Alma Mater Studiorum
University of Bologna

Artificial Intelligence - Deep Learning

Report for Deblurring Project

Dicosola Alessandro [Matr. 935563]

1 Introduction

The aim of this project is to survey deep neural network architectures that allow to remove blurring artifacts from images.
In order to do so the following datasets will be used:

- CIFAR10 [1] dataset, with progressive gaussian blur using a random standard deviation between 0 and 3.
- REDS[2] dataset , with motion blur.

2 Hardware

The hardware used is my personal computer:

1. CPU: i7-8750H@2.20GHz
2. GPU: Nvidia GTX 1060 (6 GB)

3 Autoencoders

In image-to-image tasks and in particular in *image restoration tasks* most of of the implementations studied follow the **Encoder-Decoder** architecture: the *encoder* tries to capture the latent space that represent the input image and the *decoder* use the learned information for reconstructing it.

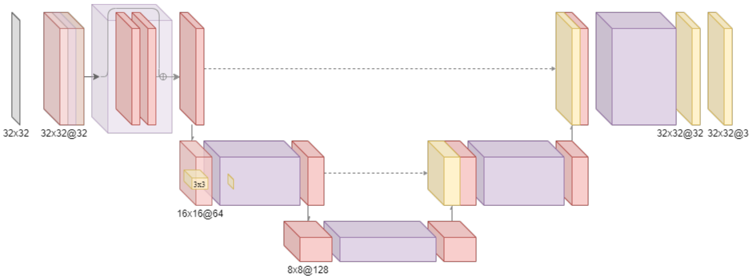
In particular, for image deblurring, the encoder learn a latent space where the blur artifact is not present (given a sharp image) and the decoder is able to output a sharper image with more fine details than the blurred one.

The following models are implemented:

- ResUNet - a simple UNet with ResBlock [3]
- EDDenseNet - an autoencoder network using DenseBlock[4]
- CAESSC - a direct implementation of Convolutional Auto Encoders with Symmetric Skip Connections[5]
- SRNDeblur - a direct implementation of SRN-DeblurNet[6]

3.1 ResUNet

The model is the following:



The U-Net [7] architecture is largely used in image segmentation due to its structure: the descending path encodes the image information and the ascending path decode the information localized at particular position inside the ground truth image.

Descending and ascending paths are connected togheter at each level in order to share the information and achive a more precise classification: this is done *concatenating* the features of each output in the descending path to the upsampled input of the ascending path.

This pattern follows the encoder-decoder architecture which can be used for image-to-image processing, although some changes are necessary:

- With respect to the original paper the downsampling and upsampling is done respectively using *Conv2D*(red) and *Conv2DTranspose*(yellow) in order to "learn" the transformation instead of using MaxPooling and UpSampling, leading to a loss of information.
- Moreover the *ResBlock* (as defined in [3]) is used in order to increment the amount of information propagated at each level in both paths. [1]

Since this network was used with *CIFAR10* only (allowing a large batch), the *Conv2D* layer is composed by:

1. Conv2D using a filter $(3, 3)$.
2. BatchNormalization [8]. It's necessary since the batch size is huge thus it allow to maintain the distribution of the inputs on each layer for each batch the same, allowing a faster training using a higher learning rate and avoiding exploding/vanising gradients.
3. Activation.

The *ResBlock* is composed by two Conv2D layers and the output is the concatenation of the last output with the input.

The networks implemented are: **ResUNet1** with 1 ResBlock and **ResUNet3** with 3 ResBlock.

[1] In the CAESSC network will be discussed the advantages of residual connections

3.2 Training

The network was trained with Adam[9] with default parameters using an early stopping equals to 7 for avoiding overfitting and MSE as loss function. SRNDeblur1 and SRNDeblur3 converged, respectively, within 40 and 50 epochs.

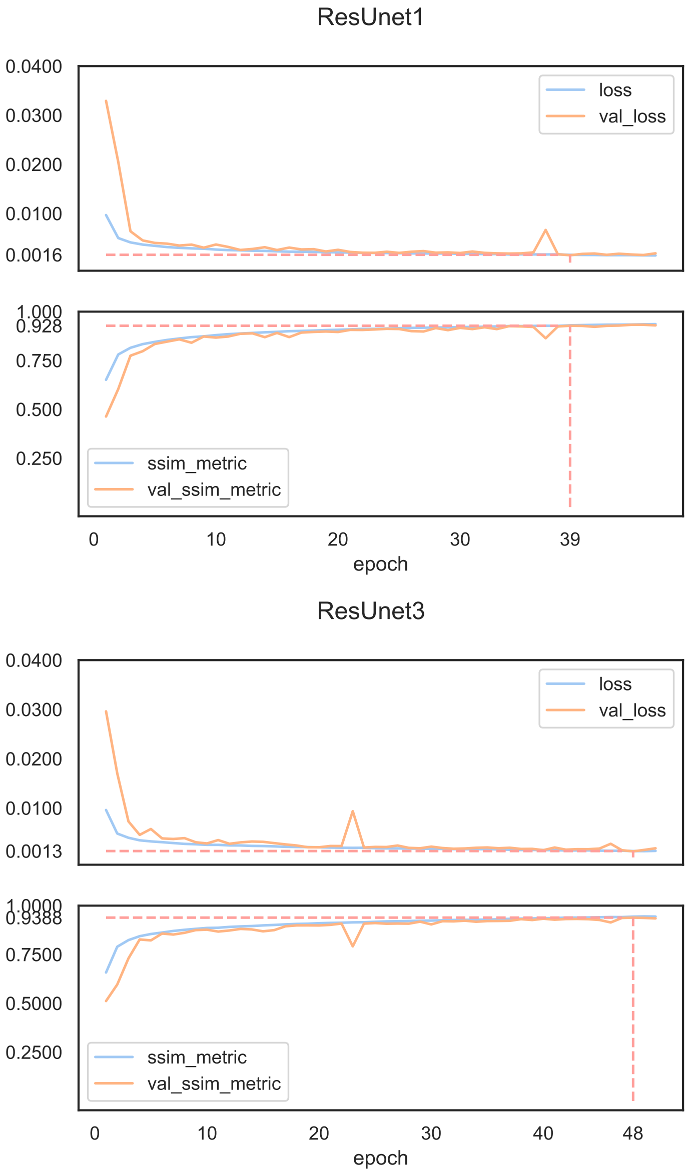


Figure 1: Training information of ResUNet

3.3 Results

Two models were tested and the evaluation on a **uniform gaussian blur** is the following:

number of ResBlock	MSE	PSNR	SSIM
1	0.0018	28.49	0.930
3	0.0016	29.03	0.935

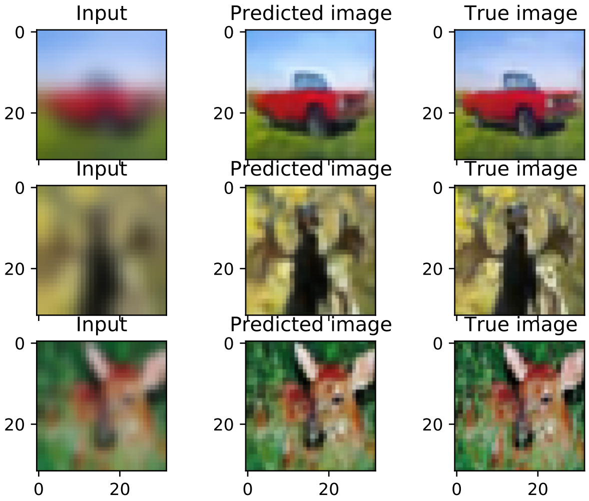


Figure 2: Test image generated by **ResUNet1**

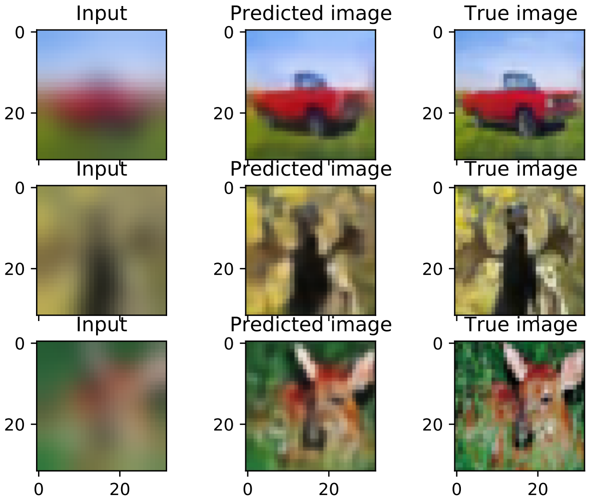
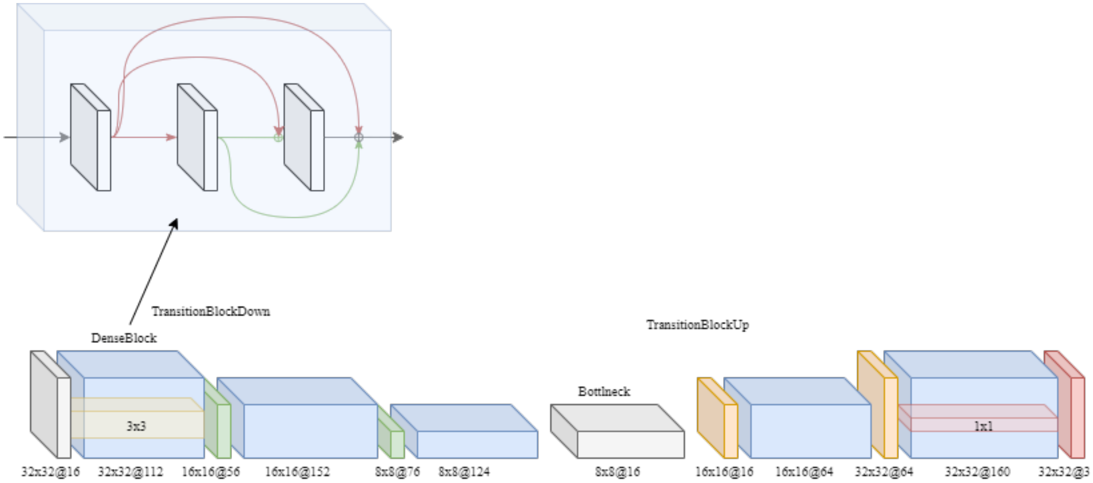


Figure 3: Test image generated by **ResUNet3**

Both networks are able to generate quite sharp images but we can see that ResUnet3 is able to restore more fine details when the input is heavily blurred.

3.4 EDDenseNet

The model is the following:



The DenseNet[4] architecture improve the ResNet skip connections allowing the concatenation instead of doing the sum: each DenseBlock contains layers whose input is the concatenation of all previous layers' channels output allowing *feature reuse* which increase the performance of the network because allow a better flow of the information on the network and at the same time allow to have more compact network. Each DenseBlock contains bottleneck layer before each 3x3 convolution in order to reduce the number of parameters.

After each DenseBlock, a TransitionBlock is inserted which downsamples the output of a dense block.

In the paper the DenseNet-121 has a growth rate of 32 and the number of blocks are [6,12,24,16]

The model implemented has different characteristics:

- There are TransitionBlocks which upsample the output of a DenseBlock in order to mimic the Encoder-Decoder architecture.
- The growth rate is 16 and the number of layers for the encoder are [6, 6, 3] and for the decoder [3, 6].
- A bottleneck layer is used for reducing the number of channels at the end of the encoder (thus reducing the number of parameters): for instance without the bottleneck layer the amount of parameters would be $124 * 3 * 3 * 124 + 124 = 138384$, instead with the bottleneck layer is $124 * 3 * 3 * 16 + 16 = 17872$ (so with bottleneck layer using a large growth rate and more layers would reduce a lot the parameters therefore reducing also the training time)
- With respect to the original paper the downsampling and upsampling is done respectively using Conv2D and Conv2DTranspose in order to "learn" the transformation instead of using MaxPooling and UpSampling leading to a loss of information.

- The last layer is a Conv2DTranspose which ”learn” to decode the output.

3.5 Training

The network was trained with Adam[9] with default parameters and MSE as loss function. The network converged within 22 epochs with early stopping equals to 5.

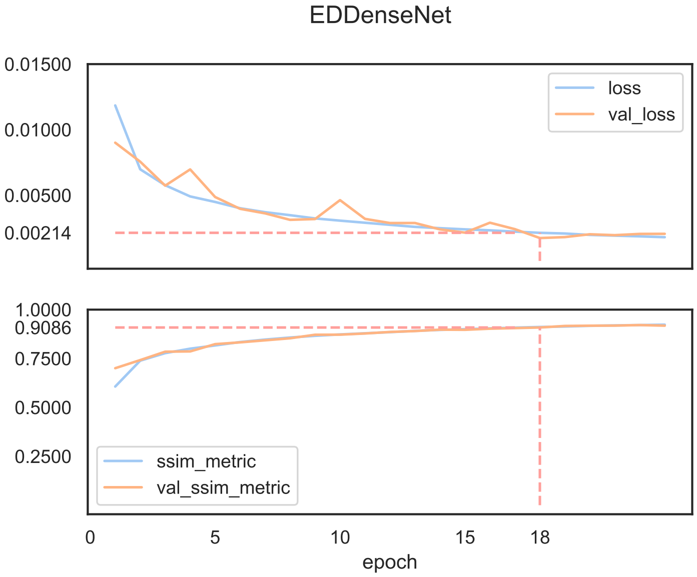


Figure 4: Training information of EDDenseNet.

3.6 Results

The evaluation of the network is the following:

kernel type	MSE	PSNR	SSIM
gaussian	0.0021	27.62	0.90

My implementation (the encoder) is not so deep as the one implemented by researchers, nevertheless the results are quite good.

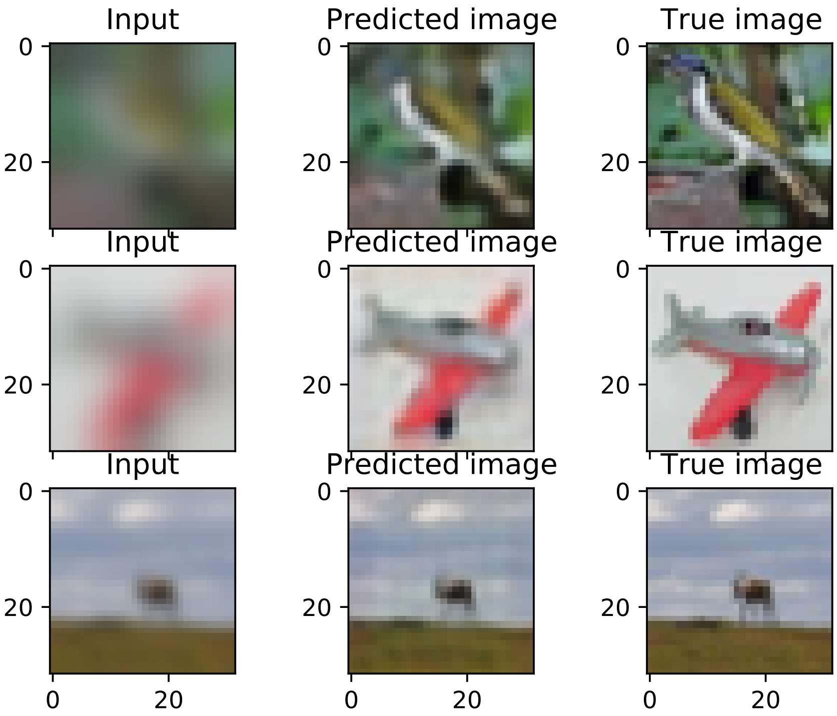
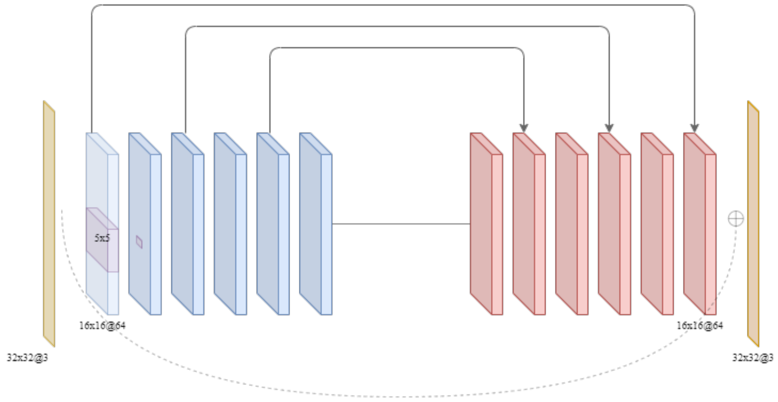


Figure 5: Test image generated by EDDenseNet.

Looking at the images I can assert that the network is able to restore fine details from the blurred image even when they are not visible (i.e. the cockpit of the plane or the details on the chest of the bird).

3.7 CAESSC

The model is the following:



In order to achieve good result for low-level image processing as deblurring, super-resolution, deep network are necessary. Nevertheless the deeper the network, the slower the training is, thus skip connections are used between symmetric convolutional and deconvolutional layers.

Features are extracted by convolutional layers (encoder) and those information are used by deconvolutional layer (decoder) for reconstructing the image; moreover symmetric skip connections are used for increasing the performance and the ability to converge of the network.

The architecture is quite simple:

- Convolutional and deconvolutional layer are defined as the following: Conv/Deconv → BN → ReLU
- The last layer use sigmoid[2] instead of ReLU (as defined in the paper)
- The first Conv2D downsample the input and the last Conv2DTranspose upsample it. The researchers observed that an earlier downsampling allow to decrease the training time with a small reduction of the performance.
- Encoder-Decoder architecture where convolutional and deconvolutional layers are connected symmetrically
- A skip connection between the input and the output allow to learn the output deblurred without the skip connection plus all the details learned by the encoder. [3]

Skip connections have important goals [3]:

1. Networks are easy to optimize using the identity function

[2] See 3.8 Training

[3] Refer to [5, 4.1 Analysis of the architecture]

2. The features information are forwarded from convolutional to deconvolutional layer facilitating the work of deconvolutional layer for reconstructing the image increasing the performance of the network.
3. Overcame **vanishing gradient**: the weights are not updated when the network is deeper because gradients became smaller and smaller the deeper the network is.
4. Overcame **exploding gradient**: the backpropagation of the gradient could lead to an exponentially increase of the gradient the deeper the network is.

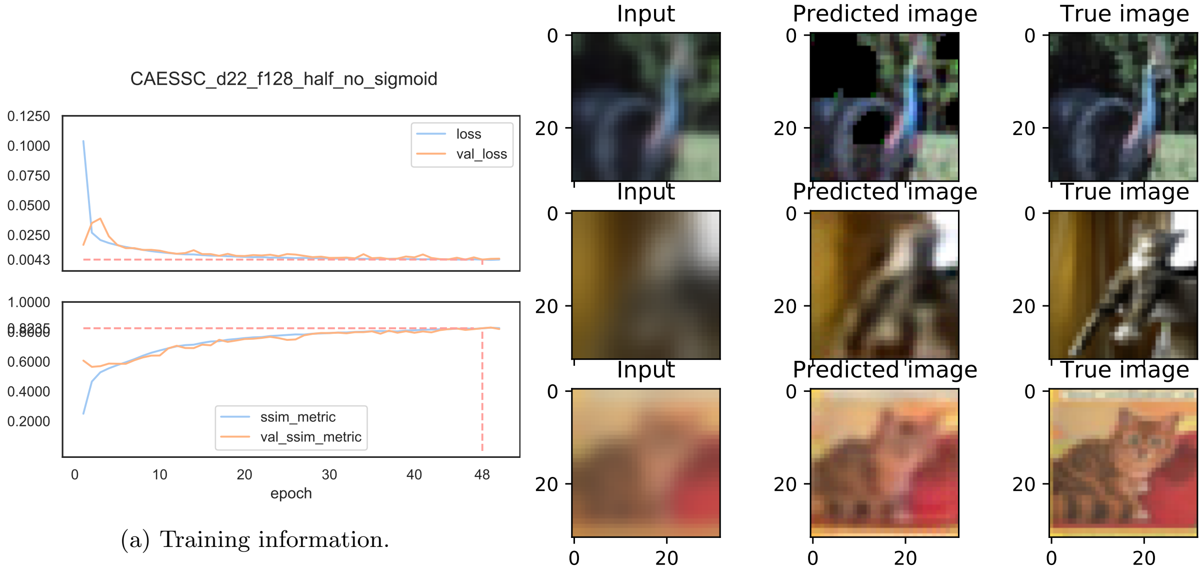
The model implemented are:

- **CAESSC_d30_f64**: depth equals to 30, filters equal to 64 and no downsampling applied.
- **CAESSC_d22_f128_half**: depth equals to 22, filters equal to 128 and downsampling applied.
- **CAESSC_d22_f128_half_no_sigmoid**: depth equals to 22, filters equal to 128, downsampling applied and last layer use ReLU.

3.8 Training

The network was trained using Adam[9] with $1e^{-4}$ as learning rate, MSE as loss function and with an early stopping equals to 7 in order to push more the training.

Using the configuration above was observed that the performance were poor with ReLU as last activation function: we can see that the network can't restore black area of the input image properly.

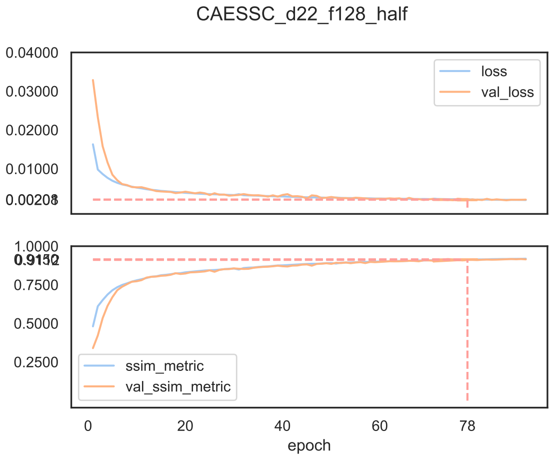


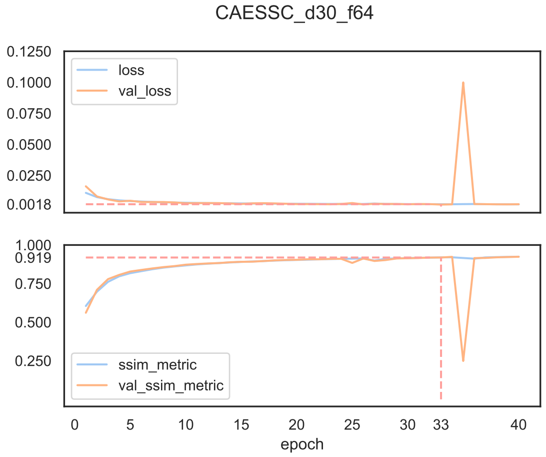
(a) Training information.

(b) Test image.

Figure 6: CAESSC with depth=22,filters=128, downsampling and ReLU as last activation function

CAESSC_d22_f128_half and CAESSC_d30_f64 converged, respectively, within 80 and 40 epochs:



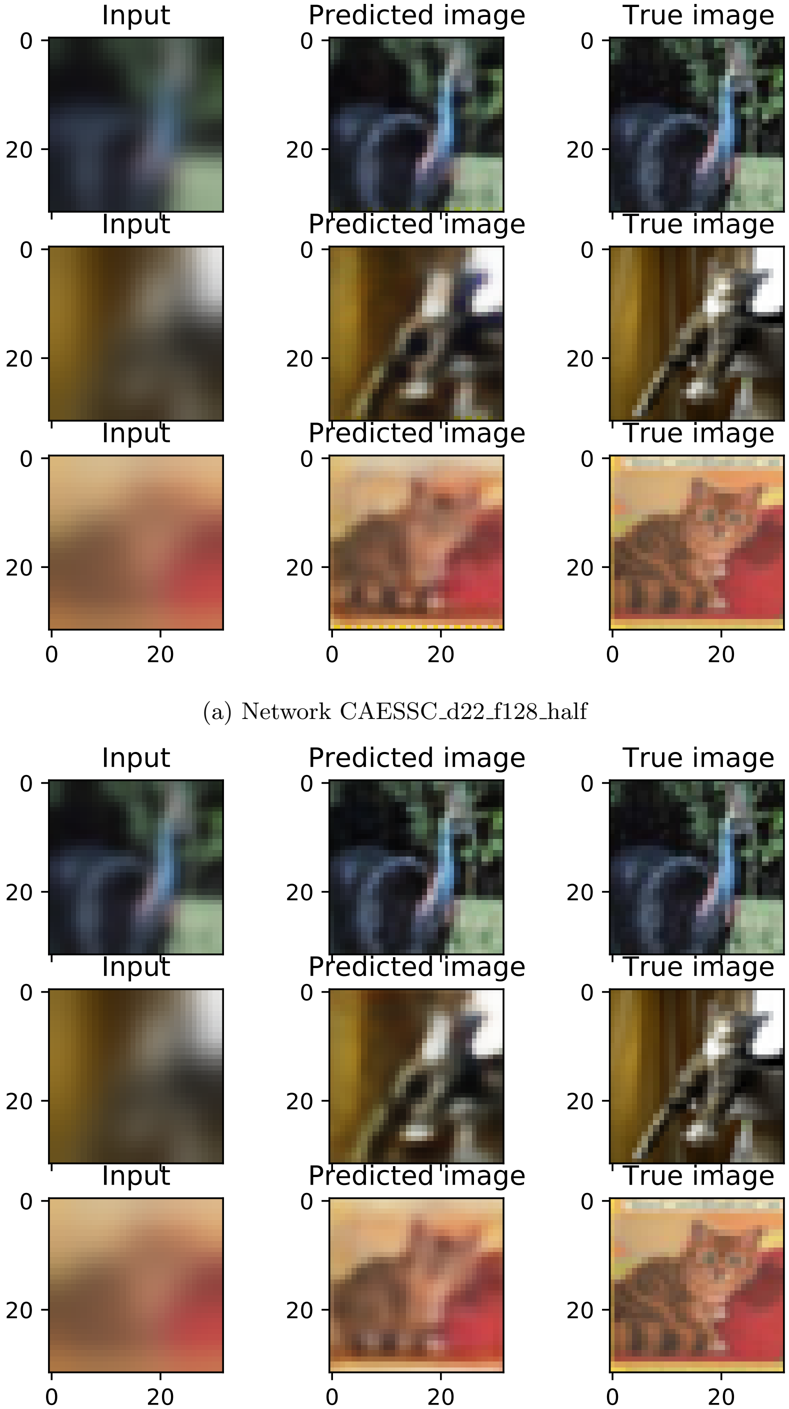


3.9 Results

The evaluation of the networks is the following:

network	MSE	PSNR	SSIM
RED30[4]	-	34.49	-
CAESSC_d22_f128_half_no_sigmoid	0.00389	24.85	0.839
CAESSC_d22_f128_half	0.0020	28.12	0.916
CAESSC_d30_f64	0.0018	28.95	0.919

[4] RED30 is composed by 30 layers with skip connections every 2 layers, 128 filters, kernel size equal to 3 and no downsampling. Taken from `https://github.com/ved27/RED-net/blob/master/model/REDNet_ch3.prototxt` and `/model/debluring/gaussian.caffemodel`



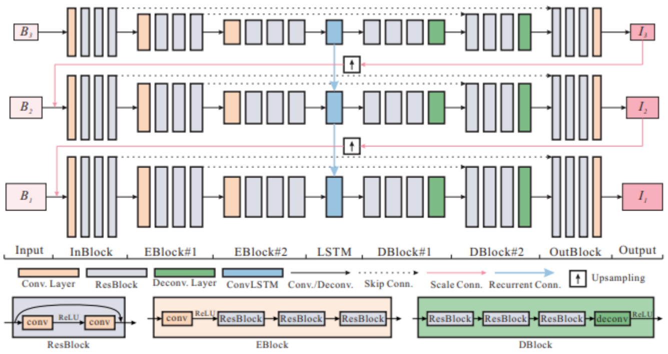
(a) Network CAESSC_d22_f128_half

(b) Network CAESSC_d30_f64

Figure 8: Test image generated by CAESSC.

4 SRNDeblur

The base model, taken from the paper [6], is the following:



In image deblurring an encoder-decoder network is not going to produce very sharp results. In order to improve this, [6] create a **recurrent network** which shares parameters among different **scale** thanks to residual connectivities which lead to better performance and faster training times.

The idea is that all scales are going to solve the same problem therefore they should use the same parameters.

The network takes blurred images at different scales and learn to produce sharp images at the corresponding level. Moreover scale connection between the output in a scale and input in the next one allow the reuse of parameters within different scales and skip connections on the same level allow to improve the quality of the result as well as the training time.

A novelty introduced by [6] is the use of a **recurrent layer** which share its internal state with the recurrent layer in the next one allowing to share information that are able to improve the quality of the deblurred image.

This architecture exploit all the technique introduced in the literature up to now:

1. Fully connected convolutional neural network which allows to handle any input (provided that the machine is able to handle the input in memory)
2. The creation of blocks which allow to scale easily the network
3. The encoder-decoder architecture
4. The use of residual blocks
5. The use of multi-scale CNN
6. The use of a recurrent layer

4.1 Architecture

The implementation of the network for both *CIFAR10* and *REDS* was done using only **two scales** in order to reduce the number of parameters and so to reduce the learning time.

Moreover due to the memory limitation the datasets are handled with different architectures:

CIFAR10

- BatchNormalization is used
- LeakyRELU activation is used
- Sigmoid activation in the final layer
- LSTM layer is used (In order to accommodate the input the following layers are added:
  Reshape → LSTM[a] → Dense[b] → Reshape)

REDS

- No BatchNormalization is used
- Relu activation is used
- No activation function is used in the final layer
- Conv2DLSTM layer is used
  (This is the configuration used inside the paper)
- Input layer is concatenation of two layers over the filter dimension instead of a single Conv2D layer.

The hyperparamters are the following:

CIFAR10

- Filters start from 32
- Batch size is 150
- Epochs are 70
- Adam learning rate: $1e^{-3}$

REDS

- Filters start from 8 (increased to 16 concatenating channels in the input in order to avoid *OutOfMemory* error).
- Batch size is 10.
- Epochs are 150 (average training time is 1h15m therefore time spent for training is 150*75=187h30m).
- Amount of patches extracted per image equals to 2 of size 256 × 256
- Adam learning rate: $1e^{-4}$

4.2 REDS Training set

In order to cope with low memory and high training time, a tradeoff between batch size and training set size was necessary.

1. The first 24 scenes on the training set were used as validation set
2. It's used half of the training set [5]
3. Two patches are taken from each image: the idea is that the more are the patches taken the greater is the probability that patches contain a motion blur. For instance with batch_size = 10 and num_patches = 2 at each batch 10//2 = 5 images are selected.
4. An analysis of the training set show that the probability that a patch is too perfect [Figure 9a and 9b] is 29% (using kernel density estimation[6] - Figure 9d) therefore the **ssim** of patches

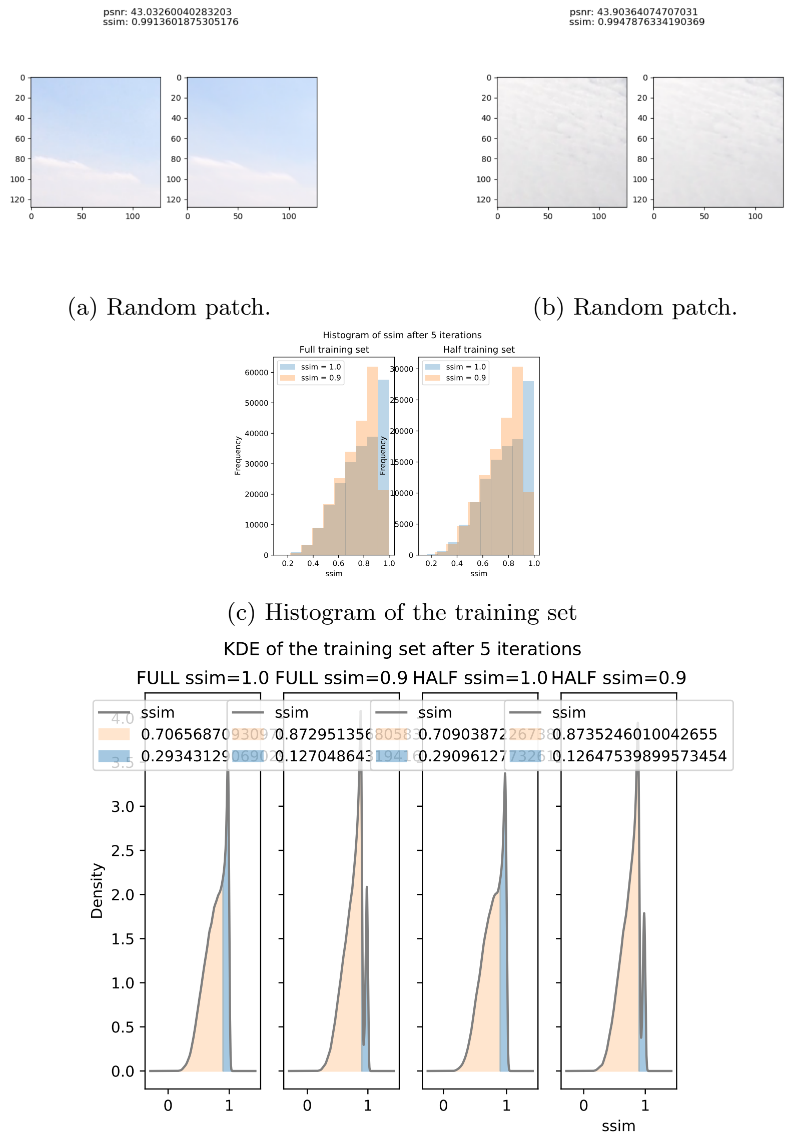
[a] 32 hidden units were used at each scale in order to reuse the internal state and reduce the overhead

[b] Hidden units equal to the multiplication of the input dimensions (*width* × *height* × *channels*)

[5] It's used the following expression self.dir_sharp.glob(**/000000[5-9][0-9].png)

[6] Refer to reds_generator_histogram_test.py

extracted is computed in order to take the ones with a ssim less than ssim_threshold = 0.9 for avoiding patches that could influence negatively the training and so use as efficiently as possible each epoch. In order to maintain a generalization during the training when the threshold is not satisfied, random patches are taken.[7]



(a) Random patch.

(b) Random patch.

(c) Histogram of the training set

(d) Kernel density estimation of the training set

Figure 9: Random patches. Histogram and kernel density estimation (using default parameters) of the full and half training set.

[7] Refer to reds_generator.py get_all_patches

4.3 Training

The network was trained with Adam[9] using a learning rate of $1e^{-4}$ and MSE as loss function for both the outputs. The network used is the one at the epoch 150 when the training was stopped. Were spent 10h/day for training resuming each time the training. **SRNDeblur_reds was trained for 150 epochs in 180h**. SRNDeblurNet[6] was trained for 2000 epochs in 72h.

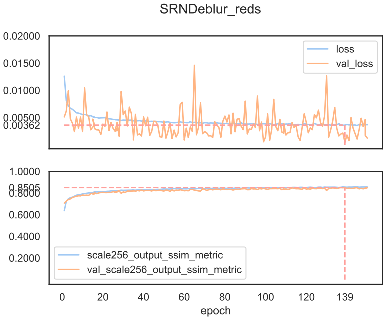


Figure 10: Training information SRNDeblur_reds

Different from the paper, the learning rate was not reduced to $1e^{-6}$ over 2000 epochs with an exponential decay, because I have thought to stop at 150 epochs since the beginning; this is, probably, the cause of the plateau of the loss.

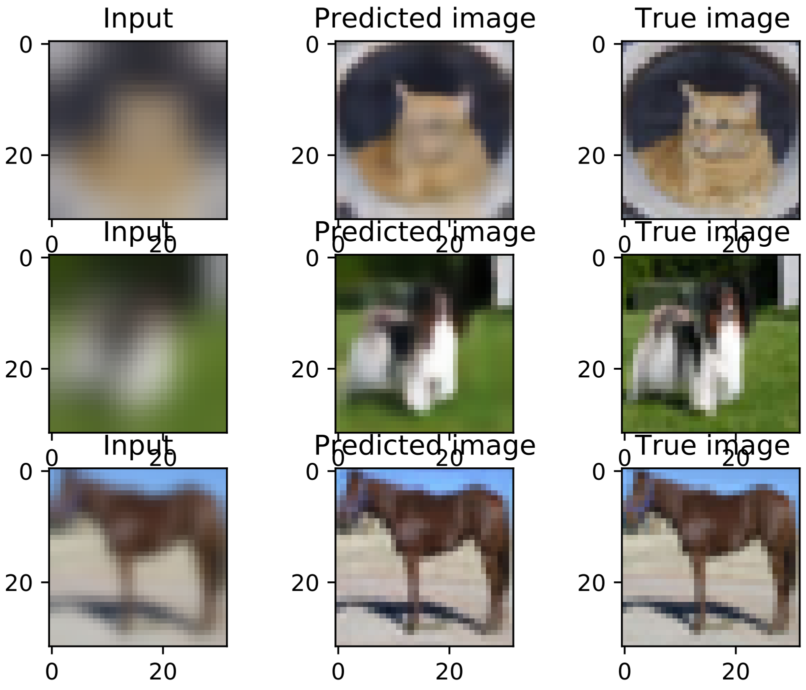
5 Results

The evaluations are the following:

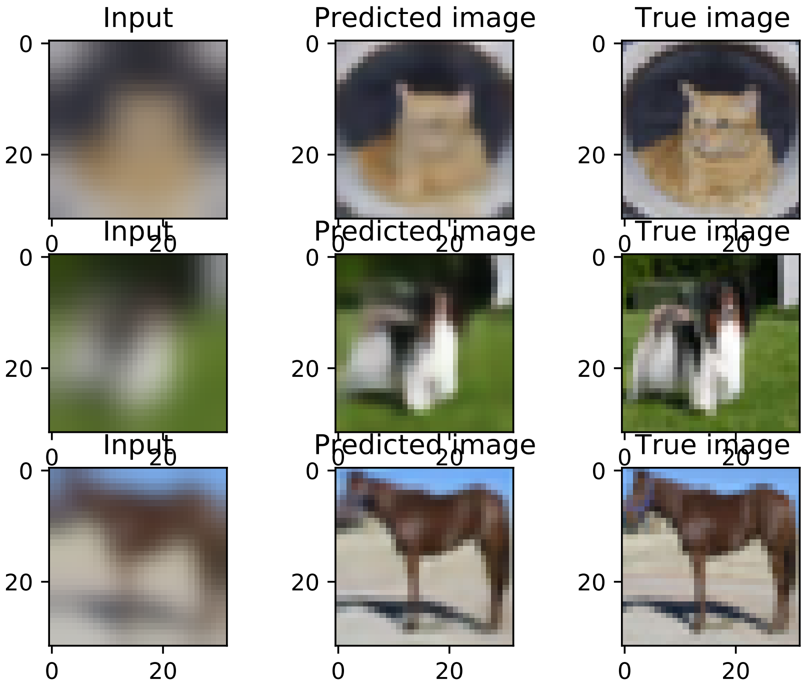
CIFAR10 network	MSE	PSNR	SSIM
Without LSTM	0.00175	29.23	0.9225
With LSTM	0.00174	29.38	0.9188

	SRNDeblur_reds[8]			SRN-DeblurNet	
dataset	MSE	PSNR	SSIM	PSNR	SSIM
REDS	0.002	27.23	0.8105	-	-
GOPro	-	-	-	30.26	0.9342

[8] The evaluation is done deblurring full hig res images and then computing their ssim. The average evaluation over 500 samples is returned.



(a) Test image generated by SRNDeblur_cifar network without LSTM



(b) Test image generated by SRNDeblur_cifar network with LSTM

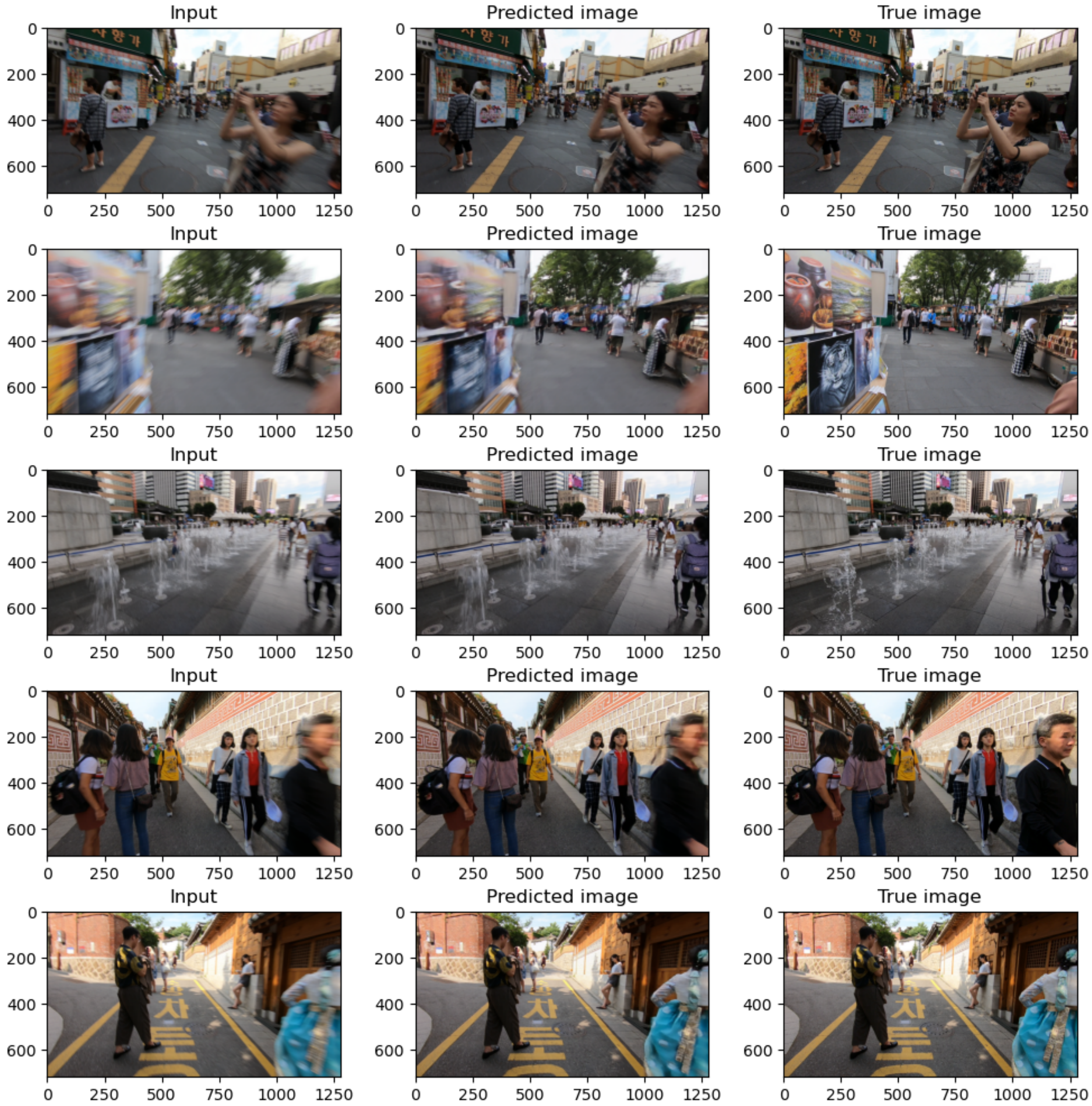


Figure 12: Test image on high resolution test set generated by REDS network.

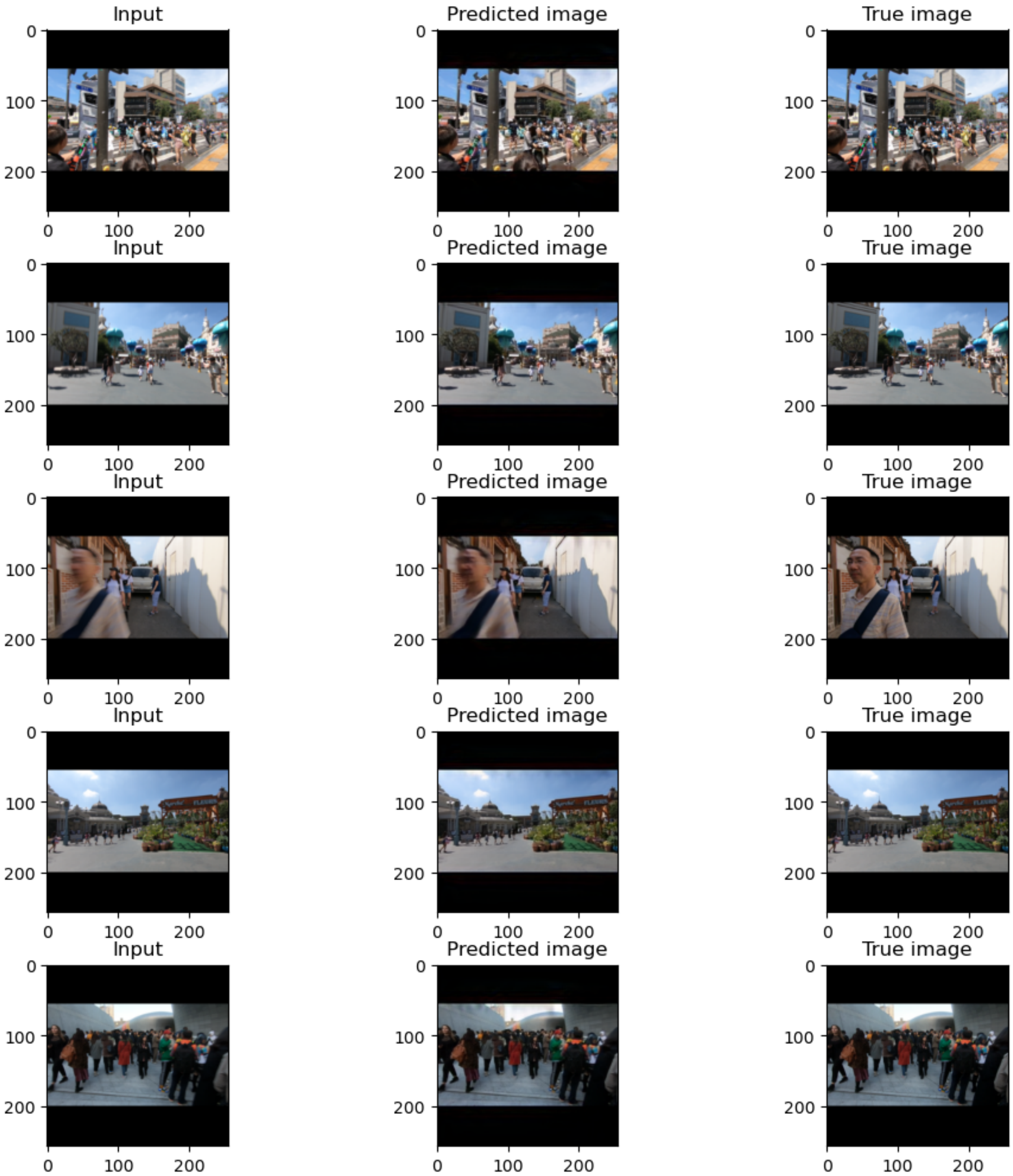


Figure 13: Test image on low resolution test set generated by REDS network.

References

[1] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[2] S. Nah, S. Baik, S. Hong, G. Moon, S. Son, R. Timofte, and K. M. Lee, "Ntire 2019 challenge on video deblurring and super-resolution: Dataset and study," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[4] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[5] X. Mao, C. Shen, and Y. Yang, "Image restoration using convolutional auto-encoders with symmetric skip connections," *CoRR*, vol. abs/1606.08921, 2016.

[6] X. Tao, H. Gao, Y. Wang, X. Shen, J. Wang, and J. Jia, "Scale-recurrent network for deep image deblurring," *CoRR*, vol. abs/1802.01770, 2018.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[9] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv e-prints*, p. arXiv:1412.6980, Dec. 2014.