University of Bologna

Project Work
Deep Learning

Deep Image Deblurring

Eleonora Mancini, Marcello Simonati

February 12, 2021

Abstract

Image deblurring has been a challenging ill-posed problem in Computer Vision for many years.
Nowadays there are various different techniques and methods that have been proposed to deblur a degraded image. For specific types of blur there are specific methods to remove it. Differently from classical Computer Vision algorithms, instead of using pre-defined filters, we want let the net learn its own filter; to do so, we focus on the application of Deep Learning techniques for the reconstruction of images affected by two types of blur: Gaussian Blur and Motion Blur. The experiments were done on the CIFAR-10 dataset and the REalistic and Dynamic Scenes (REDS) dataset. The experimental results show that the proposed method, based on the use of atrous convolutions, generally outperforms some traditional deburring methods and deep learning-based state-of-the-art deblurring methods such as Generative Adversarial Networks (GAN) in terms of such quantitative indexes as peak signal-to-noise ratio (PSNR), structural similarity (SSIM) and mean squared error (MSE).
The code is available at:
https://github.com/MarcelloSimonati/DeepImageDeblurring

Contents

1 Introduction

Degradation of images is one of the major problems in Image Processing.
Blur in images is an unwanted reduction in bandwidth which degrades the image quality and it is difficult to avoid; it occurs due to atmospheric turbulence as well as improper camera setting. Along with blur effects, noise also corrupts the captured image.
Image restoration is a technique to get rid of the blur from the degraded image and recover the original image.
Nowadays there are various different techniques and methods that have been proposed to deblur a degraded image. Differently from classical Computer Vision algorithms, instead of using pre-defined filters, we want let the net learn its own filter; to do so, we are going to focus on the application of Deep Learning techniques for the reconstruction of images affected by two types of blur: Gaussian Blur and Motion Blur.
Deep Learning has become very popular in the field of image deblurring. Up to now, different Deep Learning models, such as Generative Adversarial Network (GAN) and Convolutional Neural Networks (CNN) have been proposed.
The approach used in this project is the one of Convolutional Neural Networks.
In this report we will discuss the two problems on which we focused on and how we have addressed them, based on the analysis of the latest studies in this field and providing a summary of the performances of our models.
We further provide results on the GoPro dataset [1] (see *Appendix D*) and compare our results to state-of-the art approaches.

1.1 Gaussian Blur and Motion Blur

In Image Processing, a **Gaussian Blur** (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian kernel function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail.
Formally speaking, the Gaussian kernel function is defined as the product of two Gaussian functions (one in each dimension) and it results as follows:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution. The higher the σ, the stronger the smoothing caused by the filter.
The rule-of-thumb to define the size k of the filter consists in the following mathematical relation: $k = 3\ \sigma$.

Figure 1: Starting from the left, the sharp image and the images resulting from progressive Gaussian smoothing.

In Image Processing **Motion blur** is one of the most common factors degrading image quality. It often arises when the image content changes quickly (e.g. due to fast camera motion) or when the environment is poorly illuminated, hence necessitating longer exposure times.
Many existing deblurring methods generally describe the blur processes as:

- Gaussian Blur:

$$B = S \times K + N \tag{1}$$

- Motion Blur (or non-uniform blur):

$$B = S \times k(M) + N \tag{2}$$

where "$\times$" denotes the 2D convolutional operator, B is the blurry input image, S is the sharp image, K is the blur kernel [1], $k(M)$ are unknown blur kernels determined by motion field M [2] and N is the additive noise.[3]

[1] In this case, it is the 2-D Gaussian function described in *Section 1.1*.

2 Datasets

In the following sections we describe the datasets used in our work. In particular, a dataset was used to study Gaussian noise deblurring and a dataset was used to analyze Motion blur removal.

2.1 CIFAR-10

We tested our networks on the CIFAR-10 dataset [4], which consists of 60 000 images of resolution 32x32.

2.1.1 Preprocessing and Data Augmentation

First of all we modelled the data in order to be used by the network to learn; we did that by smoothing every image with a Gaussian Kernel with a random standard deviation (σ) value between 0 and 3. After doing that, we divided the dataset into Training, Validation and Test set, with a proportion of 80 %, 10% and 10% each and eventually we started the computation of the learning algorithm.
In particular, we used 40 000 images to train the network, 10 000 for the validation and test set respectively.

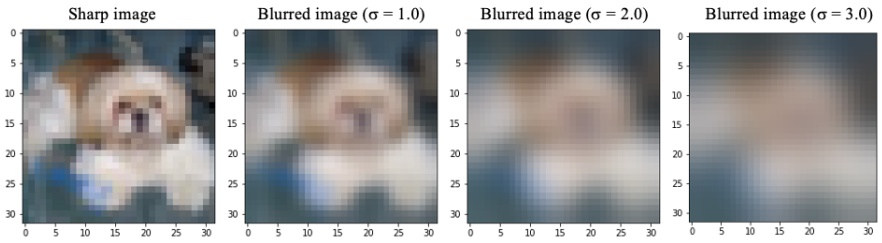

Figure 2: Image taken from CIFAR-10 dataset; the image is blurred with increasing standard deviation (from left to right).

We did not apply any data augmentation technique because after a few attempts we agreed that for all networks we obtained better results without using them.
One reason could be that given the relatively deep structure of our networks, by cutting or rotating our images, we obtained a consistent loss of information, also considering the small size of the images. All the networks used to address the Gaussian blur removal problem receive in input a single image at a time.

2.2 REDS

In order to address the Motion blur removal problem, we trained our networks on the REalistic and Dynamic Scenes dataset [5], which has been provided by the NTIRE2019 [6] and NTIRE2020 [7] Image Deblurring Challenge.
It consists of 300 videos with 100 images of size 720 × 1280 each, where 240 videos are used for training, 30 for validation and 30 for testing. The blurred images were synthesized by overlaying multiple sharp frames captured by a high-frame-rate camera. The sharp images of the test set are unknown. Because of this, in order to create a test set, we split the training set provided by the author of the dataset into training set and validation set, so that the validation set provided by the author became our test set. In this way we obtained 270 images, of which 210 are used for training, 30 for validation and 30 for testing.

2.2.1 Preprocessing and Data Augmentation

For training we use random image crops with a size of 256 × 256 pixels. This is mainly due to the enormous run-time and memory consumption when using full sized images 720×1280.
We varied image brightness by small amounts for every image crop. We also flip and rotate images at a 50% rate each. Due to the combination of all these image augmentations, it is almost impossible to get the exact same image twice during training.
All images were normalized to a range between 0 to 1.
In order to provide useful information of adjacent frames on learning the motion deblurring, the previous and next frames are concatenated to the target frame along the channel axis. The previous and next frames are shifted up to 2 pixels to the opposite directions respectively for the purpose of robust training. This **augmentation** strategy gives a small amount of random motion to stationary scenes.
In conclusion, our networks take as input a triplet of images at a time.

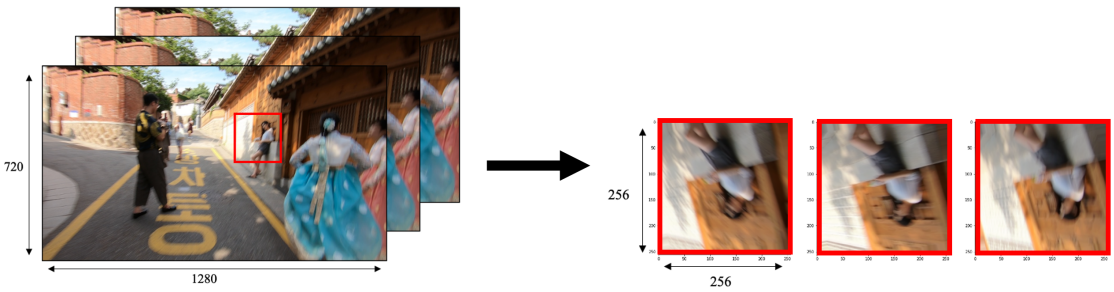

Figure 3: On the left, three consecutive frames taken from REDS training set. On the right, three images obtained by randomly cropping the frames on the left; the previously mentioned data augmentation techniques have been applied to these images.

3 Evaluation Metrics

The metrics employed to assess the similarity between the sharp images and the reconstructed ones are the following:

- **Structural Similarity Index Measure** (SSIM): is a method for predicting the perceived quality of digital television and cinematic pictures, as well as other kinds of digital images and videos. SSIM is used for measuring the similarity between two images. The SSIM index relies on structural information to evaluate image degradation.[8] The higher, the better.
- **Peak Signal-to-Noise Ratio** (PSNR): is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Because many signals have a very wide dynamic range, PSNR is usually expressed as a logarithmic quantity using the decibel scale. PSNR is most easily defined via the mean squared error (MSE). [9] The higher, the better.
- **Mean Squared Error** (MSE): in the context of image comparison, the MSE represents the cumulative squared error between the predicted and the original image. The lower, the better.

4 Description of the Architectures

In order to address the problem adequately we have decided to test many different networks. The architectures we have built are based on two main structures:

- **Convolutional Autoencoders**
- **Residual Blocks**

A synthetic representation is presented here:

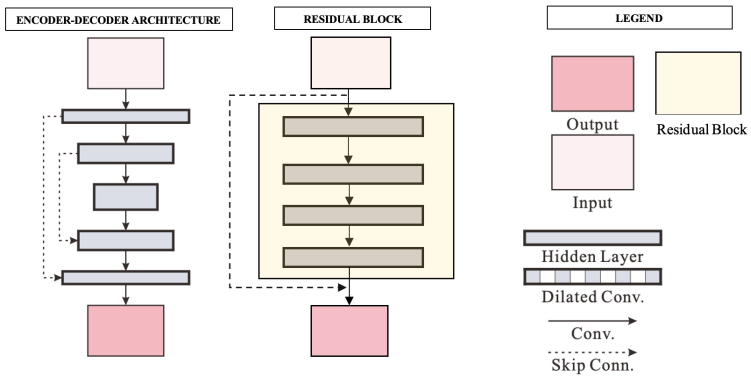

Figure 4: On the left the basic schema of a Convolutional Autoencoder, on the right the basic schema of a Residual Network.

We have built, studied and tested three different networks that we will call:

- **CARLO_NET**
  - Convolutional Autoencoder with the use of symmetric skip connections.
- **KAIST_NET**
  - Convolutional Autoencoder based on the idea of the U-Net architecture, where the covolution layers in the encoder and decoder parts are replaced with residual down-up and residual up-down blocks respectively.
- **ATROUS_NET**
  - Convolutional network based on the use of novel Residual Blocks composed of atrous (or dilated) convolutions.

In the section *6 Results* we will show a comparison between the performances of the networks. However, we can now announce that the network that showed the best results is ATROUS_NET, and in fact it is the official model that we propose in this work. First, we used different approaches to tackle the two problems (Motion Blur and Gaussian Blur) and only later we were able to find a suitable architecture to solve both.

4.1 The basic ideas behind our networks

4.1.1 Skip Connections

At present, skip connection is a standard module in many convolutional architectures. By using a skip connection, we provide an alternative path for the gradient (with backpropagation).
Using the chain rule, we must keep multiplying terms with the error gradient as we go backwards. However, in the long chain of multiplication, if we multiply many things together that are less than one, then the resulting gradient will be very small. Thus, the gradient becomes very small as we approach the earlier layers in a deep architecture. In some cases, the gradient becomes zero, meaning that we do not update the early layers at all. We commonly refer to this issue as *vanishing gradient problem.*
Skip connections in deep architectures, as the name suggests, skip some layer in the neural network and feeds the output of one layer as the input to the next layers (instead of only the next one).
It is experimentally validated that this additional paths are often beneficial for the model convergence.
In general, there are two fundamental ways that one could use skip connections through different non-sequential layers:

- **addition** as in residual architectures
- **concatenation** as in densely connected architectures

In all of our experiments we will use the first of the two methods, so we will provide a more detailed description of a ResNet in the section 4.1.2.
Additive skip connections are used in two kinds of setups:

- **short skip connections** : they are used along with consecutive convolutional layers that do not change the input dimension (see Res-Net 4.1.2)
- **long skip connections**: they often exist in architectures that are symmetrical (see CARLO_NET 4.2 and KAIST_NET 4.3), where the spatial dimensionality is reduced in the encoder part and is gradually increased in the decoder part. In the decoder part, one can increase the dimensionality of a feature map via transpose convolutional layers. The transposed convolution operation forms the same connectivity as the normal convolution but in the backward direction.

It is known that the global information (shape of the image and other statistics) resolves *what*, while local information resolves *where* (small details in an image patch).

4.1.2 Residual Blocks

Residual Blocks are skip-connection blocks that learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. They were introduced as part of the ResNet architecture.[10]
We employ this kind of building block in order to improve learning and avoid vanishing gradients.
Here is the structure of a residual block:

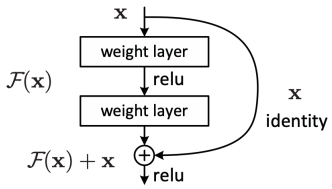

Figure 5: Residual learning - a building block [10]

4.1.3 Convolutional Autoencoders

A convolutional autoencoder is a neural network that is trained to reproduce its input image in the output layer.

- An image is passed through an **encoder**, which is a ConvNet that produces a low-dimensional representation of the image. The convolutional layers act as a feature extractor, which preserve the primary components of objects in the image and meanwhile eliminating the corruptions.
- The **decoder**, which is another sample ConvNet, takes this compressed image and reconstructs the original image. The deconvolutional layers have the capability to upsample the feature maps and recover the image details, which may be lost during the convolution process. The output of the deconvolutional layers is the recovered clean version of the input image.

The two networks that realize the idea of a convolutional autoencoder are CARLO_NET (4.2) and KAIST_NET(4.3).

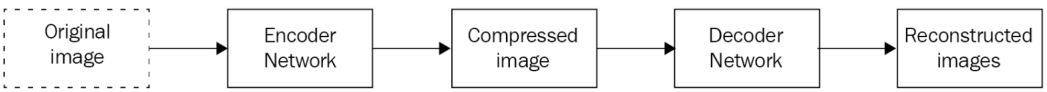

Figure 6: Encoder-Decoder network.

4.2 First experiment: CARLO_NET

After some studies and research, at first we decided to take a cue from the model presented in [11]. Although the paper [11] refers to the problem of removing Gaussian noise, it is slightly different from ours as the input data differ from ours both in terms of dataset and in terms of preprocessing, so we will not compare our results with those reported in the paper since this comparison could be inconsistent.
This model has been used only for the removal of Gaussian noise, therefore in the section *6* we will report only the training details and the results obtained on the CIFAR-10 dataset.

4.2.1 Architecture

In this section, we propose a very deep fully **convolutional auto-encoder** network for image restoration, which is a *encoding-decoding* framework with symmetric *convolutional-deconvolutional* layers.
In other words, the network is composed of multiple symmetric layers of convolution (encoder) and de-convolution (decoder) operators, learning end-to-end mappings from corrupted images to the original ones.
We propose to symetrically link convolutional and deconvolutional layers with skip-layer connections to deal with the problem that deeper networks tend to be difficult to train. Following this strategy the training converges much faster and attains better results.
More in details, the convolutional feature maps passed to the skip-connections layers are summed to the deconvolutional feature maps element-wise, and passed to the next layer after rectification. **ReLU** is added after each convolution and deconvolution. Skip shortcuts are connected every a few (in our experiments, two) layers from convolutional feature maps to their mirrored deconvolutional feature maps. The response from a convolutional layer is directly propagated to the corresponding mirrored deconvolutional layer, both forwardly and backwardly. We use neither pooling nor unpooling in the network.
We have tried different combinations in terms of number of convolutional and deconvolutional layers (5, 10, 15), number of filters (32, 64, 128) and kernel size ((3,3), (5,5)); the best results, shown in *Appendix A*, were obtained with the following combination: 10 convolutional layers, 10 deconvolutional layers, 128 filters, kernel size (3,3).

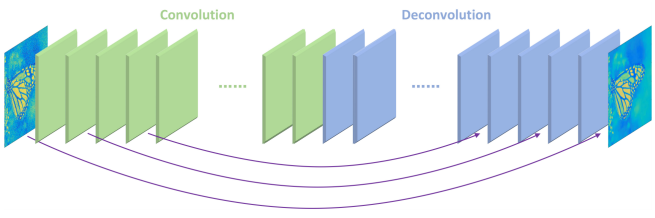

Figure 7: The overall architecture of our proposed network.

It is worth mentioning that since the convolutional and deconvolutional layers are symmetric, the network is essentially pixel-wise prediction, thus the size of input image can be arbitrary. The input and output of the network are images of the same size $w \times h \times c$, where w, h and c are width, height and number of channels.

4.3 Second experiment: KAIST_NET

Doing more research, we decided to take a cue from the model presented in [12]. The model presented in [12] tries to address the Motion Blurring problem and the network is tested on REDS dataset.
Although the proposed model is used for Motion Blur removal, we decided to try to make the same network presented in [12] to perform also the removal of Gaussian noise. In fact, in section 6 we will report the training details and the results obtained with this model on the two reference datasets (CIFAR-10 and REDS).
As a reminder, let's recall what is expressed in the section 2: for Gaussian Blur removal the input consists of a single image, while for Motion Blur removal it consists in the concatenation of three frames.

4.3.1 Architecture

This motion deblurring network is based on the U-Net structure where the convolution layers in the encoder and decoder parts are replaced with novel residual down-up (RDU) blocks and residual up-down (RUD) blocks respictively.[13]

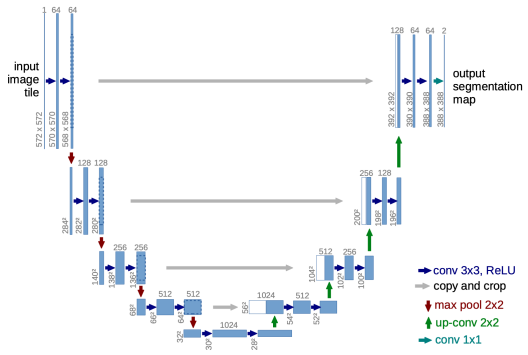

Figure 8: UNet architecture [13]

What differentiates KAIST_NET from CARLO_NET is that we do not simply have convolutional layers that are connected by long skip-connections, but we have a series of Residual Blocks (RDU for the encoder and RUD for the decoder) connected to each other through long skip-connections to prevent loss of information caused by reducing

spatial size.
In auto-encoder based regression models , like this, the spatial sizes of feature maps are reduced, while the numbers of feature map channels are increased in order to extract more various information.
Leaky ReLU is used as an activation function except for the output layers.
Nearest neighbor and average pooling are selected as up and down-sampling methods, respectively, between different scale levels. All convolutional layers have 5×5 filters and all transposed convolution layers have 4×4 to avoid the checkerboard artifact.
The **encoder** is made of 31 RDU blocks, the **decode** is made of 8 RUD blocks.
Our deblurring network generates the **output** through:

- 25-channel per-pixel kernels (adaptive kernels)
- a residual image of three channels
- a blending weight map. The weight map has a single channel and determines how to blend two preceeding outputs.

More in details, the final output is generated by the sum of the weighted residual image and the *adaptive convolution*, which is often denoted as a dot product (element-wise multiplication followed by summation). So, the final output can be represented as follows:

$$L = w \cdot B \times K_d + (1 - w) \cdot R \tag{3}$$

where B represents the input blurred image, K_d the per-pixel kernel, R the residual image, w the blending weight map, "$\times$" the adaptive convolution.[12]

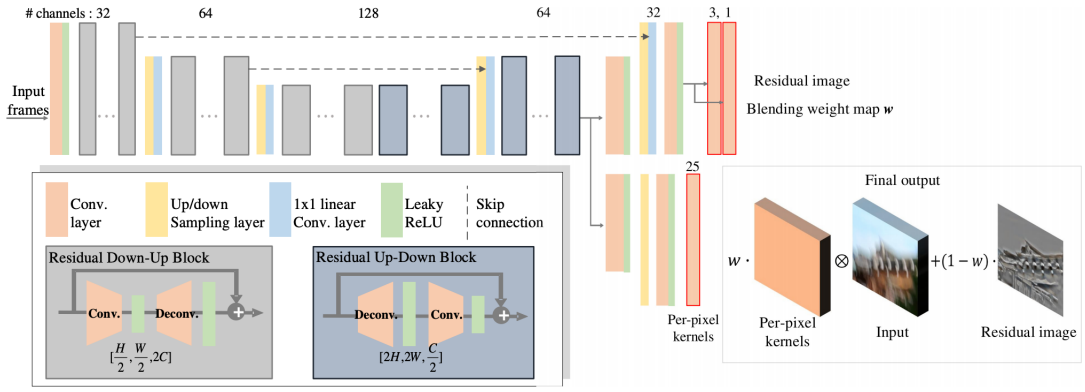

Figure 9: The architecture of the proposed network. For Gaussian Blur removal as input we have one single image instead of an input frame, differently from what is written in this figure taken from [12].

Input			Output		
Layer	H,W	C	Layer	H,W	C
Input	1	9	conv_1	1	32
conv_1	1	32	RDU_1 ($\times$9)	1	32
RDU_1	1	32	avg_pool_1	1/2	32
avg_pool_1	1/2	32	$1\times1\text{conv}_1$	1/2	64
$1\times1\text{conv}_1$	1/2	64	RDU_2 ($\times$9)	1/2	64
RDU_2	1/2	64	avg_pool_2	1/4	64
avg_pool_2	1/4	64	$1\times1\text{conv}_2$	1/4	128
$1\times1\text{conv}_2$	1/4	128	RDU_3 ($\times$4)	1/4	128
RDU_3	1/4	128	RUD_1 ($\times$4)	1/4	128
RUD_1	1/4	128	NN_1	1/2	128
NN_1, RDU_2	1/2	192*	$1\times1\text{conv}_3$	1/2	64
$1\times1\text{conv}_3$	1/2	64	RUD_2 ($\times$9)	1/2	64
RUD_2	1/2	64	conv_2	1/2	64
conv_2	1/2	64	NN_2	1	64
NN_2, RDU_1	1	96*	$1\times1\text{conv}_4$	1	32
$1\times1\text{conv}_4$	1	32	conv_3	1	32
conv_3	1	32	conv_4	1	3
conv_3	1	32	conv_5	1	1
RUD_2	1/2	64	conv_6	1/2	64
conv_6	1/2	64	NN_3	1	64
NN_3	1	64	conv_7	1	32
conv_7	1	32	conv_8	1	25

Figure 10: Configuration of our proposed network in Fig. 3. For each 'RDU' or 'RUD' block, the number in parenthesis indicates the repetition number of the same blocks in a cascade. '*' indicates concatenation of input tensors along the channel axis. The size of each estimated per-pixel kernel is 5×5, so 'conv8' layer has 25 channels.

4.4 Final model: ATROUS_NET

To build our final architecture we decided to take a cue from the model presented in [14].
We propose a novel network building block that employs multiple atrous convolutions in parallel. We carefully tune the atrous rate of each of these convolutions to achieve complete coverage of a rectangular area of the input; in particular, by using 3×3 atrous convolutions at different dilation rates we can achieve a very wide receptive field at a much lower computational cost (w.r.t. a comparable standard convolution), without losing too much spatial resolution.
We used this model both for the deblurring of Gaussian Blur and Motion Blur.
Since the results reported in paper [14] were obtained by testing the model on the REDS dataset, the comparison between our results and those obtained in [14] can only be made with the results we obtained on REDS in order to avoid inconsistencies.
Unlike the model used in the paper [14], we don't use two different networks (one for Single Image Deblurring and one for Video Deblurring). We took our cue from the network that the authors of the paper [14] use for the Single Image Deblurring and, to create the video mode, we implemented the concatenation of 3 frames as input, as indicated in the section 2.

4.4.1 Architecture

We analyze the use of atrous convolution for the task of image deblurring. Atrous convolution has already been used effectively in other dense-prediction tasks such as semantic image segmentation. [15]

4.4.1.1 Atrous Residual Block

Atrous convolutions is a method that allows increasing the spatial area covered by convolutional kernel without increasing the number of parameters. It is also known as dilated convolution. Instead of down-sampling, we use atrous convolution to increase the receptive field without reducing the spatial dimensions of the deep features of our CNN [2]. Thus we are able to maintain a high-resolution representation of our data in all layers of our proposed model.
As can be seen below, in Figure 11, we propose to use multiple parallel convolutions with different atrous rates.

[2] Thus *simulating a multiscale approach without reducing spatial resolution*

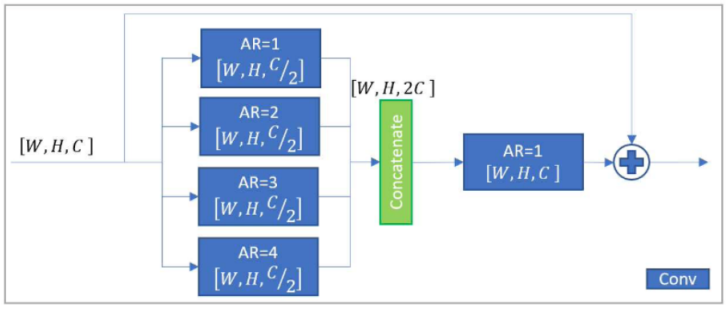

Figure 11: Residual block with atrous convolution used in our model.

Let's describe more in detail the layout of the proposed residual block is shown Figure 11:

- Each atrous residual block performs four 3×3 convolutions in parallel with 128 filters each and atrous rates of 1, 2, 3 and 4.
- We concatenate the output of all four convolutions to obtain a feature block that consists of 512 feature maps.
- Subsequently, we use another 3×3 convolution to combine these features and reduce the number of feature maps to 256 layer to the input of the block. The intuition behind this is that expanding feature depth within the residual block allows more information to pass through and can improve performance. [16]
- Each atrous block has a receptive field of size 11×11 pixels on the input feature map. Through the stacking of blocks the receptive field grows iteratively.

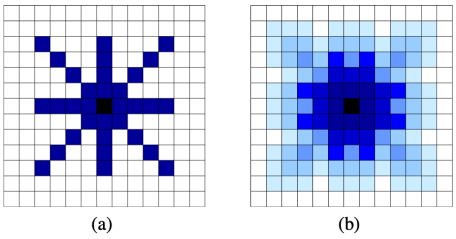

Figure 12: Receptive field of our proposed atrous residual block. (a) shows the receptive field (9×9) after concatenating the output of 4 different convolutions with atrous rates of 1, 2, 3 and 4. All layers are applied to the same input feature map. (b) shows the final receptive field (11×11) of the residual block after the second 3×3 convolution.

4.4.1.2 Overall architecture

Figure 13 gives an overview of the architecture of our final model.

- We first employ two consecutive convolution layers:
  - the first layer uses a large convolutional kernel with a size of 9×9 to extract low-level features
  - in the second layer we perform down-sampling by strided convolution with a stride of 2 and kernel size of 3×3
- Then, there is a specific number of our proposed atrous residual block to iteratively increase the size of the receptive field of the network. In particular, the number of residual blocks differs between the two applications:
  - **Gaussian Blur** removal: 5 atrous residual blocks
  - **Motion Blur** removal: 10 atrous residual blocks
- We up-sample the resulting features with a single deconvolution followed by 2 standard convolution layers.
- The output represents a residual image that we add to the blurred input image.

We use Leaky ReLU activation functions in all convolution layers except for the output layer. Further note that we do not use any common feature normalization technique such as batch-normalization because simply clipping our Adam optimizer was sufficient to keep the training stable.

We decided to use a different number of residual blocks for the two datasets due to the different image sizes. We need to carefully set the number of atrous blocks taking into account the size of the input image: images of CIFAR-10 dataset are much smaller than images of REDS dataset and the use of a network that is too deep to process very small images leads to a deterioration in performance.
We could not fully reproduce the architecture of [14] (in which 20 atrous blocks were used) due to time limitations and computational capacity. Moreover, as we have specified in the previous section, through the stacking of blocks the receptive field grows iteratively and our target patch size is lower than the one of [14] (256×256 vs 320×320); thus we need less blocks to achieve a similar receptive coverage of the input space. We therefore decided to find this compromise and we can affirm that on REDS we obtained results comparable to those of the paper [14], with a lower number of atrous blocks and a different strategy for the treatment of the input images. Please, refer to the *6 Results* section for further details.

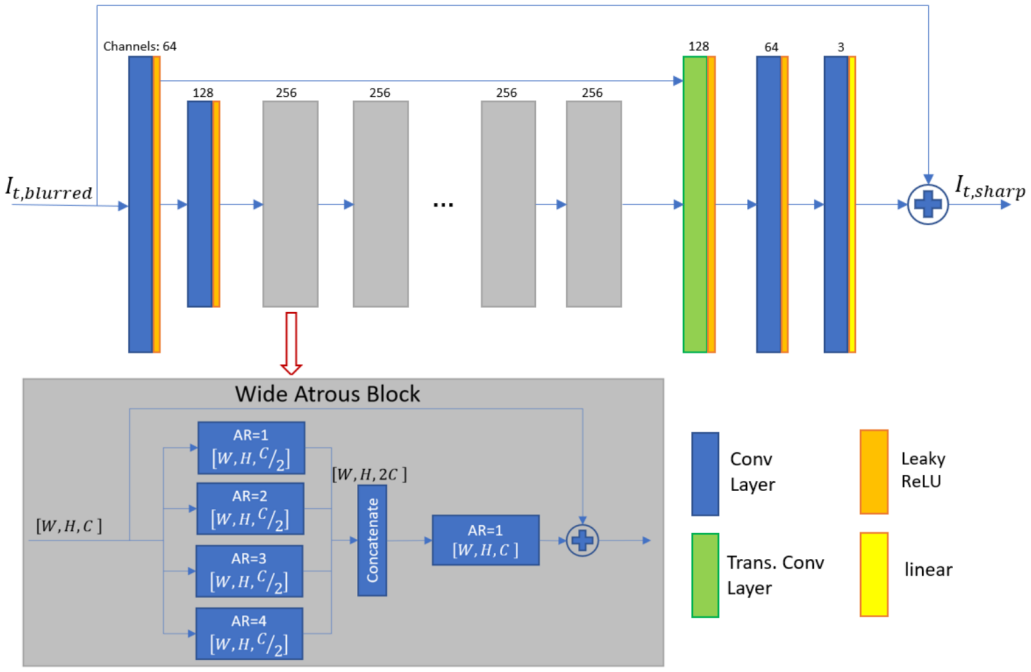

Figure 13: Proposed model architecture for atrous convolution with an enlargement of the structure of the residual atrous block.

4.4.1.3 Weight Initialization

As it is well known, a bad initialization of the weights can lead to the explosion of the gradient or the phenomenon of vanishing gradient. We initially tried to leave the default Keras initializer for each layer (Xavier/Glorot Uniform). Xavier Initialization initializes the weights in your network by drawing them from a truncated normal distribution with zero mean and a specific variance:

$$Var(\omega_i) = \frac{1}{fan\_in} \tag{4}$$

where *fan_in* is the number of incoming neurons and ω_i is the weight associated to each neuron. After about 60 epochs of training, we realized that our performance and loss function had stabilized. So we decided to change the initialization method of the weights and to restart the training; this significantly changed the performance of our model. In particular, we have opted for the use of *He Normal (He-et-al) Initialization.* This method of initializing became famous through a paper submitted in 2015 by He-et-al, and is similar to Xavier initialization, with the factor multiplied by two:

$$Var(\omega_i) = \frac{2}{fan\_in} \tag{5}$$

In this method, the weights are initialized keeping in mind the size of the previous layer which helps in attaining a global minimum of the cost function faster and more efficiently. The weights are still random but differ in range depending on the size of the previous layer of neurons.
This provides a controlled initialization hence the faster and more efficient gradient descent.[17]
Probably, our performance has improved significantly also because the Xavier / Glorot initialization is more suitable when the *tanh* activation function is used, while the He initialization is more suitable when using the $ReLU$ (as in our case).

5 Training details

5.1 CIFAR-10

Below are the details relating to the training phase of the various models.

Training details				
	Batch Size	**Iterations**	**Loss**	**Optimizer**
CARLO_NET	32	40	MSE/L2	Adam
KAIST_NET	32	30	LAD/L1	Adam
ATROUS_NET	32	29	LAD/L1	Adam

Table 1: Training details of our models on **CIFAR-10**.

Some notes:

- MSE is Mean Squared Error, LAD is Least Absolute Deviations.
- In every model we left unchanged Adam's hyper-parameters.
- The metrics employed to assess the similarity between the sharp images and the reconstructed ones are the ones mentioned in section 3.
- The results that we obtained by testing CARLO_NET, KAIST_NET and ATROUS_NET on the test set, being of 9000 images, can be found in the *6 Results* section.
- In *Appendix A*, *Appendix B* we report the trends of the loss, SSIM, PSNR and MSE obtained during the training phase with CARLO_NET and KAIST_NET. Since ATROUS_NET is the official model that we present in our work, we report here the trends of the loss and the metrics:

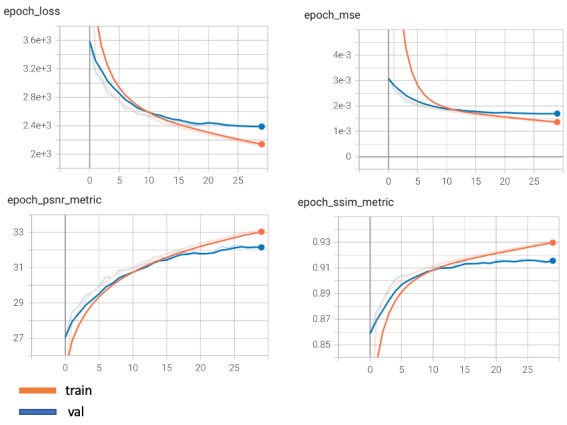

Figure 14: Loss, MSE, PSNR and SSIM trends over epochs for the model ATROUS_NET

5.2 REDS

Below are the details relating to the training phase of the various models.

Training details				
	Batch Size	**Iterations**	**Loss**	**Optimizer**
KAIST_NET	4	55	LAD/L1	Adam
ATROUS_NET	4	71	LAD/L1	Adam

Table 2: Training details of our models on **REDS**.

Some notes:

- LAD is Least Absolute Deviations.
- We have carefully decreased the value of the learning rate by hand every time we have identified a plateau in the loss function. The initial value of the learning rate for the optimizer Adam in KAIST_NET and in ATROUS_NET was 0,0001.
- The metrics employed to assess the similarity between the sharp images and the reconstructed ones are the ones mentioned in section 3.
- The results that we obtained by testing CARLO_NET, KAIST_NET and ATROUS_NET on the test set, being of 30 images, can be found in the *6 Results* section.
- In *Appendix C* we report the trends of the loss, SSIM, PSNR and MSE obtained during the training phase with KAIST_NET. Since ATROUS_NET is the official model that we present in our work, we report here the trends of the loss and the metrics:

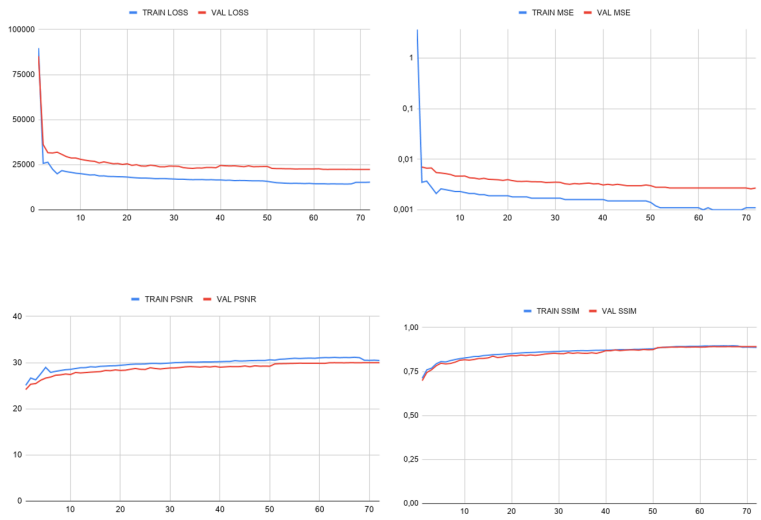

Figure 15: Loss, MSE, PSNR and SSIM trends over epochs for the model ATROUS_NET

6 Results

Here we report a summary of the results we have obtained with our networks on the two reference datasets (CIFAR-10 and REDS). We also report the results obtained by DeblurGAN and UniA team[3], which have been the reference standards for us as regards the generative approach and the convolutional approach tested on REDS dataset.
Since we took a cue from the architecture used by UniA team for Single Image Deblurring and we made the video mode working on the input, the results included in the table below refer to those obtained by UniA team in the NTIRE 2020 - Single Image Deblurring challenge.
We will then provide our considerations regarding the comparison between our results and those obtained from the two models just mentioned.

Performance on test set				
	CARLO_NET	KAIST_NET	**ATROUS_NET**	Baseline
SSIM	0.905	0.912	**0.9237**	0.7127
PSNR	29.34	31.53	**33.61**	24.64
MSE [10^{-3}]	2.11	1.77	**1.50**	6.34

Table 3: Results obtained by our models on the test set of **CIFAR-10**

Performance on test set					
	KAIST_NET	UniA team [14]	DeblurGAN-v2 [18]	**ATROUS_NET**	Baseline
SSIM	0.806	0.9412	0.8059	**0.9009**	0.7617
PSNR	28.70	34.44	28.92	**32.42**	27.048
MSE [10^{-3}]	2.42	—	—	**1.05**	3.88

Table 4: Results obtained by our models on the test set of **REDS**. Where not specified, the value of the MSE is not available for the reference network.

6.1 Considerations on the Results

Regarding CIFAR-10 we can say that we have obtained satisfactory results. In particular, we can see that as we initially announced, ATROUS_NET turned out to be the best network to achieve our goal.
Also with regard to REDS we can consider our results satisfactory. In particular, with our convolutional approach we were able to get better results than DeblurGAN (considered a standard for Generative Adversarial Networks used for motion blur removal). We also believe that the results obtained by us are comparable with those obtained by the team that inspired our work (UniA team), as both in terms of SSIM and in terms of PSNR there is not much detachment, even though we have used a network much shallower (10 atrous blocks against 20 used by UniA team), smaller input images

[3]UniA team is the group that won the NTIRE2020 competition for the Single Image Deblurring section on REDS test data.

(256×256 against 320×320 used by UniA team) and different kind of input [4].
We can therefore conclude that the use of atrous convolutions is the key point in the removal of the two types of noise considered (Gaussian Blur and Motion Blur); we also believe that the combination of atrous convolution and the use of the concatenation of three subsequent frames (current, previous and next frame), to address the motion blur problem, has represented the strength of all our work.

Regarding KAIST_NET, although the architecture created by the authors of the paper [12] is identical to ours, we need to take into account the fact that we used a larger patch size (256×256 vs 160×160) and therefore to have better results we should have either have trained it for more epochs or added a downsampling - upsampling step in order to have a suitable architecture for the patch size that we chose. We did not try these modifications, because at the same time we were already getting good results on ATROUS_NET and we decided to focus our effort on the latter model.

In Figure 16 and in the Figure 17 we report the images that we have reconstructed through our models.
It can be seen that the format of the image taken from the REDS test set and the one reconstructed by our network is slightly different. This is due to the fact that the original image resolution is 1280×720 and that we process the images using 256×256 resolution crop; therefore in order to rebuild it we used 5×2 patches with resolution 256×256.

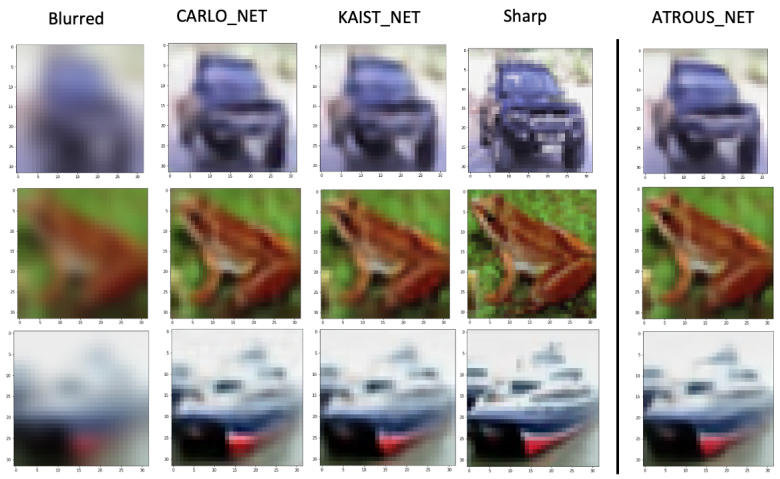

Figure 16: Comparison between the images taken from the CIFAR-10 test set and reconstructed through our models.

[4]Since we provide in input to the network a concatenation of 3 subsequent frames.

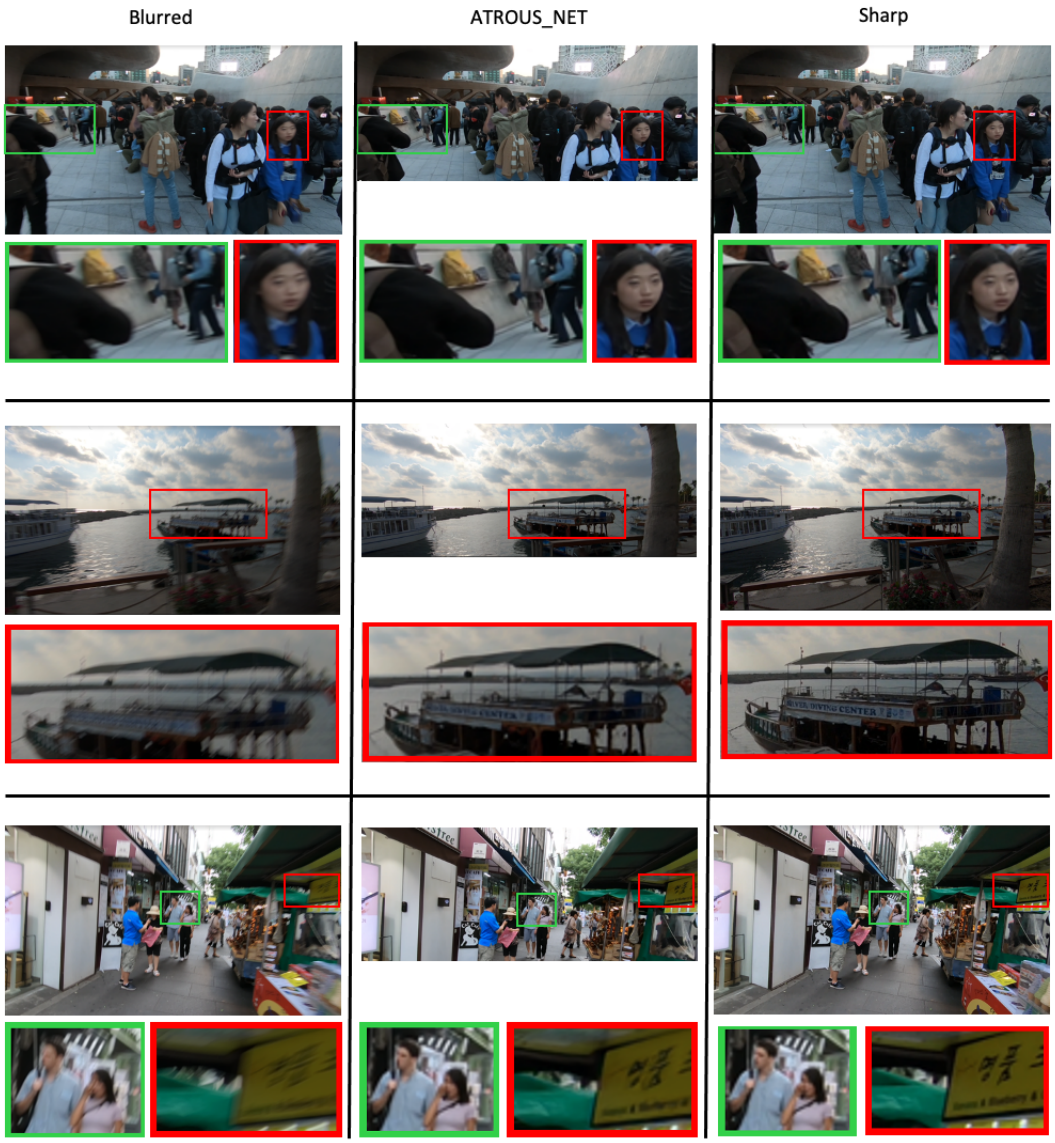

Figure 17: Comparison between the images taken from the REDS test set and reconstructed through our model.

Figure 18: Comparison between the crops of images taken from the REDS test set and reconstructed through KAIST_NET and ATROUS_NET.

7 Tools and Hardware

We used Google Colab platform, which always provided us a **GPU** NVIDIA Tesla P-100 and a dual-core (4-threads) Intel XEON **CPU** at 2.3 GHz.
During the training phase, the REDS dataset was hosted on Google Drive. To increase the training speed of our network we implemented a thread-safe generator for REDS dataset. Our networks were fully implemented using Keras.

References

[1] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. 2018.

[2] Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Jiri Matas. Deblurgan: Blind motion deblurring using conditional adversarial networks. pages 8183–8192, 06 2018.

[3] Quan Zhou, Mingyue Ding, and Xuming Zhang. Image deblurring using multi-stream bottom-top-bottom attention network and global information-based fusion and reconstruction network. *Sensors*, 20:3724, 07 2020.

[4] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[5] Seungjun Nah, Radu Timofte, Sungyong Baik, Seokil Hong, Gyeongsik Moon, Sanghyun Son, and Kyoung Mu Lee. Ntire 2019 challenge on video deblurring: Methods and results. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[6] Jianrui Cai, Shuhang Gu, Radu Timofte, Lei Zhang, Xiao Liu, Yukang Ding, Dongliang He, Chao Li, Yi Fu, Shilei Wen, Ruicheng Feng, Jinjin Gu, Yu Qiao, Chao Dong, Dongwon Park, Se Young Chun, Sanghoon Yoon, Junhyung Kwak, and Donghee Son. Ntire 2019 challenge on real image super-resolution: Methods and results. pages 2211–2223, 06 2019.

[7] Dario Fuoli, Zhiwu Huang, Martin Danelljan, Radu Timofte, Hua Wang, Longcun Jin, Dewei Su, Jing Liu, Jaehoon Lee, Michal Kudelski, Lukasz Bala, Dmitry Hrybov, Marcin Mozejko, Muchen Li, Siyao Li, Bo Pang, Cewu Lu, Chao Li, Dongliang He, and Shilei Wen. Ntire 2020 challenge on video quality mapping: Methods and results. 05 2020.

[8] Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13, 09 2004.

[9] Estimation lemma. Estimation lemma — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio, 2021. [Online; accessed 05-January-2021].

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 7, 12 2015.

[11] Yu-Bin Yang Xiao-Jiao Mao, Chunhua Shen. Image restoration using convolutional auto-encoders with symmetric skip connections. *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, page 2810–2818, 08 2016.

[12] Hyeonjun Sim. Munchurl Kim. A deep motion deblurring network based on per-pixel adaptive kernels with residual down-up and up-down modules. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 06 2016.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. volume 9351, pages 234–241, 10 2015.

[14] Stephan Brehm, Sebastian Scherer, and Rainer Lienhart. High-resolution dual-stage multi-level feature aggregation for single image and video deblurring. pages 1872–1881, 06 2020.

[15] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. 11 2015.

[16] Jiahui Yu, Yuchen Fan, Jianchao Yang, Ning xu, Zhaowen Wang, Xinchao Wang, and Thomas Huang. Wide activation for efficient and accurate image super-resolution. 08 2018.

[17] Vishnu Kakaraparthi. Xavier and he normal initialization. https://prateekvishnu.medium.com/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528, 09 2018. [Online; accessed 04-January-2021].

[18] Sungkwon An, Hyungmin Roh, and Myungjoo Kang. Blur invariant kernel-adaptive network for single image blind deblurring. 2020.

A Training results of CARLO_NET on CIFAR-10

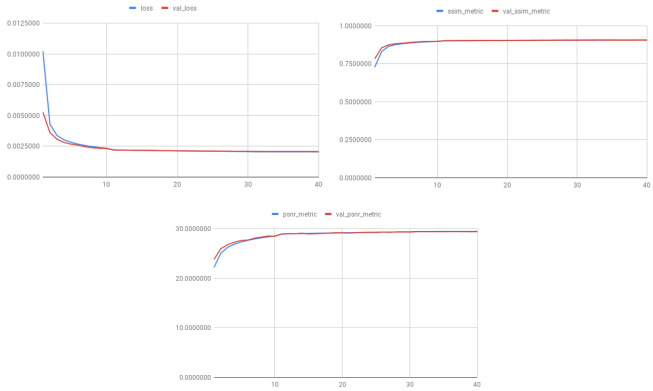

Figure 19: Loss (MSE), SSIM and PSNR trends over epochs for the model CARLO_NET.

B Training results of KAIST_NET on CIFAR-10

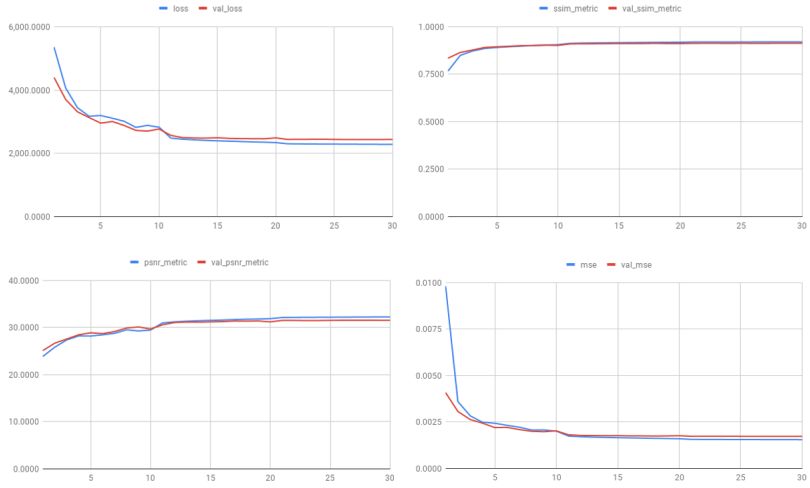

Figure 20: Loss, SSIM, PSNR and MSE trends over epochs for the model KAIST_NET.

C Training results of KAIST_NET on REDS

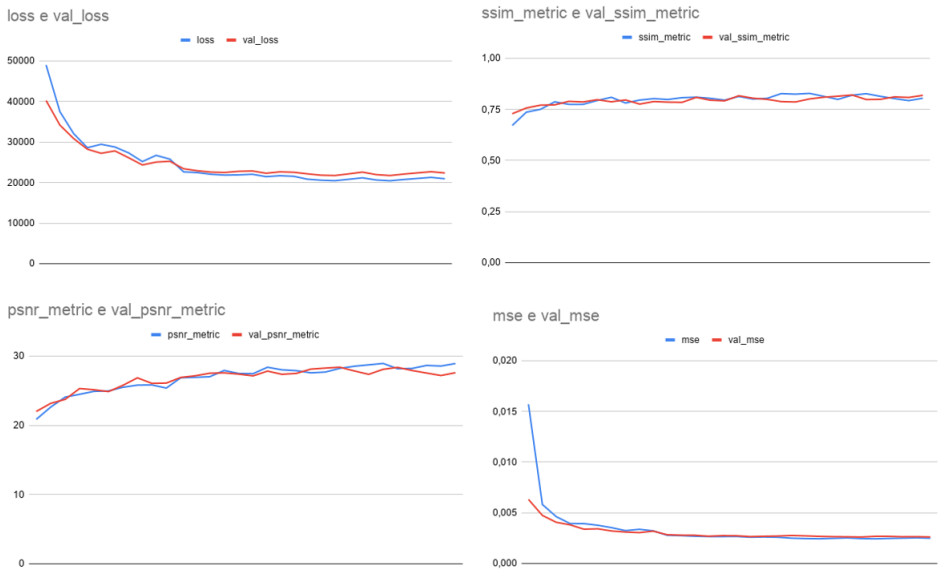

Figure 21: Loss, SSIM, PSNR and MSE trends over epochs for the model KAIST_NET.

D Results of ATROUS_NET on GOPRO

Our model has not been trained on the GoPro dataset [1], which is a standard benchmark for evaluation of deblurring algorithms.
It consists of 3214 pairs of blurred and sharpimages with the same image resolution as in the REDS dataset (2103 pairs for training and 1111 pairs for testing). Like the REDS dataset, the GoPro dataset is a synthetically created dataset for single image as well as video deblurring.
Therefore the results we report are those we obtained simply by testing ATROUS_NET with the weights obtained from training on the REDS dataset. A more correct and consistent analysis could be made by going to train ATROUS_NET on GoPro. Although the network is not trained on GoPro, we can say that it still manages to slightly remove the noise present in the images.
This makes us conclude ATROUS_NET is also characterized by a good generalization capacity and gives us good hope that we could also get good results on GoPro after training on it.

Performance on test set		
	ATROUS_NET	Baseline
SSIM	0.8240	0.7895
PSNR	27.18	26.59
MSE $[10^{-3}]$	3.18	3.56

Table 5: Results obtained by our model on the test set of **GoPro**

Figure 22: Comparison between the crops of images taken from the GoPro test set and reconstructed through ATROUS_NET.