



Capstone Project Phase B

Image Enhancement with Super-Resolution and Controlled Noise

24-1-R-12

Students:

Liza Shvachka– 206236176

Liza.shvachka@e.braude.ac.il

Avishai HersHKovitz– 209460443

Avishai.hersHKovitz@e.braude.ac.il

Supervisor: Dr. Renata Avros

Advisor: Prof. Zeev Volkovich

Table of Contents

Abstract.....	2
1. Introduction.....	3
2. Product.....	3
3. Dataset.....	4
4. Research Process.....	4
4.1 Process.....	4
4.2 Research	5
5. Models.....	6
5.1 Super Resolution	6
5.2 Noise2Noise	8
6. Results.....	11
6.1 Super Resolution	11
6.2 Noise2Noise	12
7. Testing the model.....	18
8. Conclusions.....	20
9. User Documents.....	20
9.1 User Instruction.....	20
9.2 Maintenance Guide	27
10. References.....	29

Abstract

This paper explores signal reconstruction using machine learning, with a specific focus on image enhancement tasks. The first approach uses a single model for various image reconstruction tasks, including denoising and denoising synthetic images, all based only on corrupted data. This demonstrates the effectiveness of learning recovery without explicit image or corruption models. The second approach addresses image super-resolution (SR), a challenging task due to its inherent ambiguity. Previous deep learning methods struggle with non-stationary degradations, limiting their effectiveness in some applications. To deal with this, we propose a basic U-net and a robust (RU-net) architecture that learns the relationship between degraded low-resolution images and high-resolution images. The main idea of the project is to test the most efficient approach for image enhancement tasks.

Our project files and source code will be available at: [GitHub](#).

1. Introduction

Image restoration and super resolution, techniques for converting low quality images to high quality. Dealing with the challenge of restoring images based only on corrupted images. In the past, traditional statistical models such as linear regression and wavelet transformation were used. But these models require a deep understanding of the underlying processes and statistical properties. In the real world the images are varied including, different levels of exposure (short and long exposure shots), analysis of aerial photographs and when there is a latent/hidden object in the image. Furthermore, image super-resolution (SR) has made significant strides with the advent of deep learning techniques. Adversarial network (GAN) based methods have shown promise in generating realistic results. But they rely on a Bicubic down-sampling model that does not fully represent real-world scenarios, which limits their effectiveness, and they often face instability in training.

In this project we focus on the process of turning low quality images into high quality by learning corrupted images where the machine learns the way of corruption indirectly from the training data. The methodology involves training the U-net model several times using a data set that includes pairs of corrupted input. By analyzing the nature of the corruption, the model learns to efficiently restore the images, thereby improving their quality. To address the challenge of super-resolution, a robust U-net (RU-net) architecture is proposed. This approach includes connections between distant layers within the network and a degradation model that varies between different regions of the image.

2. Product

Our product focuses on improving image quality through two models: Super-Resolution and Noise2Noise, image restoration without clean data. In the process of super-resolution, we use a Robust U-Net model to upscale low-resolution images to high resolution, reproducing fine details and improving clarity. Additionally, at noise2noise, we add synthetic noise to images to simulate real-world scenario. We use a U-net model to indirectly learn the nature of the noise and restore the image to a clean image.

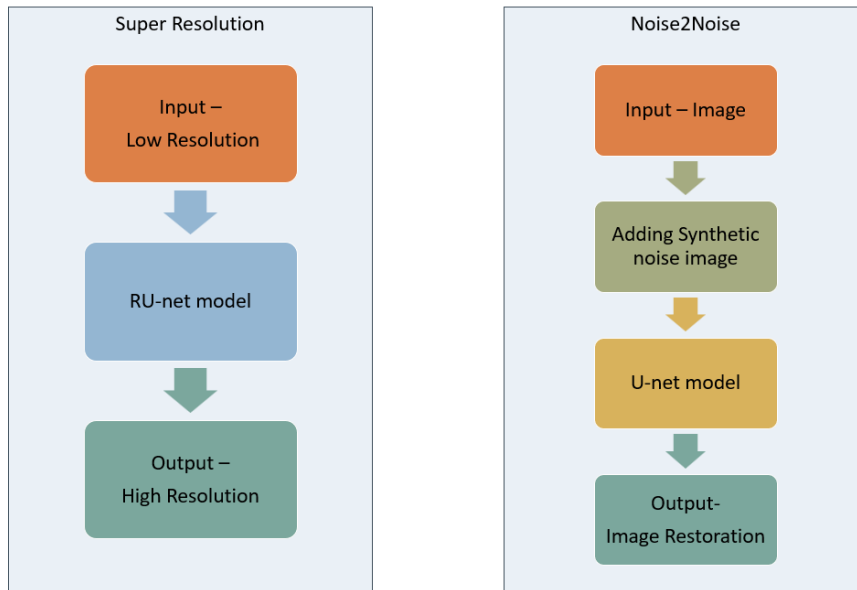


Figure 1. Product models.

3. Dataset

Super Resolution

To train our network we took some pictures from Linnaeus 5 dataset. The dataset is offered for multi-class classification between 5 labels (berry, bird, dog, flower, other) and it has 6000 colored images as training set and 2000 as test set.

We decided to take 2000 of 256x256 pixels images to train our dataset. We split it in training (85%) and validation set (15%). For the testing phase, we used 128x128 pixels images.

Noise2Noise

To train Noise2Noise network, we used images from the COCO 2017 dataset. This dataset, commonly used for object detection and segmentation tasks, includes a smaller validation set, around 1 GB in size, which we split into training and validation sets (85%/15%). For training, we used images resized to 256x256 pixels to ensure consistency across the dataset.

4. Research Process

4.1 Process

The main goal of the project is to transform an image from low resolution to high resolution. We do this by studying how noise affects the image and what types of noise can be used to restore the image to the cleanest version.

In part A of the project, we researched various image reconstruction methods, focusing on classifying noisy images and explore restoration techniques. Additionally, we

conducted a detailed study of the U-Net architecture, analyzing its functionality and testing methods to enhance image resolution for cleaner outputs.

In this Part B, we explored the Super Resolution model with RU-net architecture and the Noise2Noise (N2N) model, which leverages U-Net. Both models aim to enhance degraded images, but they approach the task from distinct perspectives. The Super Resolution RU-net model is specifically designed to upscale low-resolution images to higher resolutions, while the Noise2Noise model is used to denoise images by learning to predict clean versions without clean images examples. The primary focus of our investigation was to assess the performance of these models when subjected to three types of noise: Gaussian, Bernoulli, and Poisson. These noise types are commonly found in practical scenarios and represent diverse challenges for image restoration models. To further this study, we compared the effectiveness of the two models by applying them to the same datasets with varying noise conditions. This comparative analysis helped to determine which model more effectively restored images under different noise types.

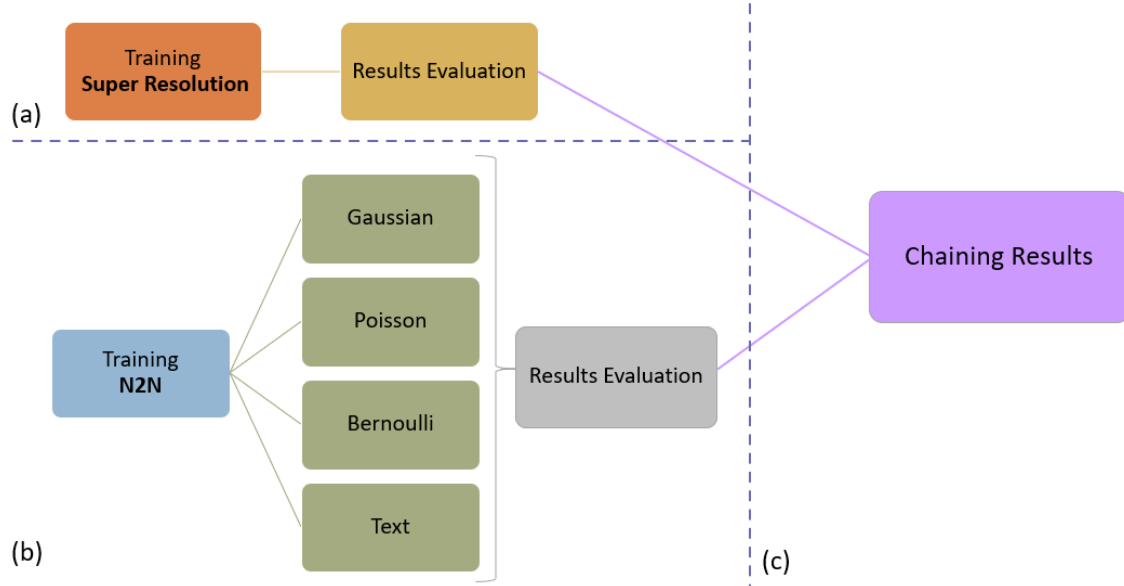


Figure 2. Process flow-chart- (a) Training and evaluate Super-resolution model. (b) Training and evaluate N2N model for each noise type. (c) Combining the results of the models, the output of the first model serves as the input to the second model.

4.2 Research

To begin the research, numerous methodologies, models, and architectures such as U-net, RU-net, and so on must be investigated. To put the various models produced to the test, they had to be implemented and trained, which took a long time. To address this constraint, each model was trained for only 10%-20% of the recommended number of iterations, and the statistical parameters were examined to determine if they were rising upward. To evaluate the effectiveness of the models, we conducted a systematic study by modifying several key training parameters- 'Hyperparameters'. One of the primary variables was the number of epochs, which directly influences how well the model learns

from the data over time. We also adjusted the size of the dataset used for training, exploring how different volumes of data affect model performance. Additionally, we varied noise intensity, adding different levels of Gaussian, Bernoulli, and Poisson noise to simulate various real-world conditions. Other parameters, such as batch size, learning rate, and optimizer choice, were also fine-tuned to observe their impact on model outcomes.

Throughout the training process, the models' performance was measured using three key metrics: Peak Signal-to-Noise Ratio (PSNR), Loss rate, and Structural Similarity Index Measure (SSIM) only for 'Super Resolution' model. By analyzing these metrics under different training conditions, we were able to draw significant conclusions regarding the robustness of each model to different noise types and parameter adjustments. This comprehensive study highlighted the trade-offs between model complexity, noise resilience, and performance optimization strategies.

5. Models

5.1 Super Resolution

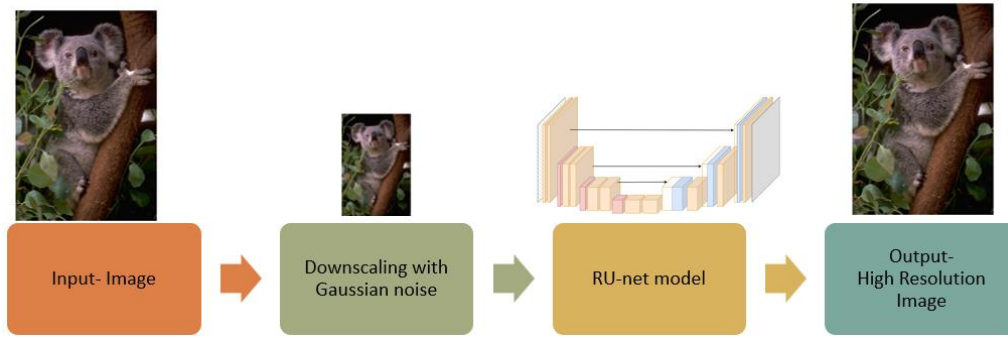


Figure 3. Workflow Super-Resolution

Input – An Image

The image size that was used is $256 \times 256 \times 3$.

Synthetic Low-Resolution

The high-resolution image is down-sampled by a factor of two, blurred using Gaussian noise, and then up-sampled by a factor of two, resulting in a blurry image with indistinct features.

RU-net Model

The RU-net model introduces a new resolution enhancement approach, consisting of a degradation module and an improved U-net architecture. The degradation module corrupts input images in various ways, while the U-net focuses on enhancing their resolution. Key features include concatenate blocks for better information exchange and pixel shuffle for efficient up-sampling without adding extra parameters.

In our convolution blocks we used the same kernel size (3,3) same as VGG net. In the convolutional layers we used zero padding in order to preserve the dimension after each

convolution. In some residual connections we rescaled the input by means of 1x1 convolutive block.

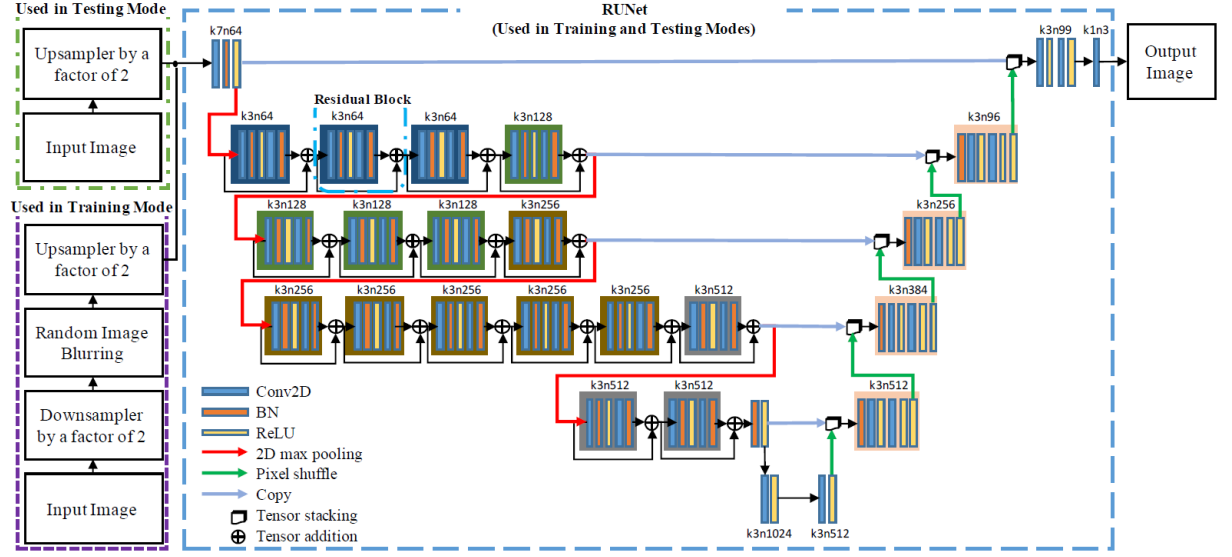


Figure 4. RU-net model [1].

Implementation was written in Google Colab [4], there was limited access to GPU.

Hyperparameters

- *Input size*–256
- *Batch size*–16
- *Adam optimizer*
- *Learning rate*–0.001
- Epochs - 20
- Perceptual *loss* function - capture the differences in image structure and features by using activations from deeper layers of a pre-trained model.

$$L^j = \frac{1}{C_j H_j W_j} \|\phi_j(\text{GroundTruth}) - \phi_j(\text{PredictedImage})\|_2^2$$

- Corrupts input by Gaussian Noise(0.1)- standard deviation of the noise added to each pixel.
- Metrics: PSNR, SSIM and (MSE).

Architecture Plot

Layer	Function	Output Shape (H × W)	Channels (C)
0	Input layer	(256, 256)	3
1	Conv2D + BN + ReLU	(256, 256)	64
2	Residual Block x 4	(128, 128)	128
3	Residual Block x 4	(64, 64)	256
4	Residual Block x 6	(32, 32)	512
5	Residual Block x 2	(16, 16)	1024
6	Residual Block	(32, 32)	384

7	Residual Block	(64, 64)	256
8	Residual Block	(128, 128)	96
9	Residual Block	(256, 256)	99
10	Conv2D	(256, 256)	3

Table 1. . RU-net architecture plot.

5.2 Noise2Noise

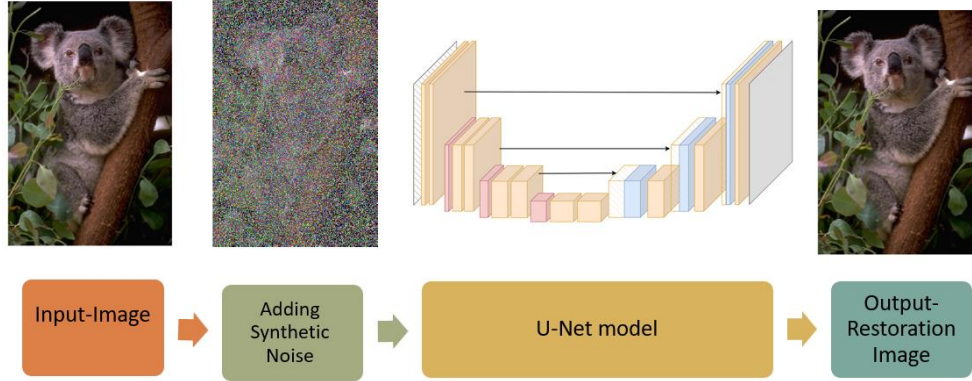


Figure 5. Workflow N2N

Input – An Image

The image size that was used is 256×256×3.

Adding Synthetic Noise

In this phase, synthetic noise is artificially introduced to the dataset containing images. We add various types of synthetic noise and trained the model for each one of them:

1. Gaussian noise

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

2. Poisson noise

$$P(x = k) = \frac{e^{-\lambda} * \lambda^k}{k!}$$

3. Multiplicative Bernoulli

$$\operatorname{argmin}_{\theta} \sum_i (m \odot (f_{\theta}(\hat{x}_i) - \hat{y}_i))^2$$

4. Text overlay- composed of several random strings in random places on the image.

U-net Model

The U-Net model is designed to map noisy input images to clean output images by learning the relationship between them during training. The model uses up-samples images using the nearest neighbor method. Averaging noisy pixel values across multiple realizations helps reduce noise in both the images and the weight gradients, making

training more stable and efficient.

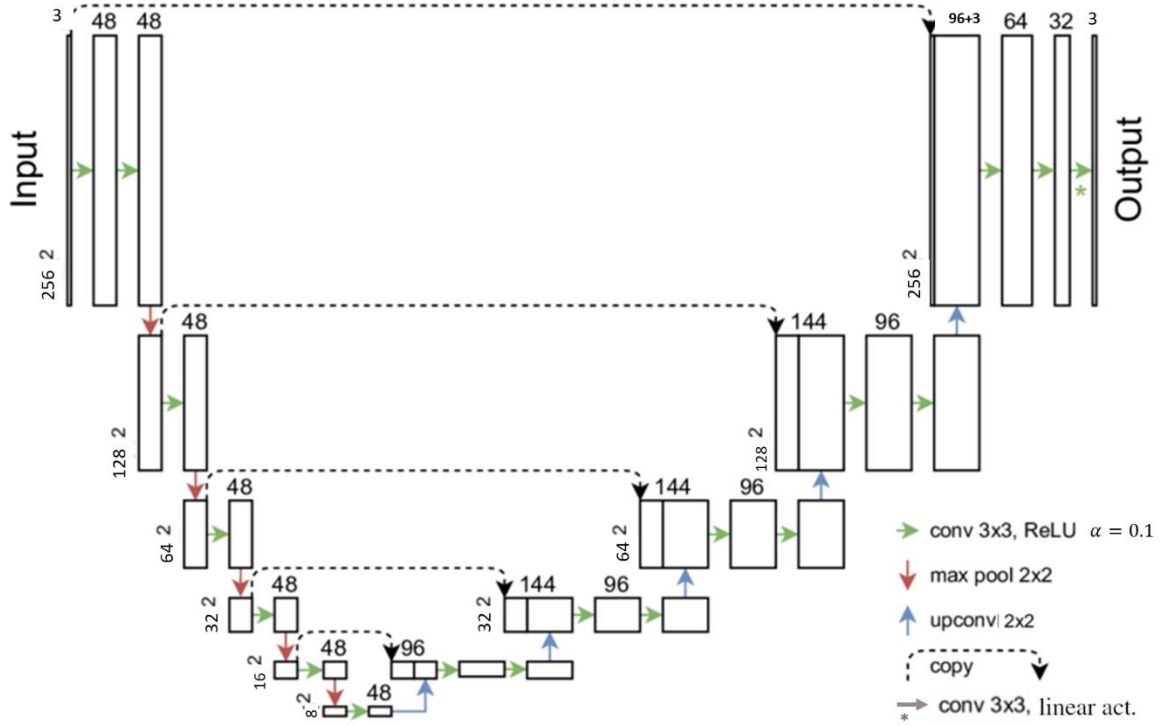


Figure 6. U-net Noise2Noise model [2]. An image is provided as input to the trained model. Then process the image through its layers, leveraging its learned denoising capabilities to remove noise while preserving important image details.

Implementation was written in Python in the PyCharm environment.

Hyperparameters

- Input size–256
- Minibatch size - 4
- Batch size–16
- Adam optimizer parameter values $\beta_1 = 0.9, \beta_2 = 0.99, \epsilon = 10^{-8}$.
- Learning rate–0.001
- L2-MSE loss function
- Epochs:
 - Gaussian, Poisson, Bernoulli - 30
 - Text - 105
- Noise params:
 - Gaussian - $\mu = 25$
 - Poisson - $\lambda = 1.2$
 - Bernoulli - $\mu = 1.2$
 - Text:
 - Font – times new roman.
 - Param – 0.2 - the proportion of pixels covered by text.
- Metrics: PSNR

Architecture plot

NAME	N_{out}	FUNCTION
INPUT	n	
ENC_CONV0	48	Convolution 3×3
ENC_CONV1	48	Convolution 3×3
POOL1	48	Maxpool 2×2
ENC_CONV2	48	Convolution 3×3
POOL2	48	Maxpool 2×2
ENC_CONV3	48	Convolution 3×3
POOL3	48	Maxpool 2×2
ENC_CONV4	48	Convolution 3×3
POOL4	48	Maxpool 2×2
ENC_CONV5	48	Convolution 3×3
POOL5	48	Maxpool 2×2
ENC_CONV6	48	Convolution 3×3
UPSAMPLE5	48	Upsample 2×2
CONCAT5	96	Concatenate output of POOL4
DEC_CONV5A	96	Convolution 3×3
DEC_CONV5B	96	Convolution 3×3
UPSAMPLE4	96	Upsample 2×2
CONCAT4	144	Concatenate output of POOL3
DEC_CONV4A	96	Convolution 3×3
DEC_CONV4B	96	Convolution 3×3
UPSAMPLE3	96	Upsample 2×2
CONCAT3	144	Concatenate output of POOL2
DEC_CONV3A	96	Convolution 3×3
DEC_CONV3B	96	Convolution 3×3
UPSAMPLE2	96	Upsample 2×2
CONCAT2	144	Concatenate output of POOL1
DEC_CONV2A	96	Convolution 3×3
DEC_CONV2B	96	Convolution 3×3
UPSAMPLE1	96	Upsample 2×2
CONCAT1	$96+n$	Concatenate INPUT
DEC_CONV1A	64	Convolution 3×3
DEC_CONV1B	32	Convolution 3×3
DEV_CONV1C	m	Convolution 3×3 , linear act.

Figure 7. U-net architecture plot [2].

6. Results

6.2 Super Resolution

Evaluation metrics

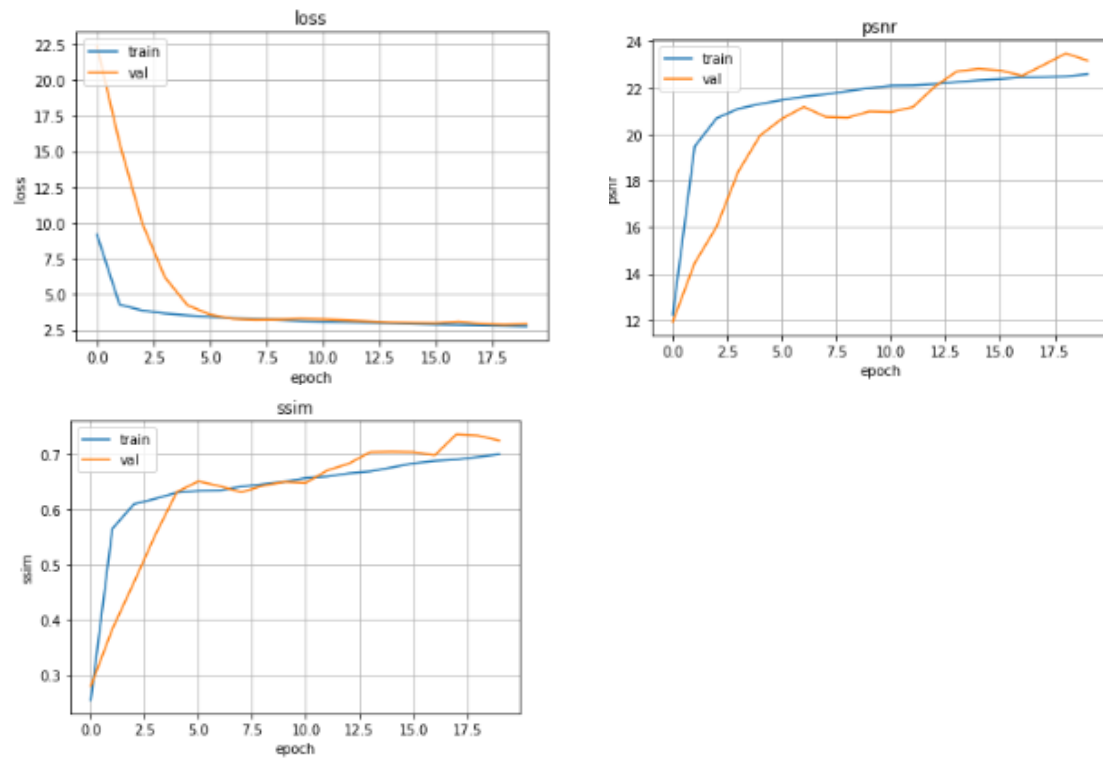
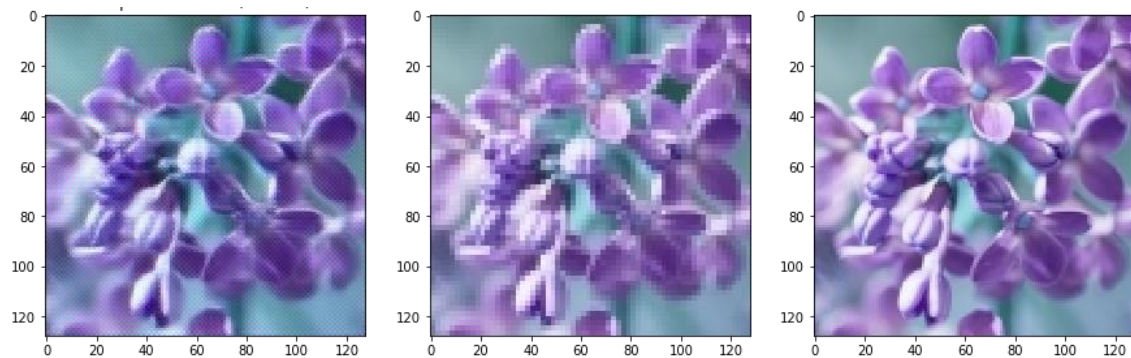
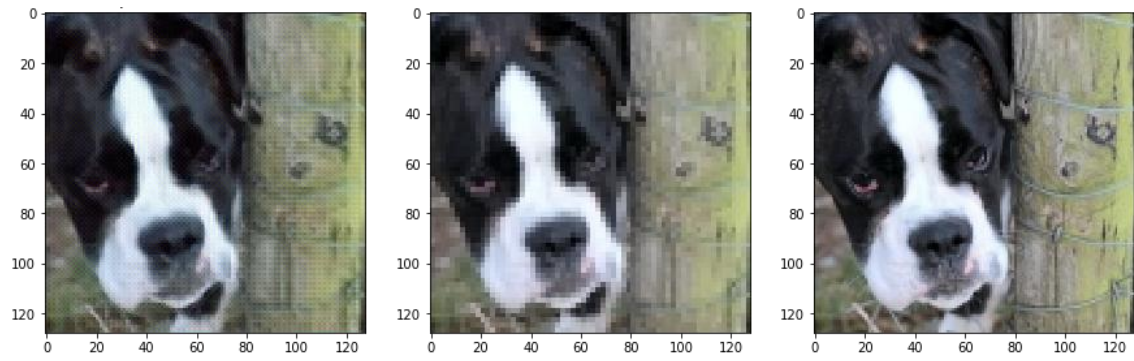


Figure 8. Evaluation metrics- Loss, PSNR and SSIM.

Examples





6.2 Noise2Noise

Training results for each noise type:

Gaussian

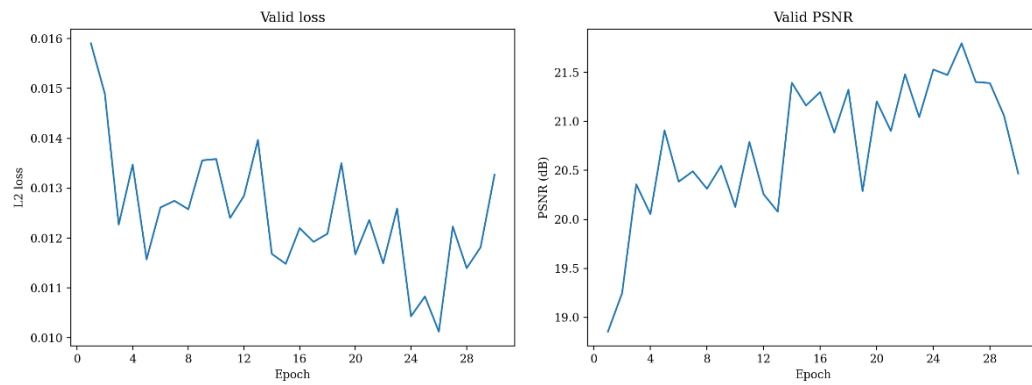


Figure 9. Evaluation metrics- Loss and PSNR.

Examples



Input: 15.60 dB



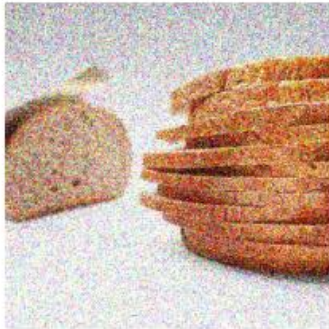
Denoised: 26.44 dB



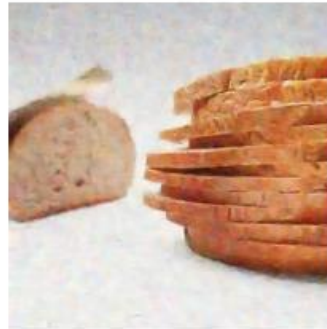
84_256



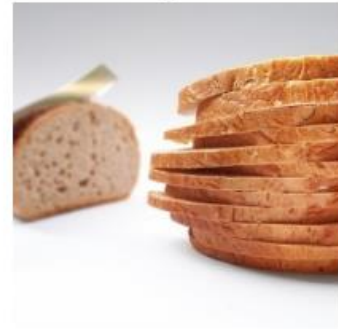
Input: 15.51 dB



Denoised: 25.93 dB



73_256



Input: 15.31 dB



Denoised: 27.83 dB



4_256



Input: 20.45 dB



Denoised: 30.65 dB



Ground truth





Poisson

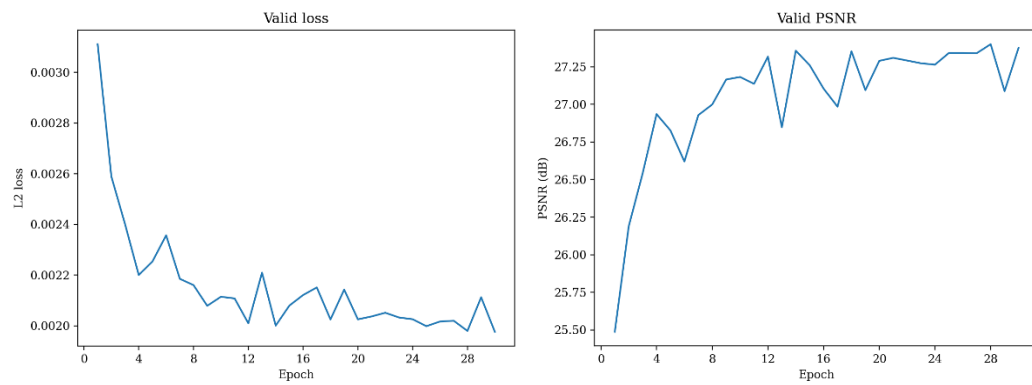
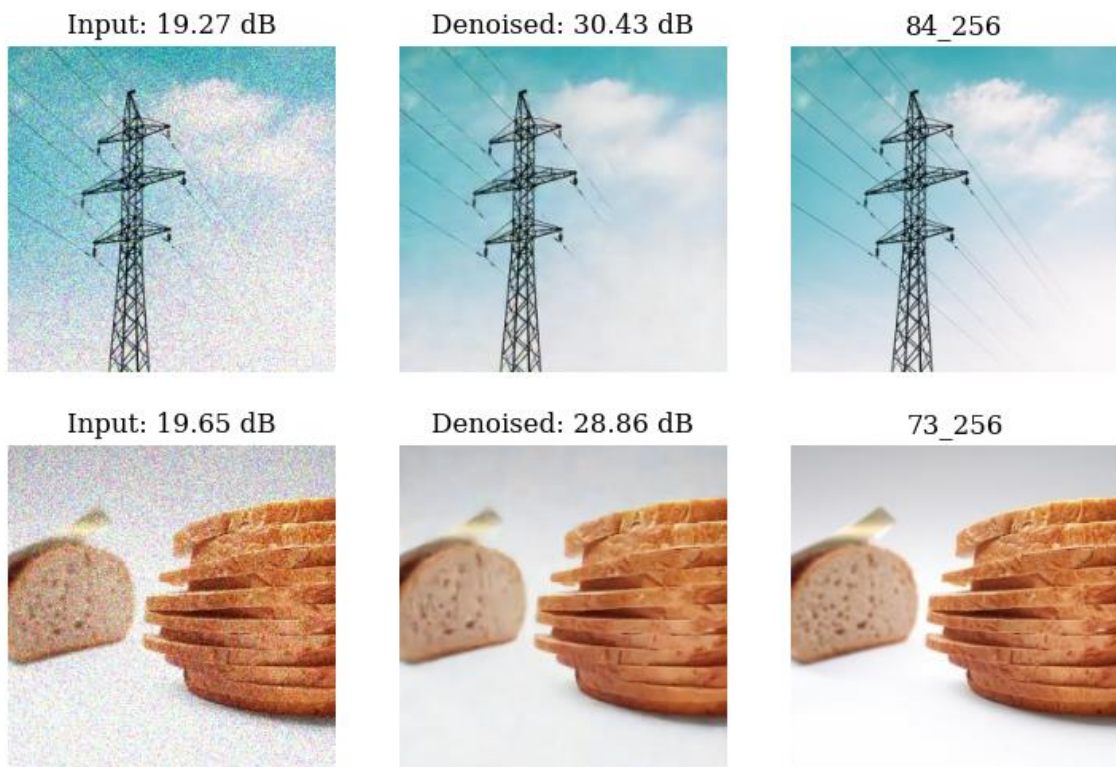
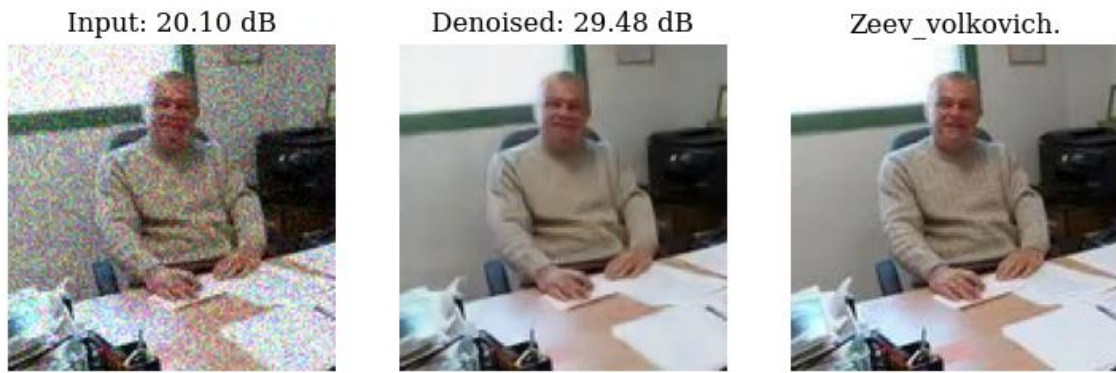


Figure 10. Evaluation metrics- Loss and PSNR.

Examples





Bernoulli

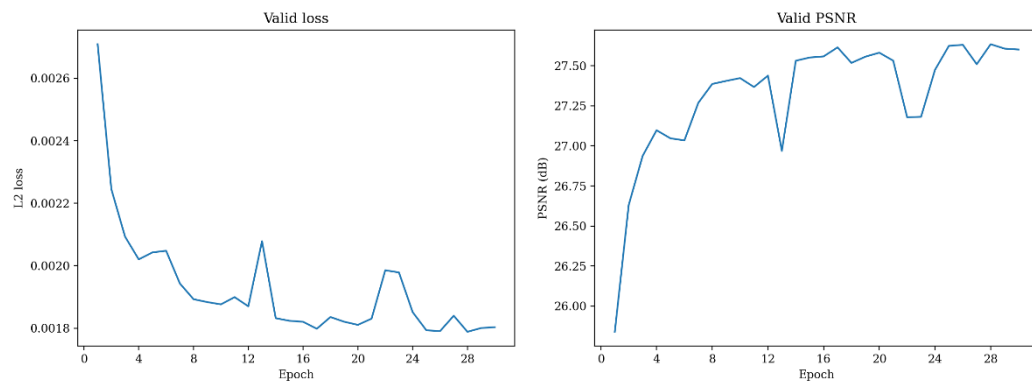
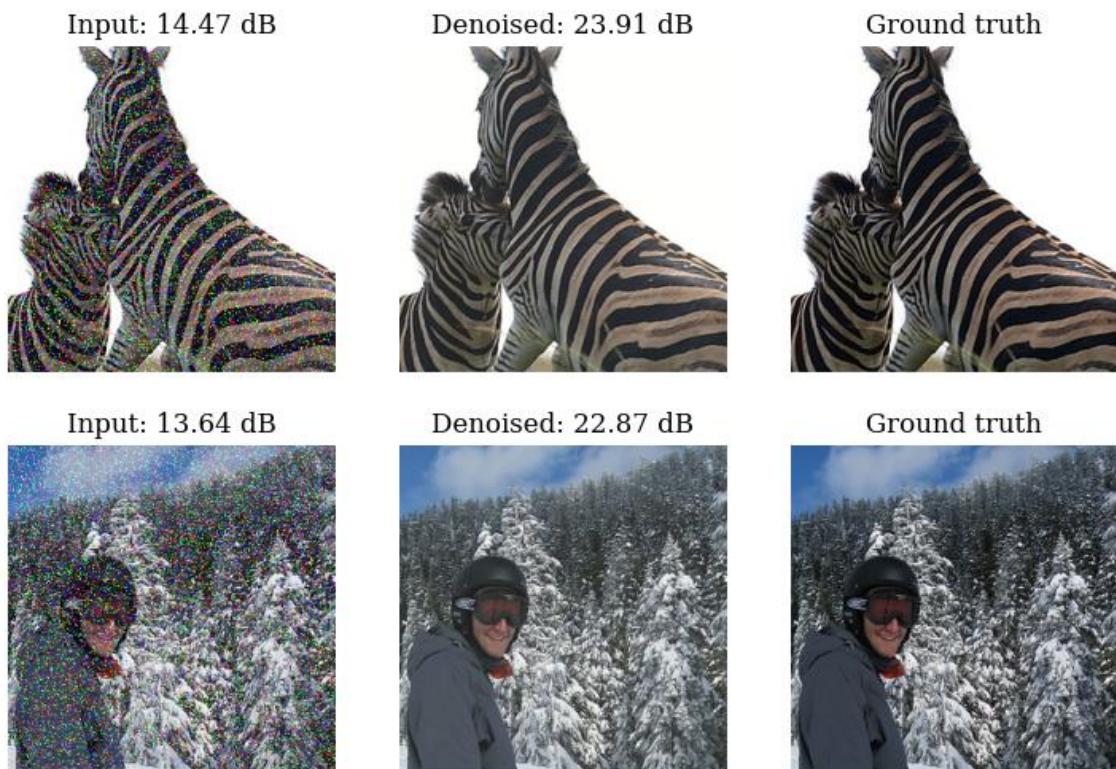
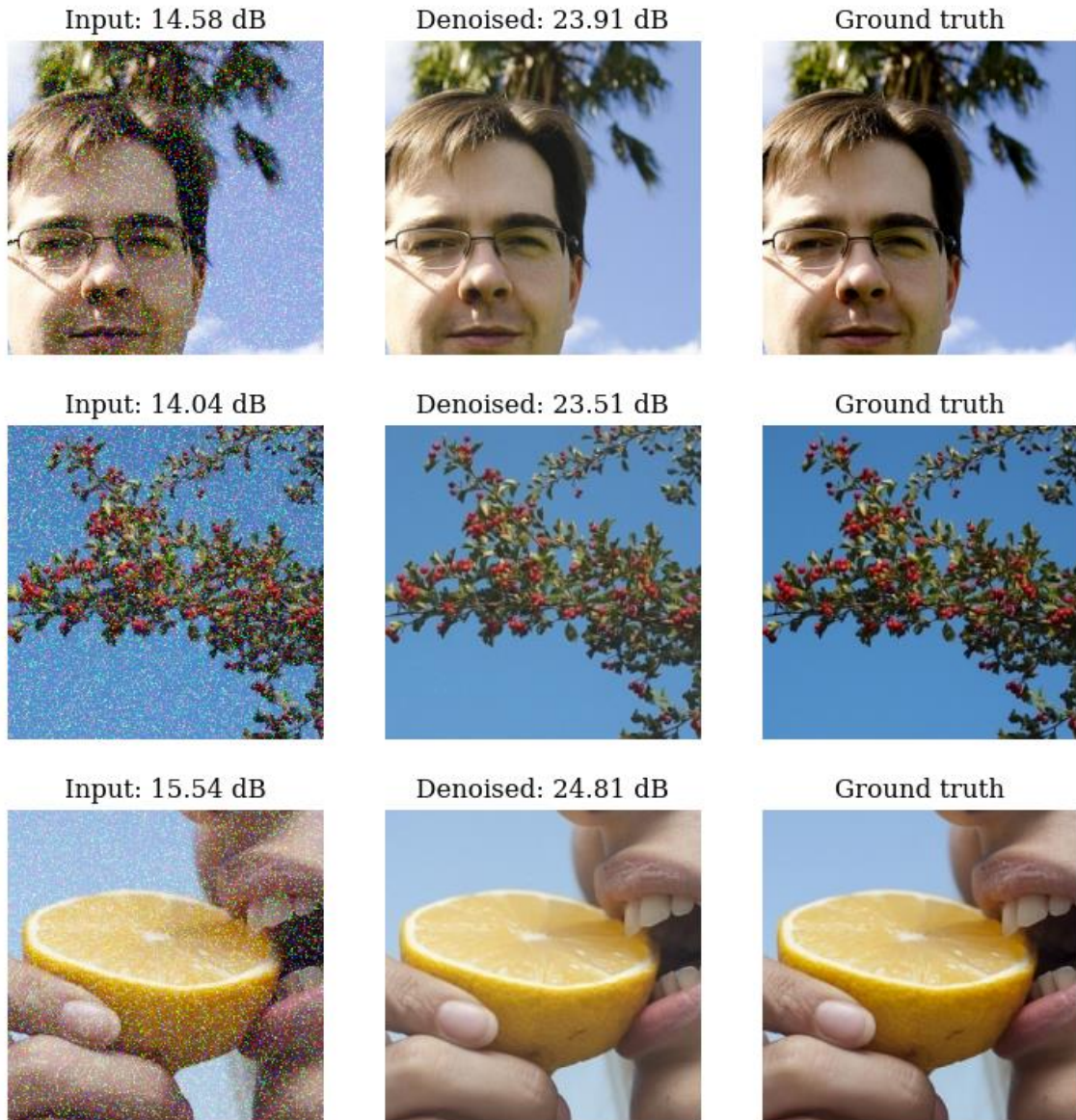


Figure 11. Evaluation metrics- Loss and PSNR.

Examples





Text overlay

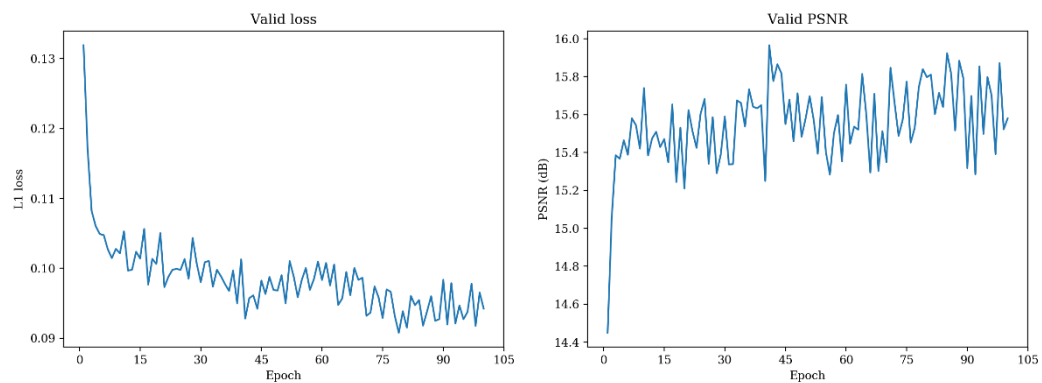


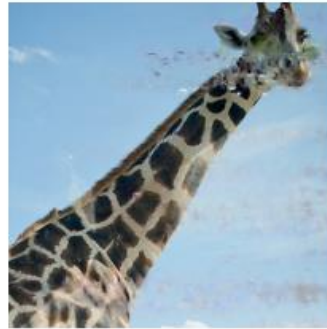
Figure 12. Evaluation metrics- Loss and PSNR.

Examples

Input: 15.79 dB



Denoised: 26.69 dB



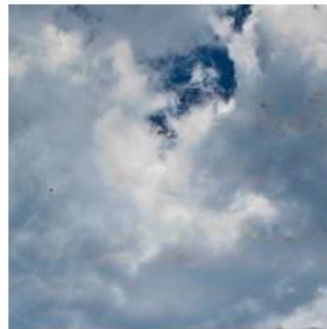
Ground truth



Input: 17.42 dB



Denoised: 33.95 dB



000000077460



Input: 16.59 dB



Denoised: 25.48 dB



Ground truth



Input: 16.96 dB



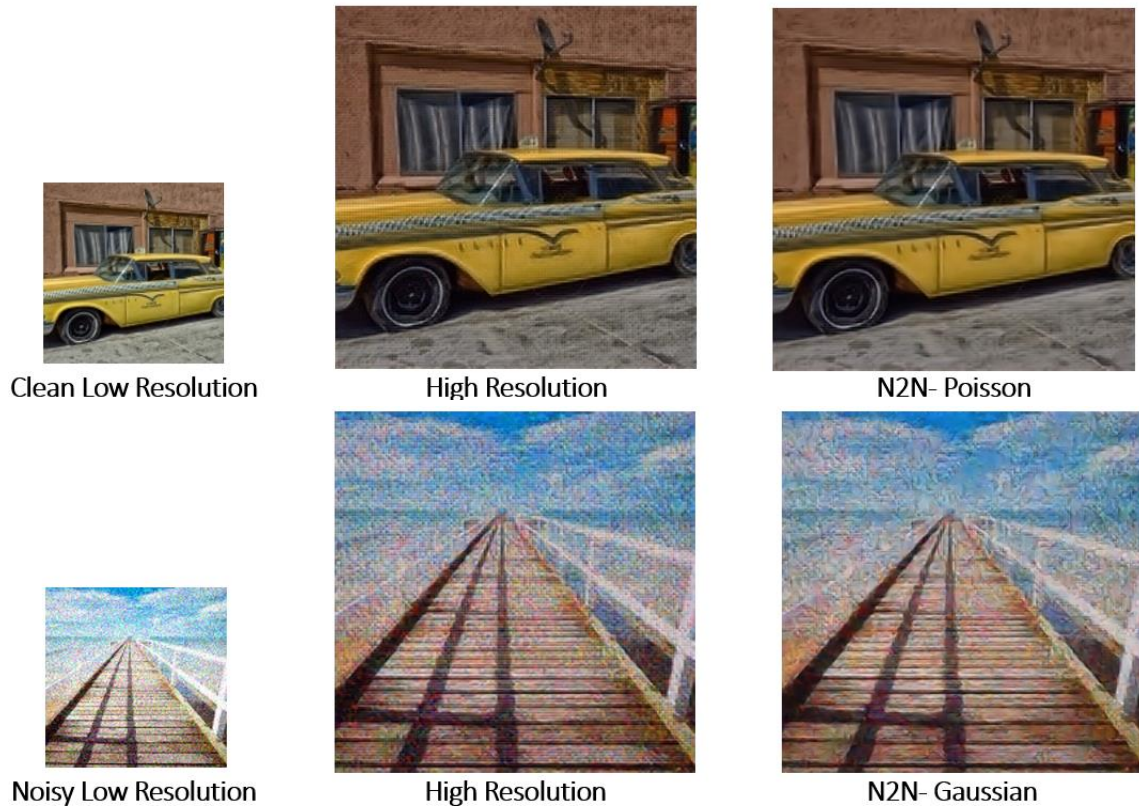
Denoised: 22.60 dB



105_128



Result Chaining- Super Resolution to N2N



7. Testing the model

For the evaluation of our results, we will apply some metrics related to image evaluation:

1. **PSNR**- Peak Signal-to-Noise Ratio - measures the quality of a reconstructed image by comparing it to a reference image, focusing on the signal-to-noise ratio. As the signal index increases, the quality of the processed image also increases, expressed in decibels (dB).
2. **SSIM**- Structural Similarity Index Measure - assesses how two images are structurally similar by considering factors such as brightness, contrast, and pixel arrangement, providing a score between 0 and 1.

Gui Testing

Test	Module	Test description	Excepted result
1.	Home page	Click 'Train Models' in navbar menu	Open 'Train Models' window
2.	Home page	Click 'Restore an image' button in navbar menu	Open 'Restore an image' window
3.	Home page	Click 'Upload dataset' button in navbar menu	Open 'Upload dataset' window
4.	Train Models window>Type of Model	Click on combo box 'Type of model' and choose 'Super resolution'	Option 'Super resolution' should be chosen.
5.	Train Models window ->Type of model	Click on combo box 'Type of model' and choose 'Noise2Noise'	Option 'Noise2Noise' should be chosen.
6.	Train Models window ->Type of model (Noise2Noise)->Type of noise	Click on combo box 'Type of noise' and choose 'Gaussian'	Option 'Gaussian' should be chosen
7.	Train Models window ->Type of model (Noise2Noise)->Type of noise	Click on combo box 'Type of noise' and choose 'Poisson'	Option 'Poisson' should be chosen
8.	Train Models window ->Type of model (Noise2Noise)->Type of noise	Click on combo box 'Type of noise' and choose 'Multiplicative Bernoulli.'	Option 'Multiplicative Bernoulli' should be chosen
9.	Train Models window ->Type of model (Noise2Noise)->Type of noise	Click on combo box 'Type of noise' and choose 'Random text.'	Option 'Random text' should be chosen
10.	Train Models window ->Type of model (Noise2Noise)->Type of noise	Click on combo box 'Type of noise' and choose 'Random-Valued'	Option 'Random-Valued' should be chosen
11.	Train Models->Dataset	Click on combo box 'Dataset (train + test)' and choose 'dataset/data_set1.'	Option 'dataset/data_set1' should be chosen
12.	Train Models window ->Train Process window	Click on 'Train' button	Open 'Train process' window with active training timer and current PSNR present window.
13.	Train Models window	Click on 'Train' button (failure case: One or more combo box not chosen)	an alert in red will pop up with message 'one or more fields are not chosen'
14.	Train Models ->->Type of model (Super resolution)->Type of noise	Click on 'Train' button (failure case: hosen Model type: 'Super resolution' and chose also type of noise)	an alert in red will pop up with message 'please remove the Type of noise in model Super resolution'.
15.	Restore an image window->Upload image	Click on 'Browse for file'	Open the file explorer to select the files
16.	Restore an image window->Clear image	Click on button 'Start process'	Advance to step of Clear image processing with active percent of the clear
17.	Restore an image window->Finish process	Click on button 'download'	Download the clean image to your computer

8. Conclusions

Considering the training time limitations, both the Super Resolution and Noise2Noise models show significant potential for improving image resolution and reducing noise, even when trained solely on corrupted images.

In the Noise2Noise model, we observed varying performance across different noise types. In certain cases, for specific noise levels, the model struggled to reconstruct the original image effectively. This indicates that the model's ability to generalize to different noise levels could be enhanced with more diverse and intensive training scenarios. The Super Resolution model demonstrated good results for most images, occasionally producing images that appeared even more enhanced than the original. Despite this, it is constrained by image size, as it excels at improving images from 128x128 to 256x256 resolution, but further scaling might pose challenges.

In conclusion, the primary limitation lies in the available resources. Removing constraints on GPU usage, increasing training iterations, and adding more complex noise scenarios would likely lead to better model performance and more robust results.

9. User Documents

9.1 User Instruction

The Pixel Pro GUI is designed to offer a comprehensive interface for a sophisticated image processing model developed within this project. The Training section enables users to configure the model's parameters and select the appropriate dataset for training.

In addition, the GUI includes an Inference tab, allowing users to apply a pre-trained model for practical testing purposes. Within the restore image tab, users can upload an image and utilize the GUI's canvas functionality to enhancement his image.

Instructions

Clone the project '*PhaseBImplementation*' from [GitHub](#).

Git clone	Cd ../ Python-server pip install -r requirements.txt	Cd ../client Npm i
-----------	---	-----------------------

Figure 14. Command to download the web project and how to install the package.

Download for this link the **Models** and **follow** the instructions below:

Super Resolution Weights (super_resolution_weights.h5):

Location in Drive: Trained Models/Super Resolution/super_resolution_weights.h5

Steps:

1. Create a folder named "models" inside python-server if it doesn't already exist.
2. Move or copy the super_resolution_weights.h5 file into this newly created models folder.

Noise2Noise Weights and Models:

Location in Drive: Trained Models/Noise2Noise/* , all the folders inside (bernoulli, gaussian, poisson, text) should be downloaded and placed into the python-server/models/ folder.

Steps:

1. Navigate to python-server/models.
2. Download and move the folders (e.g., bernoulli, brownGaussian, etc.) into this "models" folder.

Once the weights and models are in the correct locations, your web project should be able to access them for both Super Resolution and Noise2Noise tasks.

Getting Started

Run the client with the command:

```
\PhaseBImplementation>cd client
\PhaseBImplementation\client>npm run start-client
```

Figure 15. Command to run the client.

Run the server in another terminal with the command:

```
PhaseBImplementation> cd .\python-server\
PhaseBImplementation\python-server> python .\app.py
```

Figure 16. Command to run the server.

To open the PixelPro home page enter to <http://localhost:3000/>

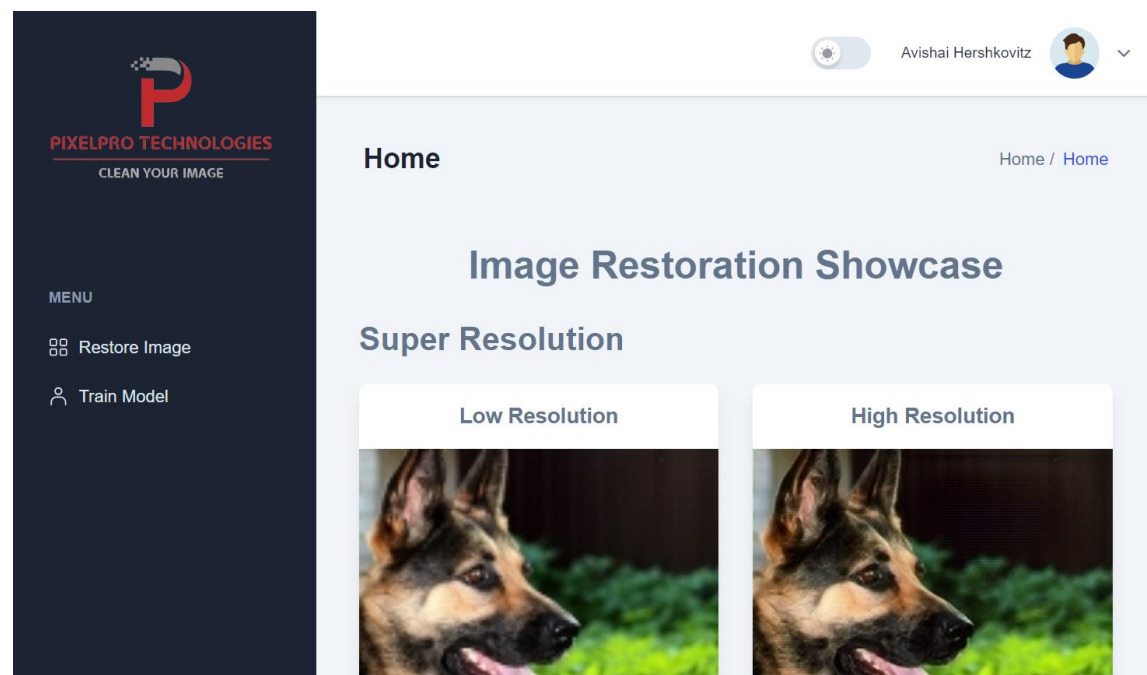


Figure 17. PixelPro- Home page.

On the Home Page, users can navigate through links to the Restore Image, Train Model, Results, and User Account sections. Additionally, there is an option to toggle Dark Mode for visual comfort.

Restore Image

The Restore Image Page uses pre-trained models to either enhance resolution with 'Super Resolution' or reduce noise with 'Noise2Noise.' Users can upload an image, select a noise type if using 'Noise2Noise,' and apply the model to enhance the image. Overfitting may occur if the model is too finely tuned to training data traits, affecting its effectiveness on new images.

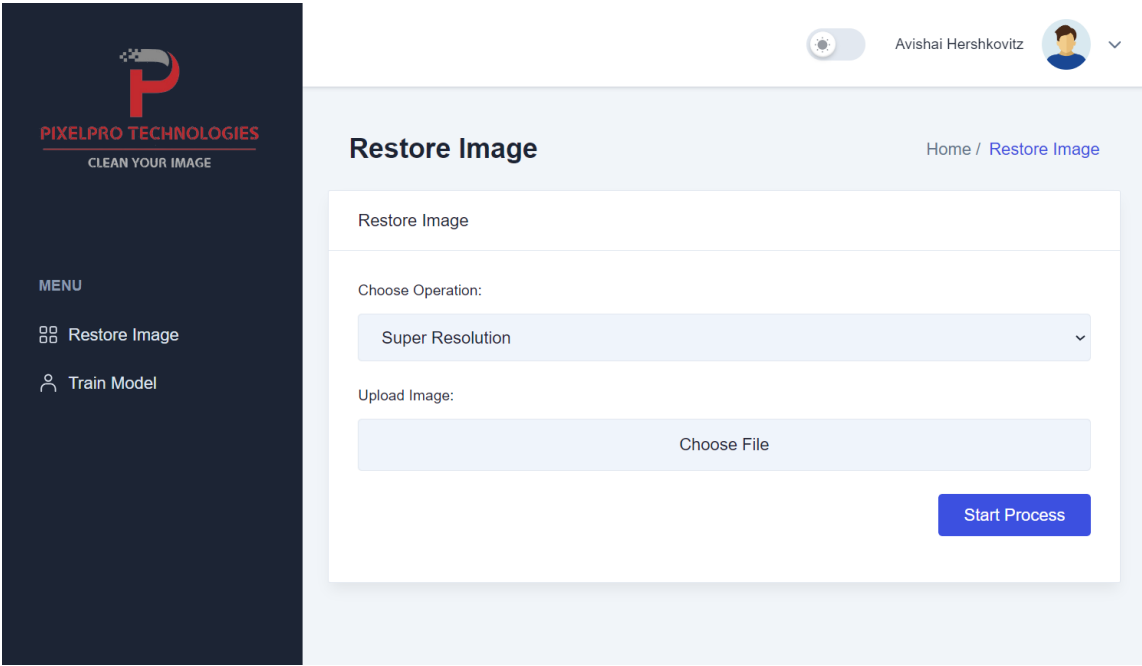


Figure 18. PixelPro- Restore Image page.

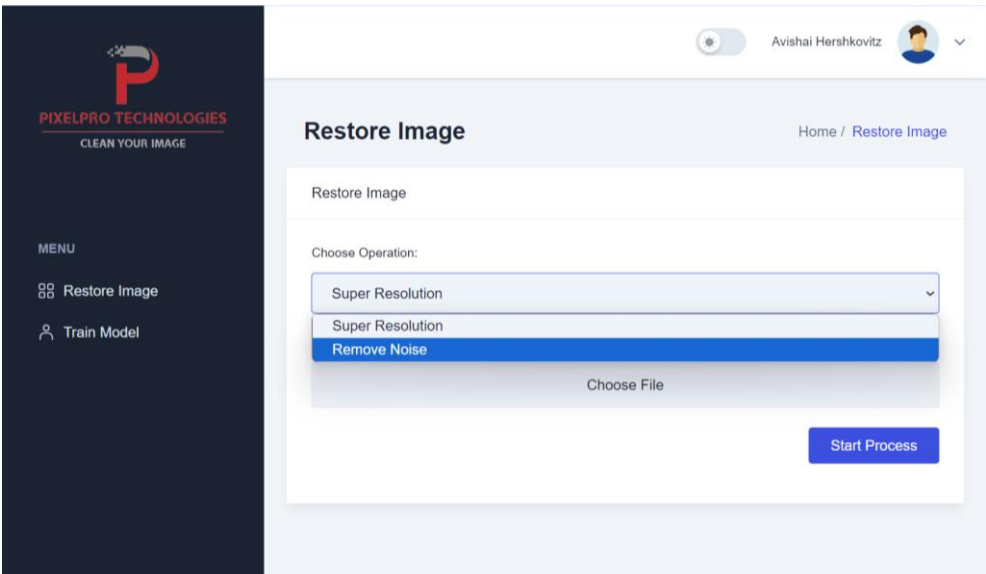


Figure 19. PixelPro- Restore Image page show restore image opetions.

Choose the restoration type you would like to apply from the dropdown menu and upload your image for processing.

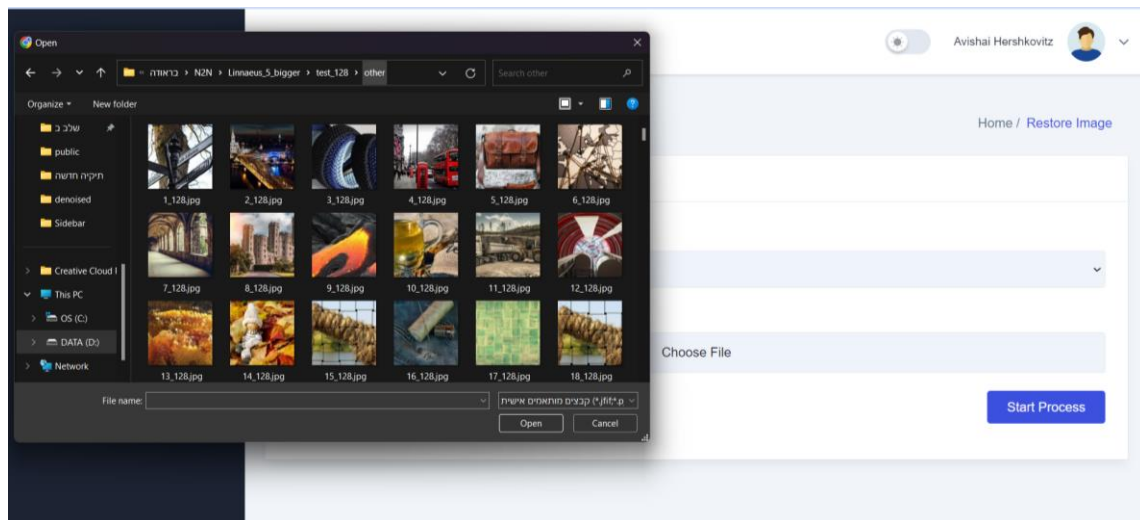


Figure 20. PixelPro- Restore Image page choose file to restore.

Navigate through your local directories to choose and upload an image file that you want to restore.

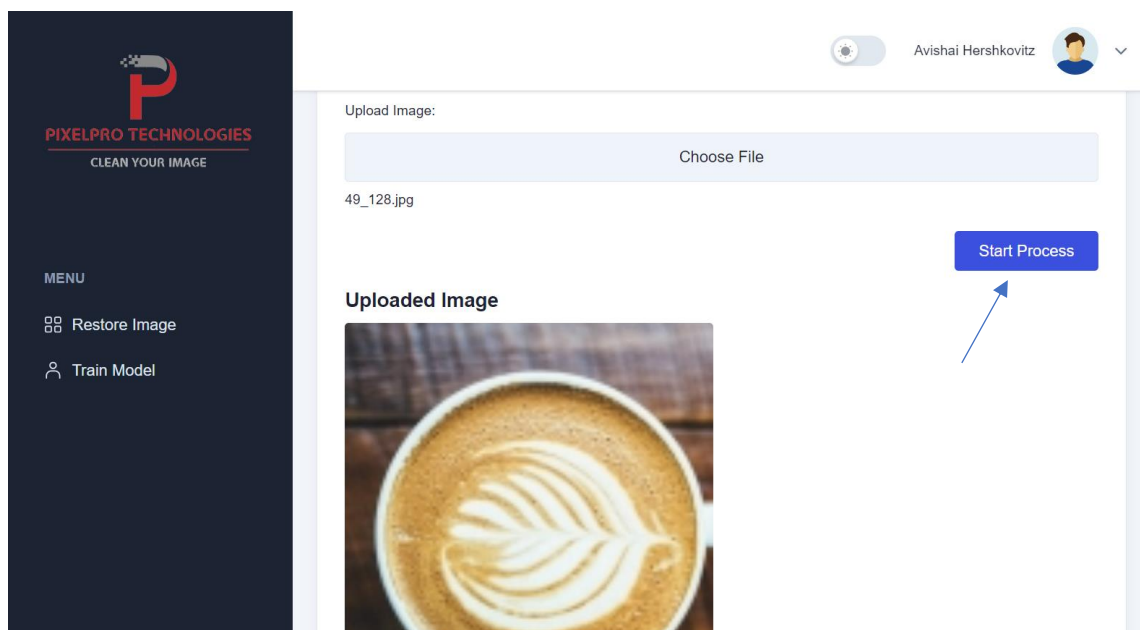


Figure 21. PixelPro- Restore Image page 'Start process' button to generate super-resolution image.

Your uploaded image is now displayed and ready for the selected restoration process. Click 'Start Process' to begin.

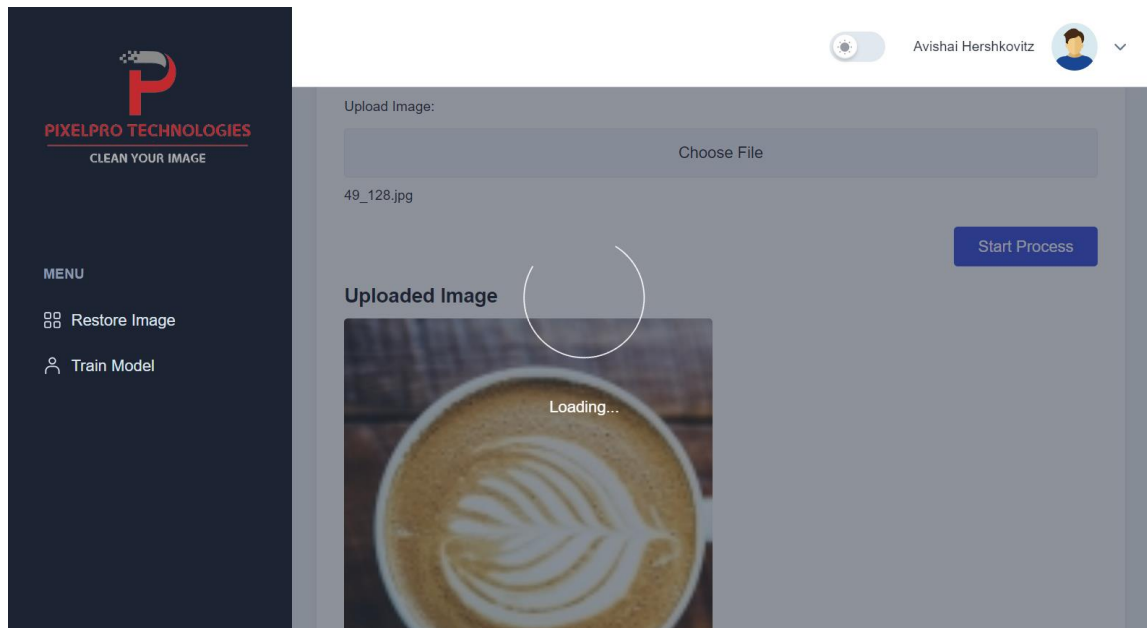


Figure 22. PixelPro- Restore Image page loading the super-resolution image.

Please wait while your image is being processed; a loading animation will display the ongoing operation.

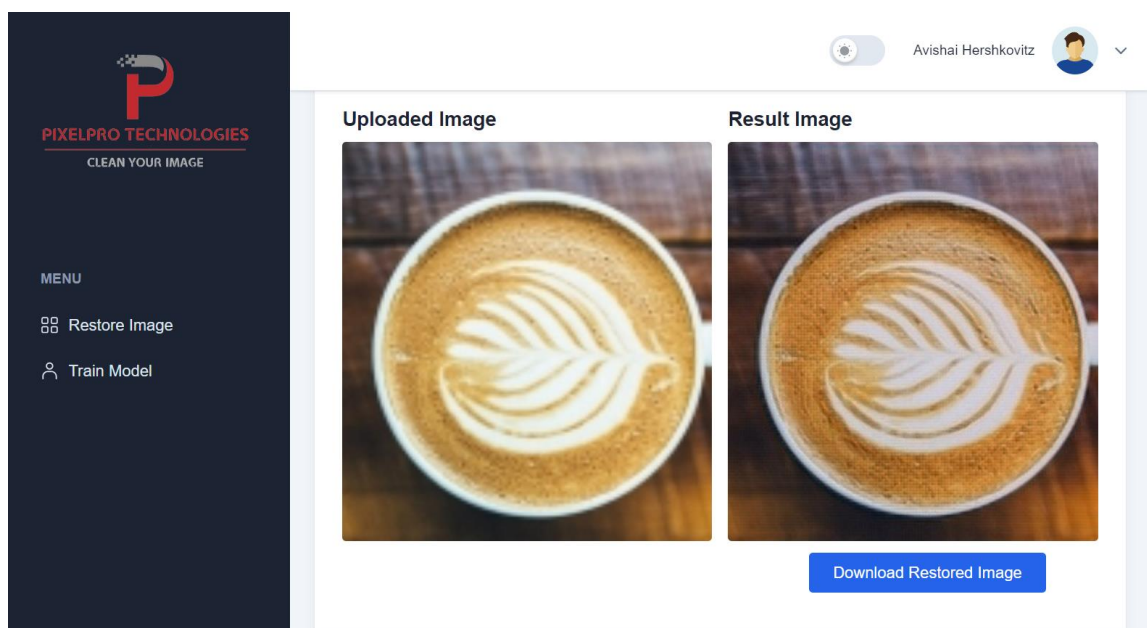


Figure 23. PixelPro- Restore Image page show the output, super-resolution image.

Your image restored and available to download. Click 'Download Restored Image' for download the image.

Training

The Train Model Page enables users to train customizable machine learning models on their datasets. Users can choose between 'Super Resolution' and 'Noise2Noise' models depending on the restoration need.

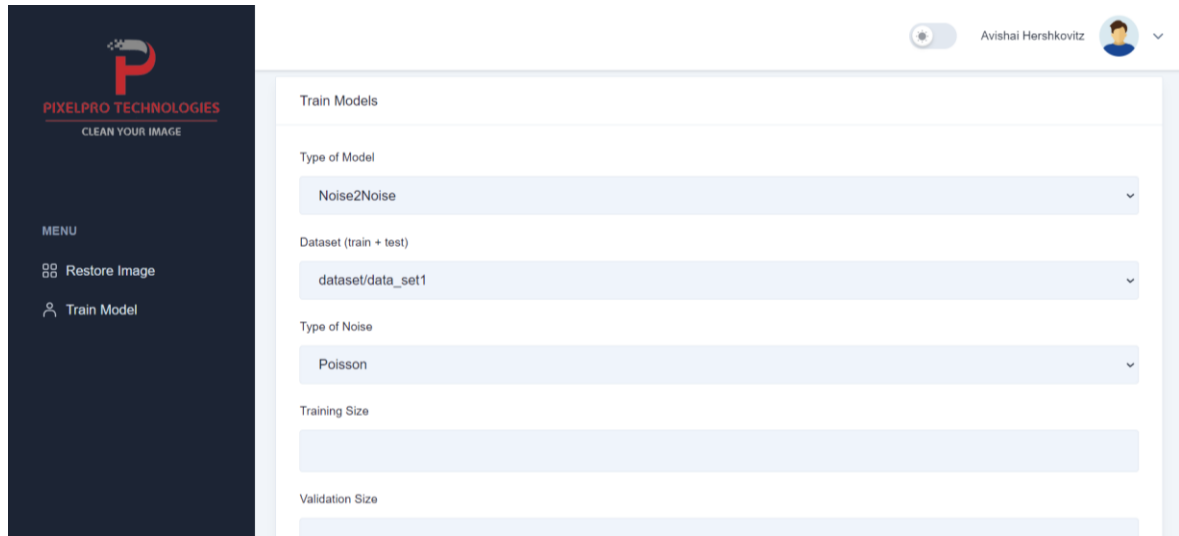


Figure 24. PixelPro- Train model Image page.

Parameter	Description
Type of model	Super Resolution / Noise2Noise
Dataset	Selection from datasets reflecting image characteristics for restoration.
Type of Noise	Type of noise; Gaussian, Poisson, Bernoulli, Text (available only for N2N)
Training Size	Number of images determining the learning scope to prevent overfitting.
Validation Size	Image count for performance validation and model tuning.
Epochs	Number of training cycles to expose the model to data patterns.
Loss Function	Determines optimization towards essential features, e.g., L1, L2, or L-Hdr.
Noise Parameter	Adjusts model's response to specific noise types in training data.
Crop Size	Image cropping dimensions influencing input size and model effectiveness.

The training configuration, the following parameters are specified to optimize the model's performance and adaptability, as outlined in table above.



Figure 24. PixelPro- Train model Image page- 'Train' button to save the configuration in the server.

After pressing 'Train,' the model undergoes a validation check to ensure that all parameters are entered correctly. If validated successfully, the training request is forwarded to the system administrator to initiate the model training process.

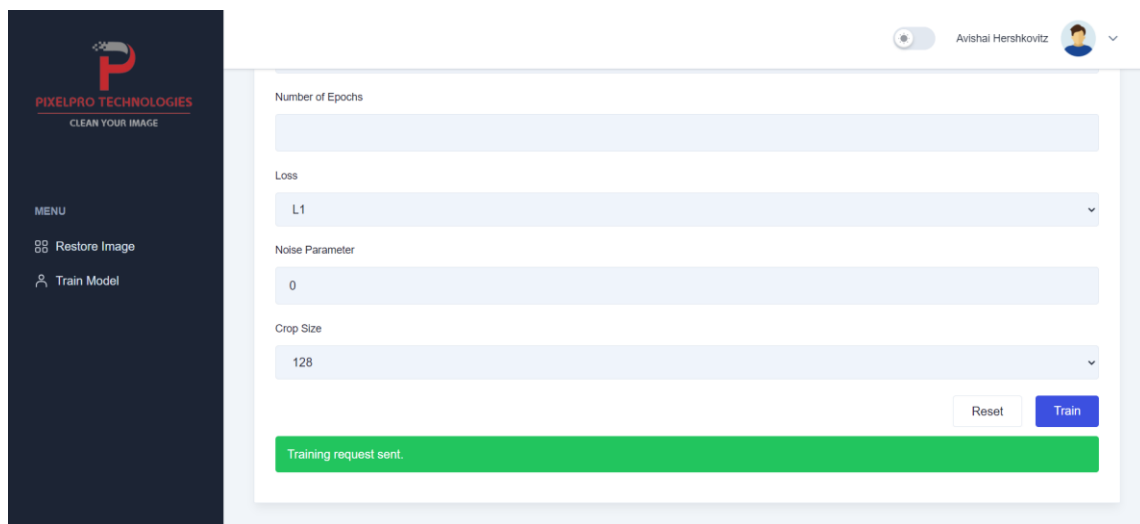


Figure 25. PixelPro- Train model Image page- success message after saving configuration to train new model.

9.2 Maintenance Guide

Use-Case diagram

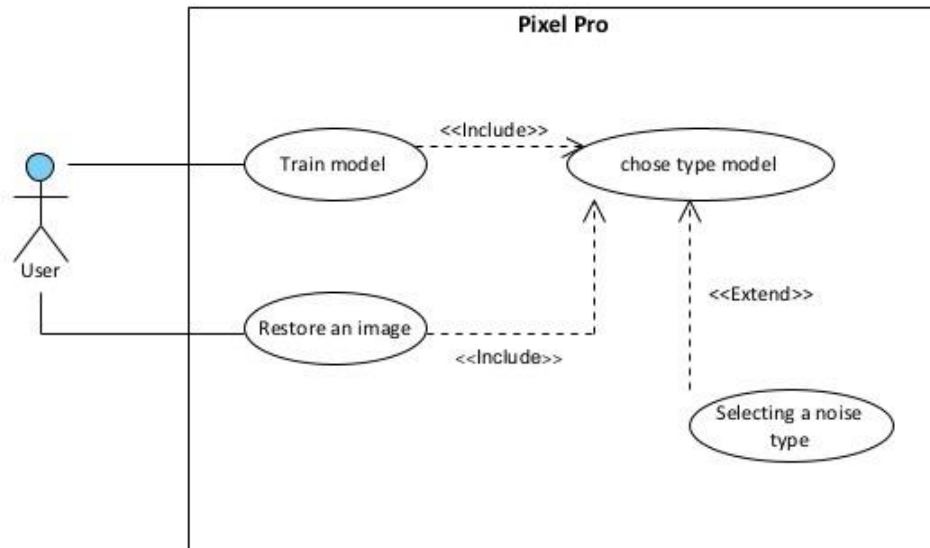


Figure 26. Use-Case diagram.

Package diagram

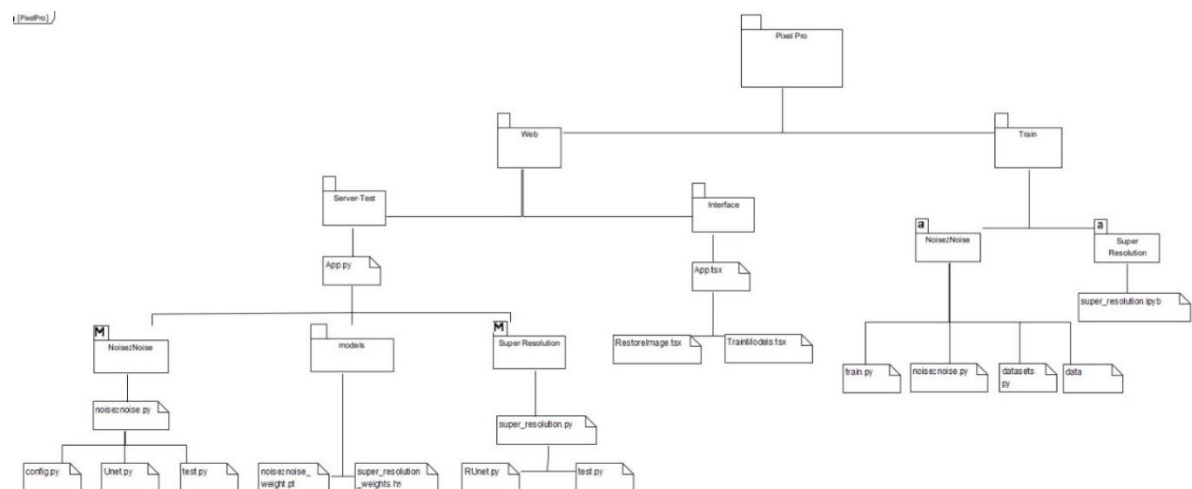


Figure 27. Package diagram.

Instructions

Clone the project 'Image-Enhancement-with-Super-Resolution-and-Controlled-Noise-24-1-R-12' from [GitHub](#).

Explanation for each of our packages: Web application, Super Resolution and N2N.

Web application

Following the instruction from [User Instruction](#).

Super Resolution

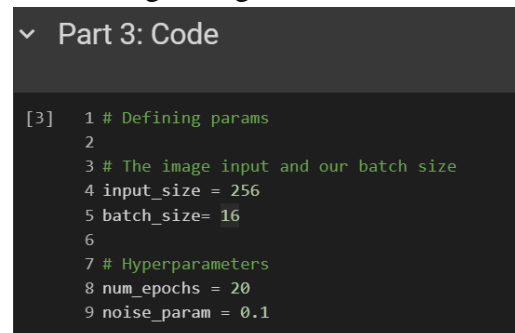
The model was written in Google Colab platform to take advantage of the available GPU units and speed up the computation. We implemented the network in python using tensor-flow v2.1 and Keras API.

Instructions

From folder 'Super-Resolution', download the file '*Super_Resolution.ipynb*' and upload to your 'Colab'.

The model was trained with Linnaeus 5 dataset. To train the model on the dataset please download the data from GitHub and saved in your Google Drive folder.

Set training configuration in the cell 'Part 3: Code'.



```
[3] 1 # Defining params
    2
    3 # The image input and our batch size
    4 input_size = 256
    5 batch_size= 16
    6
    7 # Hyperparameters
    8 num_epochs = 20
    9 noise_param = 0.1
```

Figure 28. Cell in google Colab to define parameters for training super-resolution model.

After this setting you can run the code and start training the model.

At the end you can evaluate the results in 'Part 4 - Experimental evaluation'. Additionally, the model weights will save in your Drive, you can use it on our images.

Noise2Noise

The model is PyTorch implementation using python 3.9.

Instructions

Clone the project 'N2N' from [GitHub](#).

Dependencies

- [PyTorch](#) (0.4.1)
- [Torchvision](#) (0.2.0)
- [NumPy](#) (1.14.2)
- [Matplotlib](#) (2.2.3)
- [Pillow](#) (5.2.0)
- [OpenEXR](#) (1.3.0)

To install the latest version of all packages, run:

```
pip3 install --user -r requirements.txt
```

Figure 29. Command to install requirements packages in python.

The model was trained with COCO 2017 dataset. You can download the full dataset by running 'LoadDataSet.bat' or use another dataset. Add your dataset to 'data' folder when the train images insert to 'train' folder and validation images to 'valid' folder. \triangle The dataset must contain only images and placed with an English path only! (The model can't recognize any other languages in the path) of several random strings in random places on the image.

Training

See `python3 train.py --h` for list of optional arguments, or

`examples/train.bat`

for an example.

By default, the model train with noisy targets. To train with clean targets, use `--clean-targets`. To train and validate on smaller datasets, use the `--train-size` and `--valid-size` options. By default CUDA is not enabled: use the `--cuda` option if you have a GPU that supports it.

The user can generate 'train.bat' files with all the arguments, so you can simply add the file into 'N2N\src' folder and start to train the model. At the end model checkpoints are automatically saved after every epoch in 'N2N\src\ckpts\noise_type'.

10. References

[1]. Hu, X., Naiel, M. A., Wong, A., Lamm, M., & Fieguth, P. (2019). RUNet: A robust UNet architecture for image super-resolution. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 0-0).

[2]. Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., & Aila, T. (2018). Noise2Noise: Learning image restoration without clean data. arXiv preprint arXiv:1803.04189.

[3]. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18 (pp. 234-241). Springer International Publishing.

[4]. <https://colab.research.google.com/>