate to recover some trade-off between denoising and salvation of the structure of the image.
5. **Optimization Techniques (Adam)**: Using Adam adapted optimizer helped speed up convergence; hence, the network exhibited much capability for learning, efficiently and adaptively according to gradients, thus shortening total training time.

3.10 Project Milestones and Achievements

Milestones:

1. **Project Initialization and Setup**: It covered all the initialization steps required for the project environment, which included necessary libraries like PyTorch and Skikit-Image, as well as preliminary setting up of the dataset, particularly if it is for the kind of task dealing with image denoising. Project Objectives/Scope This project would be about unsupervised image denoising techniques through deep learning models.
2. **Network Architecture Design**: Probably, the most important point is where architecture design strategy of the neural network is invented. Among plenty of alternatives chosen, a Skip Network for the conservation of important details of an image is implemented at the denoising step. Model parameters, like input depth, layer configuration, and up sampling techniques, were configured towards a denoising task.
3. **Noise Addition and Input Preparation**: To simulate conditions that are life noisy, synthetic Gaussian noise was added to clean images. Some amount of noise was defined to be sigma; it was then used to generate noisy images that the model was to take as its input to clean.
4. **Denoising Model Training**: In this stage, the model was trained for iterations set to 3000 in the primary experiment with Adam as the optimizer. It marked the culmination of the test where the noise was progressively reduced, and quality was restored to an image based on performance metrics like PSNR.
5. **Model Optimization and Regularization**: Training regularization noise the discourages the model to overfit during the training; apart from this, other important hyperparameters like the learning rate and the strength of noise optimization are carried out in such a way that it balances the removal of noise with details inside the image.
6. **Final Evaluation and Visualization**: In the last test, a visual comparison was done with noised images and images denoised, complemented by a quantitative evaluation of performance as PSNR. Of course, it resulted in impressive improvement on the quality of images from clear images output by inputs.

Achievements:

- Implemented a skip network for unsupervised image denoising that was successful in proving it to maintain details in an image very effectively.
- Designed a pipeline that achieves holistic denoising of the images and removes most noise without any reference to ground truth.
- There is a visually perceivable enhancement of the quality in the image due to high PSNR values and visually improved outputs.
- Noise regularization and adaptation of learning rates were used, resulting in a generalized solution adequate for any level of noise and type of image.

Chapter 4

Conclusion

The techniques of **Unsupervised Image Segmentation and Denoising** in Python have developed into methods which have had a very considerable impact on improving image quality by removing noise very efficiently and effective, with no input of ground truth. The flexibility that Python provides, along with a very powerful ecosystem, made the integration of a model like Skip Networks for denoising tasks smooth and engaged with many image processing situations that require fluency and flexibility of such networks.

For all of these reasons, this becomes an expansion of projects that will use Inpainting, Activation Maximization, Feature **Inversion**, **Restoration**, and **Super Resolution**-an entirely new design that would allow the model to perform missing part recovery, generate optimal feature visualizations, and up the resolution of images. The project would settle the tasks and boost its firm and stable platform to address a whole spectrum of tasks connected with image manipulation and restoration work.

Unsupervised image analysis and image restoration and enhancement would continue to be on the frontier as the technology matures, paving the way for scalable, customizable, low-cost solutions in image processing using Python-based models.

Chapter 5

Future Work

Over the years, **GANs** have more recently been positioned as an alternative for enhancing denoising. Deep learning frameworks through self-supervised learning methods further this work to provide more complete coverage of segmentation outputs, incorporating transformer models in understanding and image restoration.

One other suggestion revolves around some upscale visual tools which play a significant role in providing ease of interpretation for the output, enhanced capacity of the model for large-scale datasets, and accelerated training performance of neural networks. One of the forthcoming key domains whereby this project will yield value is real-time image processing-denoising capabilities integrated into the multifarious aspects of the 3D image denoising and segmentation processes.

This shall prove to be untenably welcoming to those domains where integrated approaches can find an instance in numerous sectors of medical imaging, remote sensing, and autonomous driving. Not to mention the open source to the Outreach of the masses towards endless possibilities for improving scalability, use of state-of-the-art evaluation metrics, furthering a framework opening up for adaptations with the crowning rewards of nudging fresh and emerging quandaries in unsupervised image processing.

By pursuing these tasks, this project optimistically anticipates several advances in the frontiers of image restoration and segmentation toward more accurate and effective solutions for several image-related tasks.

References

1] J. Paul, "Unsupervised machine learning based SEM image denoising for robust contour detection," *ResearchGate*, Sep. 2021. [Online]. Available: https://www.researchgate.net/publication/354981796_Unsupervised_machine_learning_based_SEM_image_denoising_for_robust_contour_detection. [Accessed: Oct. 9, 2024].

[2] G. Liu, X. Sun, J. Liu, and Y. Shi, "Unsupervised image segmentation via Stacked Denoising Auto-encoder and hierarchical patch indexing," *Pattern Recognition Letters*, vol. 96, pp. 77-84, Sep. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0165168417302554. [Accessed: Oct. 9, 2024].

[3] D. Zheng, "Unsupervised denoising," *GitHub*, 2019. [Online]. Available: https://github.com/zhengdihan/Unsupervised_denoising. [Accessed: Oct. 9, 2024].

[4] D. Kuang, P. Gong, and G. Wang, "PET image denoising using unsupervised deep learning," *Springer*, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s11042-020-08882-9. [Accessed: Oct. 9, 2024].

Chapter 6

Appendix

6.1 Libraries and Tools Used:

1. **Python**: Basic programming language used for writing and building the codes of the development and implementation of the algorithms for denoising and segmentation.
2. **PyTorch**: PyTorch was adopted in developing and building the neural networks in skip connection-based architectures.
3. **Scikit-Image**: The module holds primitives of class-related image processing operations like "introduction of noise" and the management of file images.
4. **Matplotlib**: plots and visualizes images during train and evaluation.
5. **NumPy**: All computations related to arrays and numerical calculations take place in image and tensors manipulations.

6.2 Datasets and Image Formats:

1. **F16_GT.png**: Testing image to test the performance of denoising algorithm. The given image can be taken as a more realistic source image, because synthetic Gaussian noise has been added.
2. **Image Formats**:
    - The images are saved and processed using both JPEG and PNG format, and in these formats, resolution and colour depth will be parameters in question thereby affecting the task of denoising.

6.3 Neural Network Architecture:

1. **Skip Connection-Based Network (U-Net)**:
    - This is based on the architecture of U-Net, which includes skip connections to combine features at low and high levels. Therefore, the denoising performance has also improved.
2. **Deep Image Prior (DIP)**:
    - This is an unsupervised model, rather than a labelled dataset for denoising, learned directly from a noisy input image.

6.4 Optimization Techniques:

1. **Adam Optimizer**:
   - This is one among gradient-based optimization algorithms used in the training process aiming at minimizing Mean Squared Error (MSE) loss.
2. **MSE Loss Function:**
   - Average squared deviation between the noisy image and the predicted or denoised image that drives the optimization.

6.5 Key Features Implemented:

1. **Unsupervised Denoising**:
   - It is an algorithm for noise removal from images, which requires no need of having labelled training data
2. **Unsupervised Image Segmentation**:
   - Pattern extraction from data is internal: It comes to represent image segmentation into meaningful regions.
3. **Adaptive Noise Removal**:
   - The variational feature, which takes an adaptive noise removal level based on the type of regions in a picture, maintains fine details inside it but focuses more on the noisier areas.

6.6 Challenges Encountered:

1. **Overfitting**: Noisy images in an overfitting format were handled by regulating techniques and careful adjustments to hyperparameters.
2. **Handling High Noise Levels**: This was quite a tough point, since it is needed to keep enough noise removal without erasing vital details in the image.

6.7 Future Improvements:

1. **GANs for Denoising**: Improved Quality of Denoising Images Using Generative Adversarial Networks (GANs).
2. **Addition of Self-Supervised Learning**: Self-supervised learning on top of the segmentation and denoising technique.