Image Restoration Using SwinIR (Machine Learning 2022 Course)

Hai Le [1] **Ivan Gerasimov** [1] **Prateek Rajput** [1] **Rustam Guseynzade** [1] **Vladimir Chernyy** [1]

Abstract

This report provides the description of our image restoration project. The goal is to restore high-quality images from low-quality images. We utilized SwinIR (Liang et al., 2021) model based on the Swin Transformer, which consisted of three main parts:

- Shallow feature extraction
- Deep feature extraction
  - Composed of many residual Swin Transformer blocks (RSTB), each has several Swin Transformers layers together with a residual connection
- High-quality image reconstruction.

We examined the performance of SwinIR with its default blind noises against our own synthetic noise. Moreover, we implemented the ISTA/-FISTA algorithms with SwinIR as a de-noising model for non-blind de-blurring problem

Github repo: https://github.com/ctrlzet/imgrestore/
Video presentation: drive.com

1. Introduction

Image restoration, a process of recovering a corrupted image - usually blurred and noisy, has long been a well-recognized computer vision problem. This fundamental image processing problem is often a test-bed for several general inverse problems. Thus, many state-of-the-art image restoration methods have been heavily researched on. The primary workhorse of these methods surrounds variation of Convolutional Neural Networks (CNN). Despite the increase in performance, there are flaws that exist with this implementation. For instance, restoring different image regions by using the same convolutional kernels can lead to poor performance. Moreover, CNN is not effective in terms of long-range dependency.

Despite the visible flaws, research solely focused on variations of CNN. Few has attempted to solve the image restoration problem with Transformers. Transformer is a deep learning model that adopts the mechanism of self-attention - weighting the significance of each part of the input data. This alternative method shows promising performance of several high-level vision tasks. However, this method also has flaws such as border issues stemming from the nature of its batch divisions.

With the flaws of both Transformers and CNN in mind, Swin Transformer combines the advantages of both said techniques. Swin Transformer can process large images due to local attention mechanism. Moreover, this method performs well in long-range dependency.

In this project, we experimented with synthetic noises for both the training and validation data. The goal was to compare the performance of the SwinIR model with pre-defined blind noises against our synthetic noise data. For the training data, standard deviation σ of noise level ranges from 10 to 55. For the validation data, we used noise level of 15, 25, and 50. Performance of both settings were measured via the PSNR and SSIM metrics.

Additionally, we further implemented the ISTA and FISTA algorithms. The goal was to combine said algorithms with SwinIR to make a de-noise models for de-blurring problems.

The main contributions of this report are as follows:

- In **Section 2**, we provide a comprehensive overview of previous state-of-the-art methods for the image restoration problem.
- In **Section 3**, we discussed the extensive architecture details of the models and algorithm that were used. Moreover, we elaborated on the parameters that were used for these models and algorithms.
- In **Section 4**, we have two main experiments: (1) blind

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Hai Le <Hai.Le@skoltech.ru>, Ivan Gerasimov <Ivan.Gerasimov@skoltech.ru>, Rustam Guseynzade <Rustam.Guseynzade@skoltech.ru>, Vladimir Chernyy <Vladimir.Chernyy@skoltech.ru>, Prateek Rajput <Prateek.Rajput@skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

vs. non-blind noises and (2) performance of ISTA and FISTA in de-blurring tasks. In both of these experiments, we provide extensive description of the obtained results and thus analyzed it.

2. Related Works

Image restoration has been a long-standing vision problem within the image processing field. With the many years of its existence, several revolutionary works have anchored Convolutional Neural Network as the main tool for image restoration. We will discuss previous researches that utilize variations of CNN.

One study proposed a deep learning method for single image super resolution (SR) (Dong et al., 2014). Super Resolution Convolutional Neural Network (SRCNN) employ a deep CNN mapping (end-to-end) between low and high-resolution images. Moreover, the method jointly optimizes all layers instead of handling each component separately. The advantages of this deep CNN includes a lightweight structure, state-of-the-art restoration quality with fast speed. However, the proposed SRCNN were tested on a small network; thus lacks in the upscaling department.

Another study proposed a discriminative model learning for image de-noising (Zhang et al., 2017). Feed-forward de-noising convolutional neural networks (DnCNN) utilizes residual learning and batch normalization to increase training efficiency and de-enoising performance. Additionally, with the introduction of residual learning, DnCNN removes latent clean image in the hidden layers. Moreover, this model can handle Gaussian de-noising with unknown noise level. This model boast in its high performance in several image de-noising tasks including Gaussian de-noising with unknown noise level, single image super-resolution (SISR) with multiple upscaling factors, and JPEG image deblocking with different quality factors.

Another approach that utilize other variation of CNN is compression artifacts introduced by Lossy compression (Dong et al., 2015). This four-layer convolutional network, AR-CNN, proved to be extremely effective in dealing with various compression artifacts. This network demonstrates efficiency that meets the speed requirement of a real-world application such as Twitter. However, since ARCNN is built upon SRCNN, the shallow CNN architecture is one of its disadvantage.

These pioneering work of SRCNN, DnCNN and ARCNN paved the way to a flurry of CNN-based models that utilize more intricate neural network architecture. One paper proposed a residual dense network for image restoration (Zhang et al., 2018). Residual dense network (RDN) make full use of the hierarchical features from the original low-quality images, thus directly equates to higher performance. Specifically, the utilization of residual dense block (RDB) allowed for direct connections from preceding RDB to each convolutional layer of current RDB. This results in a contiguous memory (CM) mechanism. This paper demonstrated the effectiveness of RDN through several image restoration tasks such as single image super-resolution, Gaussian image de-noising, image compression artifact reduction, and image de-blurring.

3. Algorithms and Models

Before diving into the details of the model and our implemented algorithms, we will discuss our preprocessing and data-handling stage.

3.1. Data Preparation

3.1.1. Dataset

For our training data, we used clean images from Berkeley Segmentation Dataset (BSD500)(Martin et al., 2001) [1]. This public dataset includes 12,000 hand-labeled segmentations of 1,000 Corel dataset images from 30 human subjects. Half of the segmentations were obtained from presenting the subject with a color image; the other half from presenting a grayscale image.

3.1.2. Data Processing

Like advised in the project description, we split the dataset into training and validation subset - 432 images for training and 68 images for validation.

Initially, we were applying normalization as well as augmentation. Our normalization and augmentation approach was the following:

```
random_transform =
[TT.Compose([
    TT.RandomCrop(320, padding=0),
    TT.Pad(2, padding_mode='reflect')
    ]),
TT.Compose([
    TT.RandomCrop(320, padding=0)
    ])
]

train_transform = TT.Compose([
    TT.RandomHorizontalFlip(p=0.5),
    TT.RandomRotation(degrees=20),
    TT.RandomChoice(random_transform),
    TT.RandomChoice(random_transform),
    TT.Resize((height, width)),
    TT.ToTensor(),
    TT.Normalize((0.5, 0.5, 0.5),
    (0.5, 0.5, 0.5))
])
```

[1] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

However, as we trained our model with the training data with this augmentation and normalization approach, we realized that any rotational transform introduced undefined behavior. For instance, there were odd pixels within the black background of the rotated input data. Therefore, we decided to simply use the `Resize` and `ToTensor` function.

3.2. Model Description

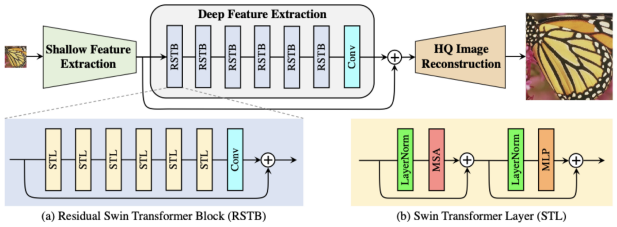

Figure 1. SwinIR Architecture for Image Restoration

As shown in the above figure, the SwinIR model consists of 3 main parts (1) Shallow Feature Extraction, (2) Deep Feature Extraction and (3) High-quality Image Reconstruction. We utilized the same extraction module but a different reconstruction module.

The model extracts data for shallow and deep features and uses these to reconstruct the final image with a convolution layer at the end and residuals connections after every RSTB block combined with another convolution layer before reconstruction.

3.2.1. Shallow Feature Extraction

For a low-quality (LQ) input $I_{\text{LQ}} \in \mathbb{R}^{H*W*C_{in}}$ where (H, W, C_{in} are the image height, width and input channel numbers, respectively), we use a 3×3 convolutional layer H_{SF} to extract shallow feature $F_0 \in \mathbb{R}^{H*W*C}$ as:

$$F_0 = H_{\text{SF}}(I_{\text{LQ}})$$

where C is the feature channel number.

3.2.2. Deep Feature Extraction

We then extract the deep feature $F_{\text{DF}} \in \mathbb{R}^{HxWxC}$ from F_0 as:

$$F_{\text{DF}} = H_{\text{DF}}(F_0)$$

where H_{DF} is the deep feature extraction module containing K residual Swim Transformer blocks (RSTB) and a 3*3 convolutional layer. The full block being described as:

$$F_i = H_{\text{RSTB}}(F_{i-1}), i = 1, 2, ..., K$$
$$F_{\text{DF}} = H_{\text{CONV}}(F_k)$$

where H_{RSTB} denotes the i^{th} RSTB and H_{CONV} is the last convolutional layer.

3.2.3. High-quality Image Reconstruction

We then reconstruct the image using these two feature extractions I_{RHQ} as:

$$I_{\text{RHQ}} = H_{\text{REC}}(F_0 + F_{DF})$$

where H_{REC} is the function of the reconstruction module. For simpler tasks like image de-noising as in replication part of our project, a single convolution layer is used for reconstruction. Apart from this residual learning is used to reconstruct the residual between the LQ and HQ images formulated as $I_{RHQ} = H_{\text{SwinIR}}(I_{LQ}) + I_{LQ}$ where H_{SwinIR} denotes the function of SwinIR.

3.2.4. Evaluation Metrics

We utilized 2 evaluation metrics: Pixel and Charbonnier loss.

(1) For image SR we use L_1 pixel loss

$$\mathbb{L} = \|I_{RHQ} - I_{HQ}\|$$

(2) For JPEG compression loss Charbonnier loss

$$\mathbb{L} = \sqrt{\|I_{RHQ} - I_{HQ}\|^2 + \epsilon^2}$$

where ϵ is set to 10^{-3}.

3.2.5. Gaussian Noises vs. Blind Noises

We used Gaussian noise and used the special projection layer for this specific type of noise to see if it affects our measurement indexes (SSIM and PSNR). Gaussian noise is added to the final labels as:

$$y = x + \sigma * n, n \in N(0, 1)$$

The original default SwinIR model utilizes blind noises and does not need any information about noise level σ. Thus, we added a Projection Layer to analyzed performance differences between blind vs. non-blind (Gaussian) noise.

3.2.6. Projection Layer

The final and novel component for the construction of the proposed architecture is the projection layer.

$$\epsilon = e^{\alpha}\sigma\sqrt{N_t - 1}$$

where σ is the standard deviation, N_t is the total number of pixels and α is a trainable parameter.

$$\nabla_{\boldsymbol{v}}\mathcal{L} = \varepsilon\gamma\left(\mathbf{I} - \beta^{+}\gamma^2(\boldsymbol{v} - \mathbf{y})(\boldsymbol{v} - \mathbf{y})^{\top}\right)\nabla_{\boldsymbol{q}}\mathcal{L}$$

where:

$$\boldsymbol{q} = \Pi_{\mathcal{C}}(\boldsymbol{v}), \beta^{+} = (1 + \operatorname{sign}(\|\boldsymbol{v} - \mathbf{y}\|_2 - \varepsilon))/2$$
$$\gamma = 1/\max(\|\boldsymbol{v} - \mathbf{y}\|_2, \varepsilon)$$

Additionally, the gradient of the parameter α w.r.t the loss function $\mathcal{L}$ is computed as:

$$\Pi_{\mathcal{C}}(\boldsymbol{v}) = \mathbf{y} + \varepsilon \frac{\boldsymbol{v} - \mathbf{y}}{\max(\|\boldsymbol{v} - \mathbf{y}\|_2, \varepsilon)}$$

3.2.7. Parameters

As previously mentioned, we implemented our own synthetic noise. For the training data, standard deviation σ of noise level ranges from 10 to 55. For the validation data, we used noise level of 15, 25, and 50.

Regarding our model's parameters, we followed the parameters mentioned in the original SwinIR paper (Liang et al., 2021). As mentioned in the 'Experimental Setup' section, the parameters used were:

- RSTB number: 6
- STL number: 6
- Window size: 8
- Channel number: 180
- Attention head number: 6

We followed these parameters without any changes. Beside the mentioned model parameter, we experimented with batch accumulations, learning rate, batch size. We were heavily limited by our available GPU. Despite that, we still took the divide and conquer methods. Our team members tried out different combination of the said parameters. After these trial and error periods, we came to the parameters that were optimal within our available GPU.

3.3. Algorithm Description

We consider the class of iterative shrinkage-thresholding algorithms (ISTA) for solving linear inverse problems arising in signal/image processing. Such methods are also known to converge quite slowly. In this paper we present a new fast iterative shrinkage-thresholding algorithm (FISTA) which preserves the computational simplicity of ISTA but with a global rate of convergence which is proven to be significantly better, both theoretically and practically.

Before diving into the details of ISTA and FISTA, we will discuss the dataset and its preparation for this algorithm part.

3.3.1. Dataset and Preparation

For this de-blurring tasks, we needed the ground truth images (clean images) and kernels. We obtained the kernels and ground truth images from the study "Edge-based Blur Kernel Estimation Using Patch Priors" (Sun et al., 2013)

3.3.2. ISTA

ISTA with **constant step-size**: - Input: $L := L(f)-$ A Lipschitz constant of ∇f.
Step 0: Take $\mathbf{x}_0 \in \mathbb{R}^n$.
Step k: $(k \geq 1)$ Compute: $\mathbf{x}_k = p_L(\mathbf{x}_{k-1})$

ISTA with **backtracking**:
Step 0: Take $L_0 > 0$, some $\eta > 1$, and $\mathbf{x}_0 \in \mathbb{R}^n$.
Step k: $(k \geq 1)$ Find the smallest non-negative integers i_k such that with $\bar{L} = \eta^{i_k} L_{k-1}$

$$F(p_{\bar{L}}(\mathbf{x}_{k-1})) \leq Q_{\bar{L}}(p_{\bar{L}}(\mathbf{x}_{k-1}), \mathbf{x}_{k-1}).$$

Set $L_k = \eta^{i_k} L_{k-1}$ and compute: $\mathbf{x}_k = p_{L_k}(\mathbf{x}_{k-1})$.

3.3.3. FISTA

FISTA **constant step-size** - Input: $L = L(f)$ - A Lipschitz constant of ∇f.
Step 0: Take $\mathbf{y}_1 = \mathbf{x}_0 \in \mathbb{R}^n, t_1 = 1$.
Step k: $(k \geq 1)$ Compute:

$$\mathbf{x}_k = p_{L_k}(\mathbf{y}_k)$$
$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2},$$
$$\mathbf{y}_{k+1} = \mathbf{x}_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\mathbf{x}_k - \mathbf{x}_{k-1})$$

FISTA with **backtracking**:
Step 0: Take $L_0 > 0$, some $\eta > 1$, and $\mathbf{x}_0 \in \mathbb{R}^n$. Set $\mathbf{y}_1 = \mathbf{x}_0, t_1 = 1$.
Step k: $(k \geq 1)$ Find the smallest non-negative integers i_k such that with $\bar{L} = \eta^{i_k} L_{k-1}$

$$F(p_{\bar{L}}(\mathbf{y}_k)) \leq Q_{\bar{L}}(p_{\bar{L}}(\mathbf{y}_k), \mathbf{y}_k)$$

Set $L_k = \eta^{i_k} L_{k-1}$ and compute:

$$\mathbf{x}_k = p_{L_k}(\mathbf{y}_k)$$
$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2},$$
$$\mathbf{y}_{k+1} = \mathbf{x}_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\mathbf{x}_k - \mathbf{x}_{k-1})$$

4. Experimental and Results

All of our experiments were carried via Google Collab platform. Validation output from training with $\sigma = 25$

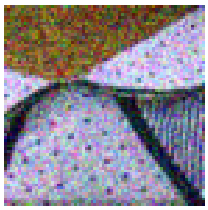

Figure 2. input

Figure 3. target

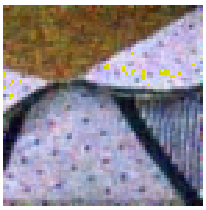

Figure 4. output

4.1. Replication Results and Analysis

Validation output from training with $\sigma = 50$

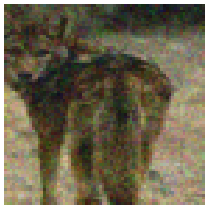

Figure 5. input

Figure 6. target

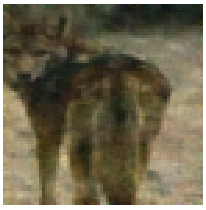

Figure 7. output

Prediction output from training with $\sigma = 15$

Figure 8. input

Figure 9. ouput

Prediction output from training with $\sigma = 50$

Figure 10. input

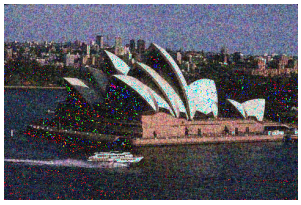

Figure 11. ouput

Analysis: The above figures showcased our model's output throughout the training as well as testing period. From a visual standpoint, despite some loss of pixels, our obtained results are overall relatively decent. The following section will comprehensively discuss our obtain results via different metrics.

4.2. Blind vs. Non-Blind Noises for SwinIR Model

Comparing Loss Functions

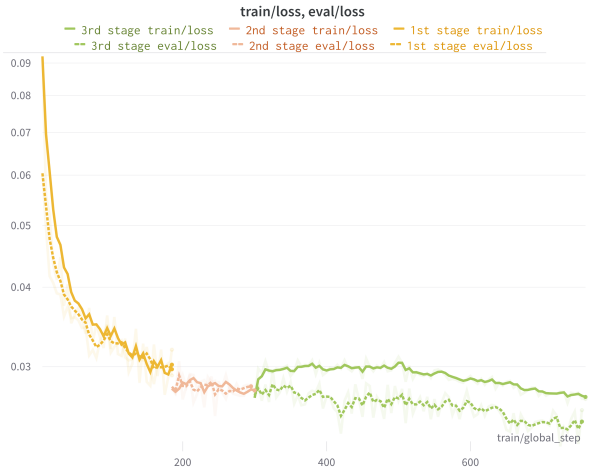

Figure 12. Loss for Train/Validation for Blind Noise

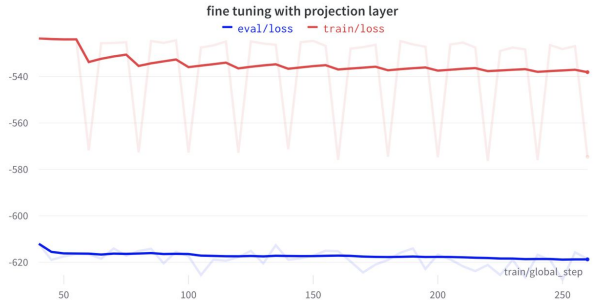

Figure 13. Loss for Train/Validation for Non-blind Noise

Analysis: The figures above are the loss data throughout the train and validation process for both blind and non-blind trained model. The loss's characteristics within the Figure 12 exhibits a common behavior for loss metrics while training/validating. The lost is decreasing fast at first and slowing down as the process proceeds. Moreover, Figure 12's behavior is also intuitive. The majority of improvement happened during the train/validation process. The addition of the projection layer only marginally improved the performance.

Comparing PSNR Functions

Analysis: In the above two figures, we plotted the metrics

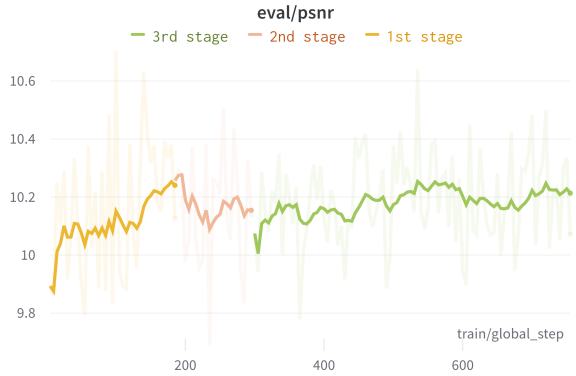

Figure 14. PSNR for Blind Noise

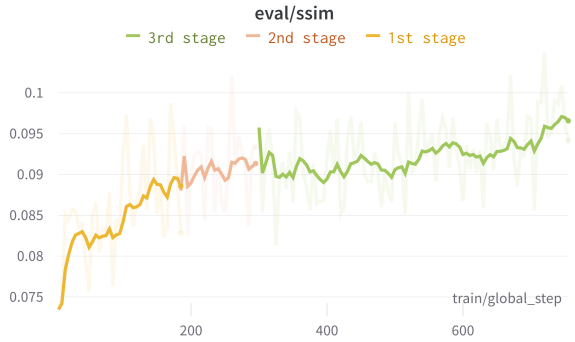

Figure 16. SSIM for Blind Noise

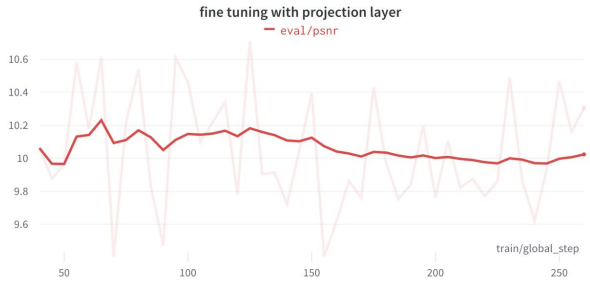

Figure 15. PSNR for Non-blind Noise

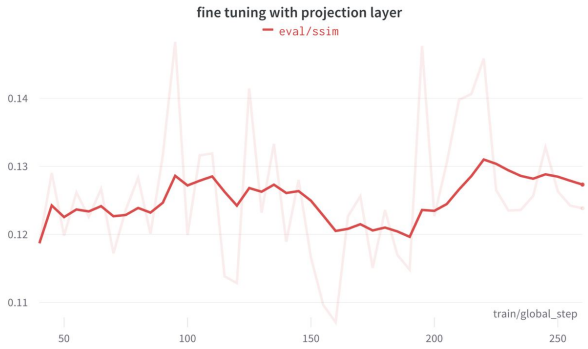

Figure 17. SSIM for Non-blind Noise

in every single step. Therefore, I will be comparing the final points of both blind noise and non-blind noise trained model against the average PSNR score within the original SwinIR paper (Liang et al., 2021). In the default blind-noise SwinIR, we obtained roughly 10.23 on this metric. On the other hand, with our own synthetic noise, we obtained 10.052. As we used "Negative PSNR" (specifically designed for projection layer), the decrease in PSNR score is valid. In other words, a lower score means that the projection layer slightly improved the performance of the model. Comparing with the average of the PSNR score of the original paper, they obtained roughly 32.72. We suspect that the transformation pipeline could be one of the vital flaw causing such issues. After we have finished our training procedure, we realized that the `Resize` function could cause major issues. We resized all of the input images uniformly the same size. Moreover, we suspected that the size that we chose to resize to was suitable for super resolution instead of de-noising. Despite realizing this fact, the lack in computational power prohibited us from training our model again, unfortunately.

Comparing SSIM Functions

Analysis: Similar to the PSNR metrics, we also plotted the metrics in every step. Thus, I will be comparing the end points against the average SSIM scores of the original paper. In the default blind-noise SwinIR, we obtained approximately 0.098 on this metric. On the other hand, we obtained roughly 0.128. With the introduction of our projection layer, our SSIM scores got higher. This does not make intuitive sense as our SSIM scores should've gotten lower with the addition of the projection layer. Moreover, compared to the original paper's score, they achieved roughly the score of 0.95. That equates to our model beating the paper's SSIM score by an order of a magnitude. However, as we have discussed in the previous section, our suspection of our flaws within our transform pipeline could be the catalyst.

4.3. ISTA and FISTA performance

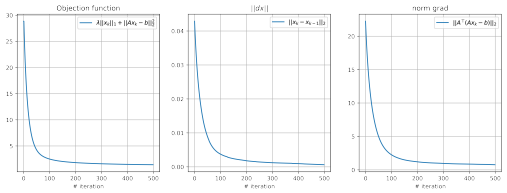

Figure 18. ISTA Performance

Analysis: The above figure is the inference time graph for the ISTA algorithm. All 3 approaches coverges at a simular rate and behaves quite similarly. Unfortunately, we were only able to implement ISTA successfully in combination with the de-noise SwinIR algorithm. We were able to sucesfully de-blur using this approach.

5. Conclusion & Future Work

In this project, we successfully implemented two major tasks. The first task revolves around replicating the results of the SwinIR model (Liang et al., 2021). This default SwinIR model utilizes blind noises. We implemented our own synthetic Gaussian noise. Within this replication tasks, we implemented a projection layer on top of the replicated implementation. With such addition, we compared the trained SwinIR on its default noise vs. our synthetic noise. The second task involves our own implementation of the ISTA and FISTA algorithms. These algorithms were used in combination with the de-noise SwinIR model to solve de-blurring tasks. However, due to the time constraint, we were only able to successfully implemented the ISTA algorithm. We then analyzed the performance of the two implemented algorithms.

There are several directions/improvements to this project in future research. For instance, we suspect that our `Resize` function may have degraded the performance/efficiency of the SwinIR model. We resized all input data to the same size. Furthermore, as we have mentioned, we suspect that the size of our images are more suitable for super resolution instead of denoising. We hypothesized that rescaled images transform smooth areas to sharp. Moreover, we picked around 400 epochs for our training process. The large amount of epoch may have lead to overfitting issues. Lastly, experimentation of the SwinIR model parameters could be more extensively looked at. We were limited in our computing resources, thus directly inhibits our parameters tuning process. Thus, this issue also prohibited us from retrain our model despite our suspection and hypothesis.

References

Dong, C., Loy, C. C., He, K., and Tang, X. Learning a deep convolutional network for image super-resolution. *Computer Vision – ECCV 2014*, pp. 184–199, 2014. doi: 10.1007/978-3-319-10593-2_13.

Dong, C., Deng, Y., Loy, C. C., and Tang, X. Compression artifacts reduction by a deep convolutional network. *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015. doi: 10.1109/iccv.2015.73.

Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., and Timofte, R. Swinir: Image restoration using swin transformer. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021. doi: 10.1109/iccvw54120.2021.00210.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pp. 416–423, July 2001.

Sun, L., Cho, S., Wang, J., and Hays, J. Edge-based blur kernel estimation using patch priors. In *Proc. IEEE International Conference on Computational Photography*, 2013.

Zhang, K., Zuo, W., Chen, Y., Meng, D., and Zhang, L. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017. doi: 10.1109/tip.2017.2662206.

Zhang, Y., Tian, Y., Kong, Y., Zhong, B., and Fu, Y. Residual dense network for image restoration. *CoRR*, abs/1812.10477, 2018. URL http://arxiv.org/abs/1812.10477.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Hai Le (20% of work)

- Experimented with model parameters for the SwinIR model
- Reviewing literate on the topic (4 papers), wrote the Abstract, Introduction, Relevant Works, Data Preparation, Parameters, and Conclusion.
- Data augmentation and normalization
- Coded the toy example for ISTA and FISTA

Ivan Gerasimov (20% of work)

- Coded the main SwinIR model.
- Experimented with model parameters for the SwinIR model.
- Trained the model.
- Worked on/coded the projection layer.
- Coded the main example for ISTA and FISTA

Prateek Rajput (20% of work)

- Experimented with model parameters for the SwinIR model.
- Coded the Gaussian noises.
- Wrote the description of the SwinIR architecture.
- Wrote the description for the ISTA and FISTA algorithms.
- Prepared the slides.

Rustam Guseynzade (20% of work)

- Experimented with model parameters for the SwinIR model.
- In charge of github repository and merge conflicts.
- Worked on/coded the projection layer.
- Wrote the description of the projection layer.

Vladimir Chernyy (20% of work)

- Experimented with model parameters for the SwinIR model.
- Prepared and load the data.
- Coded the main SwinIR model.
- Worked on/coded the projection layers
- Prepared the video presentation.

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: Roughly 60-70 percent of the code were written on our own

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   ☐ Yes.
   ☐ No.
   ☑ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

12. Clearly defined error bars are included in the report.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

13. A description of the computing infrastructure used is included in the report.

    - [x] Yes.
    - [ ] No.
    - [ ] Not applicable.

    Students' comment: None