UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Single Image Motion Deblurring for Mobile Devices

Author:
Nazarii KUSPYS

Supervisor:
Volodymyr KARPIV

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences

Lviv 2023

Declaration of Authorship

I, Nazarii KUSPYS, declare that this thesis titled, Single Image Motion Deblurring for Mobile Devices and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

You have to stay faithful to what you're working on.

Stephen King

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Single Image Motion Deblurring for Mobile Devices

by Nazarii KUSPYS

Abstract

Motion blur is a common issue in image processing and video production. It is crucial to have a profound pre-processing method to address this obstacle for various image processing applications, e.g., object detection. Many existing state-of-the-art methods are limited to their computational complexity and memory requirements if one wants to use them in practice on mobile devices. This work proposes several ideas to overcome this issue and optimize existing solutions via architectural changes and graph optimizations. Moreover, we introduced an adapted version of Soft Attention inside skip connections and achieved a PSNR raise of 0.22 (dB) for the selected baseline. Finally, we derived the optimized model for mobile real-time applications without a significant drop in accuracy, e.g., obtaining 31.36 dB PSNR on GoPro (for image deblurring) with 24 FPS on the mobile application.

Acknowledgements

I feel immense gratitude and appreciation for my family, who provided me with a great deal of motivation, inquiring about the progress of my thesis every week. I also want to thank my supervisor Volodymyr KARPIV , who inspired me to start this research.

Contents

List of Figures

List of Tables

List of Abbreviations

DNN	**D**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
SOTA	**S**tate **O**f **T**he **A**rt
ML	**M**achine **L**earning
NLP	**N**eutral **L**anguage **P**rocessing
TLC	**T**est-time **L**ocal **C**onverter
SA	**S**oft **A**ttention

Chapter 1

Introduction

Motion blur is a frequent issue in image processing and video production when the camera or subject moves during exposure time. It leads to a blurry image that can obscure important details, making it hard to analyze or recognize objects. It can significantly diminish the image quality and hinder its usefulness for diverse applications.

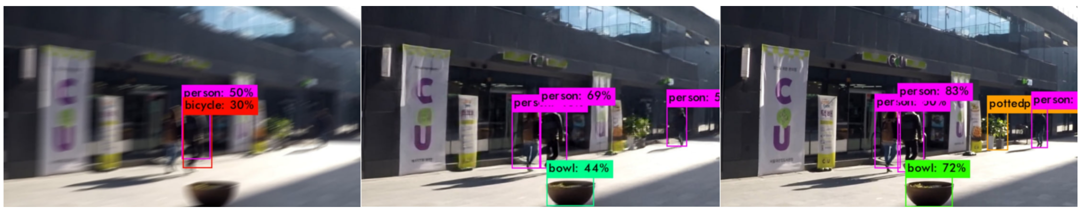

FIGURE 1.1: Redmon et al., 2016 detections on the blurred image (left), the Kupyn et al., 2018 restored (middle) and the sharp ground truth image (right)

For example, a motion blur makes it difficult for most object detection approaches to detect and identify objects precisely, as shown in Figure 1.1. Removing motion blur through image restoration strategies can enhance object detection accuracy and make locating and tracking objects in the scene simpler.

A wide variety of Image restoration strategies can be utilized to tackle the issue of motion blur. However, this work will be focused only on the DNN approaches for this task. One of the advantages of DNNs, compared to the classical computer vision algorithms, is that they can abstractly process complicated motion blur patterns. Typical methods usually assume the motion blur follows a definite mathematical formula, which may not be valid for all cases — especially if it needs to be modified for a specific production application. Also, the deep learning approaches firmly rammed the SOTA for motion image deblurring.

1.1 Motivation for optimization

Over the last few years, one has seen the deep learning tendency to increase the number of parameters, as well as the computational complexity of architectures, as shown in Figure 1.2.

Many approaches tend to use too heavy and complex DNN models for the competitions without explicit justification of the selected architecture, e.g., fully-connected architecture Zeng et al., 2022 outperforms the existing CNN/Transformer-based approaches for the Time Series Forecasting on ETTh1/2 datasets, introduced in Zhou et al., 2021. Moreover, the top 100 models, which achieve SOTA accuracy for Image

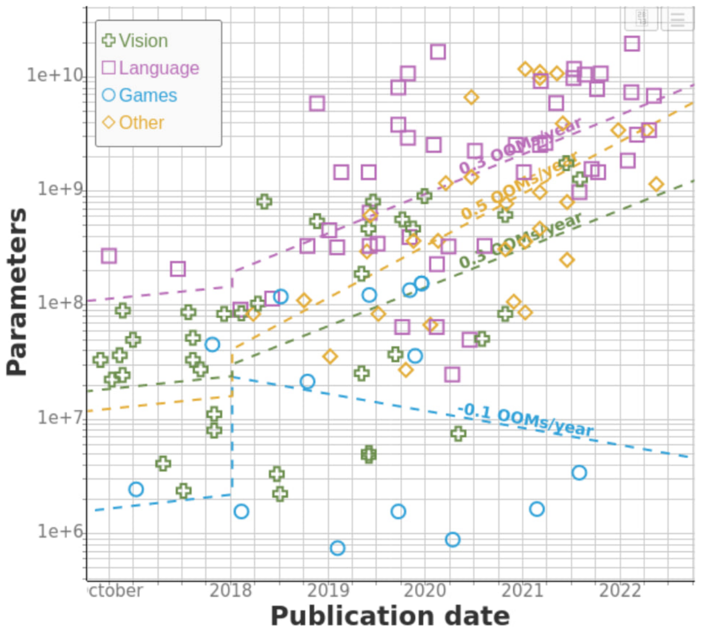

FIGURE 1.2: Villalobos et al., 2022 global (below-gap) trends in deep learning

classification on ImageNet, Russakovsky et al., 2015, have at least 100M of trainable parameters. This trend raises a limitation of the usage of such approaches in production. Thus they cannot be applicable to edge devices or mobile phones.

Chapter 2

Background Information

2.1 Image Restoration

Image restoration involves the recovery of a degraded image by using prior knowledge of the degradation process. This process entails estimating the model and using inverse filtering to restore the original image as accurately as possible. As the model aims to estimate the degradation process and find inverse transformation for the corrupted image, it always results in the approximation of the degraded image.

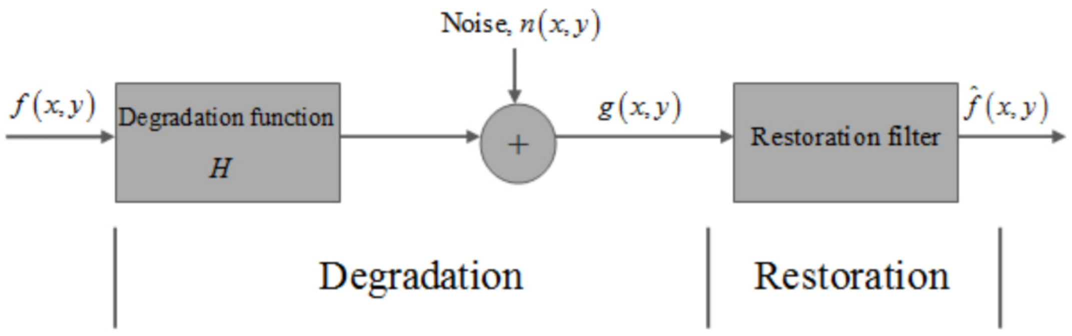

FIGURE 2.1: Khan et al., 2018 image degradation and restoration procedure outline

Generally, the process of degradation and restoration procedure can be mathematically formulated as follows, as shown in Figure 2.1:

$$g(x,y) = h(x,y) * f(x,y) + n(x,y) \tag{2.1}$$

where (x,y) denotes spacial pixel coordinates, $f(x,y)$ - original image signal, $h(x,y)$ - function of degradation, and $n(x,y)$ - noise. In order to restore the original image, our algorithm or DNN should estimate the $\hat{f}$ restoration function (filter).

2.2 Convolutional Neural Networks

The vanilla fully-connected neural network has some serious limitations for computer vision: if one changes the object's location, rotates, or shifts, it struggles to extract features correctly. The CNN architecture comes in handy, as it guarantees equivariance with respect to the translation. Also, it tends to learn better generalizable image priors from large-scale images, as well as, shows better performance in computer vision as it has a lower risk of overfitting and requires fewer parameters to train.

2.2.1 Simple CNN architecture for the image-classification problem

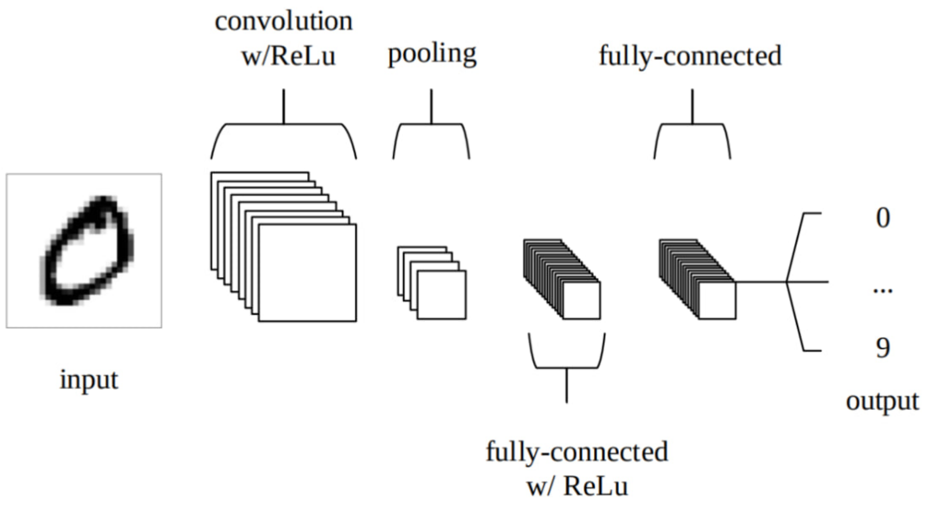

FIGURE 2.2: O'Shea and Nash, 2015, a simple CNN architecture, comprised of just five layers

Generally, the CNN architecture is composed of Convolutional, Pooling, Fully-connected layers, as shown in Figure 2.2 The purpose of trainable convolutional layer is related to extracting features from the image, like shapes, and edges. By stacking multiple convolutions, a CNN is able to learn abstract and complex features in the deeper layers.

The pooling layer has no trainable parameters and carries the purpose of reducing the size of feature maps, thus leading to less chance of overfitting and better generalization. For the image restoration tasks, the works tend not to use fully-connected layers, as it projects features maps to the space, from which further restoration is difficult.

Over the years, the CNN-based approaches achieved SOTA performance in multiple fields of computer vision. Starting form ResNet He et al., 2015, VGG16 Simonyan and Zisserman, 2015, AlexNet Alom et al., 2018. However, nowadays, practical use of vanilla CNN architecture is rare and attention-specific mechanisms are common in SOTA approaches.

2.2.2 Problems

Each convolutional kernel is operated upon a small neighborhood of pixels (or features); thus, it struggles to model long-range pixel dependencies. Increasing the kernel size (filter) or adding more depth to the CNN can put off the problem; however, it increases the model complexity, thus increasing the chances of overfitting.

Also, it applies filters to features without taking into consideration their importance, as usually, not all features are equally important. For example, the beautiful tree in the background can be less important than a cat's whiskers in case of a cat/dog classification problem. In addition, vanilla CNN architecture cannot determine which features are more relevant to each other, e.g., a cat's whiskers crop of feature maps should be closer to the corresponding feature of the cat texture itself than the background tree feature.

The above problems result in a limited receptive field and a lack of global understanding. That is why the attention mechanisms (or the Transformer blocks) are widely used to overcome these issues.

2.3 Transformer architecture

As the CNN architecture was developed due to limitations of Fully-connected, the same happened to Transformers, as it came after RNNs and CNNs. Among other advantages, It was designed to handle long-range dependencies in sequence models like RNNs more effectively and be computationally faster.

The Transformer architecture introduces the self-attention mechanism as a key innovation, allowing the model to take into account different parts of the input sequence when predicting the output. This is unlike RNNs, which must consider preceding elements sequentially. Additionally, multi-head attention further enhances this process and enables the model to capture intricate correlations between elements of the sequence more successfully than other architectures. Overall, the Transformer architecture has been shown to be highly effective for several natural language processing tasks, such as machine translation and language modeling, and it is now being applied to other areas, e.g., computer vision.

In this section will be described the usage of Transformer for NLP tasks, as some works in the image restoration field modify the Transformer block differently and the further application of this architecture will be discussed in the section of Related works.

2.3.1 Transformer attention block example for NLP case

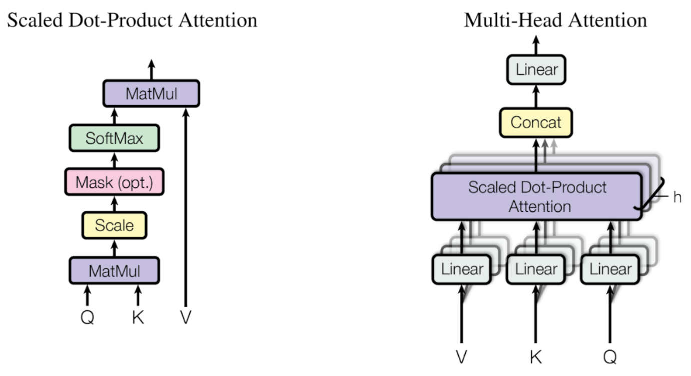

FIGURE 2.3: Vaswani et al., 2017 (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel

The whole process can be split into 3 steps, as shown in Figure 2.3:

- Obtain tokens: encode the words of the sequence to the Embeddings space, along with its location in the sentence
- Obtain Q, K, V vectors: for each token, multiply it with three weights matrices (W_Q, W_K, W_V) obtained from the training process. This will yield three vectors for each of the inputs: a key vector, a query vector, and a value vector.

- Apply attention via formula:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.2}$$

For the Multi-Head attention, the only difference is in step 2: one needs to stack h such matrices; thus, it results in multiple outputs of length = h.

Originally, authors affirmed that adding a Linear layer after Attention raises better generalizability in terms of features. The Transformer architecture has demonstrated remarkable proficiency in various natural language processing applications, as well as computer vision tasks, e.g., classification on ImageNet, where most part of Top 100 approaches used ideas of Transformer architecture.

2.3.2 Problems

However, the primary obstacles are the high computational cost and memory requirements which are particularly problematic for larger models and datasets. This is due to the self-attention mechanism used in the Transformer that necessitates each input token to attend to all other tokens - resulting in a quadratic complexity with regard to the number of tokens. Thus, the computational cost and memory requirements of the Transformer can become challenging to manage as sequence length increases.

Also, Transformer architecture learns long-range dependencies well. Thus, in practice, training on a large dataset was required to show good performance.

Chapter 3

Related Works

In this chapter will be described approaches and ideas, which are settled towards the goal – making the models more lightweight and preserving accuracy.

3.1 Multi-stage architectures

In this section will be discussed types of models, which use multi-scale inputs and output multi-scale outputs.

3.1.1 MPRNet

The proposed approach, Zamir et al., 2021 consists of the first two stages - encoder-decoder networks and the third stage - feedforward network. The two early stages help the third with scale-based supervision, thus resulting in better accuracy.

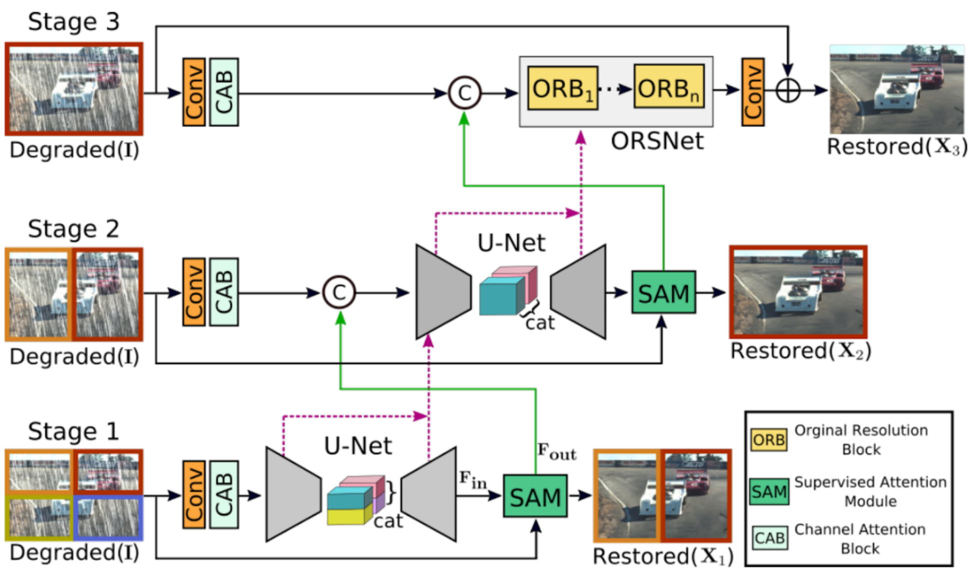

FIGURE 3.1: MPRNet architecture overview

The earliest two stages extract multi-scale features using classical UNet and propagate the features to the third stage via CSFF (cross-stage feature fusion) blocks and SAM (self-attention module), as shown in Figure 3.1. Also, the authors apply CAB (channel attention block) to encode more useful features.

3.1.2 MIMO-Unet

The previous approach 3.1.1 shows SOTA performance. However, it is limited to its computational complexity, as it has multiple stages with UNet architecture. That is why the authors of MIMO-UNet Cho et al., 2021 explored the issue of computational complexity in MPRNet and proposed the method, which uses only one U-shaped-like architecture with a modification of AFF (asymmetric feature fusion), as shown in Figure 3.2. After conducting the experiments, they proved that SOTA performance could be achieved without the need to use multi-cascaded U-Nets.

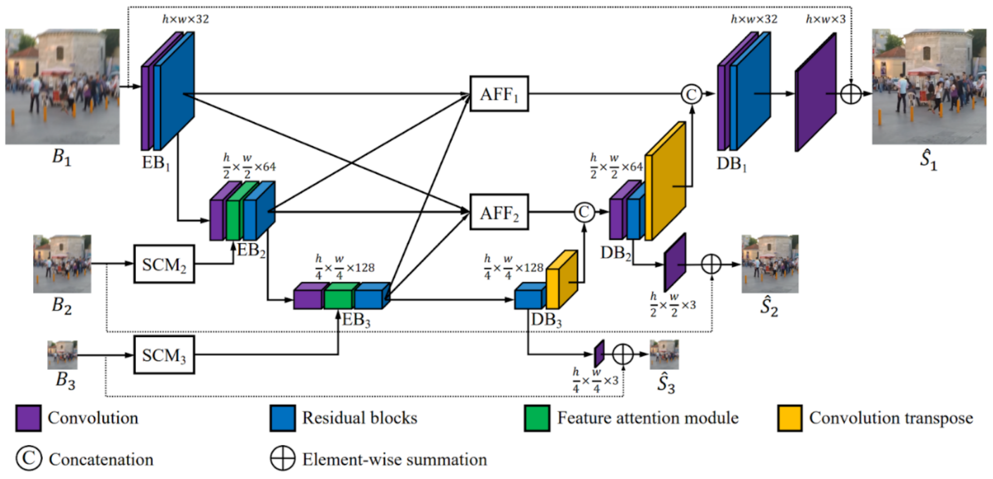

FIGURE 3.2: MIMO-Unet architecture overview

3.2 Vision Transformer Architectures

CNNs have been widely used for image deblurring due to their aptitude for learning generalizable image priors from large datasets. On the other hand, Transformers have demonstrated impressive performance in NLP and tasks involving long-range feature dependencies.

However, while Transformers address some of the issues associated with CNNs (e.g., limited receptive fields and inadequacy to input content), their computational complexity increases quadratically as the spatial resolution increases, making them unfeasible for most image restoration problems requiring high-resolution images. The major computational overhead in Transformers comes from the self-attention layer, and it results in $O(W^2H^2)$ for images of HxW.

The following three approaches apply different ideas of how to deal with the problem stated above. In this section, we will discuss only the proposed ideas that optimize the standard self-attention mechanism in Transformer architecture.

3.2.1 Uformer

The authors of Dosovitskiy et al., 2021 have utilized the power of Transformer architecture for computer vision, specifically, image classification. On the other hand, motivated by previous work authors of Wang et al., 2021 adapted this idea – using non-overlapping window-based self-attention instead of global self-attention to generate tokens, as shown in Figure 3.3. They achieved the computational complexity $O(M^2HW)$, where M is the size of a window.

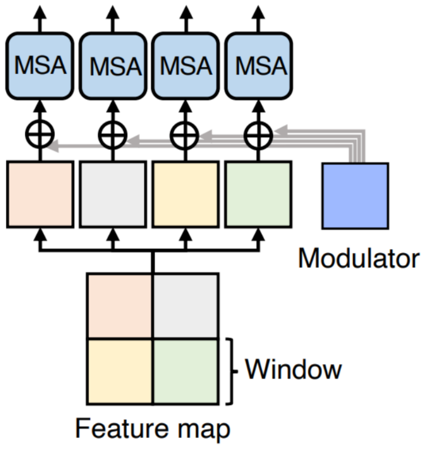

FIGURE 3.3: Uformer self-attention block overview

3.2.2 Restormer

Authors of Zamir et al., 2022 raised an alternative ingredient – apply self-attention across channels rather than the spatial dimension in MDTA (self-attention module), as shown in Figure 3.4. Thus, they achieved linear complexity rather than quadratic.

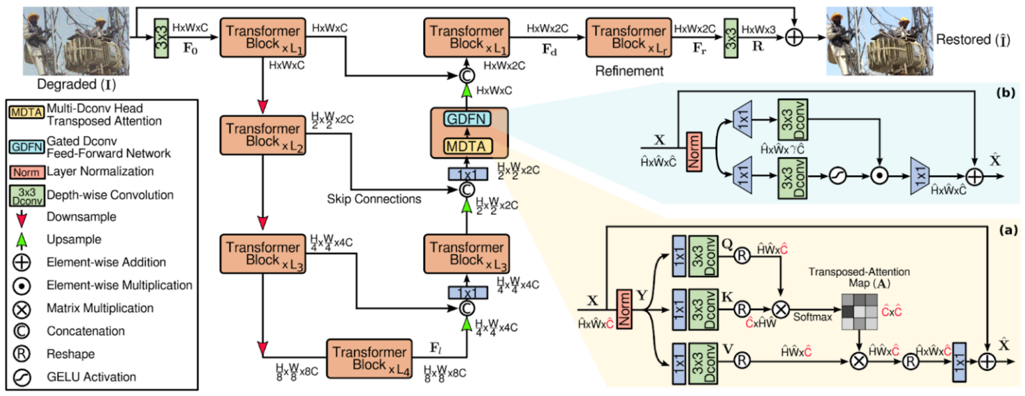

FIGURE 3.4: Restormer architecture overview. a) MDTA: Self-attention module b) Gated-Dconv Feed Forward Network

3.2.3 Stripformer

The authors of Tsai et al., 2022 aim to create a fast strip Transformer that achieves SOTA accuracy for different Image Restoration benchmarks. In order to fasten the attention mechanism, they apply attention layers in the horizontal/vertical axis separately (as shown in Figure 3.5).

The intra-strip tokens carry local pixel-wise blur features, while the inter-strip tokens – global region-wise blur information. By combining Intra-SA and Inter-SA

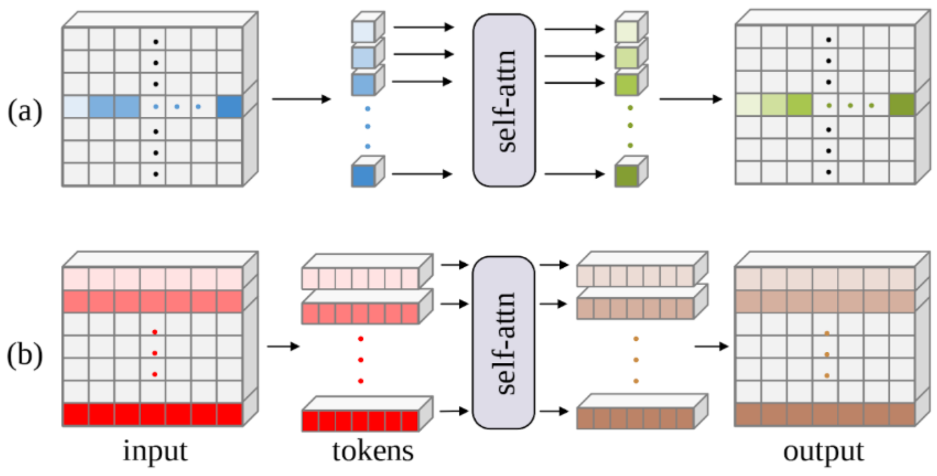

FIGURE 3.5: (a) Horizontal intra-strip attention (Intra-SA-H) – encode pixel dependence within the same horizontal strip. Vertical intra-strip attention (Intra-SA-V) is symmetrically constructed. (b) Horizontal inter-strip attention (Inter-SA-H) – capture stripwise correlations. Inter-SA-V is similarly established for vertical strips.

blocks, the authors got a hybrid Transformer architecture that gathers both global and local features inside their self-attention, as well as the model, operating in a linear complexity of image size.

3.3 NAFNet

The authors of Chen et al., 2022 have called into question the use of heavy models for Image Restoration and proposed a simple, lightweight baseline architecture, which outperforms existing solutions in terms of accuracy, along with providing a fast lightweight model for Image Deblurring on the GoPro dataset. Specifically, they exceed the previous SOTA of 0.38 dB with only 8.4% of their computational costs.

Starting with the plain U-Net block, as shown in Figure 3.6, they made two main modifications, as shown in Figure 3.7:

- Adding SCA, named “Simplified channel attention”
- Replacing ReLU activation with Simple Gate, which is a simple feature map multiplication:

$$\text{Simple Gate}(X, Y) = X \odot Y \tag{3.1}$$

This work is precious for our goal, as it is not problem-specific and provides ideas that can be used for optimization network architecture itself.

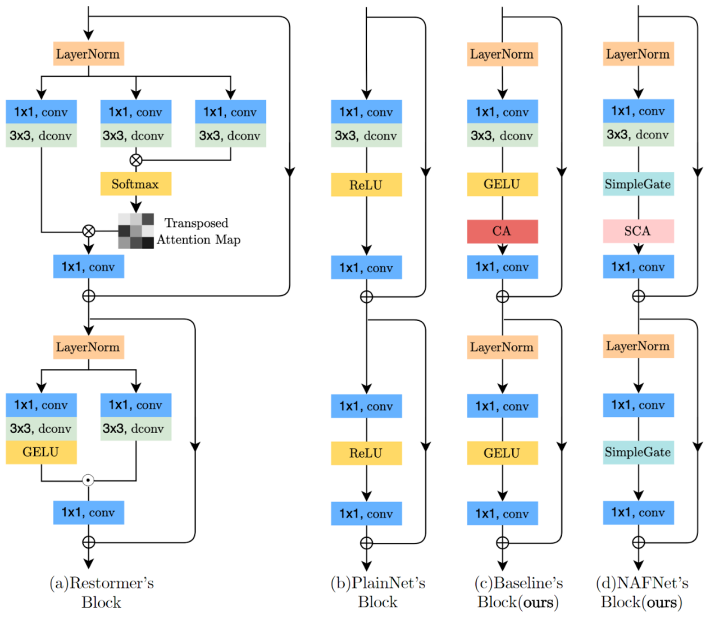

FIGURE 3.6: NAFNet U-Net block modifications

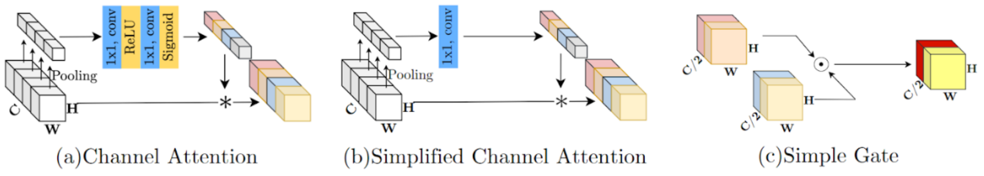

FIGURE 3.7: Illustration of (a) Channel Attention, (b) Simplified Channel Attention, and (c) Simple Gate

3.4 TLC

Image restoration models are commonly trained on cropped patches from images and test the performance directly on full-resolution images. The process of training is made in this way due to memory and time constraints, as well as the ability of CNN models to be automatically scaled with regard to the input size.

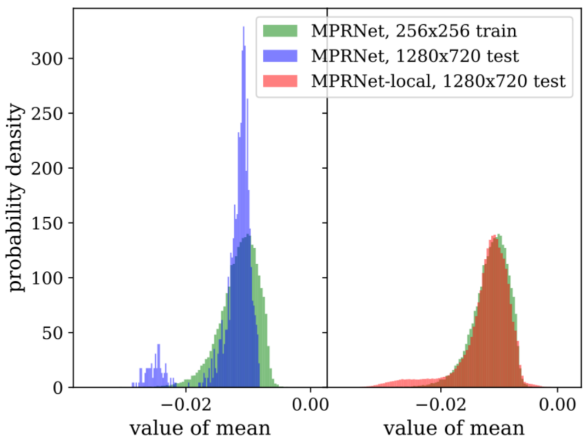

FIGURE 3.8: The effectiveness of TLC on MPRNet 3.1.1

The authors of Chu et al., 2022 began with the analysis of train-test distributions of feature maps inside global information aggregation modules(as shown in Figure 3.8), and they have found a significant difference between them. Thus, they proposed a tentative solution – Test-time Local Converter that modifies such aggregation modules in a way that they compute the aggregation function inside some local window (kernel).

Chapter 4

Methods

In this chapter will be described approaches and ideas, which are settled towards the goal – making the models more lightweight and preserving accuracy.

4.1 Proposed Approach

Overall, our pipeline infrastructure consists of steps illustrated in Figure 4.1:

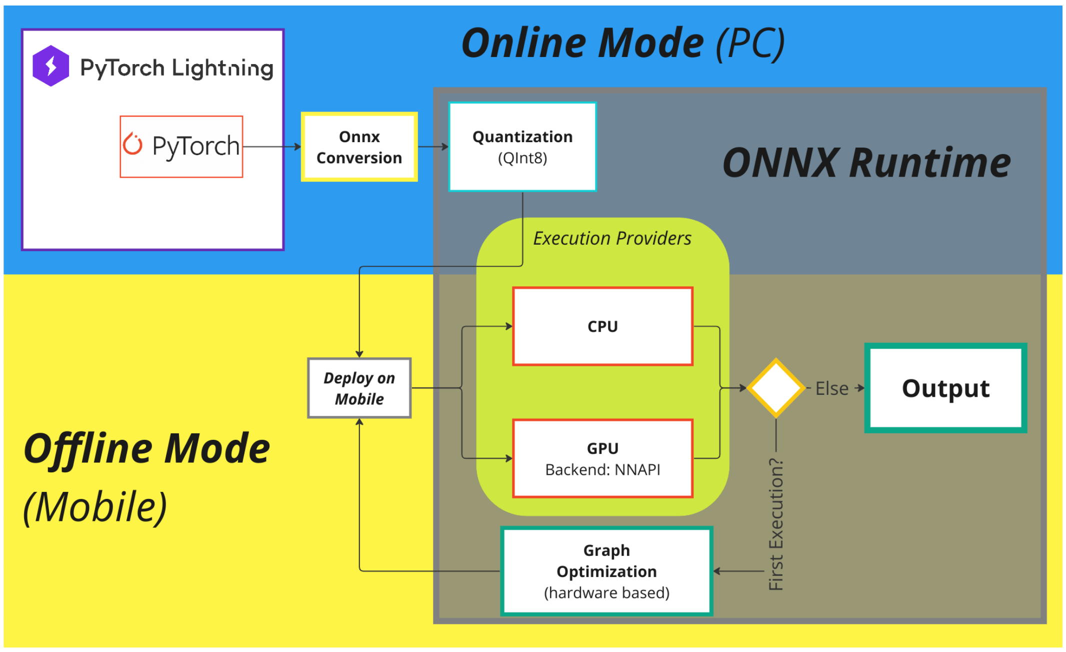

FIGURE 4.1: Pipeline Infrastructure Diagram

- Convert the trained pytorch model to the .onnx format.
- Quantize the model graph to the QInt8 data type.
- Deploy on the target mobile device the quantized .onnx (or .ort) model.
- Select the execution provider for the device. By this time, in case the model is successfully converted to .onnx, we do not expect any issues with further execution on the mobile CPU. Onnx Runtime currently supports only one backend for GPU – NNAPI, and it is in the experimental phase of development.
- Apply all graph optimizations, including the hardware-based, by executing the model with Onnx Runtime once and saving the optimized version on the specified device storage.

- Deploy the optimized version of the model on the target device for a previously selected execution provider.

After these steps, one should be able to obtain the model output with a fully optimized model.

In case one desires to use the pipeline for non-tested in this work architecture, we recommend one should check the success of the quantization step and the list of NNAPI layers, which are supported. The overview of the proposed pytorch model architecture will be in Section 4.3.

4.2 Onnx Runtime Optimization

4.2.1 Quantization

The API offers a scheme of 8-bit linear quantization, where the floating point values are mapped into 8-bit quantization space using the formula:

$$value_{fp32} = scale * (value_{quantized} - point_{zero}) \tag{4.1}$$

, where *scale* is a positive real number that maps the float point space to a quantization space. $point_{zero}$ is used to represent the zero point of float space in the quantization space.

There are actually two types of quantization in Onnx Runtime: static and dynamic. As the static one doesn't use computational overhead during inference to do quantization steps, we propose its use for all of our experiments. For the static quantization, we need to pass to the API the calibration data, which is used to determine values $point_{zero}$ and *scale*. During this process, the algorithm computes the quantization parameters for each activation in a way that the difference metric between activations of the original model and quantized will be minimal.

Empirically we have chosen that the 1000 samples from the training dataset are enough not to have a significant drop in accuracy. If some layers are not supported by Onnx Runtime in the architecture, the algorithm automatically covers it with DeQuant and Quant layers so that the full model can be converted.

4.2.2 Graph Optimization

We have used all available Onnx Runtime optimizations for a model graph by default, including general:

- Operator fusion – combining multiple consecutive operations (like Relu, Convolution, Batch normalization) into a single one, thus resulting in fewer intermediate tensors and leading to less computational/memory overhead.
- Constant folding - evaluating constant graph expressions and replacing them with values before the runtime.
- Removing unused layers from the graph, e.g., Dropout or Identity.

Also, it is worth mentioning that Onnx Runtime offers a set of optimizations that aim to select the best kernel, i.e., operation implementation, based on the input shape and available mobile hardware. However, this feature can only be used on the target hardware, e.g., we cannot apply an efficient optimization in advance on the PC (offline) in order to use an optimized model on the mobile device (online). There are other optimization tricks, which are used, but we noted here the most important ones.

4.3 Network Architecture

The network architecture diagram can be seen in the Figure 4.2.

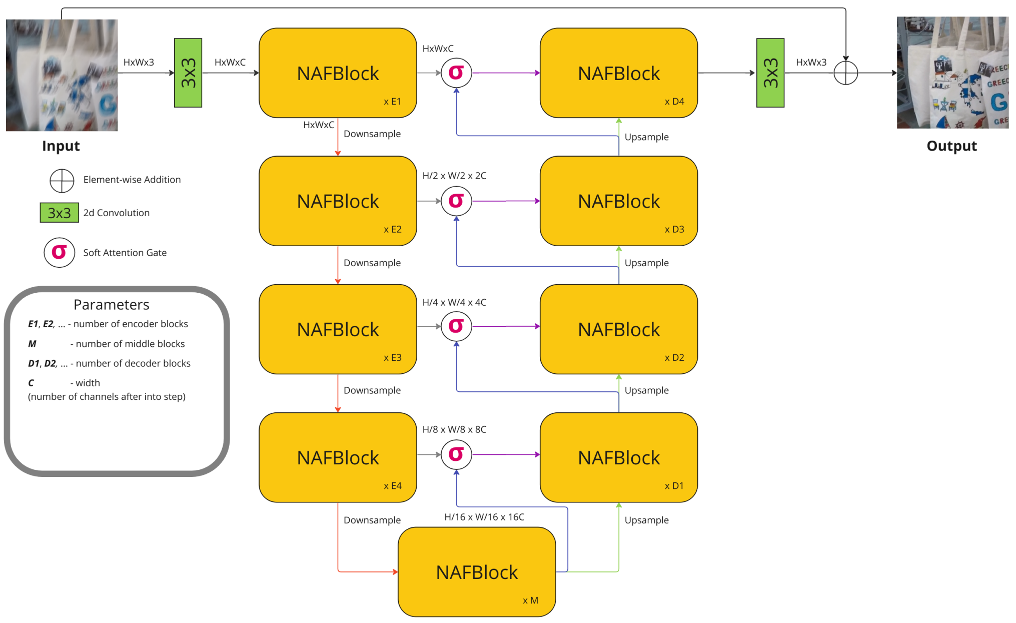

FIGURE 4.2: Proposed network architecture diagram

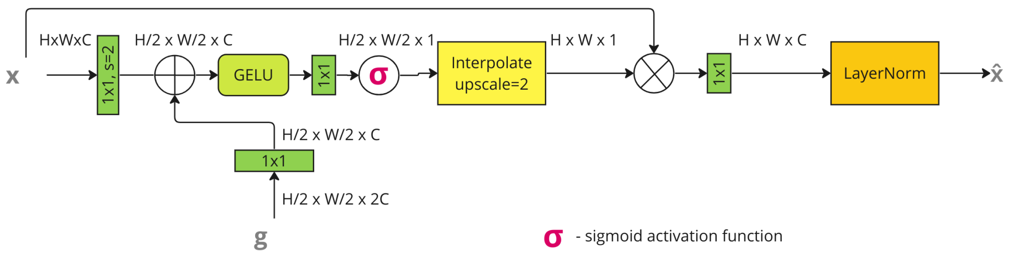

FIGURE 4.3: Modified Soft Attention Gate

The degraded blurry $I \in HxWx3$ image is firstly propagated through the $3x3$ convolution to get the embedded activation map $\in HxWxC$; then, it is passed to the series of encoder blocks and middle (bottleneck) blocks. Afterward, the features are passed to the series of decoder blocks, following the U-shape strategy with the modified Soft Attention Gate (Figure 4.3) inside each skip connection. The SA Gate is used to propagate the most representative spacial features. After the decoder phase, $3x3$ convolution is used to get back to the image space, obtaining $\hat{B}$ (estimated blur factor). The $\hat{B}$ is then added to the input image I to estimate the deblurred image $\hat{I} = \hat{B} + I$.

For the training, we propose using the combination of Charbonnier and Focal Frequency loss.

4.4 Datasets

For training and testing, we selected the GoPro open-source image dataset, as it is the most popular for the motion image deblurring benchmark and provides the largest number of non-synthetic training and testing images. It contains 3214 blurred images with the original size of 1280*x*720, which are divided into 2103 training images and 1111 test images. Originally, the dataset was obtained by capturing the sequence of sharp frames of a dynamic scene with a GoPro camera and then averaging them to create a blurred image. As the ground truth, i.e., restored image, the authors use the middle frame of the sharp frames sequence. The samples from the training dataset are visuallized in the Figure 4.4.

During training, we apply a crop of size 256*x*256 for the training set in a sliding window manner, thus obtaining 16824 blurry and restored image pairs for one epoch. Also, we apply flip and rotate augmentations during the training process.

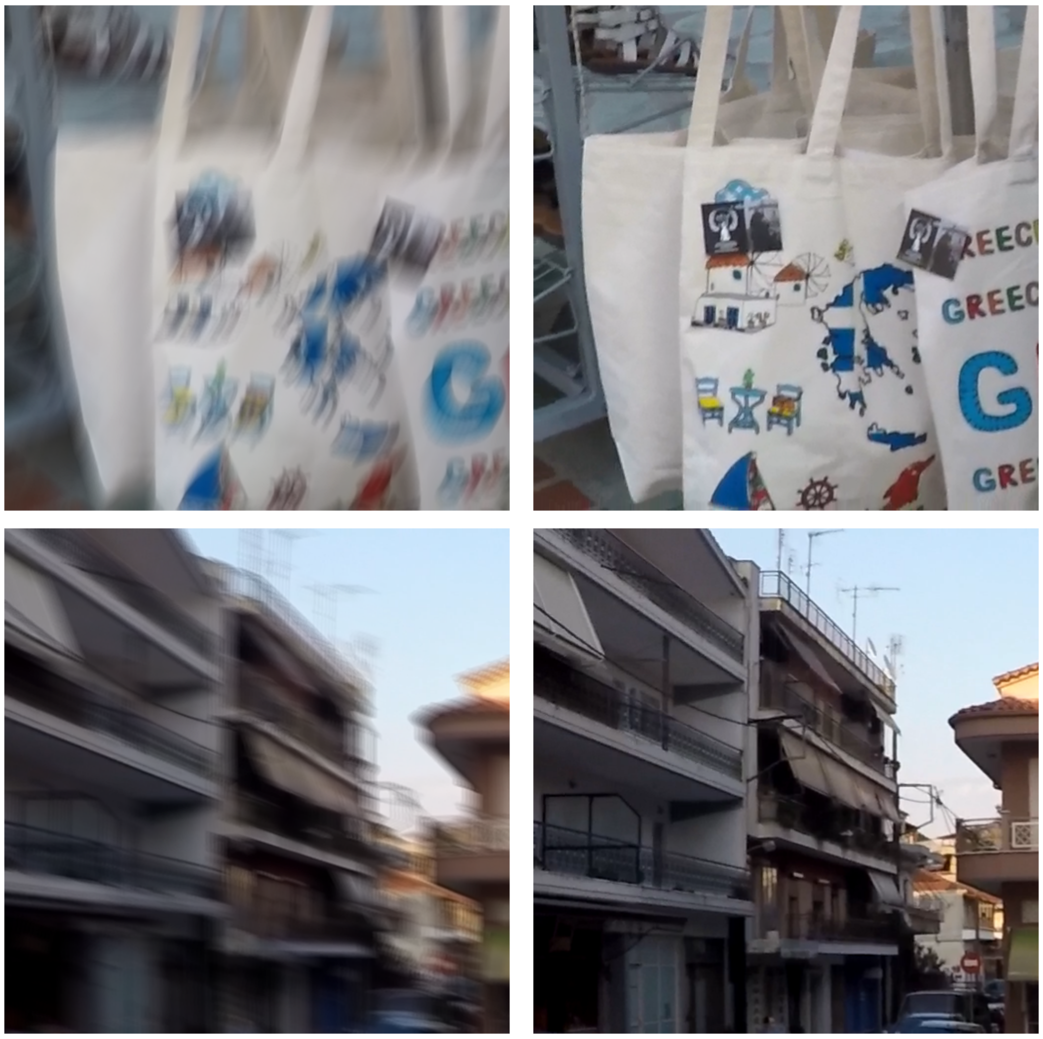

FIGURE 4.4: Training Samples Example. Left - blurry image, right - sharp (ground truth) image

4.5 Metrics

Suppose we have the ground truth sharp image S and the predicted $\hat{S}$. For the peak memory consumption and inference time, we measure them for the crop patches of 256x256.

4.5.1 PSNR

Peak Signal-to-Noise Ratio (dB) is a frequently used metric to evaluate the degradation of the signal, e.g., the quality of the deblurred image compared to the sharp ground truth image. The higher value refers to the higher quality and performance of the deblurring model.

$$\text{PSNR}(\hat{S}, S) = 10 * \log_{10}\left(\frac{MAX(S)^2}{MSE(\hat{S}, S)}\right) \tag{4.2}$$

, where MAX is related to the maximal value of input signal, e.g., 255 in case of 8-bit image, and MSE - Mean Squared Error.

4.5.2 SSIM

The Structural Similarity Index is a score metric that evaluates the similarity between two images by taking into consideration luminance, contrast, and structural differences. In general, the calculations for the samples x and y can be splitted into:

$$Luminance = \text{l}(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4.3}$$

$$Contrast = \text{c}(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{4.4}$$

$$Structure = \text{s}(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{4.5}$$

In these equations, μ refers to the mean value, σ - standard deviation, and C-s are the constants to avoid division by zero. Suppose that maximal value of a sample is MAX, then $C_1 = (0.01 * MAX)^2$, $C_2 = (0.03 * MAX)^2$, and $C_3 = \frac{C_2}{2}$ Finally, the formula for the SSIM:

$$\text{SSIM}(x, y) = l(x, y) * c(x, y) * s(x, y) \tag{4.6}$$

4.5.3 MACs

The number of Multiply-Accumulate operations is the metric that measures the computational complexity of an algorithm – provides a way to estimate the speed of the model. A single MAC operation computes a multiplication and adds it to the accumulator. This metric is not hardware based, thus allowing the authors of the research to compare their results to other works. However, the lower MACs metric does not guarantee less computational time, as we proved in some of our experiments.

4.5.4 Peak Memory Consumption

Another essential criterion for selecting a model for mobile deployment is related to the memory constraint. We propose to benchmark RAM consumption, especially peak value, during execution. This metric can give the lower bound of the RAM size, which device should have available to perform the deblurring.

We discard the model storage size metric. Usually, the storage size is not a defining criterion for the model selection due to the easy use of flash drives, e.g., external storage of various sizes.

4.6 Loss Functions

4.6.1 PSNR

The basic idea behind this loss type is quite simple – optimize the PSNR metric directly by using backward propagation through the tensor. Moreover, the authors of NAFNet use the PSNR loss for all their experiments. Thus we had to use this loss for further architecture comparisons. The formula is described in the 4.5.1

4.6.2 Charbonnier

This type of loss is a smoothed version of the well-known L1 loss that has a continuous gradient near zero. Also, it was practically proved more useful than the standard L1, as it is more robust to outliers and noisy images. Thus it is better for image restoration and suitable for our dataset, as GoPro contains real, slightly noisy images.

$$\text{Loss}(S,\hat{S}) = \frac{\sqrt{\sum((S-\hat{S})^2+\epsilon^2)}}{N} \tag{4.7}$$

, where N - is the size of the image and $\sum$ is over pixels.

4.6.3 FFT loss

Motivated by the recent results of work Jiang et al., 2021 for the image restoration task, we explored the possible usage of focal frequency loss for our task. In general, we use the basic version of this loss, which can be prescribed in the form:

$$\text{Loss}(S,\hat{S}) = F(\hat{S}) - F(S) \tag{4.8}$$

, where F - the mapping function from the image to the real frequency domain. The loss is the distance between blurry and restored image in the frequency domain. It was also proved to be highly effective for learning motion blur kernels, as the authors Mao et al., 2022 stated.

4.6.4 Full Objective

For the final tuning of the model, we made a set of experiments with the following loss combinations:

- Single PSNR loss (as in the baseline setup) $L = L_{PSNR}$
- Charbonnier, FFT losses combined $L = L_{Charbonnier} + \lambda_1 * L_{FFT}$
- PSNR, FFT losses combined $L = L_{PSNR} + \lambda_2 * L_{FFT}$

Note that λ_1, λ_2 are the trade-off hyperparameters empirically set to 0.01 and 0.01, respectively.

4.7 Default Training Setup

We follow the experiment's setting of original NAFNet authors to compare the changes for architecture to the original easily. More specifically, we train modified models with Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.9$, weight decay 1e-3) for a total of 200K iterations.

We started with a learning rate of 1e-3 that gradually decreased to 1e-6 using a cosine annealing schedule. The size of image patches used for training is $256x256$ pixels. We use a batch size of 32. In case the GPU limitation does not allow us to set up the exact same batch size for a memory-intensive architecture, we use the accumulate gradient batch strategy without any simplification.

However, there is one important modification to the training setup: we have changed the training environment from BasicSR to Pytorch Lightning because we encountered many hidden issues, the codebase was not well documented and tested, as well as didn't provide easy scalability. We did previous modifications to retrain the original model in our new setup, and all metrics coincided with the author's results.

Chapter 5

Experiments

5.1 Specifications

For our further experiments, we use OnePlus 8 Pro as the mobile device for benchmarking. It is equipped with GPU - Adreno 650, CPU - Snapdragon 865 5G, and 8 GB of available RAM. Moreover, as the PC GPU, we use NVIDIA RTX 4090 with 24 GB of VRAM.

5.2 Baseline Selection

Our initial goal was to achieve a fast model for further mobile deployment. Thus we have benchmarked most of the SOTA methods for the Image Deblurring on GoPro in terms of metrics and inference time on both platforms: a PC GPU and a mobile CPU. Due to the issue of the limited number of supported layers for the NNAPI GPU execution backend, not all current SOTA architectures can be used on the mobile without a drastic change in the architecture. Therefore, we provide our comparison only for mobile CPUs instead of mobile GPUs.

Analysis of both Figures 5.1 and 5.2 clearly shows that NAFNet architecture offers the best tradeoff between metrics results and inference speed. Moreover, it achieves the best results in terms of metrics. Thus we have chosen as the baseline the lightweight version of NAFNet, where the width and the number of blocks are 32 and 36, respectively. It should be noted that NAFNet's authors have already adopted the TLC (3.4) (NAFNetLocal version) for their model architecture. We have used this modification out of the box, as it mainly avoids the training by patches and testing by full resolution inconsistency.

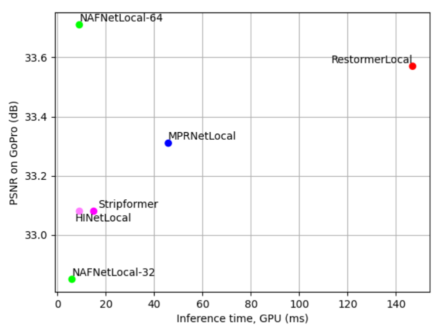

FIGURE 5.1: SOTA methods inference time on PC GPU

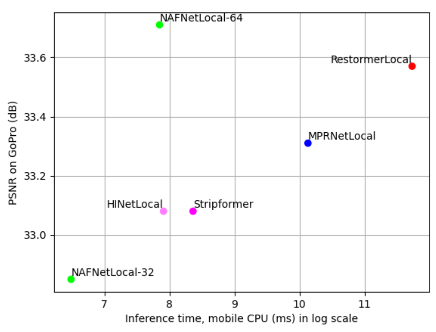

FIGURE 5.2: SOTA methods inference time on Mobile CPU

5.3 Optimizing the Architecture Strategy

Firstly, we started with an ablation study of the selected baseline. After the goal of an acceptable inference time and low memory consumption on the mobile device was reached, we started the second phase by replacing and combining additional ideas to the architecture, which could result either in better metrics or a faster model. Also, we tuned the obtained final model architecture by introducing different combinations of loss functions.

5.4 Baseline Ablation Study

We began our ablation study, by reducing the *C* - width and *E*4 – number of the last encoder's NAFBlocks of our baseline model, illustrated in Figure 5.3. All the missing information regarding the NAFblock is described in Section 3.3.

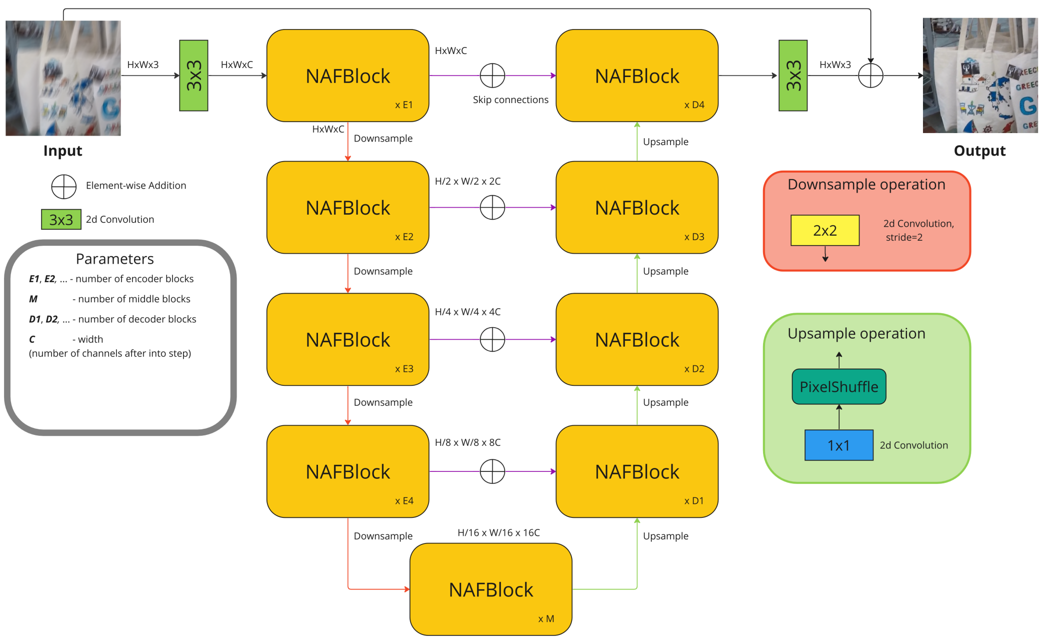

FIGURE 5.3: Selected Baseline Architecture

5.5 Depth-wise Separable Convolutions

Our baseline model has a CNN architecture with many 2d Convolutions inside, so we replaced them with Depth-wise Separable Convolutions, which are illustrated in Figure 5.4.

This type of layer decomposes the standard 2d convolution into two separable sequential operations:

- Depth-wise convolution: single convolution is applied for each input channel separately
- Point-wise convolution: 1x1 convolution is applied for input channels

This computational trick offers better computational complexity compared to the usual convolutional operation. Also, it can lead to a better generalization of the

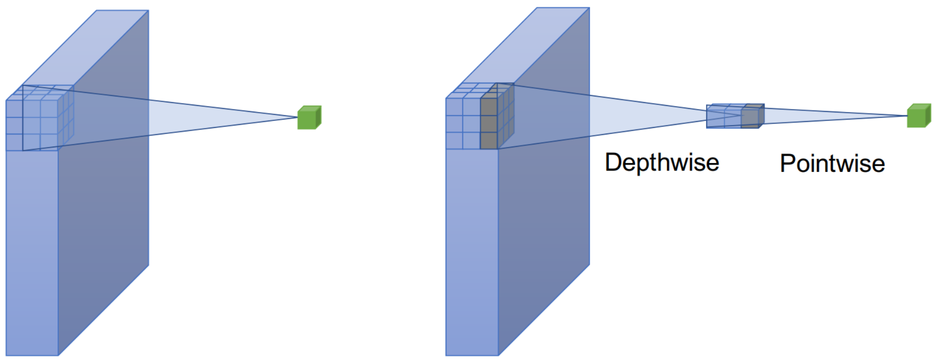

FIGURE 5.4: Depth-wise Separable Convolutions. Left - usual convolution, right - Depth-wise Separable Convolution

model due to decreasing the number of trainable parameters in the Image Classification, as the authors of Guo et al., 2019 stated.

5.6 Soft Attention

Motivated by the authors of Oktay et al., 2018, who managed to improve the accuracy of the standard U-Net architecture for the medical image segmentation task, we added the original Attention Gate (Figure 5.5) inside the skip connections for our baseline architecture. Moreover, we improve its performance by making engineering replacements for more modern layers.

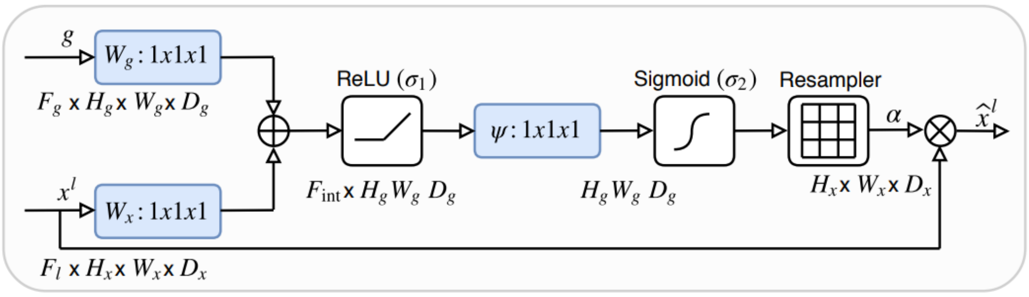

FIGURE 5.5: Original Soft Attention Gate

The idea behind the soft attention inside the skip connection is majestic and simple at the same time: the later activation maps have a higher level of feature representation compared to the starting ones, thus propagating the low-level spacial features related to the corresponding high-level features inside the skip connections, can improve the performance.

Also, the authors stated that this approach can be applied to a variety of computer vision tasks, e.g., image restoration in our case.

Chapter 6

Results

In this section, we present the metrics from our experiments and unclose the details of the experiments. For each of the experiments, we proceeded with all of the pipeline steps of the proposed approach, thus providing CPU/GPU (mobile) inference time, memory consumption, metrics, etc.

In all Sections, we use the quantized and fully optimized model to calculate the inference time, except only one Section – Onnx Runtime Quantizations.

Also, as the memory consumption for the GPU and CPU runtime may differ, we benchmarked both values. The memory consumption is evaluated via Android Studio Runtime Profiler for the whole application, which executes the model several times. From the profiler, we took only the peak value from the "Native" category, which relates to the memory objects executed in C/C++, as it is an exact place where the built Onnx Runtime lives. Unfortunately, as some of the buffers for the model execution are preallocated in the initialize phase, we could not determine the exact peak memory consumption during the model execution, as the difference between the peak value and the memory before the execution would not give us the whole memory, which the model used. However, this approach correctly estimates the upper bound of this value and is acceptable for our goal.

The optimization methodology we present for architecture optimization can be reused for any other architectures that are U-shape-like. However, one should not forget that only a small number of layers are supported by Onnx Runtime quantization, especially the GPU Executor - NNAPI backend.

6.1 Baseline Ablations Comparison

After composing several experiments with the modifications of the baseline and training each model from scratch with a similar default training setup, we achieved the following results in Table 6.1. In the column "Model name," the entry "Model-X-Y" has the following description: X and Y refer to the width and the total number of NAFBlocks, respectively.

One can see that the target PSNR and inference time on GPU lie between [32.87, 29.00] (dB) and [293, 19] (ms), respectively. The goal was to achieve at least 24 fps, so we chose the NAFNetLocal-16-18 as the baseline for our future experiments. We did not select more lightweight versions due to a much more significant drop in accuracy, e.g., the decrease in PSNR between NAFNetLocal-16-9 and NAFNetLocal-16-18 is 1.06 (dB), which is a bad trade-off for 9 (ms) in our opinion.

After selecting the baseline for future experiments, we got another 6.7 milliseconds to spare for changing it for modification. In general, we achieved a speedup of x4.45 for CPU, and x8.37 for GPU, while a PSNR accuracy drop of 1.72 (dB) compared to the original NAFNetLocal-32 to NAFNetLocal-16-18. The following Section will give a more detailed analysis of inference time and memory consumption.

Model name	Metrics (PSNR, SSIM)	Inference time Mobile CPU (ms)	Inference time Mobile GPU (ms)
Original NAFNetLocal-32-36	(32.87, 0.960)	423	293
NAFNetLocal-32-18	(32.17, 0.951)	170	95
NAFNetLocal-16-36	(31.80, 0.948)	147	89
NAFNetLocal-16-18	(31.14, 0.942)	95	35
NAFNetLocal-16-9	(30.08, 0.931)	75	24
NAFNetLocal-8-9	(29.00, 0.915)	43	19

TABLE 6.1: Baseline modifications inference time on mobile device

6.2 Onnx Runtime Optimizations impact

This section aims to observe the impact of the Onnx Runtime quantization from FP32 to QInt8 and hardware-based graph optimizations. To do so, we measured the inference time and memory consumption on the mobile device for various models from the previous section.

Also, one can clearly see from Table 6.2 and Table 6.3 ("Default" column means - without appliance of any optimizations and quantization): with the Onnx Runtime, we managed to achieve a speedup of x3.6 on average (Default, CPU vs Quantized + Optimized, GPU), as well as less memory usage x2.33 for GPU runtime and x2.95 for CPU runtime.

During the analysis of the peak memory consumption in Table 6.3, we found out that the memory consumption for the GPU's execution provider was higher than for the alternative - CPU. We expect this happens because the preloader allocates more RAM, as the GPU executor is faster than the CPU. Also, after optimizations, all models have relatively low peak memory consumption, which is acceptable for our mobile device.

Model name	Default, CPU (ms)	Quantized QInt8, CPU (ms)	Optimized FP32, CPU (ms)	Quantized + Optimized, CPU (ms)	Quantized + Optimized, GPU (ms)
NAFNetLocal-32-36	930	540	780	423	293
NAFNetLocal-32-18	455	222	321	170	95
NAFNetLocal-16-36	310	183	220	147	89
NAFNetLocal-16-18	150	121	140	95	35

TABLE 6.2: Impact on inference time after optimizations for baseline modifications

Model name	Default, CPU (MB)	Default, GPU (MB)	Quantized + Optimized, CPU (MB)	Quantized + Optimized, GPU (MB)
NAFNetLocal-32-36	630	740	190	305
NAFNetLocal-32-18	321	380	95	148
NAFNetLocal-16-36	220	253	84	121
NAFNetLocal-16-18	140	211	75	103

TABLE 6.3: Impact on memory consumption after optimizations for baseline modifications

6.3 Depth-wise Separable Convolutions Modifications Impact

For the selected modification of baseline - NAFNetLocal-16-18, we conducted a series of experiments:

- Replace the Convolutions inside the first two encoders and the last two decoders. For these layers, the activation map's number of channels exceeds at most 2C - 32 in our case, and - the highest operated resolution - at least $\frac{H}{2}x\frac{W}{2}$.
- In the opposite way to the first experiment, replace the Convolutions inside the last two encoders and the first two decoders, as well as in all middle blocks.
- Replace all Convolutions inside the model architecture when possible (e.g., if the number of groups for the Conv2d is larger than 1, we cannot make a replacement).

The above modifications are named "High resolution," "Low resolution," and "All," respectively in the Table 6.4 (Original states for no modification applied).

Modification	Metrics (PSNR, SSIM)	MACs	Inference time Mobile CPU (ms)	Inference time Mobile GPU (ms)
Without	(31.14, 0.942)	2.27	35	95
High resolution	(30.62, 0.937)	2.23	45	120
Low resolution	(30.85, 0.938)	2.25	41	112
All	(30.51, 0.935)	2.22	52	130

TABLE 6.4: Depth-wise Separable Convolutions Modifications speed performance

We found an inconsistency between the MACs number and the inference time during the experiments: after the replacement, we observed a drop in inference speed. It seems like parallelization inside the convolution took action, and even though the overall time complexity is higher, the inference time remains lower. Unfortunately, the performance dropped in our case. We believe it happened due to the specifics of the image restoration task – the model should not only encode the signal as accurately as possible in some space but also decode from this space into the restored signal accurately, in contradistinction to the image classification.

6.4 Soft Attentions Modifications Impact

The accuracy metrics dropped after adding the naive Soft Attention (SA) inside the NAFNetLocal-16-18. Thus, we did several experiments to encounter this issue by replacing operations:

- Activation: ReLU -> GELU
- Normalization: Batch Normalization -> Layer Normalization
- Upsample: Bilinear interpolation -> Transpose Convolution

Ideally, it would be better to experiment with different interchanges of these replacements. However, due to limited resources, we set the experiment in the way of adding consecutive changes.

Modification	Metrics, (PSNR, SSIM)	Inference time Mobile GPU (ms)	Memory consumption Mobile GPU (MB)
Without	(31.14, 0.942)	35	103
Naive SA	(30.85, 0.940)	38	110
Activation	(31.10, 0.941)	39	112
Activation + Normalization	(31.24, 0.943)	42	120
Activation + Normalization + Upsample	(31.16, 0.942)	45	139

TABLE 6.5: Soft attention modifications impact on performance

One can see from Table 6.5 that replacing the activation function and normalization gives us the PSNR increase of 0.10 (dB) with an inference time increase of 7 ms 20% relative to the initial, which is acceptable for our goal. On the other hand, replacing the Upsample operation raises the accuracy drop and inference time increase. Thus we discard this exact modification.

For the final model, we select "NAFNetLocal-16-18-SA," which stands for NAFNetLocal-16-18 with the modified Soft Attention, as it offers a great trade-off between accuracy and mobile inference time.

6.5 Losses tuning

For this experiment we select different types of training losses for our final model. One can see from the Table 6.6, that the training with a combined Charbonnier and FFT loss gives the best performance in terms of accuracy metrics.

Model name	Loss type	Metrics, (PSNR, SSIM)
NAFNetLocal-16-18-SA	PSNR	(31.24, 0.943)
	Charbonnier + FFT	(31.36, 0.944)
	PSNR + FFT	(31.25, 0.942)

TABLE 6.6: Final model performance for different training losses

6.6 Conclusions

We have selected two models for the final comparison:

- Original baseline: "NAFNetLocal-32" (NAFNetLocal-32-36)
- Final model: "NAFNetLocal-16-18-SA"

We have accomplished a 23.8 $\approx$ 24 frames per second (FPS) speed on a mobile device while maintaining an acceptable peak memory consumption (120 (MB)) for the entire application (Table 6.7). Our proposed approach resulted in a PSNR drop of 1.51 and a speedup of x6.7 times faster than the original architecture.

Although there was a slight decrease in accuracy, the visual performance of the model remained sufficient, as one can see in the Figure 6.1. Therefore, this approach can be utilized for various tasks such as the pre-processing algorithm for object detection, semantic segmentation, and more.

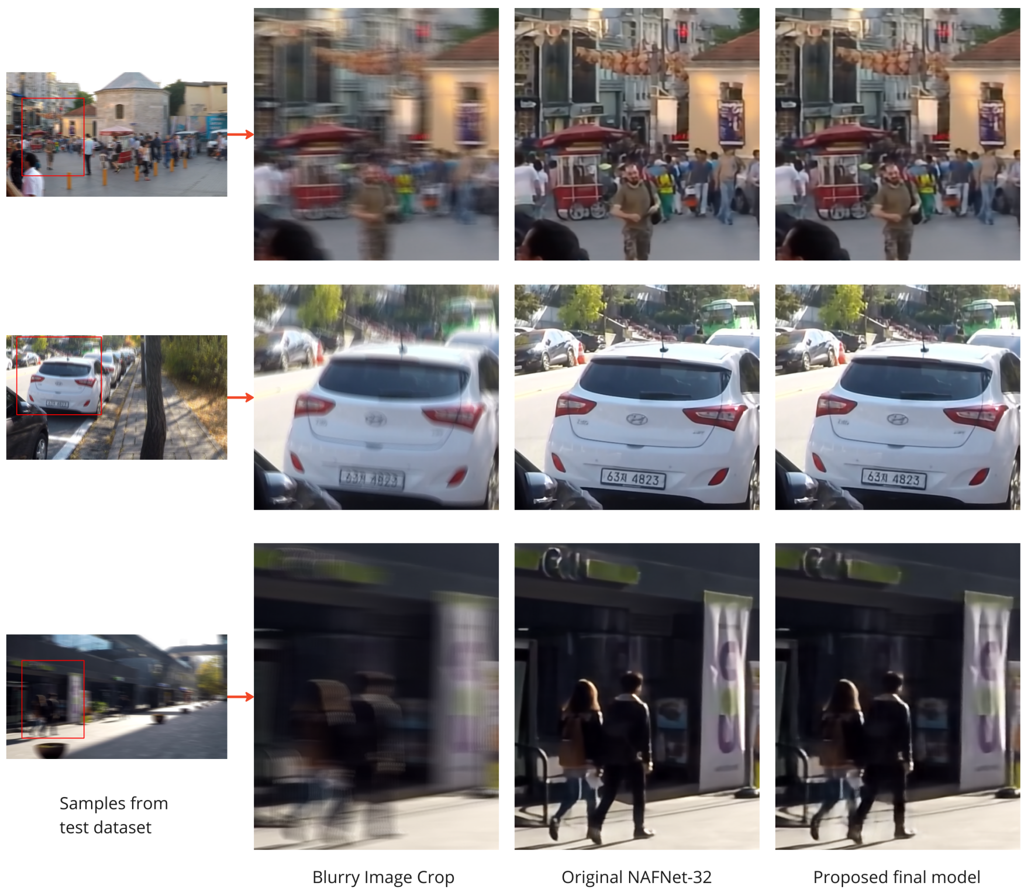

FIGURE 6.1: The proposed final model visual performance

Model name	Metrics, (PSNR, SSIM)	Inference time Mobile GPU (ms)	Memory consumption Mobile GPU (MB)
Original NAFNetLocal-32	(32.87, 0.960)	293	305
Proposed NAFNetLocal-16-18-SA	(31.36, 0.944)	42	120

TABLE 6.7: Final model performance comparison

Chapter 7

Summary

7.1 Contributions

The main contributions of this thesis are as the following:

- Conducted an investigation into the practical application of state-of-the-art (SOTA) solutions for mobile devices designed to deblur images.
- Proposed an efficient U-Net-like approach for a real time mobile application.
- Assessed the effectiveness of Onnx Runtime optimizations for the mobile application.
- Reintroduced the use of Soft Attentions in skip connections for U-shaped architectures.
- Highlighted the issues related to using MACs metric for comparing model speed, using depth-wise separable convolutions as an example.

7.2 Future Work

- The single blurry image has infinite possible solutions; thus, learning the distribution can improve visual performance, as the training process is not so strict.
- Use the Tensorflow-Lite for graph optimizations for better speed acceleration and more supported layers.
- Investigate the attention maps inside the Soft Attentions to achieve better performance.
- Use the modern encoder pre-trained on a large dataset.
- Improve the model generalization by using various augmentation methods.

Bibliography

Alom, Md Zahangir et al. (2018). *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. arXiv: 1803.01164 [cs.CV].

Chen, Liangyu et al. (2022). *Simple Baselines for Image Restoration*. arXiv: 2204.04676 [cs.CV].

Cho, Sung-Jin et al. (2021). *Rethinking Coarse-to-Fine Approach in Single Image Deblurring*. arXiv: 2108.05054 [cs.CV].

Chu, Xiaojie et al. (2022). *Improving Image Restoration by Revisiting Global Information Aggregation*. arXiv: 2112.04491 [cs.CV].

Dosovitskiy, Alexey et al. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv: 2010.11929 [cs.CV].

Guo, Yunhui et al. (2019). *Depthwise Convolution is All You Need for Learning Multiple Visual Domains*. arXiv: 1902.00927 [cs.CV].

He, Kaiming et al. (2015). *Deep Residual Learning for Image Recognition*. arXiv: 1512.03385 [cs.CV].

Jiang, Liming et al. (2021). *Focal Frequency Loss for Image Reconstruction and Synthesis*. arXiv: 2012.12821 [cs.CV].

Khan, Mohammad Mahmudur Rahman et al. (2018). "Digital Image Restoration in Matlab: A Case Study on Inverse and Wiener Filtering". In: *2018 International Conference on Innovation in Engineering and Technology (ICIET)*, pp. 1–6. DOI: 10.1109/CIET.2018.8660797.

Kupyn, Orest et al. (2018). *DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks*. arXiv: 1711.07064 [cs.CV].

Mao, Xintian et al. (2022). *Intriguing Findings of Frequency Selection for Image Deblurring*. arXiv: 2111.11745 [cs.CV].

Oktay, Ozan et al. (2018). *Attention U-Net: Learning Where to Look for the Pancreas*. arXiv: 1804.03999 [cs.CV].

O'Shea, Keiron and Ryan Nash (2015). *An Introduction to Convolutional Neural Networks*. arXiv: 1511.08458 [cs.NE].

Redmon, Joseph et al. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv: 1506.02640 [cs.CV].

Russakovsky, Olga et al. (2015). *ImageNet Large Scale Visual Recognition Challenge*. arXiv: 1409.0575 [cs.CV].

Simonyan, Karen and Andrew Zisserman (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv: 1409.1556 [cs.CV].

Tsai, Fu-Jen et al. (2022). *Stripformer: Strip Transformer for Fast Image Deblurring*. arXiv: 2204.04627 [cs.CV].

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Villalobos, Pablo et al. (2022). *Machine Learning Model Sizes and the Parameter Gap*. arXiv: 2207.02852 [cs.LG].

Wang, Zhendong et al. (2021). *Uformer: A General U-Shaped Transformer for Image Restoration*. arXiv: 2106.03106 [cs.CV].

Zamir, Syed Waqas et al. (2021). *Multi-Stage Progressive Image Restoration*. arXiv: 2102.02808 [cs.CV].

Zamir, Syed Waqas et al. (2022). *Restormer: Efficient Transformer for High-Resolution Image Restoration*. arXiv: 2111.09881 [cs.CV].

Zeng, Ailing et al. (2022). *Are Transformers Effective for Time Series Forecasting?* arXiv: 2205.13504 [cs.AI].

Zhou, Haoyi et al. (2021). *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting*. arXiv: 2012.07436 [cs.LG].