Replicating Simple Baselines for Image Restoration

Sajad Rahmanian Ashkezari[1] and Seyed Alireza Hosseini[1]

[1]York University, CA

Abstract. The process of image restoration is obtaining high quality images from low quality ones. These processes are run on edge devices like smartphones. Thus, it is imperative that they have low computational cost. The SOTA methods propose deep learning based models to solve this problem. The goal of this paper is to propose a model that is much simpler than previous methods, but maintains and even improves their performance. To achieve this goal, they break the complexity of previous methods into two parts called inter-block complexity and intra-block complexity. To reduce the inter-block complexity they pick U-Net as their architecture. They then minimize the intra-block complexity by starting from a plain block and then adding more modules to it only if they are necessary. The paper tests its models for denoising and deblurring tasks and shows that they beat the SOTA. We make multiple attempts to reproduce their results for denoising. Specifically we test what they call NAFNet model with width 32 and successfully replicate their results. Throughout this process we find some discrepancies between the paper and their implementation which we show do not have noticeable effect. Our code can be found at our GitHub repository.

1 Introduction

1.1 Problem and Contributions

The images taken from a scene usually have some artifacts which makes them of low quality. Some of these artifacts are a result of imaging device, e.g., noise, and some are a result of the scene itself, e.g., blur. Even the best cameras cannot fully remove these artifacts without a post processing step. The process of obtaining a high quality image from a low quality image is called image restoration which includes tasks like deblurring and denoising. This task is important because highly important because first high quality images look more similar to the real scene and have better visual quality which improves user experience. Moreover, some images are used as input to other systems. For example, they might be used for image classification. Having images with higher quality might help the performance of those systems. However, these restoration models should not be too complex. This is because they are implement on devices with low computational budget and/or limited battery like smartphones. Thus, it is of great importance to make these models as simple as possible.

In recent years, many deep learning based methods have been proposed for image restoration tasks [see [2] and references therein]. The goal of [2] is to create a simple baseline model whose complexity is the same as previous methods but beats their performance. As mentioned, the goal of the paper is to propose a model with low complexity. To do so, it decomposes the model complexity into two parts, inter-block complexity and intra-block complexity. If we think of the model as a connection between different blocks, inter-block complexity corresponds to how complex the connection between these blocks is. SOTA methods have used different intra-block complexities shown in Figure 1. The paper selects a simple U-Net which aligns with its goal. The intra-block complexity is about the complexity of design structure inside each block. The novelty of the paper is in their design of the blocks such that not only they get high performance, but they are also simple. The paper starts from building a plain block which contains most common components (convolutional layers, ReLU, etc.) then adds more layers only if they improve the results.

The paper has two main contributions:

1. It extracts essential parts of previous SOTA methods by conduction multiple experiments to create a baseline that has low complexity yet outperforms SOTA methods.
2. It further simplifies the baseline by removing nonlinear activation functions and replacing them with other nonlinearities while also keeping and even improving the performance on multiple image restoration tasks.

1.2 Architecture

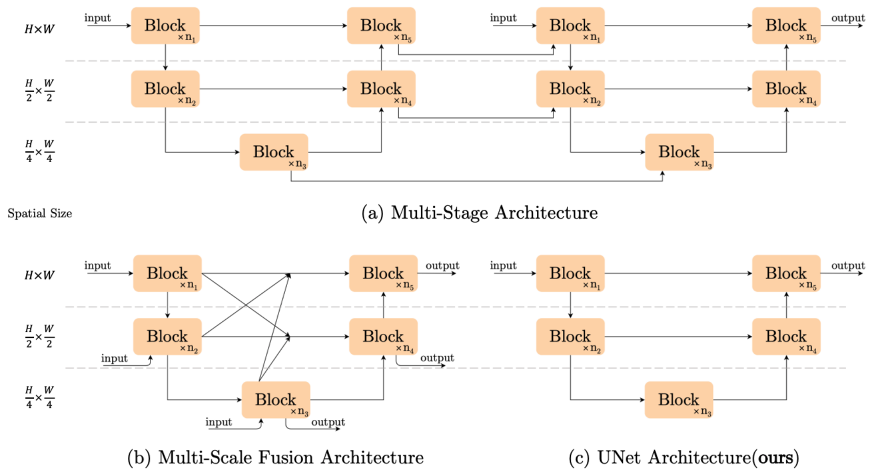
For inter-block complexity, as we said before, the paper uses a single-stage U-Net. This architecture is simple but does not prevent performance improvements. Thus, the paper focuses on intra-block complexity. The Upsample and Downsample layers are similar to previous papers. For downsampling, a convolutional layer with kernel size of 2 and stride of 2 is used. For upsampling, first the number of channels is doubled then pixel shuffling is used. Also, the skip-connections are just a simple point-wise addition unlike previous papers.

To create a simple baseline block, the paper follows a simple rule; starting from a simple block, add a layer to a block only if it is necessary. The necessity of each block is determined by empirical evaluations. For the plain block the paper just uses the most common layers which are convolutional layers and ReLU layer. They also use skip-connections. They do not use transformers because some papers claim transformers might not that much effective [4]. They also mention that depth-wise convolution is simpler that self-attention.

The next widely used layer, is the normalization layer. Following most of the previous SOTA methods and due to their stability the paper uses Layer Norm instead of Batch Norm and Instance Norm. The stability allows them to increase the learning rate which in turn brings more performance.

The activation function in plain block is the ReLU layer which is widely used in the literature. However, recent SOTA methods use GELU activation. The paper also tests GELU. The experiments show that using GELU improves the results on deblurring tasks while not changing the performance on denoising. Thus, they decided to use this activation layer instead of ReLU.

While the depth-wise convolutional layers capture local information, the paper follows some SOTA methods and uses channel-wise self-attention to mix global information. This layer is called the channel attention module, but it is essentially the squeeze and excite module that we saw in the course. Adding this module also improves the performance. These modules form the baseline block which is shown in Figure 2(c).



H×W
H/2×W/2
H/4×W/4
Spatial Size
input
output
Block ×n1
Block ×n2
Block ×n3
Block ×n4
Block ×n5
(a) Multi-Stage Architecture
(b) Multi-Scale Fusion Architecture
(c) UNet Architecture(ours)

Fig. 1: Models with different inter-block complexity. Figure is taken from [2].

Next, the paper tries to simplify the model even more. First, the authors replace the GELU layer by a much simpler layer called SimpleGate.GELU is defined as:

$$GELU(x) = x\Phi(x) \tag{1}$$

Where Φ is CDF of standard normal distribution, which is approximated by a complex function. Instead, SimpleGate is just a point-wise multiplication of the first half of feature map with the second half.

Finally, the authors simplify the Channel Attention layer by keeping only one convolution layer. The resuling module is called Simplified Channel Attention and is shown in Figure 3. These changes make the block acitvation free and also simpler. The resulting block is called NAFNet block, which is shown in Figure 2(d).

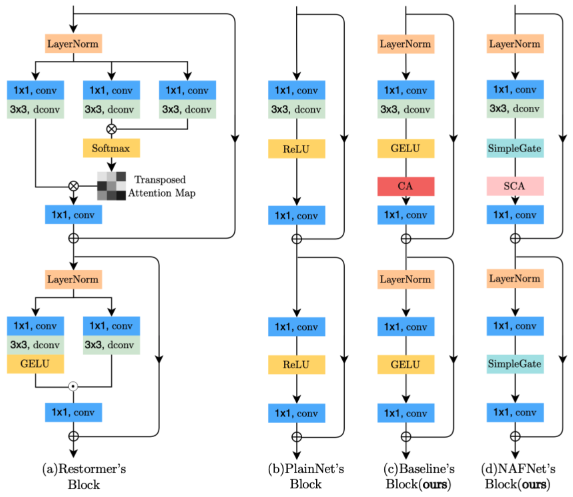


Fig. 2: Different choices for building blocks. Figure is taken from [2].

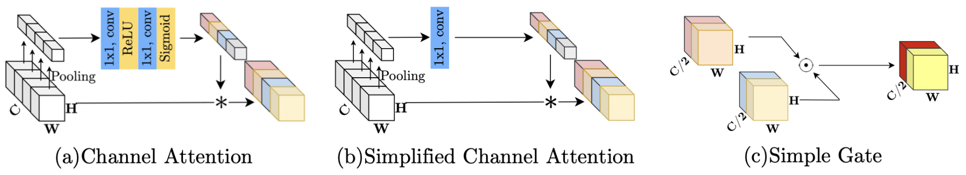


Fig. 3: Simplified modules. Figure is taken from [2].

2 Experiments

The paper includes multiple experiments for different restoration tasks. In our experiments, we focus on image denoising and trying to reproduce the last row of Table 2 in [2].

The dataset used for this experiment is SIDD dataset [1]. We use, with some modifications, the dataloader implemented in the official repository of the paper. As mentioned in the paper, they use random cropping for training with patch size of 256×256.

We attempt to reproduce their results for NAFNet and Baseline models with width 32 and with 36 blocks. But due to restrictions on our computational budget, we did the full training only for NAFNet. The U-Net architecture consists of four layers. We use 2, 2, 4, 8 blocks for encoder layers, 12 blocks for middle layer, and 2 blocks for each of the decoder layers. The number of blocks was not mentioned in the paper and we looked at the GitHub repository of the paper for this hyperparameters with permission of the professor. We implemented the U-Net architecture based on our knowledge and searching the web. The skip connections are just a pointwise addition as explained in appendix A.3 in the paper. For downsample and upsample layers we followed appendix A.4. For LayerNorm and skip-init we followed [4].

For training, we use the same setting as they did. We use Adam ($\beta_1 = 0.9, \beta_2 = 0.9$, and 0 weight decay) for optimizer and with initial learning rate of 10^{-3} with cosine annealing scheduler ($\eta_{min} = 10^{-6}$). We train for 200K iterations, which is 209 epochs. The paper mentions they follow [3] for gradient clipping, however, [3] does not mention using gradient clipping in their paper, so we do not use it in our first experiment.

The training and validation loss of our first attempt is shown in Figure 4. Also we compare our results we those of the paper in Table 1. As we can see, there is a 0.87 dB difference. After rereading the paper and debugging our code, we found some small issues and fixed them. We also followed Figure 2 in [5] for our implementation of the U-Net structure, which is slightly different from ours. This was mentioned in paper, but we had missed in our first reading. After retraining our model with these changes we did not see much difference so we did not fully train our model after these changes. Looking at the loss plot in Figure 4 we thought the problem is model overfitting. As training our model takes a long time, we decided to see if there are any discrepancies between the paper and their code for hyperparameters like learning rates and optimizers which might affect overfitting. We found some discrepancies, explained in the next section, and applied them. We also applied gradient clipping. The train and validation loss plots are shown in Figure 5. As we can see, the results did not noticeably change so we stopped training after 70 epochs.

In our third attempt, we found the problem, which was a difference over a hyperparameter. We will explain this further in the next section. As the results depicted in Table 1 shows, our final attempt was successful.

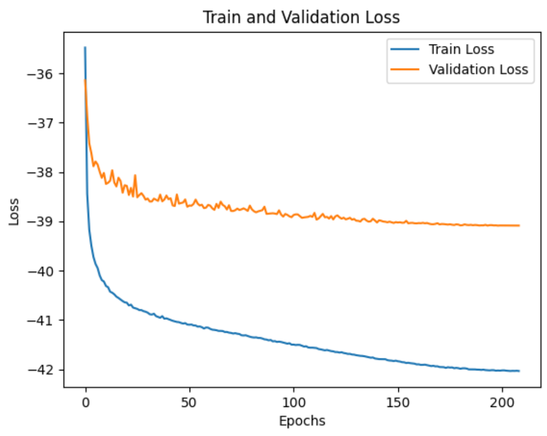


Fig. 4: The training and validation loss plot for our first attempt.

Table 1: Comparison our first attempt result with that of the paper. This result from the table is taken from the last row of Table 2 in the paper and shows performance of NAFNet with width 32.

Implementation	PSNR on SSID
Paper	39.96
Ours (first attempt)	39.09
Ours (third attempt)	39.99

3 Discussion

As we mentioned in the previous section, we found several discrepancies between the paper and its official implementation and also some missing information. The missing information were the number of U-Net layers and number of blocks in each layer. As it was not to possible to make a fair comparison without knowing the number of blocks, we took this information from the official repository of the paper with permission of the professor. The discrepancies were the following:

– The paper mentions using Adam while in their code they use AdamW.
– The paper mentions reducing the learning rate to 10^{-6}, however, they reduce it to 10^{-7} in their code.

Although we thought discrepancies were the reason our first attempt failed, as we showed in the second attempt, they did not have a noticeable effect on the result. So what was the problem?

The problem was a discrepancy between the hidden dimension (number of convolutional filters) of our model versus theirs. We found this problem using torch-summary to compare the number of parameters and output dimensions of our implemented architecture with those of the paper and noticed that while

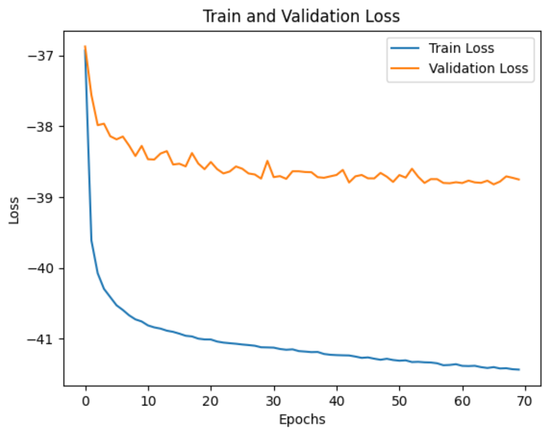


Fig. 5: The training and validation loss plot for our second attempt. We stopped after 70 epochs as we did not see much difference.

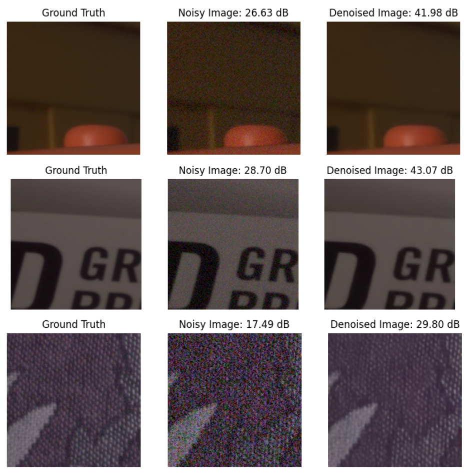


Fig. 6: Testing our first attempt implementation on some images.

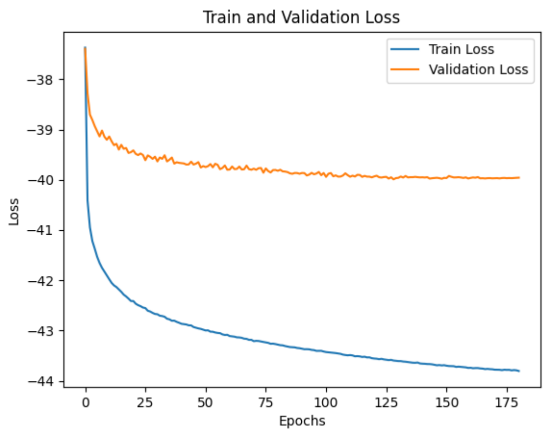


Fig. 7: The training and validation loss plot for our third attempt. We stopped at 180 epochs as the result were not improving anymore. It is interesting to note that the best validation loss happens at epoch 128.

our network has around 1.5M parameters, theirs has around 29M parameters. By inspecting the layer outputs using the same library, we found that we had used a smaller number for the hidden dimension of the blocks. We had used 6 as it is twice the number of input dimension, however, it seems the paper meant twice the number of input dimension for that layer not the input images, which might be our misunderstanding and we do not blame the paper for it.

In conclusion, we believe the paper was actually reproducible from the information provided in the paper alone maybe except for the number of blocks in different layers which was missing from the paper. We included our different attempts because for the second and third attempt we used the paper repository as mentioned above.

4 Team Members Contributions

Both team members contributed and discussed different parts of the project, however, we each focused on some parts more.

Sajad was mostly responsible for preparing the dataset, implementing the Baseline and building modules and Alireza focused more on NAFNet implementation and final visualizations. Both of us contributed to the train and test codes. We also would like to thank Irteza, another student of the course, for helping us with the computing resources.

References

1. Abdelhamed, A., Lin, S., Brown, M.S.: A high-quality denoising dataset for smartphone cameras. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1692–1700 (2018). https://doi.org/10.1109/CVPR.2018.00182
2. Chen, L., Chu, X., Zhang, X., Sun, J.: Simple baselines for image restoration. In: European Conference on Computer Vision. pp. 17–33. Springer (2022)
3. Chen, L., Lu, X., Zhang, J., Chu, X., Chen, C.: Hinet: Half instance normalization network for image restoration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 182–192 (2021)
4. Liu, Z., Mao, H., Wu, C.Y., Feichtenhofer, C., Darrell, T., Xie, S.: A convnet for the 2020s. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11976–11986 (2022)
5. Zamir, S.W., Arora, A., Khan, S., Hayat, M., Khan, F.S., Yang, M.H.: Restormer: Efficient transformer for high-resolution image restoration. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 5728–5739 (2022)