Project Report

Yalchin Aliyev (K12228282),
Christian Feichtinger (K51902301),
Nathanael Harmetzky (K11916566),
Florian Haslauer (K12008697),
Johannes Kröpfl (K11707630),
Ellena Pfleger (K12010235)

January 29, 2024

1 Abstract

Given a focal stack, we input one image at a focal height of 0 m to our own variant of the SwinSR model, incorporating a custom position-enhanced loss to refine accuracy and enhance detail around the person. The introduction of a self-ensemble technique further elevated performance. Since passing the models output back to itself to further denoise the image yielded even better results, we extended the training of the model to optimized for a multi-pass through the model. This approach culminated in an optimized SwinSR model that combines self-ensemble and multi-pass strategies for superior denoising efficacy as can be seen in Section 5.2. Other contributions include measures to combat the overall model complexity and facilitate hardware acceleration.

2 Dataset Description

With approximately 33,000 simulations, totaling around 48 GB, the dataset provides a rich source of data for training and testing. Each simulation includes 11 images with their respective parameters and a ground truth image.

2.1 Position Dataset

With the idea of using focal stacks in mind we decided to generate the integrals at different heights for each complete sample and stored them bundled up in one `.tiff` image file per sample. Furthermore, we extracted the position of the person in the image from the parameters file and stored the x and y position in the name of the files, or indicated that there was no person. This way, to load an entire focal stack and the persons position, we only had to load one file,

reducing IO interactions. Ground truths were simply copied as separate files to prevent accidental data leakage.

We converted both batch 1 and 2 of the provided files to this format. We split batch 1 into: 10 % test data, 10 % validation data and 80 % training data. The entirety of batch 2 was also training data. This resulted in the validation and test sets being 1,064 samples each and 19,306 samples for training.

To load the data, we implemented a PyTorch Dataset. This loads the desired focal lengths from the `.tiff` files, allowing us to experiment with different combinations without having to re-compute the integrals every time. Additionally, it extracts the person's position from the name and converts it to pixel coordinates. Experimentally, we found the function $f(x) = \lfloor 256\left(1 - \frac{x}{16}\right)\rfloor$ to be a good fit for this conversion. Finally, it also loads the ground truth for the sample. All images are normalized to the range $[0, 1]$. In conclusion, the dataset returns the focal stack with shape $(\#\mathrm{FL} \times 512 \times 512)$, where #FL is the number of focal lengths, the ground truth with shape $(1 \times 512 \times 512)$, and the person's x and y position. If there is no person present, then the middle of the image is used instead.

We also use random augmentation for the training dataset. Specifically, we use random rotations in 90° intervals and random horizontal and vertical flips. This augmentation of course equally affects the integrals, the ground truth and the person positions. We decided against using, *e.g.*, random noise to avoid muddying our data, as we are not too familiar with the AOS integrator and how disturbances in the original camera images would/should affect the resulting integrals.

Although much of the following architecture can most likely handle focal stack data, after our subpar initial results we decided to "go back to basics" and focus on the single integral case at $\mathbf{0\,m}$. However, as we saw extremely promising results with this setup and had already invested quite a lot of compute into a model, we decided to stick to it.

3 Algorithm Description

3.1 Initial Approach

During literature research, we encountered two promising state-of-the-art techniques, IFCNN [1] and SwinSR [2], to tackle image fusion and reconstruction, respectively. Originally, we had an end-to-end pipeline in mind that aimed at combining both methods, *i.e.*, an architecture composed of separate fusion and denoising steps.

Image fusion is achieved by means of IFCNN, accepting any number of integral images as input. However, an early test run revealed that this approach does not yield desirable results. We found both the training and validation loss to stagnate after a low number of updates. The suspected cause is insufficient weight updates owing to the overall model depth and complexity. Unfortunately, we had no independent measure to verify the success of the fusion step.

In fact, this is crucial for the subsequent SwinSR module to make sense of its input. Retrospectively, skip connections from fusion to denoising or a loss function individually taking fusion into account might be able to guarantee nonzero gradients at lower layers. This discussion is subject for further research.

Faced with time and computational constraints, we decided to allocate our limited resources to SwinSR only, considering that the main objective is indeed image denoising. To preserve our initial architectural layout, we used a single focal length at ground level as a substitute for fused images. Another idea was to take the mean, minimum, or maximum along the focal dimension of the stack. This pseudo-fusion did not lead to improved results, since 0 m appears to contain enough information to perform an acceptable reconstruction. The detailed approach is outlined throughout the following sections.

3.2 SwinSR Model

As mentioned above, the SwinSR model is a crucial component of our approach, focusing on compressed image super-resolution and restoration. SwinSR is a novel successor to SwinIR [3] and based on the SwinV2 Transformer. Although SwinSR is specially designed to realize super-resolution, we only take advantage of its improved image-restoration capabilities.

3.2.1 Model Overview

Our proposal is illustrated in Figure 1. The adopted SwinSR model is structured as a sequence of modules, each performing a specific function in the process of image reconstruction. When not using an upsampler, *i.e.*, when doing pure denoising, those modules are:

- **Feature Extraction Layer (CONV-E)**: The model begins by embedding input images from ($\#\text{FL} \times 512 \times 512$) to the embedding dimension (32), which are then processed through the transformer layers. In particular, a single focal length was set for training. Given sufficient computing resources, it can be extended to any multi-focus domain.
- **Residual Swin Transformer Block (R-STB)**: This block integrates the window attention mechanism with MLPs, facilitating the learning of robust feature representations. Chained residual connections are geared towards a balanced gradient flow at scale. In addition, it includes layers for normalization and dropout, ensuring stability and efficiency in the learning process.
- **Multi-Layer Perceptron (MLP)**: Implements two fully connected layers with a GELU activation function in between. This module is used for processing feature vectors within the architecture.
- **Window Attention (W-MSA)**: Window-based multi-head self-attention module with relative position bias. Specifically, a window size of 8 and 4 attention heads were chosen.

- **Reconstruction Layer (CONV-R)**: A CNN layer to transform the embedding feature maps back to the input dimension ($\#\text{FL} \times 512 \times 512$). The assumption being that the desired output shape is the same as the input shape, *e.g.*, denoising an RGB image should return an RGB image. The output is added to the model's input as a residual connection.
- **Grayscale Layer (CONV-G)**: A further CNN layer was added to the end of the network to transform the output from ($\#\text{FL} \times 512 \times 512$) to a single ($1 \times 512 \times 512$) grayscale image which is the model's final output. Due to our later choice to focus on a single focal length, this layer was not strictly necessary.

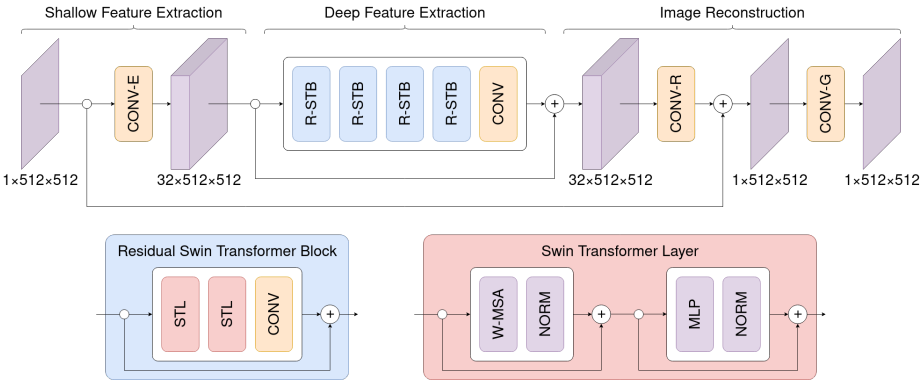


Figure 1: Schematic diagram of the proposed model architecture based on SwinSR. Notice that both the input image and the extracted feature maps are skip-connected to the subsequent reconstruction layers, a layout reminiscent of U-Net.

3.2.2 Checkpointing

In our experiments, SwinSR, no matter the hyperparameters, always required extremely large amounts of GPU memory to run. One reason is the quadratic space complexity the self-attention mechanism of SwinV2 Transformers exhibits. Reducing the model size and batch size helped somewhat, but it was still not trainable. Exactly for cases like this, PyTorch provides the checkpointing interface, which the SwinSR already implements and can be turned on with a Boolean flag. With checkpointing, regions of the model are re-calculated for the backward pass instead of storing their state the entire time. This reduces the GPU memory requirements, but also slows down training. We turned this on, as it was necessary to run the model on our machines.

4 Training Procedure Overview

4.1 Optimizer and Scheduler

An AdamW optimizer is employed for its balance of momentum and regularization. A learning rate scheduler, is used to adjust the learning rate based on validation loss.

Specifically, we start with a learning rate of 5×10^{-4} and multiply it by the factor 0.1 every time the model's validation loss plateaus for 5 consecutive evaluation steps.

4.2 Position Enhanced Loss

Subpar performance with our initial ideas drove us to re-consider using standard loss functions for this project. The problem we perceived is thus: common loss functions like MAE (L1), MSE (L2) and Charbonnier losses are calculated pixel by pixel and then averaged over the entire image. While this would likely lead to good average results, it might also mean that persons get ignored due to the drastic difference in the size of a person compared to the entire image.

Our solution is a custom loss function, Position Enhanced Loss, which increases to loss for a region around the person. It effectively calculated two losses, one for a patch around the person and one for the rest of the image. These two losses are then weighted and added together. Figure 2 illustrates this process.

The pixel-wise losses are still calculated using a PyTorch loss function, just with the reduction turned off. We decided on the MAE (L1) loss, due to promising results early on. For similar reasons, the size of the "person patch" was set to 96 and the weighting factor to 0.5. That is, given a denoised image $\hat{Y}$ and target image Y of size (512×512) each, the pixel-wise L1 loss can be written as

$$L_1(\hat{Y}, Y) = |\hat{Y} - Y|.$$

Now, let P_h and $P_w \subset \{1, 2, \ldots, 512\}$ be the sets of height and width indices of the cut-out person in the image, respectively. Note that $|P_h|$ and $|P_w| \leq 96$. If the window exceeds the image margin, the pixel-wise loss is assumed to be zero. We define

$$L_{\mathrm{P}}(\hat{Y}, Y) = \frac{1}{96^2} \sum_{i \in P_h} \sum_{j \in P_w} L_1(\hat{Y}, Y)_{ij}$$

and

$$L_{\mathrm{F}}(\hat{Y}, Y) = \frac{1}{512^2 - 96^2} \sum_{i \notin P_h} \sum_{j \notin P_w} L_1(\hat{Y}, Y)_{ij}$$

in terms of the reduced loss of the person patch and the frame surrounding it.

From this, we obtain the Position Enhanced Loss, *i.e.*,

$$L_{\mathrm{pos}}(\hat{Y}, Y) = 0.5 \times L_{\mathrm{P}}(\hat{Y}, Y) + (1 - 0.5) \times L_{\mathrm{F}}(\hat{Y}, Y).$$

Note that $L_1(\hat{Y}, Y)$ is again given by the weighting factor $96^2 \div 512^2$.

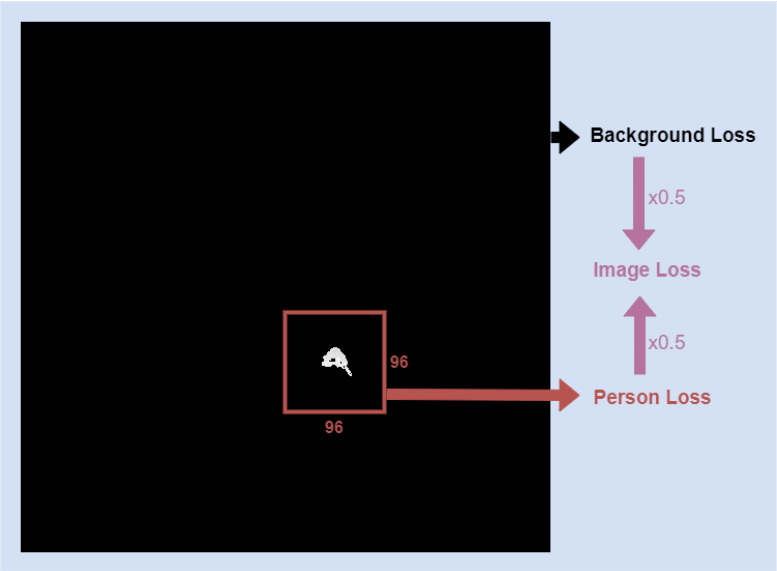


Figure 2: Illustrative schematic of how the Position Enhanced Loss is calculated.

As mentioned in Section 2.1, when no person is present in the image, the position is arbitrarily set to the middle of the image. It might have been better to not focus on any particular patch in such cases, but would have required further information sharing between the dataset and the loss function, which we did not find desirable.

4.3 Postprocessing

To enhance a trained model's performance even further, we employ Self-Ensemble and Multi-Pass. The approximate procedure at inference is shown in Figure 3.

4.3.1 Self-Ensemble

When calculating the model's output, a rotated input should always result in the same rotated output. However, this is usually not the case. To extract the best possible restoration from the model, we feed several versions of the input, rotated by a multiple of 90°, through the model, undo the rotations and aggregate the results [4]. One option for this aggregation is the mean, but we saw better results when using the median.

Interestingly, using random data augmentation during training as we do could be seen as training the model for such a use case.

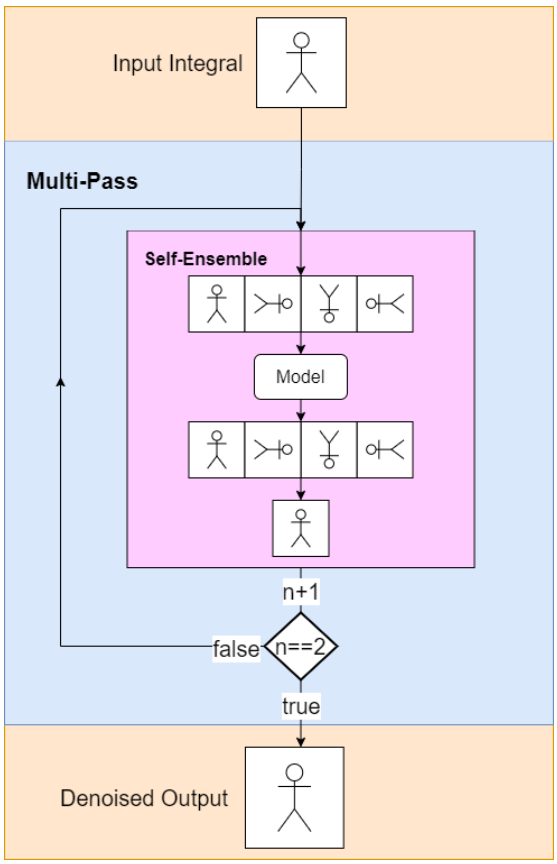


Figure 3: Approximate denoising procedure at inference time (Multi-Pass improvement check is missing).

4.3.2 Multi-Pass

While the model can get quite good at denoising inputs, it is rare that all noise is removed. However, as we were already training and using a denoising model, the idea of passing the output through the model a second time came up.

Initially, while the overall performance increased, we saw some cases where doing multiple passes made the output progressively worse, creating large patches of black or white. This was alleviated by implementing a check on whether the standard deviation of the double-denoised image was lower than that of the single-denoised image. If it is lower, then we accept the new output, otherwise we use the single-denoised image as our prediction.

As denoising an already denoised image is not something the model was ever trained on, we were intrigued by the idea of "finetuning" a model for this task. More on this in Section 4.4.6.

4.4 Training Loop

4.4.1 Batch Processing

The training loop involves processing batches of images, where each batch consists of a focal stack, ground truth, and position information. The model predicts denoised images from the focal stacks, and the loss is computed against the ground truth.

4.4.2 Gradient Accumulation

Due to the reasons discussed in Section 3.2.2, we had to reduce our batch size to 2. To still obtain good gradient estimations, we implemented gradient accumulation. This means, that instead of the standard of doing one update per batch, we accumulate the gradient over several batches (8) before doing a single update.

Gradient accumulation requires us to count the number of batches to know when we have to perform an update, so we decided to use the total number of used batches for logging and plotting, instead of epochs as usual.

4.4.3 Automatic Mixed Precision

As a further optimizations, we use PyTorch's Automatic Mixed Precision feature, which automatically converts data and weights between `float16` and `float32` as needed, to both speed up computation and lower the memory required while still keeping the precision acceptable.

4.4.4 Validation and Checkpointing

Periodically, the model undergoes validation where it is evaluated against the validation dataset. Key metrics such as loss, PSNR, and SSIM are calculated. Model checkpoints are saved, when a new best validation loss is achieved.

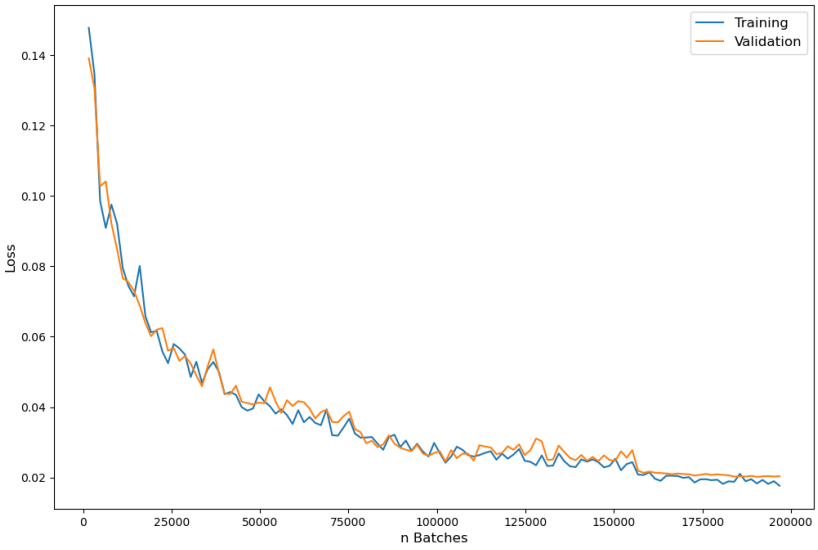


Figure 4: Loss of the single-pass SwinSR model.

Unfortunately, due to a bug in our implementations of PSNR and SSIM, the values we calculated during training cannot be used for plots to show the training progress. Therefore, only the training and validation loss curves will be reported here.

4.4.5 Base Model Training

In this step, the model is trained for simple denoising, though of course using the custom loss and other optimizations discussed in previous sections. The model was trained for around 196k batches of 2 samples each, resulting in around 25k weight updates. The training progress is visualized in Figure 4.

4.4.6 Multi-Pass Finetuning

Starting with the weights of the base model, we adapted our training procedure slightly to also include a loss term based on denoising an image twice and trained for approximately another 25k batches, *i.e.*, around 3.1k weight updates. The training progress can be seen in Figure 5.

In our experimentation, we observed that simply denoising the image a second time and adding the new loss to the existing loss does not give good results. Ultimately, we found a strategy of alternating between calculating a single-denoised loss and a double-denoised loss to be more effective. As our batch size is 2, this can be seen as single-denoising one sample and double-denoising the other sample.

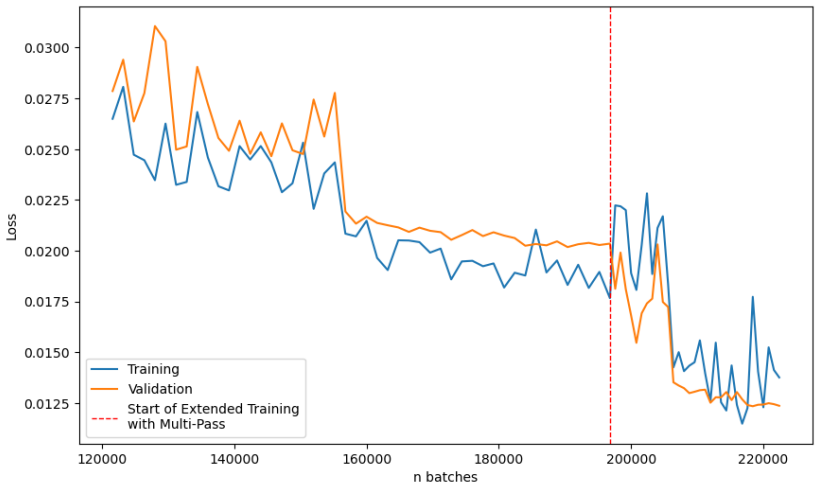


Figure 5: Loss of the finetuned multi-pass SwinSR model.

4.5 Hyperparameters

- **Batch Size**: Set to 2 for both training and validation.
- **Number of Updates**: 5,000 gradient updates are targeted.
- **Samples per Update**: 8, dictating the number of samples for a single gradient update.
- **Learning Rate**: Initialized for the AdamW optimizer and adjusted by the scheduler based on validation performance.

5 Discussion of Limitations and Results

5.1 Limitations

Due to limited time and computational resources, we could not properly experiment with focal stacks using this architecture and were likely hurt by the loss of information. However, training a model with focal stack inputs should be possible, although some thought would have to be invested in how to use and train Multi-Pass in that case.

A larger limitation of this architecture is the high GPU memory consumption and the lack of speed, both during training and inference. We experimented with several ideas, such as reducing the input size and doing super-resolution on the output, but while some helped to alleviate the aforementioned problems, none seemed to learn as well as the architecture we ultimately stuck with.

5.2 Results

Using our final model, *i.e.*, the model obtained by finetuning the base model, we achieved the results listed in table 1. As can be seen, our postprocessing methods have a large impact on the results.

	MAE (L1)	MSE (L2)	PSNR	SSIM
Simple Denoised	0.0176	0.0020	29.257	0.899
+ Self-Ensemble	0.0168	0.0018	30.255	0.909
+ Multi-Pass	**0.0084**	**0.0006**	**35.327**	**0.952**

Table 1: Results of the final model on the unseen test dataset. All values calculated for $[0, 1]$ normalized images. "Simple Denoised" refers to using the model without any postprocessing. Each postprocessing method is added to the previous line, so the row "+ Multi-Pass" shows the results for our final model with both postprocessing methods applied.

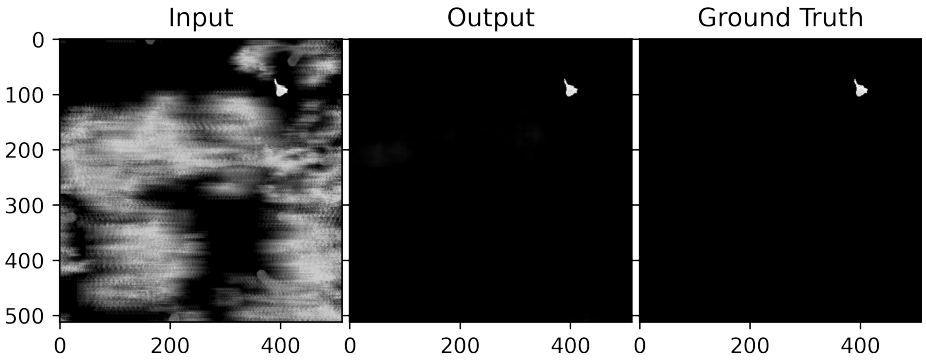


Figure 6: First test set example (batch ID 0, index 1425).

Figures 6 and 7 show some example results. Further examples can be found in Appendix A. Visually, it is remarkable how detailed some of the denoised persons are, with little to no blurriness around the edges. While it is quite common to see artefacts in the output, they are usually localized and not very severe.

Overall, this was a challenging project with many ups and downs, but we have all learned a lot and are proud of our results.

References

1. Zhang, Y., Liu, Y., Sun, P., Yan, H., Zhao, X. & Zhang, L. *IFCNN: A General Image Fusion Framework Based on Convolutional Neural Network* 2020. `https://doi.org/10.1016/j.inffus.2019.07.011`.

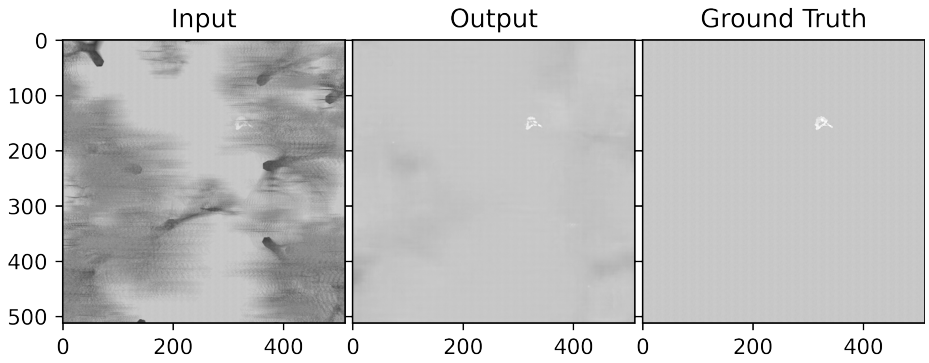


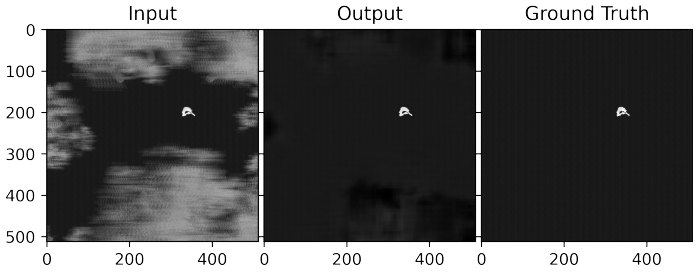
Figure 7: Second test set example (batch ID 0, index 3127).

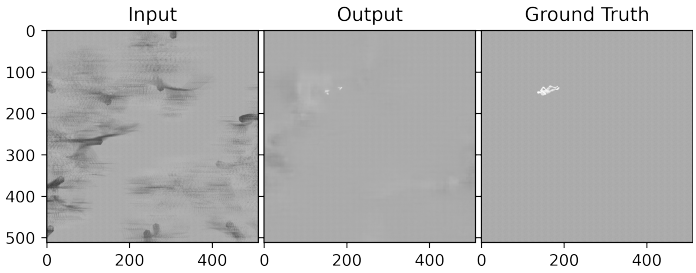
2. Conde, M. V., Choi, U.-J., Burchi, M. & Timofte, R. *Swin2SR: SwinV2 Transformer for Compressed Image Super-Resolution and Restoration* 2022. arXiv: 2209.11345 [cs.CV].

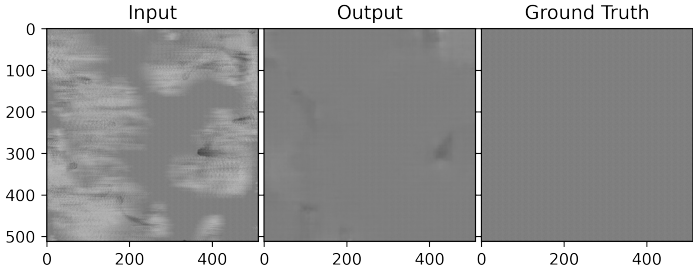
3. Liang, J., Cao, J., Sun, G., Zhang, K., Gool, L. V. & Timofte, R. *SwinIR: Image Restoration Using Swin Transformer* 2021. arXiv: 2108.10257 [eess.IV].

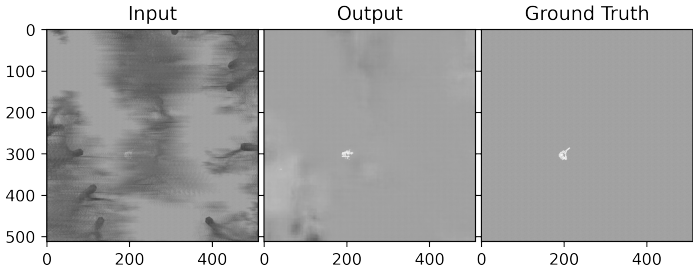
4. Timofte, R., Rothe, R. & Gool, L. V. *Seven Ways to Improve Example-Based Single Image Super-Resolution* 2015. arXiv: 1511.02228 [cs.CV].

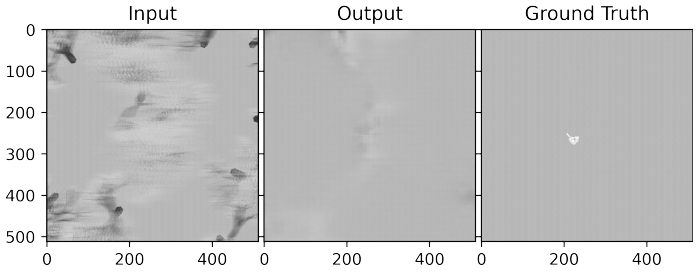
A Further Examples

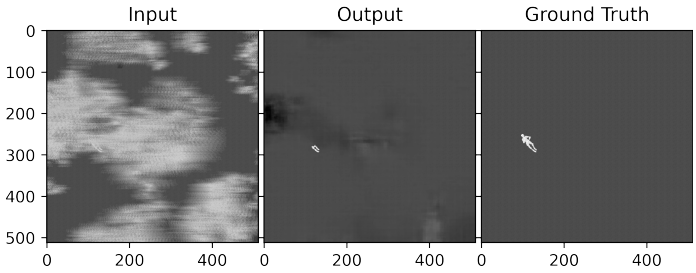


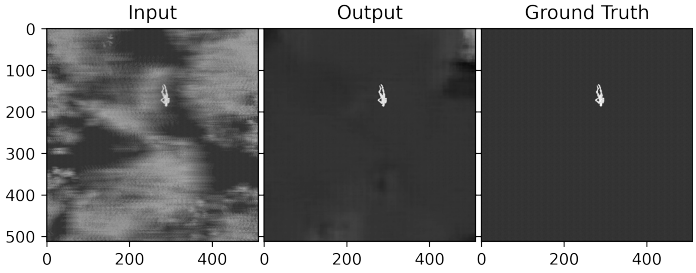

Input
Output
Ground Truth
0
100
200
300
400
500
0
200
400

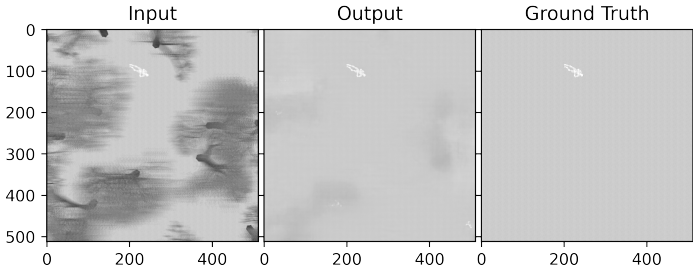

Input
Output
Ground Truth
0
100
200
300
400
500
0
200
400

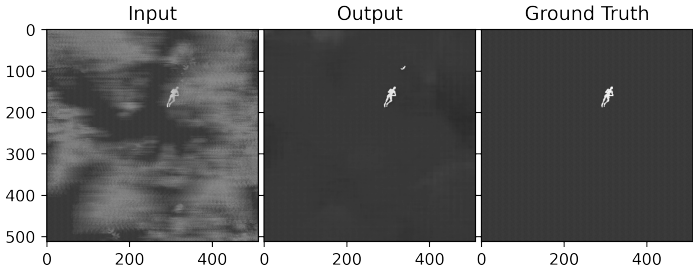

Input
Output
Ground Truth
0
100
200
300
400
500
0
200
400


Input
Output
Ground Truth
0
100
200
300
400
500
0
200
400


Input
Output
Ground Truth
0
100
200
300
400
500
0
200
400


Input
Output
Ground Truth
0
100
200
300
400
500
0
200
400


Input
Output
Ground Truth
0
100
200
300
400
500
0
200
400


Input
Output
Ground Truth
0
100
200
300
400
500
0
200
400