Application of Deep Learning on Image Deblurring

Master in Artificial Intelligence - University of Bologna

A.Y. 2019-2020

Authors:

- Mattia Orlandi
- Giacomo Pinardi

Abstract

Image deblurring is a challenging task in computer vision: it consists in enhancing the quality of images by removing blur artifacts.

In the following work we investigate deep neural models to tackle both Gaussian blur and motion blur; to do so, a modified version of CIFAR-10 dataset and REDS dataset are used for the training. In particular, all our models perform blind deblurring, namely they try to restore the sharp image without any cue on the blur kernel.

We experiment with both Convolutional Autoencoders and Wasserstein Generative Adversarial Networks with Gradient Penalty. Moreover, we also rely on some techniques such as residual learning, multi-scale approach and patch-based analysis.

In order to evaluate our models' performances, we employ Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR) metrics, which are good indicators to assess similarity between the original sharp image and the reconstructed one.

The experiments show that, both visually and in terms of metrics, our best performing model (a convolutional autoencoder with symmetric skip connections) obtains remarkable results in both datasets. Finally, in the appendix we show that it's able to reconstruct also smaller details in the unseen GOPRO dataset.

Contents

1 Introduction

Image deblurring is one of the most common tasks in image restoration: it consists in removing blurring artifacts in order to enhance the image's overall quality. It can be seen as a particular case of denoising.

In this project we investigated two different kinds of blur:

- Gaussian blur
- Motion blur

The former can arise when the subject of a photo is not on focus, and can be artificially obtained by applying a Gaussian kernel on a sharp image, where the kernel size affects the amount of blurring artifacts:

$$\mathbf{B} = \mathbf{K} * \mathbf{S} \tag{1.1}$$

where $\mathbf{B}$ and $\mathbf{S}$ are the blurred and sharp images, respectively, $\mathbf{K}$ is the Gaussian kernel, and $*$ denotes the convolution operator.

The latter is a very common type of image degradation, since it naturally arises when taking a photo of moving objects, or when the camera itself is moving:

$$\mathbf{B} = \mathbf{KS} + \mathbf{n} \tag{1.2}$$

where $\mathbf{B}$, $\mathbf{S}$, $\mathbf{n}$ are respectively the blurred image, sharp image, and noise, whereas $\mathbf{K}$ is a sparse matrix whose rows contain a local blur kernel; it can be obtained by averaging a sequence of images taken rapidly one after the other [1].

The aim of this project is to use Deep Neural Networks (DNNs) to restore images affected by both gaussian and motion blur. To do so, we relied on Convolutional Neural Networks (CNNs) and the latest research papers on such topic.

1.1 Related works

In the past, several papers have been presented to address deblurring using CNNs.

As far as motion deblurring is concerned, Sun et al. [2] first tried to estimate the motion kernel, and then motion blur was removed using a non-uniform deblurring model (non-blind deblurring).

On the other hand, with blind deblurring it is not required to gather information beforehand about the motion kernel, but instead the latent sharp image is estimated directly from the blurred one [3].

Blind deblurring can be achieved using very deep CNNs combined with residual learning, as in Mao et al. [4]. They proposed an architecture in which an end-to-end mapping from blurred to sharp images is learnt using multiple convolutions and deconvolutions

with skip-layer connections. Convolutions are useful in order to extract important features and remove corruptions, whereas deconvolutions upsample the feature maps and recover details. Such architecture can also be used to address Gaussian blur and other types of image degradation.

Recently, Generative Adversarial Networks (GANs) [5] have been exploited in this kind of task, leading to remarkable results: unlike in the original GAN paper, to address blind deblurring the generator is fed with the blurred image (instead of random noise) and outputs the restored image, which is then analysed by the discriminator.

Nah et al. [1] proposed a GAN with a multi-scale CNN generator in which deblurring at finer scale is aided by coarser scale features; moreover, the content loss employed is also multi-scale, as it takes into account the Mean Squared Error (MSE) between the generated and the sharp images at all scales.

Recently, an improved version of GANs, namely Wasserstein GAN (WGAN) (and in particular WGAN with Gradient Penalty, WGAN-GP), has been proposed [6] (more details about GANs and WGANs can be found in the appendix A). Such WGAN-GP model has been exploited by Kupyn et al. [7] to perform motion deblurring.

2 | Datasets

In the following sections we will describe the two datasets employed in our study, one to address Gaussian blur and the other to address motion blur.

2.1 CIFAR-10

The CIFAR-10 dataset [8] contains 60'000 color images, each one having a resolution of 32x32 pixels. The images in the collection are divided into classes, but as we are not performing a classification task we simply ignore them.

2.1.1 Dataset creation

For each dataset image (referred to as *sharp* image) it was required to construct the associated *blurred* version. To do so we applied gaussian blurring with random standard deviation between 0 and 3.

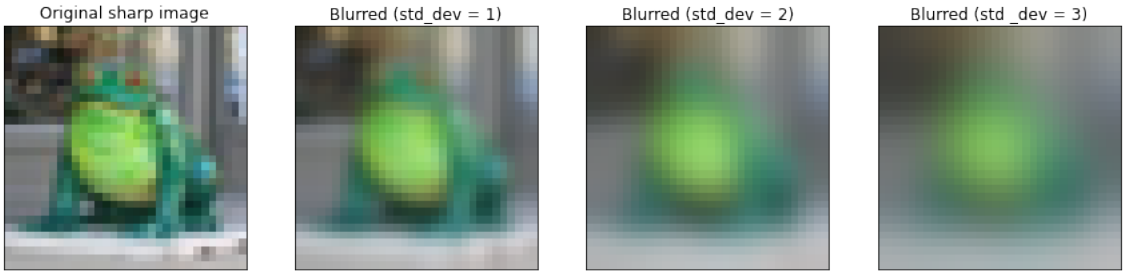

Figure 2.1: Example of the same CIFAR-10 image blurred with increasing standard deviation.

We apply some basic data augmentation in order to reduce overfitting and increase the amount of training examples analysed by the networks [9]. In particular horizontal and/or vertical flip is applied randomly to some of the images belonging to the training set.

2.2 REDS

The REalistic and Dynamic Scenes dataset [10] contains 27'000 color images already divided into train, validation and test set (respectively 21'000, 3'000 and 3'000). Each image has a resolution of 1270x720. This dataset was used in two Real Image Denoising Challenges: NTIRE 2019 and NTIRE 2020.

To construct the dataset the following procedure was used: a high speed camera (240fps) recorded videos of different scenes, then each blurred image was produced computing the average of 7-13 successive frames and the middle frame in the sequence was used as the corresponding sharp image (more details are available in the original paper [1]).

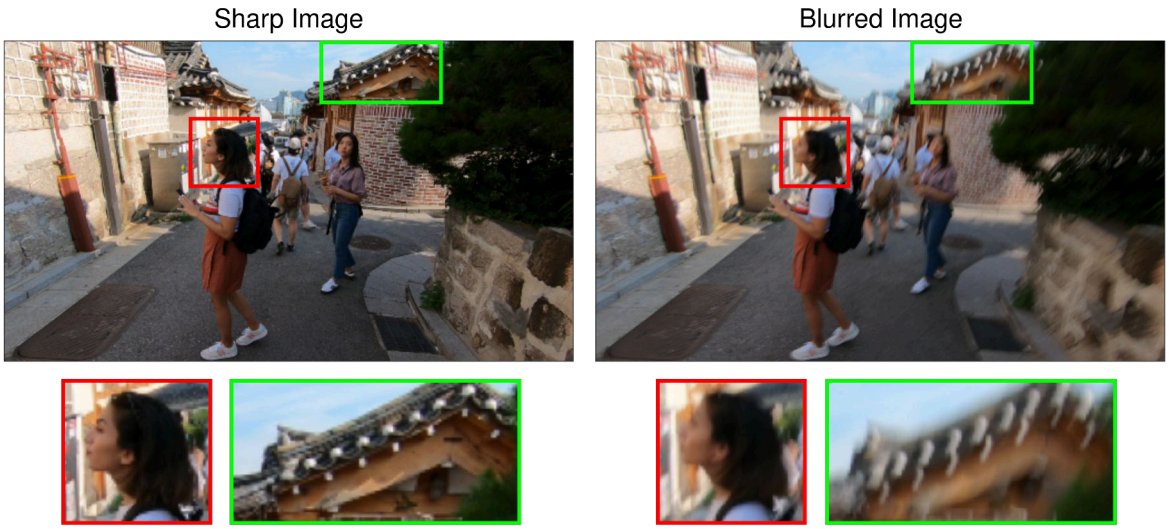

Figure 2.2: An example of a sharp and blurred images belonging to the REDS dataset.

2.2.1 Dataset manipulation

Due to the very limited computational resources available it was necessary to greatly reduce the resolution of the images: from 1280x720 pixels to 512x288, so only 16% of the original pixels are kept.

In addition, because of the memory limitation we encountered, each image is split into 12 patches forming a 4x3 grid. The resolution of each patch is 128x96.

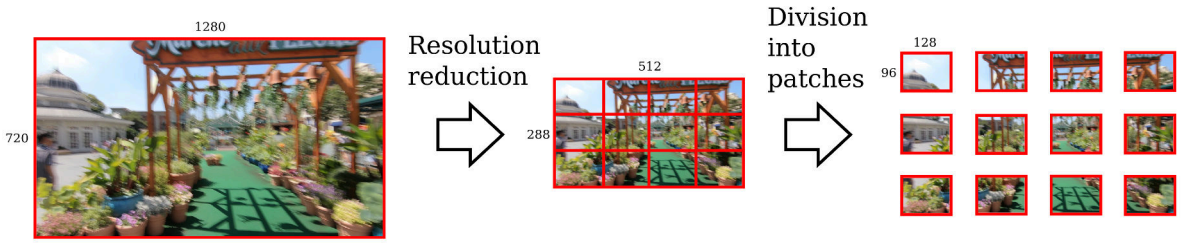

Figure 2.3: Initial process of dataset manipulation.

REDS dataset is augmented: similar to CIFAR-10 horizontal and/or vertical flip is applied randomly to some of the images belonging to the training set.

3 | Convolutional Autoencoders

Our first approach relied on convolutional autoencoders (CAEs):

- the encoder part of the network should extract features from the blurred image received in input and remove blurry artifacts, producing a down-sampled latent image representation;
- the decoder part of the network should up-sample and reconstruct the image starting from its latent representation.

In order to improve learning and avoid vanishing gradients we employed residual connections [11], which allow the flow of information from the initial layer to the last layer of the residual block using an alternative path, different from the standard one.

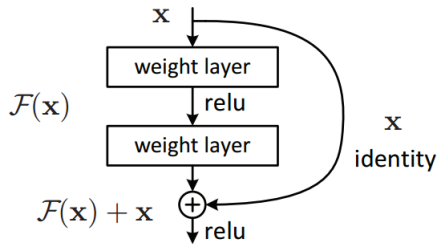

Figure 3.1: Example of residual block from [12]

3.1 Models

In the following sections we will describe the main CAE architectures that we tested in our work.

3.1.1 ResNet16

The first model developed was ResNet16: its architecture comprises an encoder and a decoder, each made of 4 residual blocks.

The structure of the residual blocks of the encoder is defined as in figure 3.2. As it can be seen, the input image is fed into a convolutional layer with kernel 3×3, $\mathbf{f}$ features and stride $\mathbf{s}\times\mathbf{s}$ (where $\mathbf{f}$, $\mathbf{s}$ can vary), followed by a batch normalization layer (which according to Ioffe et al. should speed-up training by reducing the internal covariance shift [13]), and ReLU activation function. Then, it is fed into a second series of convolution, batch normalization and ReLU identical to the first one, apart from the stride which is set to 1×1. Finally, the output of the ReLU is summed with the output of the skip

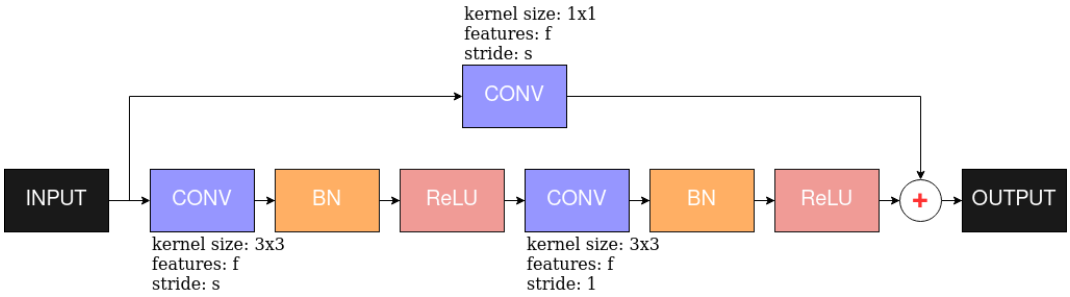

Figure 3.2: ResBlock of the encoder.

connection: to make the number of features and the size of the image match, we inserted a convolutional layer with kernel 1×1, **f** features and stride **s**×**s** (where **f**, **s** are the same as the ones in the main branch of the block).

The structure of the ResBlock of the decoder (in figure 3.3) is very similar to the one of the encoder: the only differences is that we employed transposed convolution (in order to upscale the image), and that the strides of such blocks were fixed to 1 (for the first layer) and 2 (for the second layer and for the transposed convolution on the skip connection).

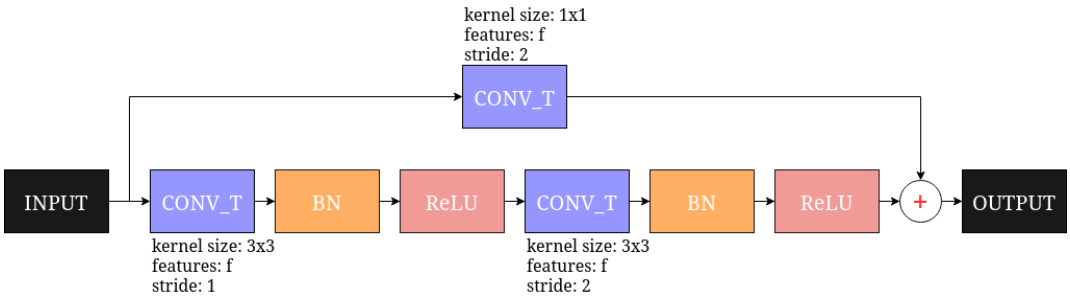

Figure 3.3: ResBlock of the decoder.

The architecture of the whole model is depicted in figure 3.4.

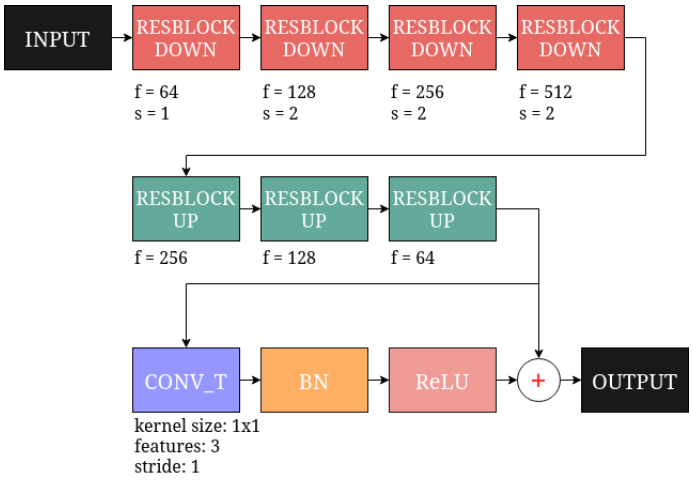

Figure 3.4: ResNet16's architecture.

As it can be seen, the input image goes through the 4 ResBlock of the encoder, which progressively shrinks its size and increases its features:

$$32 \times 32 \times 3 \rightarrow 32 \times 32 \times 64 \rightarrow 16 \times 16 \times 128 \rightarrow 8 \times 8 \times 256 \rightarrow 4 \times 4 \times 512$$

Therefore, the latent representation of the image has size $4 \times 4 \times 512$; such latent image is fed into the 3 ResBlocks of the decoder, which in turn progressively increases its size and reduces its features:

$$4 \times 4 \times 512 \to 8 \times 8 \times 256 \to 16 \times 16 \times 128 \to 32 \times 32 \times 64$$

Finally, the image goes through a ResBlock with only one sequence of convolution (with kernel 1×1, 3 features and stride 1×1), batch normalization and ReLU activation (instead of two), such that the output is $32 \times 32 \times 3$ again.

3.1.2 UNet16

The UNet16 architecture was studied and implemented starting from the work of Ronneberger et al. [14]. Their approach was particularly successful in processing biomedical images for both speed and the small number of training data required.

The architecture is made of two major parts:

- the left side of the U, called the contracting path, is characterized by convolutional layers together with max pooling operations for down-sampling
- the right side of the U, called the expansive path, applies transposed convolution (in order to up-sample) and common convolutions

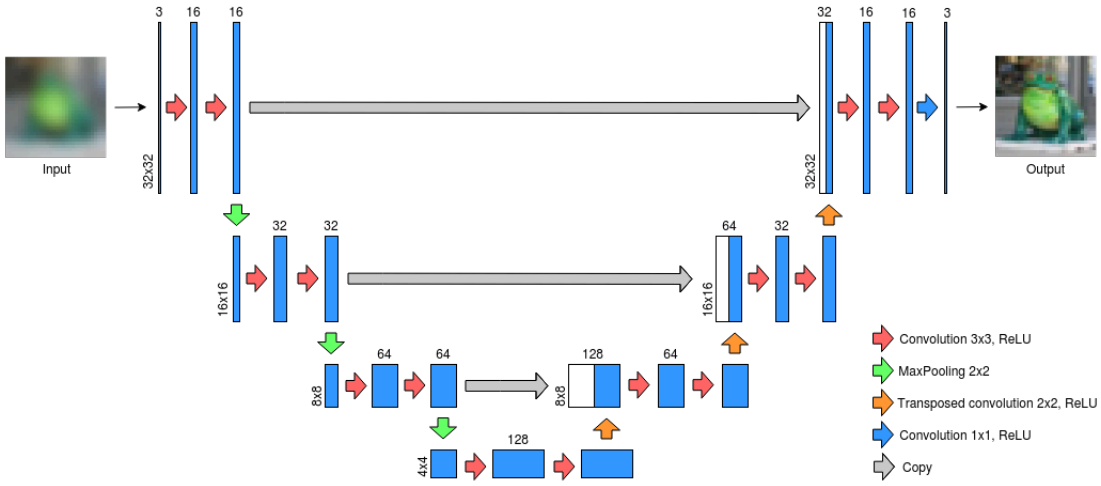

Figure 3.5: UNet16 architecture.

The architecture we propose is described below. In the contracting path there are successive application of 3×3 convolutions followed by a ReLU; the max pooling operation is used with window size 2×2 and stride 2 in order to down-sample the image (i.e. from 32×32 to 16×16 in the first application). Before applying the max pooling operation, the feature map is copied and concatenated with the corresponding feature map in the expansive path. After each down-sampling step, the number of channels is doubled and in the bottom part we end up with 128 feature channels. In the expansive path, 2×2 transposed convolutions with stride 2 up-sample the feature map, that is then concatenated with the corresponding feature map coming from the contracting path. After the concatenation the channel number is divided in half and 3×3 convolution followed by ReLU is applied twice. Finally, in order to get a $32 \times 32 \times 3$ image as output a 1×1 convolution is applied to the $32 \times 32 \times 16$ feature map.

3.1.3 REDNet30

The REDNet30 (Residual Encoder-Decoder Network) architecture we propose (in figure 3.6) is a slightly modified version of the one proposed by Mao et al. [4]: it consists in a series of a convolution with kernel 3×3, 64 features and stride 1×1, followed by an exponential linear unit (ELU), and a batch normalization layer; such block of three layers is replicated 30 times, and symmetric skip connections are added every two blocks, in order to avoid vanishing gradients.

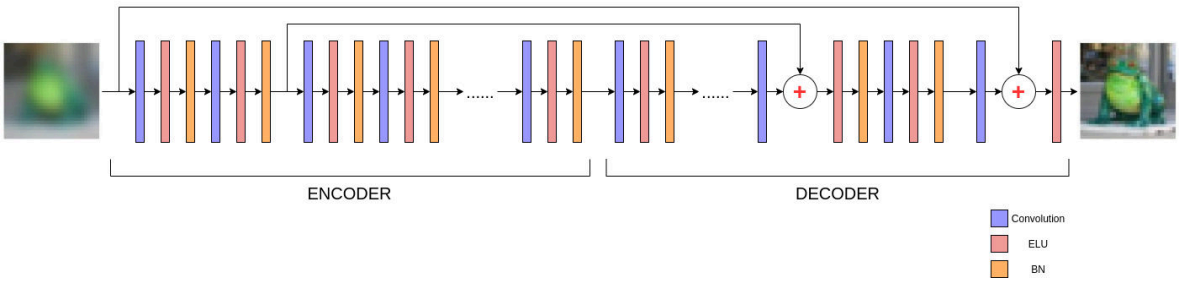

Figure 3.6: Architecture of REDNet30.

We chose to rely on ELU instead of ReLU because the latter suffers from the dying ReLU problem: if most inputs of the ReLU are negative then during backpropagation its gradient will be zero, and therefore it won't be able to adjust its inputs. On the other hand, when most ELU's inputs are negative, it will still be able to update them (however, it will saturate for very large negative values) [15]. We could have faced the dying ReLU problem with other activation functions like Leaky ReLU (LReLU) or Parametric ReLU (PReLU), but empirically we experienced better results with ELUs.

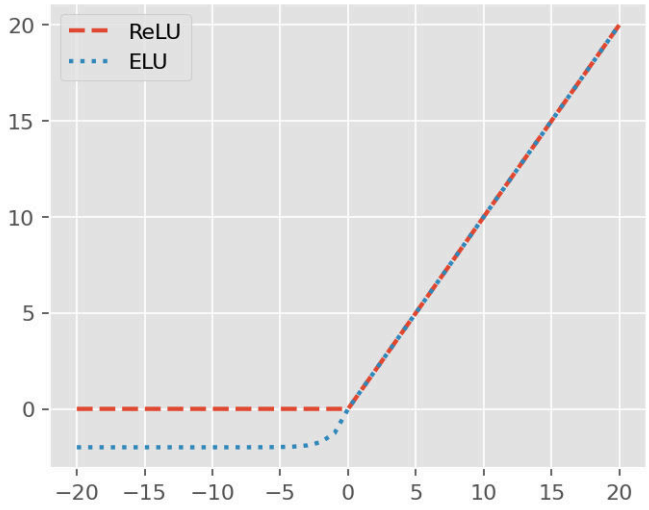

Figure 3.7: Comparison of ReLU and ELU plots.

3.2 Experimental results on CIFAR-10

3.2.1 Training phase

Each model was trained using a batch size of 32 images for each of the 8 replicas of the TPU, so that 256 images were processed at once (cf. Tools). The training lasted for a total of 120 epochs.

We tested several loss functions:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Log Hyperbolic Cosine (LogCosh)

We experienced better results when using LogCosh: this can be explained by the fact that such loss function behaves as MSE for small values while it behaves as MAE for large values; thus, it is more robust to outliers and it is also differentiable in zero [16]. For the above reasons, we decided to use LogCosh as the loss function.

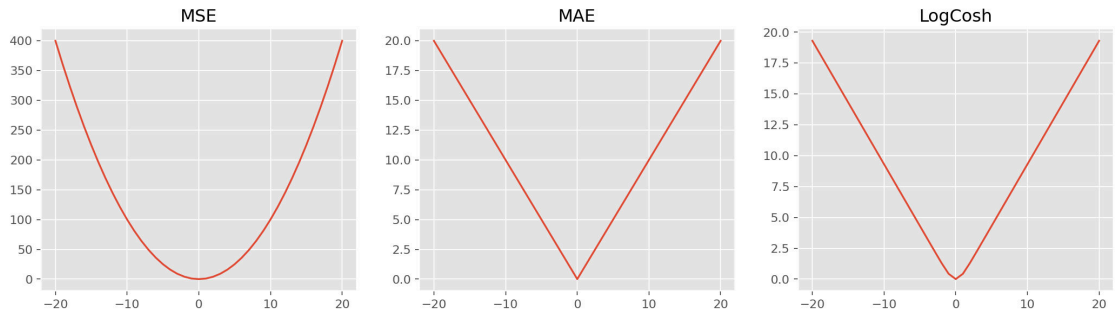

Figure 3.8: Visual comparison between MSE, MAE and LogCosh.

The Adam optimizer [17] was chosen because of its features:

- simple to use;
- computationally efficient;
- little memory requirements;
- adaptive learning rate;
- remarkable empirical results.

We left unchanged Adam's hyper-parameters apart from the learning rate, which we set to 0.001.

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

- Peak Signal-to-Noise Ratio (PSNR): ratio between the maximum possible power for signal and the power of the corrupting noise, in decibel scale; it can be defined in terms of MSE.

- Structural Similarity Index Measure (SSIM): perception-based model that relies on structural information to evaluate image degradation [18].

In figure 3.9 and 3.10 we show, respectively, the loss trends of the three proposed models and the comparison for SSIM and PSNR computed on the validation set. As it can be seen, REDNet30 performed far better than the others in both metrics.

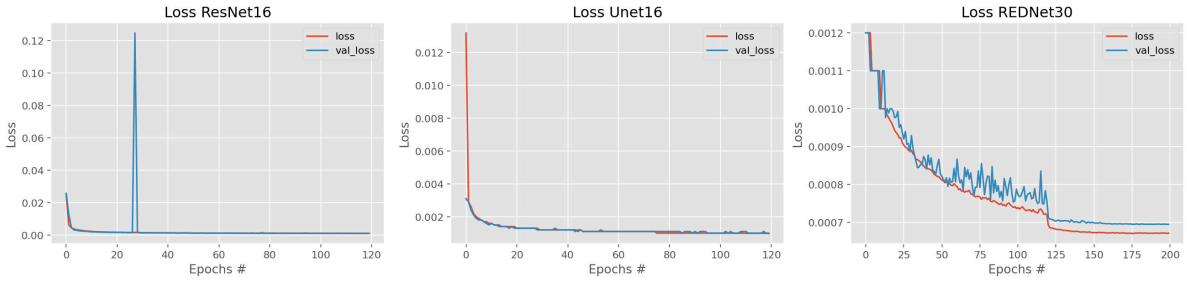

Figure 3.9: Loss trends over epochs.

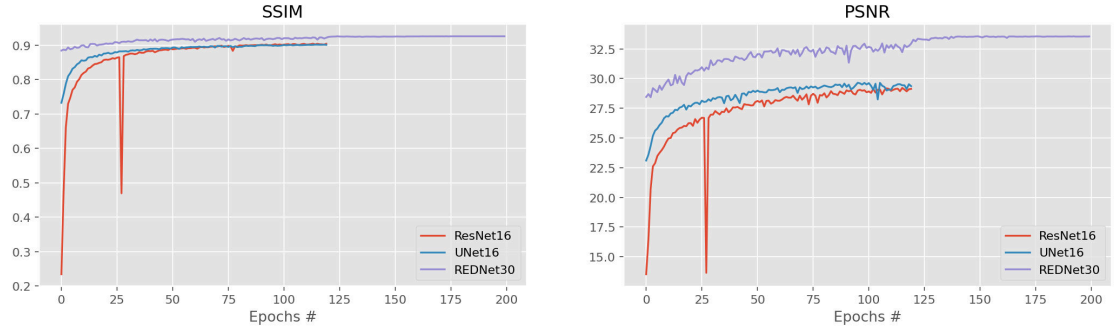

Figure 3.10: SSIM and PSNR comparison between the three proposed models on the validation set.

Because of its superior performance, we additionally trained REDNet30 until 200 epochs: in the last 80 epochs, we exponentially decayed the learning rate (starting from 10^{-4}) by a factor of 0.7 every 5 epochs. This led to a further improvement in both PSNR and SSIM metrics.

3.2.2 Performance evaluation

As far as the performance on test set is concerned, as we can see in table 3.1, ResNet16 and UNet16 are similar in terms of SSIM and PSNR, whereas REDNet30 performs clearly better. All models show a significant increase in all metrics with respect to the baseline (namely, the metrics calculated between the sharp image and the blurred, non-restored one).

In figure 3.11 we show the visual comparison between the reconstructed images: as it can be seen, all the networks are able to reconstruct large-scale features; however, smaller details are reconstructed differently by the different models. Finally, despite REDNet30 having a higher SSIM and PSNR, from a purely visual inspection it seems to perform equally to ResNet16 and UNet16.

Performance on test set				
Model	ResNet16	UNet16	**REDNet30**	Baseline
Loss (LogCosh) [10^{-4}]	9.794	10.44	**6.966**	34.48
SSIM	0.9034	0.9001	**0.9258**	0.7135
PSNR	29.09	29.32	**33.52**	24.67
MSE [10^{-3}]	1.965	2.095	**1.398**	6.317
MAE [10^{-2}]	2.888	2.898	**2.165**	4.517

Table 3.1: Results obtained on the test set by our CAE models in CIFAR-10.

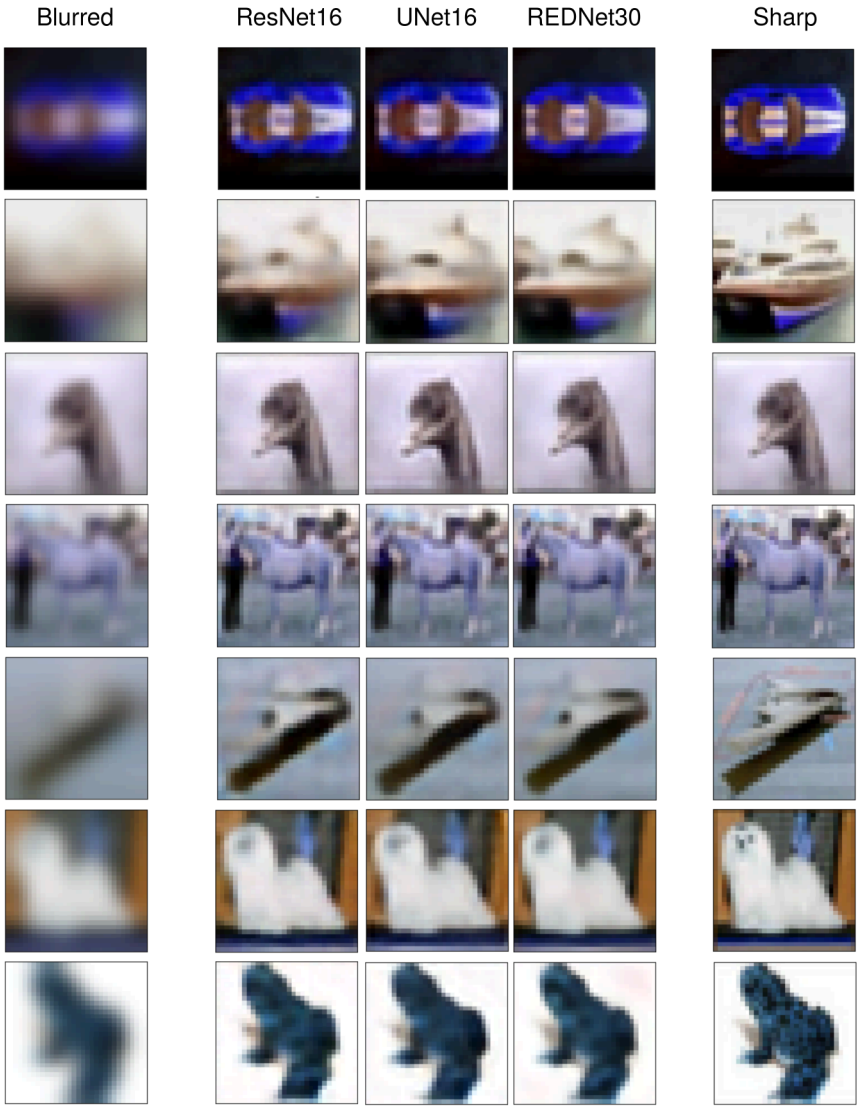

Figure 3.11: Visual comparison between reconstructed images from CIFAR-10's test set.

3.3 Experimental results on REDS

3.3.1 Training phase

All the three models were trained using a batch size of 32 patches, so 256 patches were processed at once by the 8 replicas of the TPU (cf. Tools). The training lasted for a total of 100 epochs.

The loss function employed was, again, the Log Hyperbolic Cosine (LogCosh). The Adam optimizer's learning rate was set to 0.001.

As it was done for CIFAR-10 the two metrics used to assess the model performance are PSNR and SSIM.

In figure 3.12 we show the training and validation loss during the training phase of the networks, whereas in figure 3.13 it's possible to see SSIM and PSNR metrics trends over the epochs during the training.

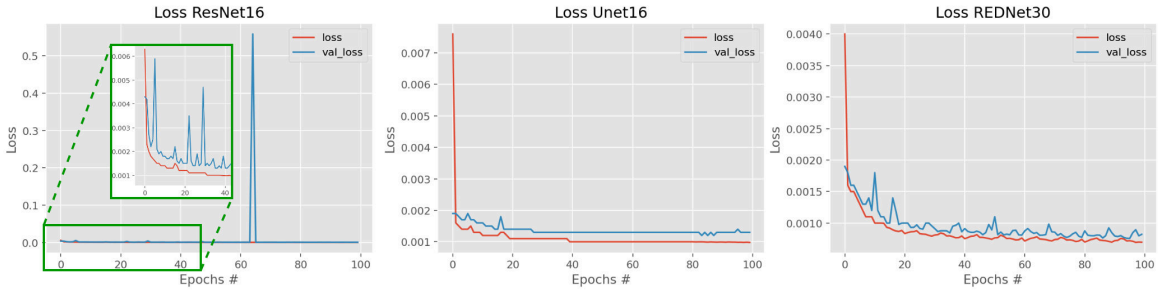

Figure 3.12: Loss trends over epochs for the models on REDS dataset.

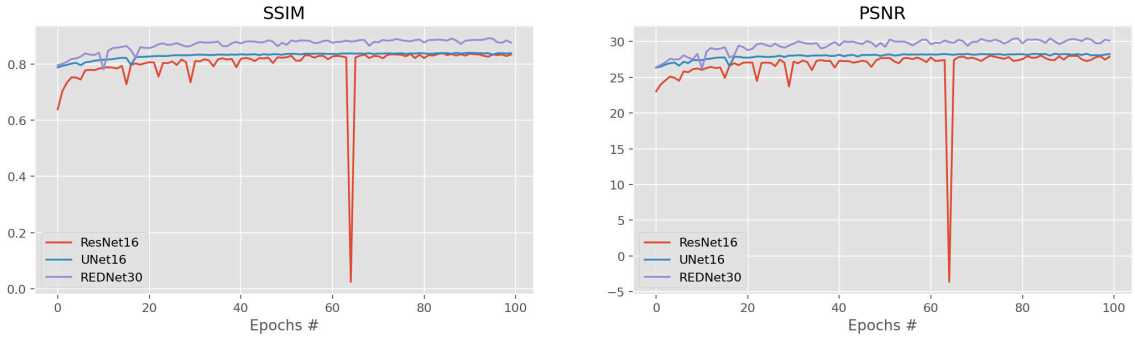

Figure 3.13: Metrics trends over epochs for the models on REDS dataset.

In terms of both loss and metrics trends, REDNet30 again performs better than ResNet16 and UNet16. In particular, ResNet16 is more unstable than the others: in fact, its training is characterized by some spikes, one of which is huge.

3.3.2 Performance evaluation

The results of the evaluation on the test set are shown in the table 3.2. Differently from the CIFAR-10 case, ResNet16 seems to perform a little worse with respect to UNet16 according to the metrics. REDNet30 on the other hand achieves the best performance, like before.

All models' performances are better w.r.t. the baseline apart from MAE metric: in fact, both ResNet16 and UNet16 show a worse MAE than the baseline. This can be explained by the fact that MAE is a more robust indicator than MSE.

Performance on test set				
Model	ResNet16	UNet16	**REDNet30**	Baseline
Loss (LogCosh) [10^{-4}]	8.669	8.763	**5.793**	19.54
SSIM	0.8554	0.8646	**0.8976**	0.8237
PSNR	29.49	30.41	**31.98**	28.80
MSE [10^{-3}]	1.742	1.761	**1.163**	2.594
MAE [10^{-2}]	2.536	2.312	**1.884**	2.118

Table 3.2: Results obtained on the test set by our CAE models in REDS.

In figures 3.14, 3.15 and 3.16 we show the reconstruction of some images taken from the test set. The regions of interest are highlighted in order to ease visual comparison. As it can be seen, REDNet30 is the best in reducing motion blur artifacts (like those on the roof in figure 3.14, or the truck's wheel in figure 3.16), but still it's not able to fully restore smaller details (like the man's face in figure 3.15 or the truck's plate in figure 3.16).

We further investigated the capabilities of REDNet30 by testing it also on the GOPRO dataset. Such results are shown in the appendix B.

Figure 3.14: Visual comparison between reconstructed images from REDS' test set (1/3).

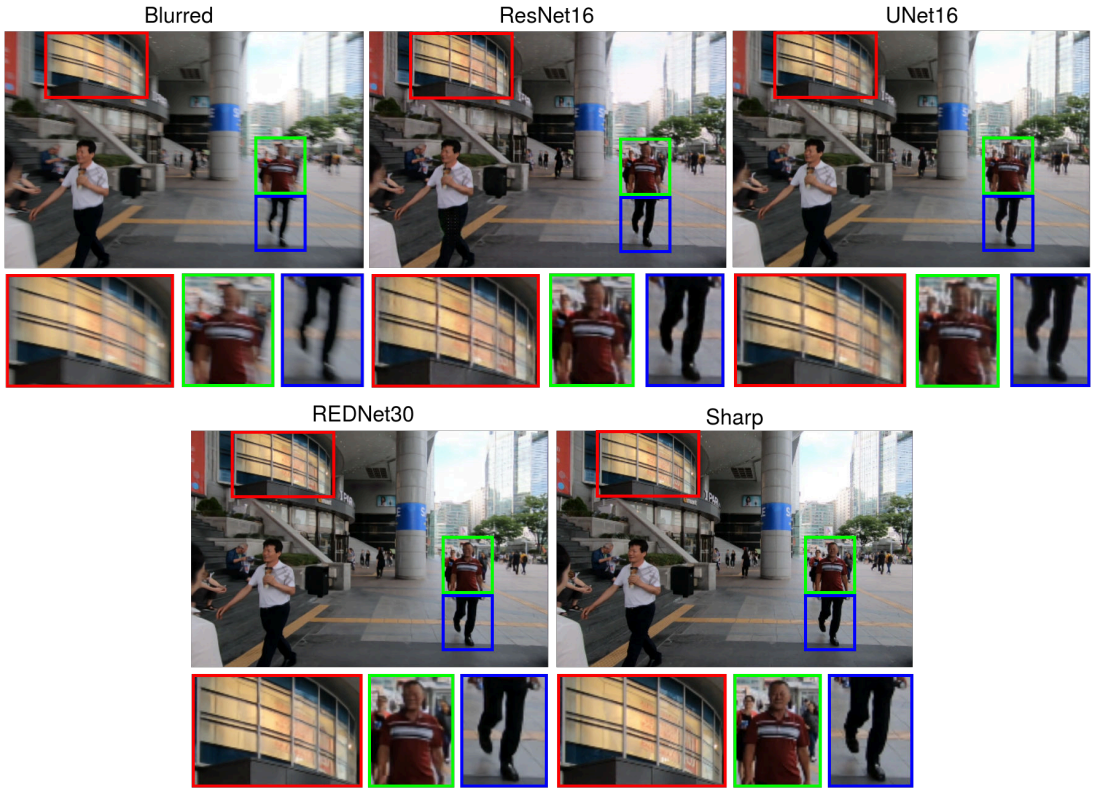

Figure 3.15: Visual comparison between reconstructed images from REDS' test set (2/3).

Figure 3.16: Visual comparison between reconstructed images from REDS' test set (3/3).

4 | Generative Adversarial Networks

Another approach that we decided to test consisted in using GANs to face the deblurring task. In particular, we started from the work by Nah et al. [1]: his team developed a multi-scale CNN generator, which takes in input a pyramid of 3 blurred images at different scales, such that each scale branch uses information from the previous one. We were also inspired by the work of Kupyn et al. [7], who relied on WGAN-GP [19] in order to obtain a more stable GAN for deblurring.

Therefore, in our first presented model we use a multi-scale generator together with a WGAN-GP discriminator (called *critic*). We also describe a second model, called RED-Net30WGAN, whose generator makes use of the previously presented REDNet30.

4.1 Models

In the following sections we will describe the main WGAN-GP architectures that we tested in our work.

4.1.1 MSDeblurWGAN

As we are using a multi-scale structure, the original blurred image is manipulated in order to obtain also the 16×16 and 8×8 versions of it. The three images are fed into the network, which consists of three branches, one per image of the pyramid: in each branch, an initial convolution with kernel size 5×5 and 64 features is applied, followed by 19 ResBlocks; then, a final convolution with kernel size 5×5, 3 features and *tanh* activation is performed. In the coarsest and middle branches, the output image is also up-scaled (using a transposed convolution with kernel size 5×5, 64 features and stride 2×2) and concatenated with the input image of the upper level. Back-propagation is performed at each level.

In the ResBlock of the generator (shown in Figure 4.2) the input is fed into a Convolutional layer (64 features and kernel size 5×5), then Batch Normalization is applied with ReLU activation. Another Convolution and Batch Normalization layer are then used, and finally the residual connection from the input is added to the result of the second Batch Normalization operation. Normally, an additional ReLU activation is applied after the residual connection, but Nah et al. [1] showed that such ReLU lead to worse results; therefore, we did not include it. Nah et al. also removed batch normalization layers because they used mini-batches of 2 samples; in our case, we trained our model with larger batches, and so the batch normalization layer is needed.

As stated before, the task of the critic is to compute an approximated version of the Wasserstein distance between the sharp images and the restored ones. In particular, in our model the critic analyses only the outputs of the finer branch (32×32 resolution). As in the work by Kupyn et al. [7], our critic is implemented as a PatchGAN discriminator: such

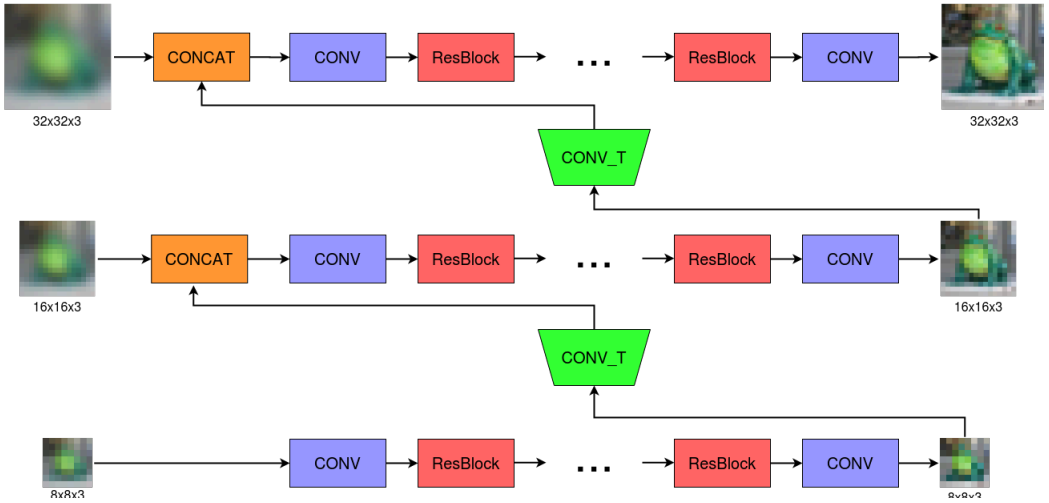

Figure 4.1: The generator of the GAN.

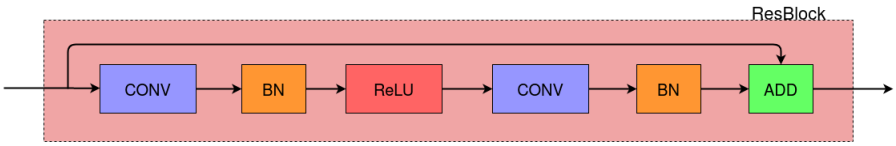

Figure 4.2: The ResBlock of the generator.

architecture was developed by Isola et al. [20]. The original implementation of PatchGAN presented a sigmoid as last activation function: however, since in WGANs the critic must output a score rather than a probability, a linear activation is preferred. Moreover, as suggested by Gulrajani et al. [19], batch normalization in the critic is replaced by layer normalization [21].

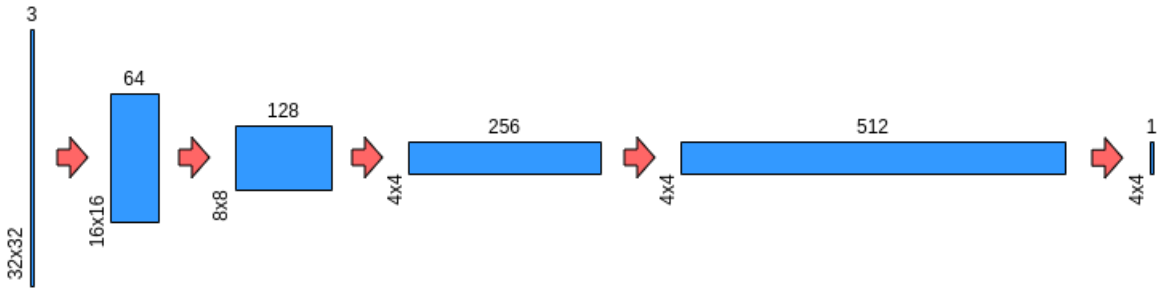

Figure 4.3: Architecture of the PatchGAN: each layer consists in a convolution with kernel 4×4 and stride 2×2 (apart from the second to last one, which has stride 1×1), followed by a layer normalization and a Leaky ReLU.

4.1.2 REDNet30WGAN

REDNet30WGAN model uses architectures already presented in this paper:

- the generator is simply the REDNet30 CNN described before, with a *tanh* activation function in the last layer instead of an ELU;

- the critic employed is the same used for the MSDeblurWGAN.

4.2 Experimental results on CIFAR-10

4.2.1 Training phase

Before the actual training phase, we normalized the images in the dataset in the range $[-1, 1]$ (and therefore, as stated before, we used a *tanh* activation function in the generator) [22].

In order to enforce 1-Lipschitz continuity, a penalty on the gradient norm for random samples $\mathbf{x_{int}}$ is applied on the critic's loss.

$$L = \mathbb{E}_{\mathbf{x_{pred}}}[C(\mathbf{x_{pred}})] - \mathbb{E}_{\mathbf{x_s}}[C(\mathbf{x_s})] + \lambda \cdot \mathbb{E}_{\mathbf{x_{int}}}[(\|\nabla C_w(\mathbf{x_{int}})\|_2 - 1)^2] \tag{4.1}$$

The training proceeds as in the following pseudo-code algorithm (the Adam's parameters were chosen according to the paper: $\alpha = 2 \cdot 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.9$).

Data: $\mathbb{D}$ dataset, ep number of epochs, n_{critic} number of critic updates, λ weight of the gradient penalty, μ weight of the adversarial loss

for $n \leftarrow 1$ **to** ep **do**
  Extract batch **bat** of m pair of sharp and blurred images from $\mathbb{D}$;
  for $k \leftarrow 1$ **to** n_{critic} **do**
    for $i \leftarrow 1$ **to** m **do**
      Extract sharp image $\mathbf{x_s}$ and blurred image $\mathbf{x_b}$ from **bat**;
      Sample random number $\epsilon \sim U[0, 1]$;
      $\mathbf{x_{pred}} \leftarrow G_\theta(\mathbf{x_b})$;
      $\mathbf{x_{int}} \leftarrow \epsilon \mathbf{x_s} + (1 - \epsilon)\mathbf{x_{pred}}$;
      $L_C^{(i)} \leftarrow C_w(\mathbf{x_{pred}}) - C_w(\mathbf{x_s}) + \lambda \cdot (\|\nabla C_w(\mathbf{x_{int}})\|_2 - 1)^2$;
    end
    $w \leftarrow$Adam$(\nabla \frac{1}{m} \sum_{i=1}^{m} L_C^{(i)}, w)$;
  end
  for $i \leftarrow 1$ **to** m **do**
    Extract sharp image $\mathbf{x_s}$ and blurred image $\mathbf{x_b}$ from **bat**;
    $\mathbf{x_{pred}} \leftarrow G_\theta(\mathbf{x_b})$;
    $L_G^{(i)} \leftarrow$ content_loss$(\mathbf{x_s}, \mathbf{x_{pred}}) - \mu C_w(\mathbf{x_{pred}})$;
  end
  $\theta \leftarrow$ Adam$(\nabla \frac{1}{m} \sum_{i=1}^{m} L_G^{(i)}, \theta)$;
end

Algorithm 1: Training phase of our model: we used $n_{\text{critic}} = 5$, $\lambda = 10$

As it can be seen, the loss function is composed by two terms, the content loss and the adversarial loss:

$$L_G = L_{\text{content}}(\mathbf{x_s}, \mathbf{x_{pred}}) + 10^{-4} \cdot L_{\text{adv}}(\mathbf{x_{pred}}) \tag{4.2}$$

The adversarial loss L_{adv} is simply equal to the (negated) score of the critic, whereas the content loss L_{content} is computed using LogCosh. In the special case of MSDeblurWGAN

a multi-scale version of LogCosh is needed, as shown in equation 4.3:

$$L_{\text{content}}(\mathbf{x_s}, \mathbf{x_{pred}}) = \frac{1}{2K} \sum_{k=1}^{K} \frac{\text{LogCosh}(\mathbf{x_s}, \mathbf{x_{pred}})}{c_k w_k h_k} \tag{4.3}$$

where K is the number of scales (in our case 3), c_k is the channel number, w_k is the width and h_k is the height of the image.

The loss trends of MSDeblurWGAN and REDNet30WGAN models are shown, respectively, in figures 4.4 and 4.5. MSDeblurWGAN was trained for 150 epochs, even if after the 100th epoch its generator's loss was approximately flat. On the other hand, REDNet30WGAN showed a more promising loss trend: hence, we continued the training until the 350th epoch.

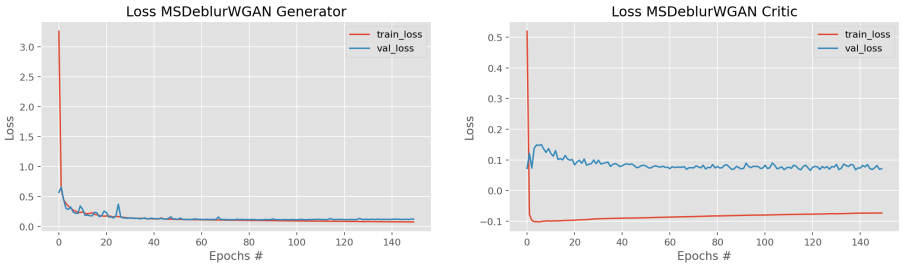

Figure 4.4: Loss trends over epochs for the model MSDeblurWGAN.

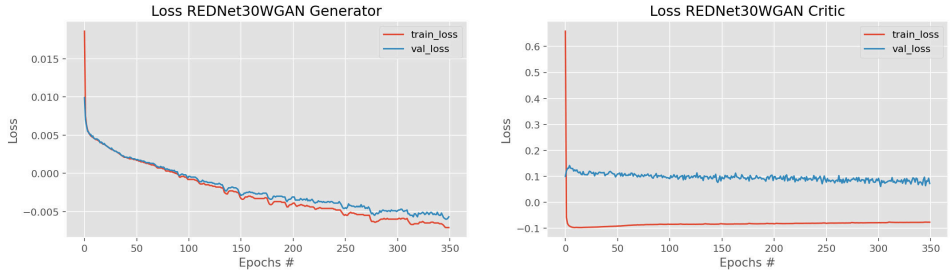

Figure 4.5: Loss trends over epochs for the model REDNet30WGAN.

In figure 4.6 we show the comparison for SSIM and PSNR computed on the validation set using the models described above: MSDeblurWGAN and REDNet30WGAN.

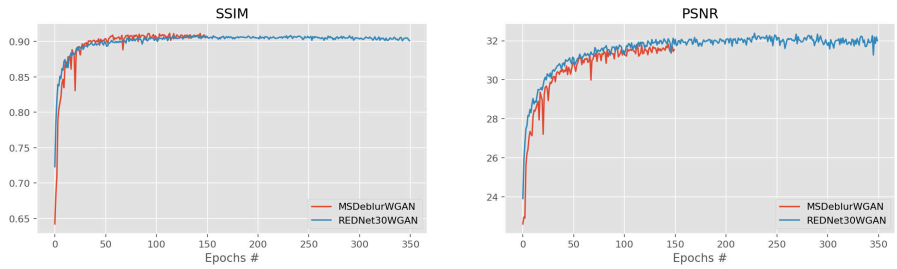

Figure 4.6: SSIM and PSNR comparison between the two GAN proposed on the validation set.

4.2.2 Performance evaluation

As it can be seen from table 4.1, both MSDeblurWGAN and REDNet30WGAN achieve very similar performances: however, standard REDNet30 still performs better than both WGANs, showing that in this particular task CAE approach seems more effective. As for CAEs, WGANs show a dramatic improvement w.r.t. the baseline.

Performance on test set				
	MSDeblurWGAN	REDNet30WGAN	**REDNet30**	Baseline
Loss (Adv.)	0.1254	-0.0057	—	—
SSIM	0.9054	0.9012	**0.9258**	0.7135
PSNR	31.49	31.98	**33.52**	24.67
MSE [10^{-3}]	1.734	1.900	**1.398**	6.317
MAE [10^{-2}]	2.471	2.482	**2.165**	4.517

Table 4.1: Results obtained on the test set by our WGAN models in CIFAR-10.

By a visual inspection (figure 4.7), we can observe that, while CAEs (like REDNet30) restores the image's general structure, WGANs tend to restore smaller details, sometimes in a creative but inaccurate way (e.g. the car's stripe in the first row or the ship in the second row). Nevertheless, the WGANs' reconstruction capabilities are overall comparable to those of CAEs.

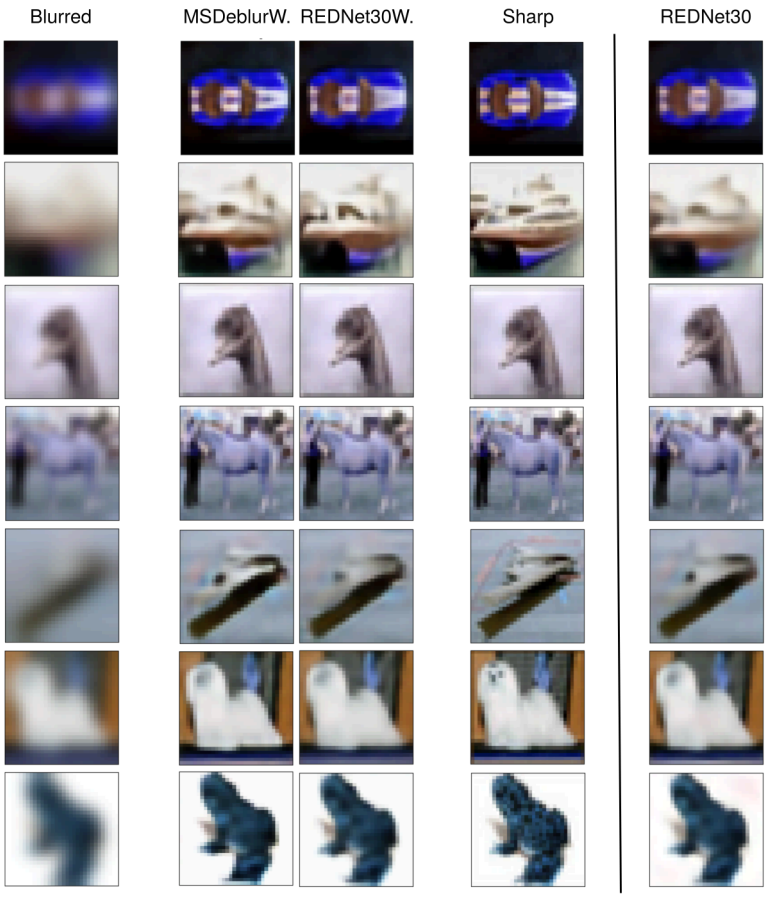

Figure 4.7: Visual comparison between reconstructed images from CIFAR-10's test set.

4.3 Experimental results on REDS

Since MSDeblurWGAN makes use of the multi-scale approach, it is particularly recommended for higher resolution images [1]: therefore, we decided to prioritize the training of such model, and to avoid the use of REDNet30WGAN with this dataset.

4.3.1 Training phase

The training phase proceeded as for CIFAR-10: the only differences are that, in this case:

- we employed a patch-based approach;
- we further reduced the resolution from 512×244 to 256×144;
- we reduced the number of ResBlocks from 19 to 5.

The reason behind such choices is that, otherwise, the training would have been intractable even using a TPU ($\sim$ 2 hours/epoch).

Unfortunately, as it can be seen in figures 3.12 and 4.9, such simplifications heavily impacted on the model's performance on the validation set: in fact, it is not able to generalize well. Moreover, after the 60th epoch SSIM and PSNR trends started oscillating.

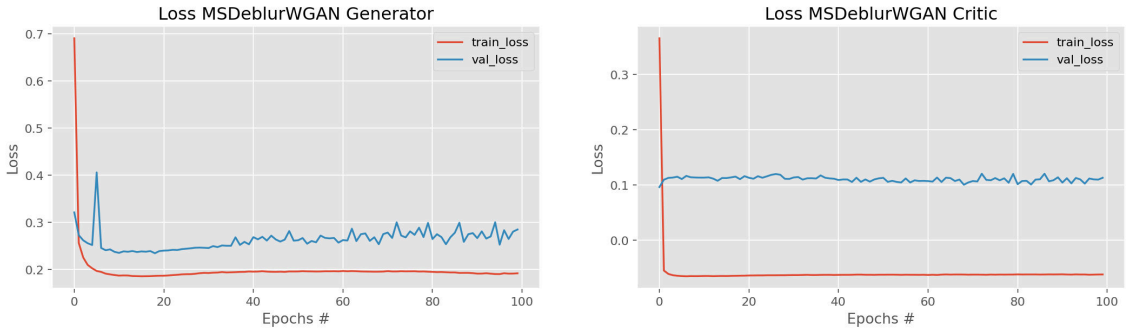

Figure 4.8: Loss trends over epochs for the model MSDeblurWGAN.

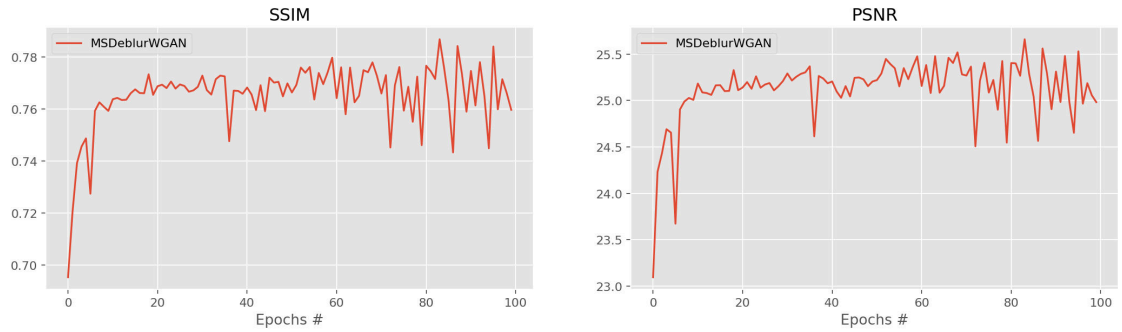

Figure 4.9: SSIM and PSNR metrics for the MSDeblurWGAN model.

4.3.2 Performance evaluation

The poor performance of the network on the validation set is reflected even during the evaluation phase: as it can be seen, SSIM, PSNR and MAE metrics are slightly worse for MSDeblurWGAN w.r.t. the baseline, whereas MSE is far better. This could be explained by the fact that the network is trying to minimize the adversarial loss, which comprises largely LogCosh: thus, the model manages to slightly adjusts pixel values in the restored image, without actually improving the other metrics.

On the other hand, from a purely visual comparison (figure 4.10), we can observe that there is a small but visible improvement in the processed images: in fact, blur artifacts are reduced (e.g. the roof in the first image, or the man's legs in the second image).

Performance on test set			
Model	MSDeblurWGAN	**REDNet30**	Baseline
Loss (Adv.)	0.2195	—	—
SSIM	0.8031	**0.8976**	0.8359
PSNR	27.11	**31.98**	27.68
MSE $[10^{-3}]$	1.729	**1.163**	3.120
MAE $[10^{-2}]$	2.495	**1.884**	2.390

Table 4.2: Results obtained on the test set by our WGAN models in REDS.

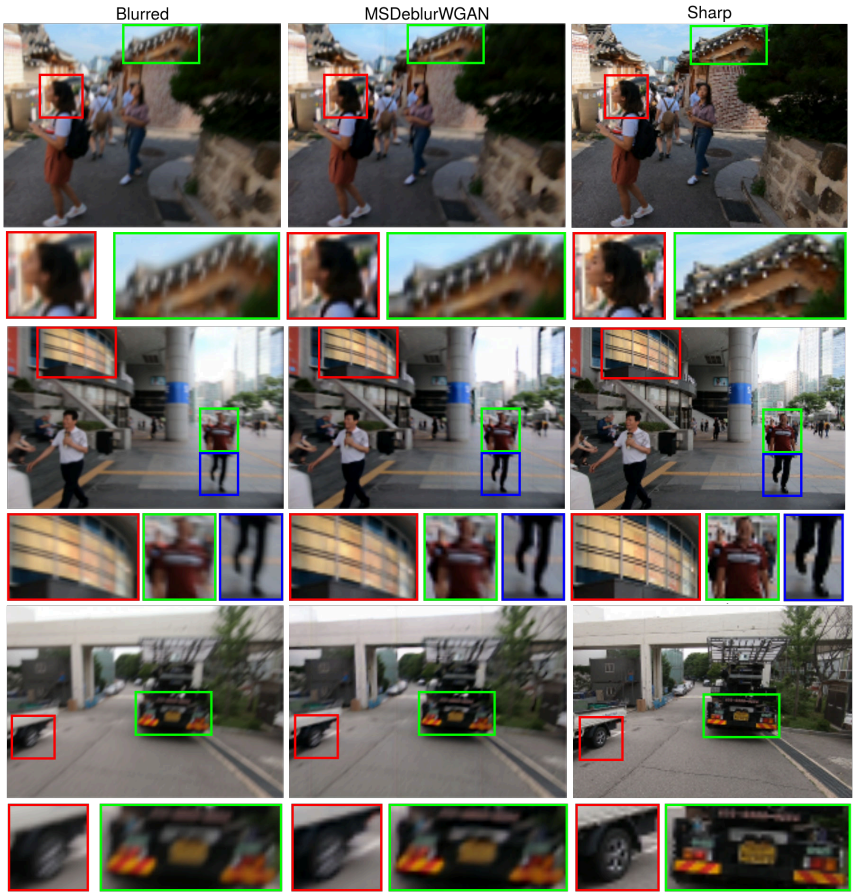

Figure 4.10: Visual comparison between reconstructed images from REDS' test set.

5 | Tools

We implemented our models in Python using TensorFlow 2.3 library. All the experiments were performed on Google Colab using the TPU backend.

Tensor Processing Units (TPUs) are application-specific integrated circuits (ASICs) developed by Google and used to accelerate machine learning workloads [23].

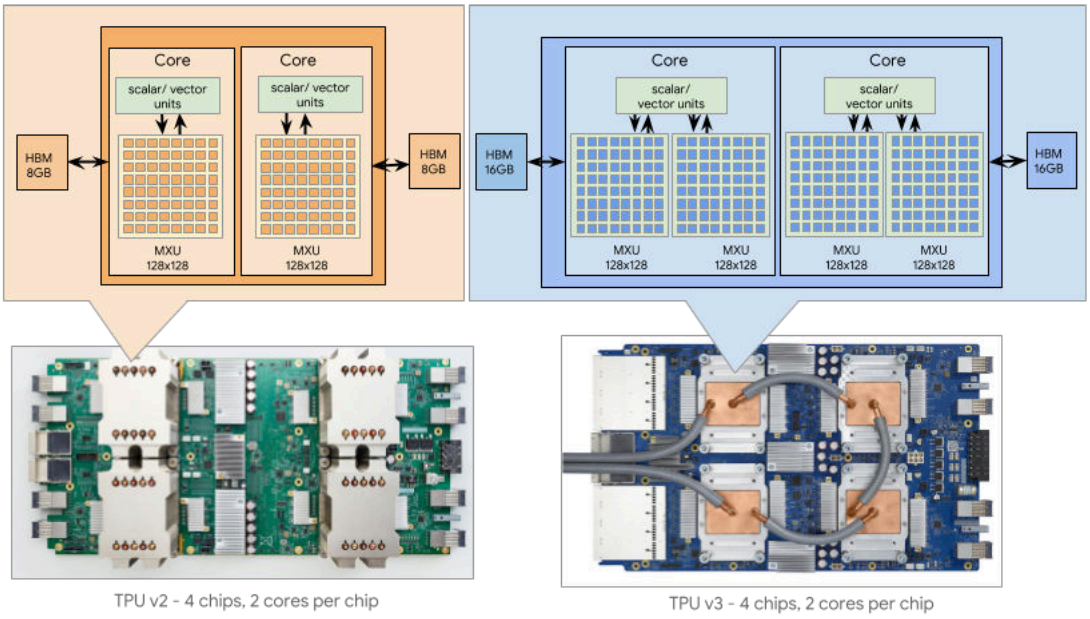

Figure 5.1: TPU's architecture from [24]

TPUs work in a distributed environment: training is performed in parallel on 8 different replicas, each working on a batch of data; the loss and metrics values computed by each replica are then reduced to a single value. As a consequence, TPUs does not support training using datasets stored locally: therefore, we uploaded the two datasets on Google Cloud Storage (GCS) using *tfrecords* format.

In order to reduce the memory occupation and further speed up training, we employed the **Brain Floating Point** (`BFLOAT16`) format for tensors: such format uses half of the bits of single-precision floating points `FP32` (16 instead of 32), but ensures the same dynamic range (from $\sim 1 \cdot 10^{-38}$ to $\sim 3 \cdot 10^{38}$) at the cost of slightly reduced precision; as a comparison, standard half-precision floating points `FP16` have a much smaller dynamic range (from $\sim 5.9 \cdot 10^{-8}$ to $\sim 6.5 \cdot 10^{4}$) ([25]).

Moreover, in order to avoid loading the whole datasets into memory (especially REDS, which was ~70 GB), we relied on a feature provided by TensorFlow, which was the class `tf.data.Dataset`. In particular, we implemented an input pipeline with the following

characteristics:

1. Initial parsing of tfrecords file to obtain the image in png format.
2. Images modifications when necessary (i.e. normalization, resolution reduction, division into patches).
3. Training and validation split.
4. Caching.
5. Data augmentation.
6. Shuffling.
7. Division into batches.

As a side note, we also tried using Microsoft Azure's virtual machines instead of Colab, but the speed-up obtained with Colab TPUs and the price of Azure subscription determined our choice towards Google Colab.

All the code is available on GitHub.

6 | Conclusions

With our experiments we investigated the capabilities of Convolutional Autoencoders and Generative Adversarial Networks in image deblurring tasks. As far as CAEs are concerned, we've seen that REDNet30 performs far better than ResNet16 and UNet16 both for CIFAR-10 and for REDS: this can be due to the sparse latent representation of REDNet30 (opposed to the dense one of the other models), combined with symmetric residual connections; also the greater depth of REDNet30 may have played a role in its better performances.

Apart from PSNR and SSIM metrics, the visual difference between the images reconstructed by ResNet16, UNet16 and REDNet30 is not so significant, at least for CIFAR-10.

For what concerns the WGANs that we proposed, they outperform ResNet16 and UNet16 in PSNR metric (although they're similar in SSIM metric) but they do not reach REDNet30's performance. Moreover, the time required to train WGANs is obviously much greater because of the GAN paradigm, that requires managing two competing networks.

An observation which can be made about MSDeblurWGAN is that a multi-scale analysis of low-resolution images (as in CIFAR-10) does not improve the network's performance: in fact, REDNet30WGAN leads to similar results even if it is based on a single-scale analysis.

As far as REDS is concerned, the poor performance of MSDeblurWGAN can be explained by the heavy simplifications made: in fact, we reduced the number of ResBlocks from 19 to 5 in order to make training faster.

After all, REDNet30 turned out to be the winning formula, combining very good results (both visually and from the metrics point of view) with a tractable training time.

A | GAN and WGAN

As previously hinted, GANs consists of two models:

- a discriminator D which, given a sample x, estimates the probability of such sample coming from the real dataset (distribution P_r) or from the generated one (distribution P_g);
- a generator G which, given a latent variable $z \sim P_z$, outputs synthetic samples $x \sim P_g$ as similar as possible to those coming from a real dataset.

D is optimized to tell fake images from the real ones, while G is trained to capture the real data distribution, generate realistic data and thus trick D; in other words, D and G play a minimax game [5]:

$$\begin{aligned} &\min_G \max_D \mathbb{E}_{x \sim P_r}[\log D(x)] + \mathbb{E}_{z \sim P_z}[\log(1 - D(G(z)))] \\ =&\min_G \max_D \mathbb{E}_{x \sim P_r}[\log D(x)] + \mathbb{E}_{x \sim P_g}[\log(1 - D(x))] \end{aligned} \tag{A.1}$$

Goodfellow et al. [5] showed that the optimal discriminator D^* is:

$$D^*(x) = \frac{P_r(x)}{P_r(x) + P_g(x)} \tag{A.2}$$

Moreover, when the discriminator is optimal the GAN's loss becomes:

$$\begin{aligned} L(G, D^*) &= -\log(4) + D_{KL}\left(P_r \middle\| \frac{P_r + P_g}{2}\right) + D_{KL}\left(P_g \middle\| \frac{P_r + P_g}{2}\right) \\ &= -\log(4) + 2 \cdot D_{JS}(P_r \| P_g) \end{aligned} \tag{A.3}$$

where D_{KL} and D_{JS} are, respectively, the Kullback-Leibler (KL) and Jensen-Shannon (JS) divergences:

$$D_{KL}(P\|Q) = \int_x P(x) \log \frac{P(x)}{Q(x)} dx \tag{A.4}$$

$$D_{JS}(P\|Q) = \frac{1}{2} D_{KL}(P \middle\| \frac{P+Q}{2}) + \frac{1}{2} D_{KL}(Q \middle\| \frac{P+Q}{2}) \tag{A.5}$$

However, Jensen-Shannon divergence is not robust when the discriminator performs better than the generator: in fact, Arjovsky et al. [26] showed that, when the discriminator is optimal but the generator is not, the gradient of the loss function diminishes and the learning stops. Therefore, they proposed to use Wasserstein's distance (also known as Earth-Mover's distance), which is not affected by the problems of D_{KL} and D_{JS}, and

which can be informally described as the minimum cost required to convert a distribution into another:

$$W(P,Q) = \inf_{\gamma \sim \Gamma(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] \tag{A.6}$$

where $\Gamma(P,Q)$ is the set of all possible joint probability distributions between P and Q. A single joint distribution $\gamma \sim \Gamma(P,Q)$ represents the "transport plan" which modifies the original distributions.

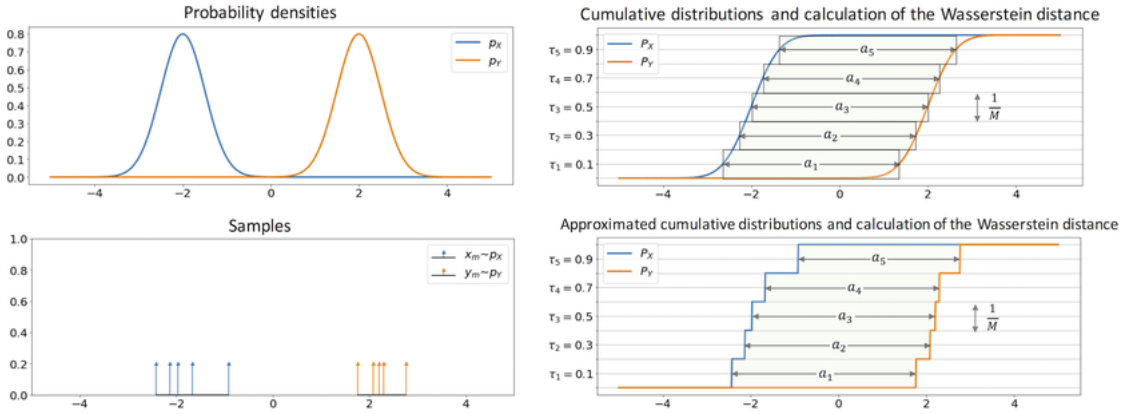

Figure A.1: Example of Wasserstein's distance between two distributions from [27].

Computing the Wasserstein distance as it is requires searching exhaustively for all possible joint distributions $\gamma \sim \Gamma(P,Q)$, and hence it is intractable; therefore, Kantorovich-Rubinstein duality [26] is exploited in order to simplify the problem, obtaining the following formulation:

$$W(P,Q) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim P}[f(x)] - \mathbb{E}_{y\sim Q}[f(y)] \tag{A.7}$$

where f is required to be K-Lipschitz continuous, which means that its gradient must have norm at most K everywhere.

In light of these considerations, Arjovsky et al. [6] conceived the Wasserstein GAN (WGAN): let's suppose the function f comes from a family of 1-Lipschitz continuous functions $\{f_w\}_{w\in\mathcal{W}}$, parameterized by w; the discriminator in WGANs (called *critic*) is trained to learn w in order to compute the optimal f_w, whereas the generator tries to minimize the Wasserstein's distance.

As stated before, the function computed by the critic must be 1-Lipschitz continuous. To enforce such constraint weight clipping was initially employed; however, such method was neither elegant nor perfect, thus Gulrajani et al. [19] developed an improved version which relies on gradient penalty (WGAN-GP) instead of weight clipping.

B | REDNet30 on GOPRO

To further investigate the capabilities of our best model, REDNet30 (which was trained on REDS dataset), we tested it on some images from the GOPRO dataset. In particular, we compared its performances with Nah et al.'s model, DeepDeblur (figure B.1). As we can see, despite the fact that REDNet30 is not able to reach DeepDeblur's visual reconstruction performances, it still manages to reconstruct smaller details (like the man's legs in the first image, or the man's face in the third one). Moreover, it is fair to notice that REDNet30 was not trained on GOPRO, whereas DeepDeblur was.

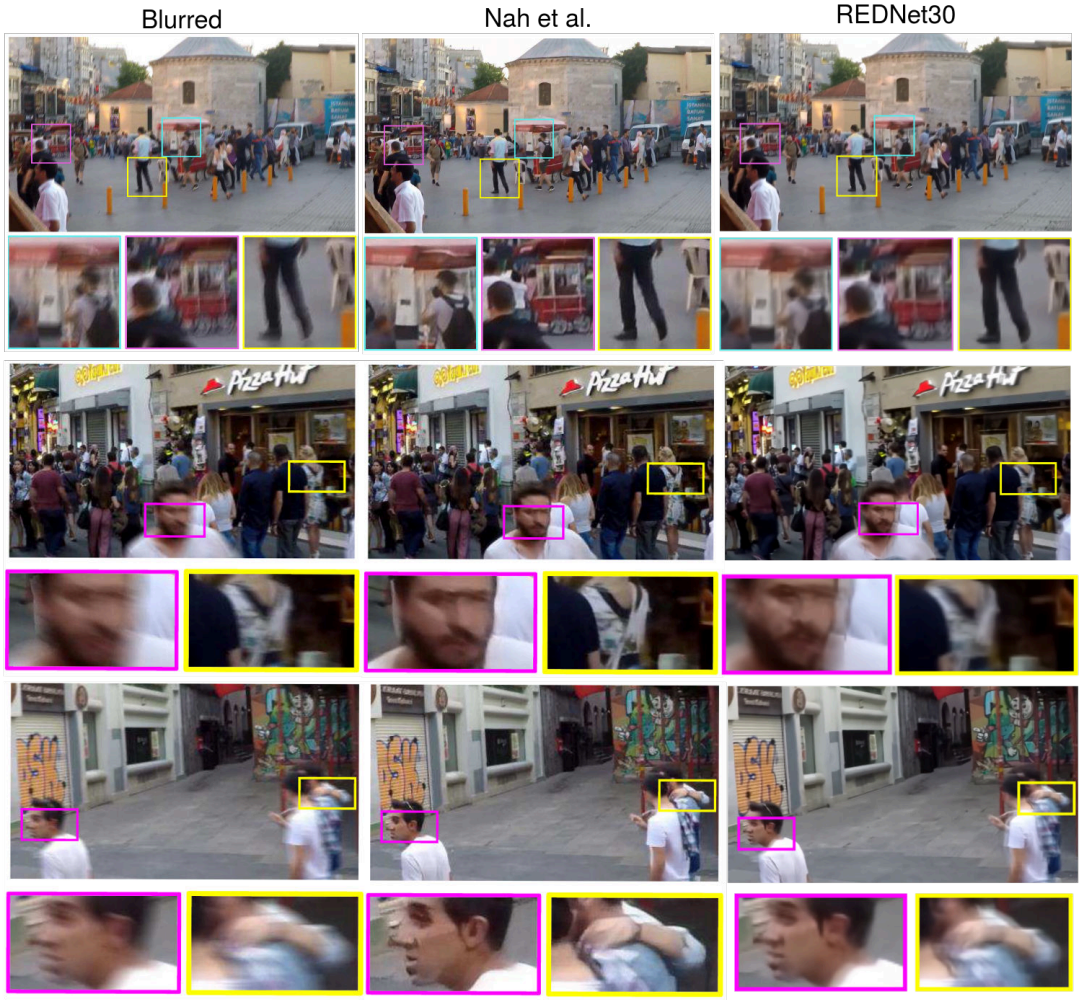

Figure B.1: Visual comparison between reconstructed images from GOPRO dataset. The first column contains the blurred images, the second column is obtained using the work of Nah et al. [1], the third column contains the reconstructed images obtained using our network REDNet30.

Bibliography

[1] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. *Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring*. 2018. arXiv: 1612.02177.

[2] Jian Sun et al. *Learning a Convolutional Neural Network for Non-uniform Motion Blur Removal*. 2015. arXiv: 1503.00593.

[3] Siddhant Sahu, Manoj Kumar Lenka, and Pankaj Kumar Sa. *Blind Deblurring using Deep Learning: A Survey*. 2019. arXiv: 1907.10128.

[4] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. *Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections*. 2016. arXiv: 1606.08921.

[5] Ian J. Goodfellow et al. *Generative Adversarial Nets*. 2014. arXiv: 1406.2661.

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875.

[7] Orest Kupyn et al. *DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks*. 2018. arXiv: 1711.07064.

[8] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. 2009.

[9] Sebastien C. Wong, Adam Gatt, and Victor Stamatescuand Mark D. McDonnell. *Understanding data augmentation for classification: when to warp?* 2016. arXiv: 1609.08764.

[10] Seungjun Nah et al. *NTIRE 2019 Challenge on Video Deblurring and Super-Resolution: Dataset and Study*. 2019.

[11] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385.

[12] Sabyasachi Sahoo. *Residual blocks - Building blocks of ResNet*. Accessed: 2020-08-28. URL: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec.

[13] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167.

[14] Olaf Ronneberger et al. *U-Net: Convolutional Networks for BiomedicalImage Segmentation*. 2015. arXiv: 1505.04597.

[15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. arXiv: 1511.07289.

[16] G. Ayzel et al. "All convolutional neural networks for radar-based precipitation nowcasting". In: *Procedia Computer Science* (2018).

[17] Diederik P. Kingma and Jimmy Lei Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980.

[18] Zhou Wang et al. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *IEEE Transactions on Image Processing* (2004).

[19] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028.

[20] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004.

[21] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450.

[22] James Briggs. *Beating the GAN Game*. Accessed: 2020-09-03. URL: https://towardsdatascience.com/beating-the-gan-game-afbcce0a20be.

[23] *Cloud Tensor Processing Units (TPUs)*. Accessed: 2020-08-27. URL: https://cloud.google.com/tpu/docs/tpus.

[24] *System Architecture*. Accessed: 2020-08-27. URL: https://cloud.google.com/tpu/docs/system-architecture.

[25] Grigory Sapunov. *FP64, FP32, FP16, BFLOAT16, TF32, and other members of the ZOO*. Accessed: 2020-10-08. URL: https://medium.com/@moocaholic/fp64-fp32-fp16-bfloat16-tf32-and-other-members-of-the-zoo-a1ca7897d407.

[26] Martin Arjovsky and Léon Bottou. *Towards principled methods for training Generative Adversarial Networks*. 2017. arXiv: 1701.04862.

[27] Soheil Kolouri et al. *Sliced-Wasserstein Autoencoder: An Embarrassingly Simple Generative Model*. 2018. arXiv: 1804.01947.