Nankai University

Artificial Intelligence: Algorithms and Applications

Machine Vision Practical Project

Johnny Agosto, Chiara Malizia

July 23, 2021

Agenda

Introduction

Image deblurring is the process of removing blurs and restoring the high-quality latent image. It is one of the most common tasks in image restoration. Blur can be of various types like Motion blur, Gaussian blur, Average blur, Defocus blur etc.
More formally, deblurring can be described as the problem of recover the sharp image S from a given blurred image B. It can be mathematically represented with the equation

$$\mathrm{B} = \mathrm{S} * \mathrm{K} \tag{1}$$

where K represents the unknown linear shift-invariant Point Spread Function (PSF), and $*$ denotes the convolution operator.
In this project we investigated the Gaussian blur and therefore K is the Gaussian filter in the formula (1).

In Image Processing, a Gaussian blur is the result of blurring an image by a Gaussian kernel function, with the main purpose to reduce the details of an image.
Formally speaking, the Gaussian kernel function $\mathrm{G(x, y)}$ is defined as the product of two 1D Gaussian functions (one in each dimension) and it results as follows:

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

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.

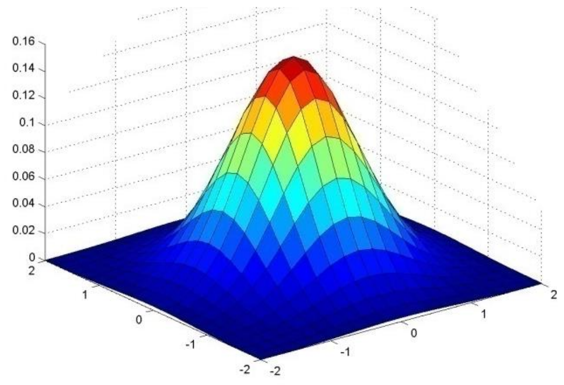


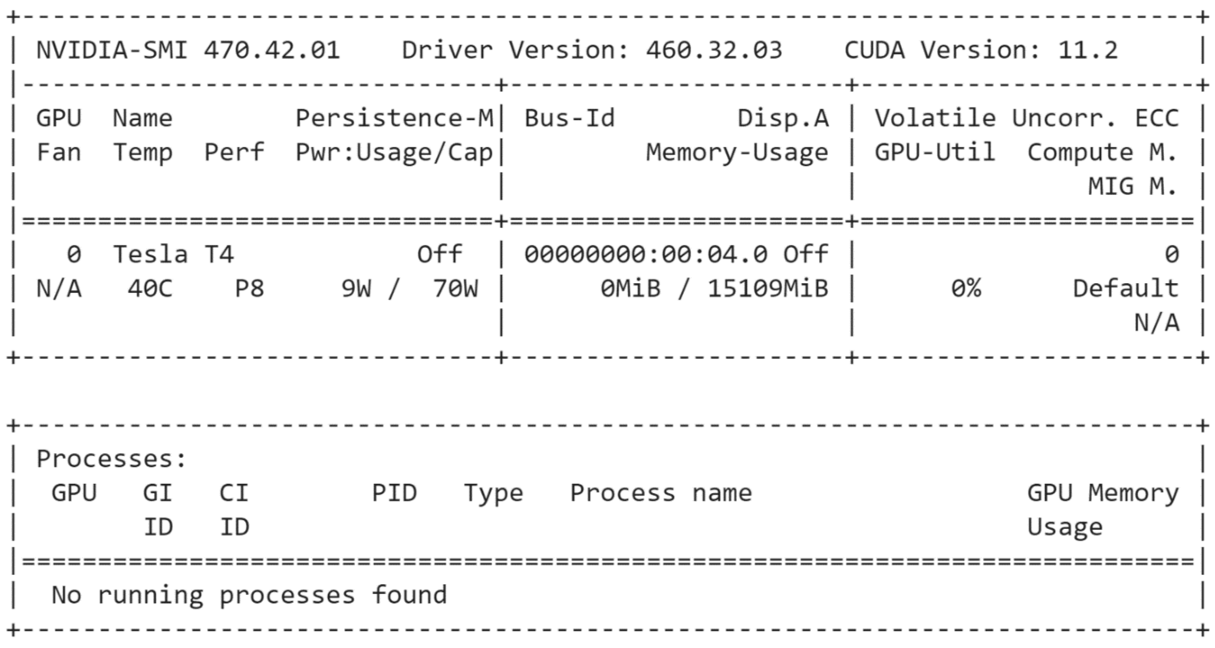
Fig 1. 2D Gaussian distribution

Moreover, we can distinguish two deblurring techniques: *blind deconvolution methods* and non-*blind deconvolution methods.* The former focus on recovering the unknown PSF, while the latter rely on a known PSF to perform robust deconvolution. In our work we considered the blind deconvolution method and we dealt with a deep neural network approach to recover the PSF by solving an inverse ill-posed problem, where the PSF function is represented by mean of the neural network. In practice, given a set of blurred images in input, the neural network should first recover the unknown PSF and then output the corresponding sharp images. To do so, we created a dataset starting from CIFAR-10 and we exploited a U-Net autoencoder architecture, whose main features are described in the section below. The idea of manipulating the dataset came from other related works [1][2].

In the next section the tools used for the project are described, while in the section 2 we describe the CIFAR-10 dataset and its manipulation. Section 3 is dedicated to the introduction of some metrics we used to train and assess our model. Section 4 provides a brief description of Convolutional Autoencoders with the aim of introducing the U-Net architecture. Section 5 and 6 present the training details and the results obtained during the inference phase, respectively.

1. Tools

We implemented our model in Python using TensorFlow 2.5 and Keras 2.5 libraries. Numpy, Pandas, Matplotlib and OpenCv are used too. All the experiments were performed on Google Colab using the GPU backend. More information is shown in the figure 2.



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.42.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   40C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Fig 2. GPU provided by Google Colab platform

2. Dataset Creation

The CIFAR-10 dataset contains 60000 colour images of resolution 32x32 [3]. The images in the collection are divided into 10 classes, which we simply ignored. Indeed, we used the dataset to perform an image restoration task and not a classification task.

To do so, for each dataset image we constructed the associate blurred version by applying a Gaussian filter with standard deviation between 0 and 3. In the figure 3 an example of the same CIFAR-10 image blurred with increasing standard deviation is shown.

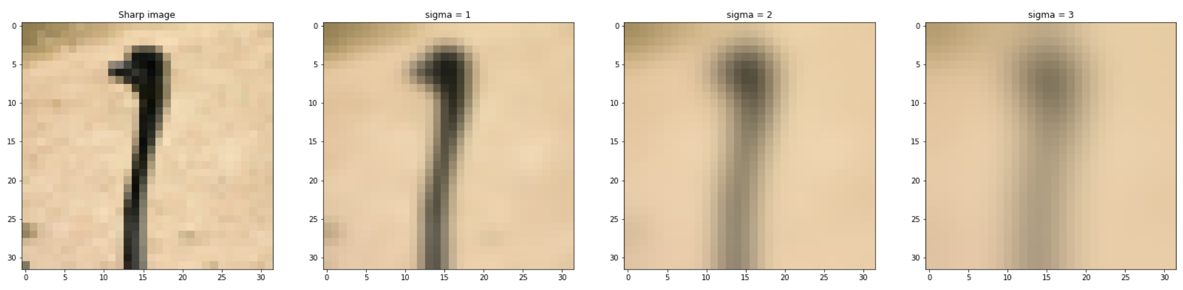


Fig 3. Example of the same CIFAR-10 image blurred with increasing standard deviation

After, we split the dataset into Training, Validation and Test sets, with a proportion of 80%, 10% and 10% respectively. In this way, we used 40000 images to train the network, 10000 for the validation phase and 10000 to test the network.

3. Evaluation Metrics

The metrics employed to evaluate the performance of our network are the following:

- *Structural Similarity Index Measure (SSIM)*: it is a method to measure the *similarity* between two images which relies on structural information to evaluate image degradation [4]. The higher, the better.
- *Peak Signal-to-Noise Ratio (PSNR)*: it is the ratio between the maximum possible power of a signal and the power of corrupting noise [5][6]. The higher, the better.
- *Mean Squared Error (MSE)*: it represents the cumulative squared error between the predicted image and the original one. The lower, the better.

4. Convolutional Autoencoders

Our approach relied on convolutional autoencoders, which consist of:

- an *encoder*, which is a ConvNet producing a low-dimensional representation of the image. The convolutional layers act as a feature extractor, which both preserve the primary components of objects in the image and remove blurry artifacts.
- a *decoder*, which is another ConvNet. It takes the compressed image outputted from the encoder and reconstructs the original clean image. The deconvolutional layers up-sample the feature maps and recover the image details eventually lost during the convolution process.

A network realizing the idea of a convolutional autoencoder is U-Net.
U-Net is a convolutional neural network that was developed for biomedical image segmentation at the Computer Science Department of the University of Freiburg [7].
It was particularly successful for both speed and the small number of training data required. The U-Net architecture proposed by Ronneberger et al. is shown in the figure 4.

The architecture consists of two major parts:

- the left side of the U, called "*contracting path*", is characterized by repeated application of convolutions, each followed by a ReLU and a max pooling operation for downsampling.
- the right side of the U, called "*expansive path*", applies transposed convolution useful to up-sample and common convolutions.

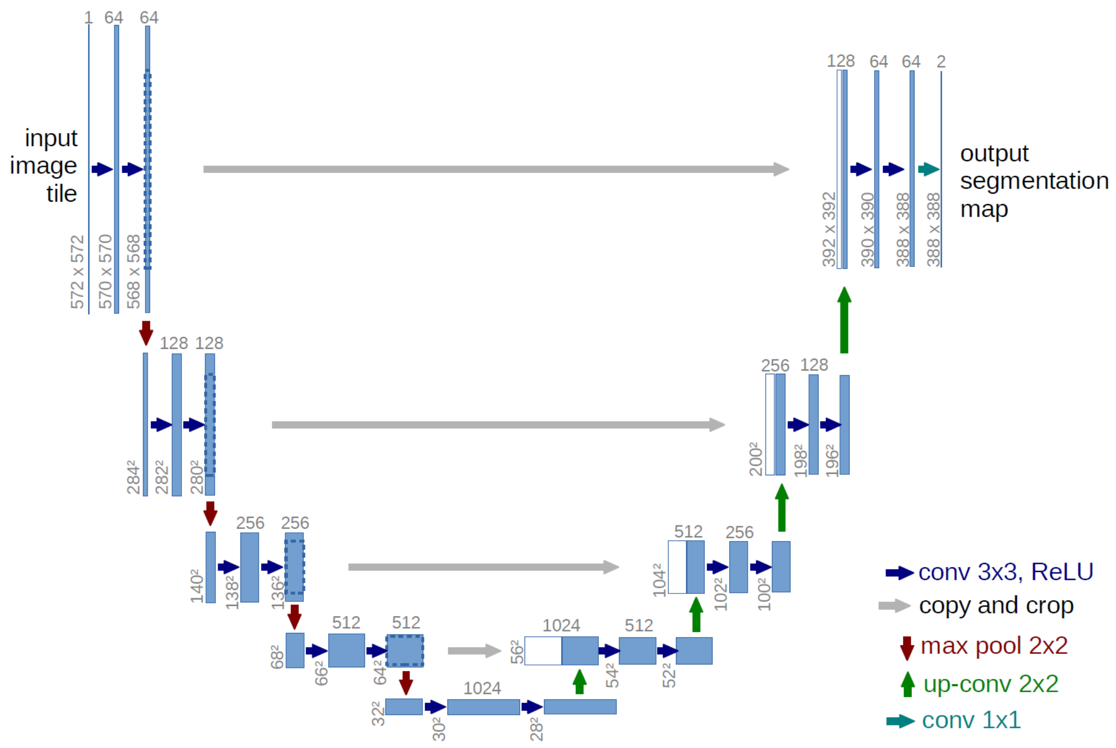


Fig 4. U-Net architecture (example for 32x32 pixels in the lowest resolution) [7]

The architecture we used in our work is U-Net20, which is described below and displayed in the figure 5.

In the contracting path successive application of 3x3 convolutions are followed by a ReLU and by the max pooling operation. The latter is used with window size 2x2 and stride 2 in order to down-sample the image. But 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 256 feature channels. In the expansive path every step involves first of all an upsampling of the feature map followed by a 2x2 convolution, thanks to which it is possible to halve the number of feature channels. Then a concatenation with the corresponding feature map coming from the contracting path is performed, and two 3x3 convolutions, each followed by a ReLU, are applied. Finally, there is a 1x1 convolution layer in order to get a 32x32x3 image as output.

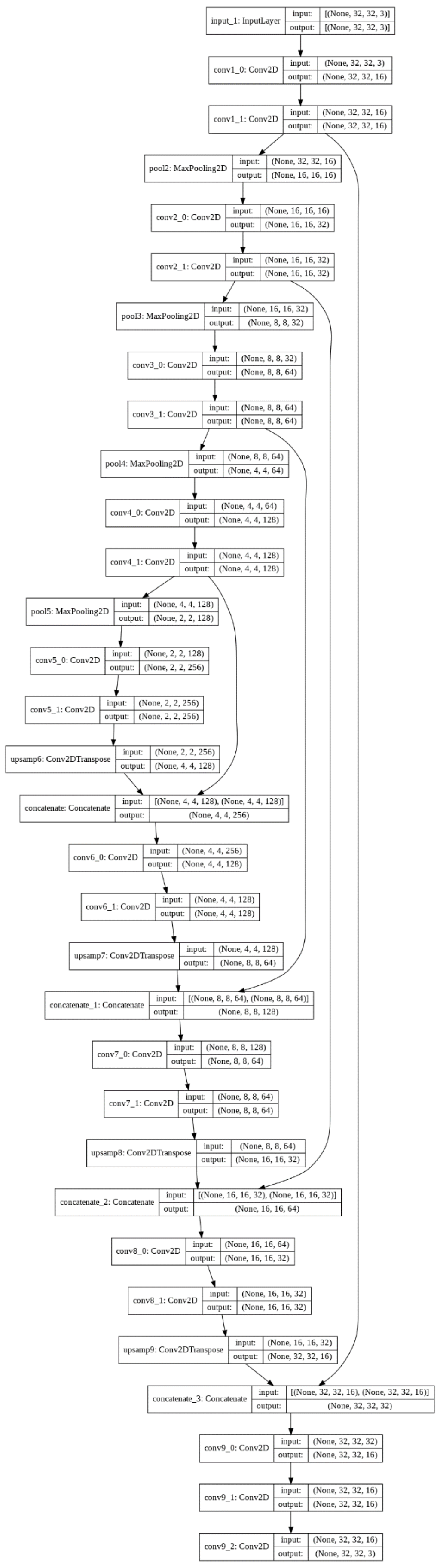

input_1: InputLayer | input: [(None, 32, 32, 3)] | output: [(None, 32, 32, 3)]
conv1_0: Conv2D | input: (None, 32, 32, 3) | output: (None, 32, 32, 16)
conv1_1: Conv2D | input: (None, 32, 32, 16) | output: (None, 32, 32, 16)
pool2: MaxPooling2D | input: (None, 32, 32, 16) | output: (None, 16, 16, 16)
conv2_0: Conv2D | input: (None, 16, 16, 16) | output: (None, 16, 16, 32)
conv2_1: Conv2D | input: (None, 16, 16, 32) | output: (None, 16, 16, 32)
pool3: MaxPooling2D | input: (None, 16, 16, 32) | output: (None, 8, 8, 32)
conv3_0: Conv2D | input: (None, 8, 8, 32) | output: (None, 8, 8, 64)
conv3_1: Conv2D | input: (None, 8, 8, 64) | output: (None, 8, 8, 64)
pool4: MaxPooling2D | input: (None, 8, 8, 64) | output: (None, 4, 4, 64)
conv4_0: Conv2D | input: (None, 4, 4, 64) | output: (None, 4, 4, 128)
conv4_1: Conv2D | input: (None, 4, 4, 128) | output: (None, 4, 4, 128)
pool5: MaxPooling2D | input: (None, 4, 4, 128) | output: (None, 2, 2, 128)
conv5_0: Conv2D | input: (None, 2, 2, 128) | output: (None, 2, 2, 256)
conv5_1: Conv2D | input: (None, 2, 2, 256) | output: (None, 2, 2, 256)
upsamp6: Conv2DTranspose | input: (None, 2, 2, 256) | output: (None, 4, 4, 128)
concatenate: Concatenate | input: [(None, 4, 4, 128), (None, 4, 4, 128)] | output: (None, 4, 4, 256)
conv6_0: Conv2D | input: (None, 4, 4, 256) | output: (None, 4, 4, 128)
conv6_1: Conv2D | input: (None, 4, 4, 128) | output: (None, 4, 4, 128)
upsamp7: Conv2DTranspose | input: (None, 4, 4, 128) | output: (None, 8, 8, 64)
concatenate_1: Concatenate | input: [(None, 8, 8, 64), (None, 8, 8, 64)] | output: (None, 8, 8, 128)
conv7_0: Conv2D | input: (None, 8, 8, 128) | output: (None, 8, 8, 64)
conv7_1: Conv2D | input: (None, 8, 8, 64) | output: (None, 8, 8, 64)
upsamp8: Conv2DTranspose | input: (None, 8, 8, 64) | output: (None, 16, 16, 32)
concatenate_2: Concatenate | input: [(None, 16, 16, 32), (None, 16, 16, 32)] | output: (None, 16, 16, 64)
conv8_0: Conv2D | input: (None, 16, 16, 64) | output: (None, 16, 16, 32)
conv8_1: Conv2D | input: (None, 16, 16, 32) | output: (None, 16, 16, 32)
upsamp9: Conv2DTranspose | input: (None, 16, 16, 32) | output: (None, 32, 32, 16)
concatenate_3: Concatenate | input: [(None, 32, 32, 16), (None, 32, 32, 16)] | output: (None, 32, 32, 32)
conv9_0: Conv2D | input: (None, 32, 32, 32) | output: (None, 32, 32, 16)
conv9_1: Conv2D | input: (None, 32, 32, 16) | output: (None, 32, 32, 16)
conv9_2: Conv2D | input: (None, 32, 32, 16) | output: (None, 32, 32, 3)

Fig 5. U-Net20

5. Training Details

Below are the details relating to the training phase of our model.

Training details of our model				
Batch Size	Iterations	Loss	Optimizer	Learning Rate
32	60	LAD	Adam	0.001

Table 1. Training details of our model

Some notes:

- LAD is Least Absolute Deviations.
- In order to find the optimal learning rate, we used the ReduceLROnPlateau provided by Keras in the module callbacks. The initial value of the learning rate was 0.001, that decreased by a 0.2 factor every time a plateau in the validation loss function was identified, namely after 4 epochs in which no improvement was seen.
- The value of the epochs was 500, but the number of actual iterations done by the network is 60. This is because we exploited the EarlyStopping strategy supported by the Keras callbacks.
- The metrics employed to assess the similarity between the original images and the reconstructed ones are those already mentioned in section 3.
- The results that we obtained on the test set can be found in the Results section presented below.

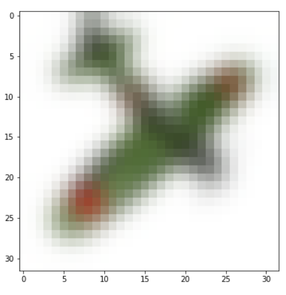
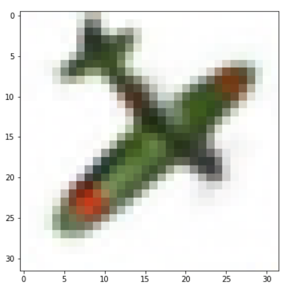
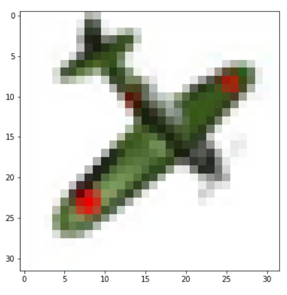
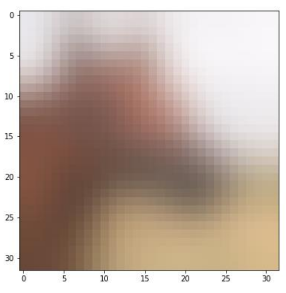
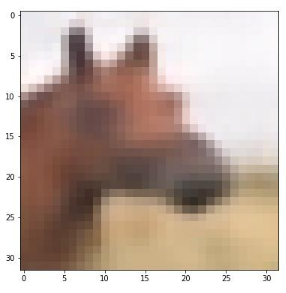
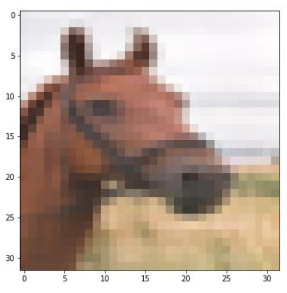
6. Results

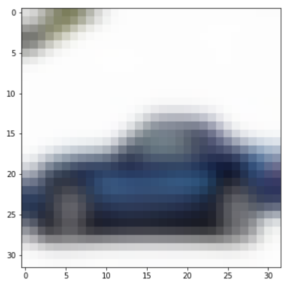
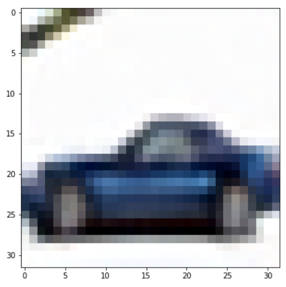
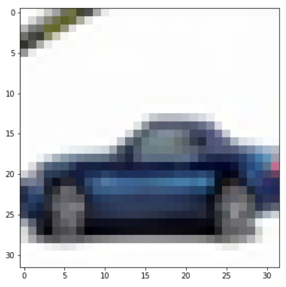
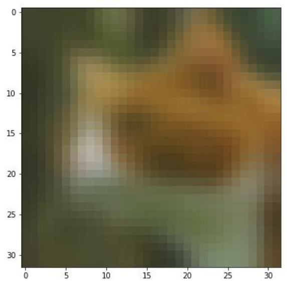
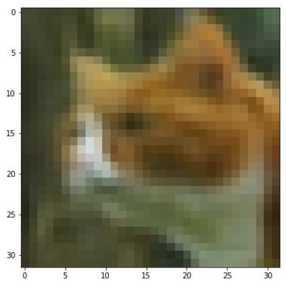
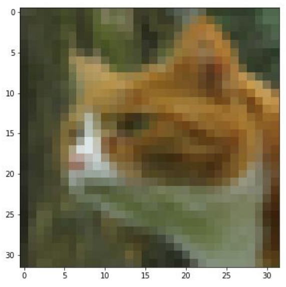
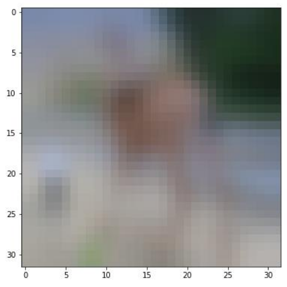
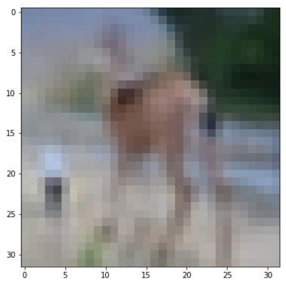
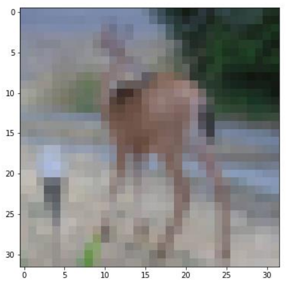
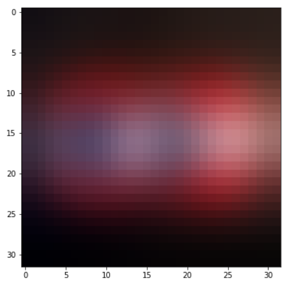
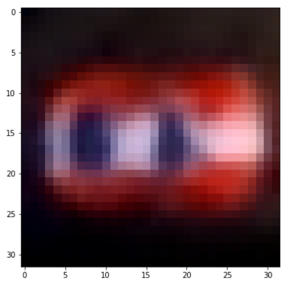
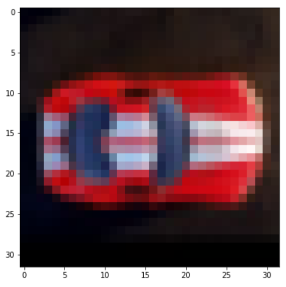
Here we report a summary of the results we have obtained with our network.

Performance on test set	
	Values
Ssim	0.874477
Psnr	27.181793
Mse	0.002962

Table 2. Performance on test set of our model

In Table 3 some of the images that we have reconstructed through our models are shown.

Output examples		
blurred	***predicted***	***sharp***
		
		

blurred	***predicted***	***sharp***
		
		
		
		

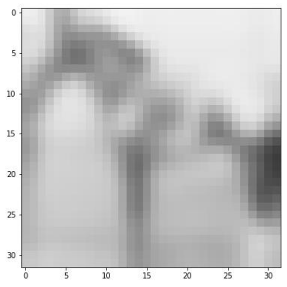
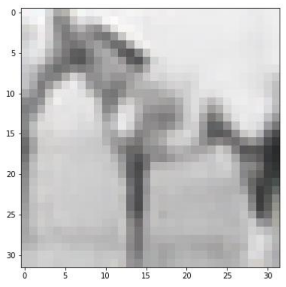
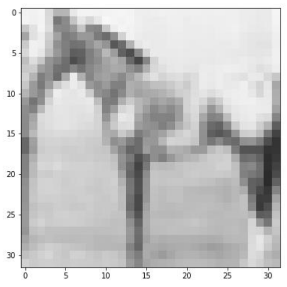
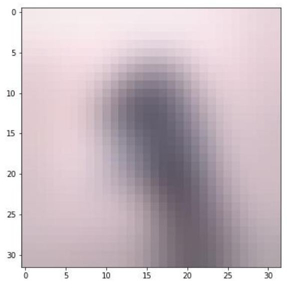
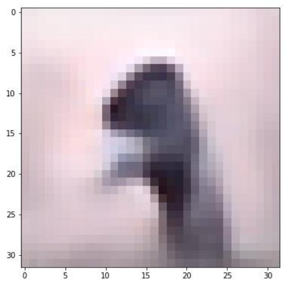
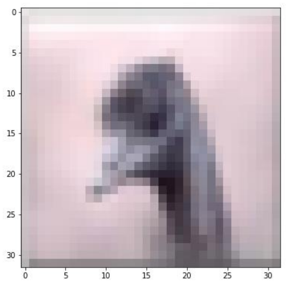
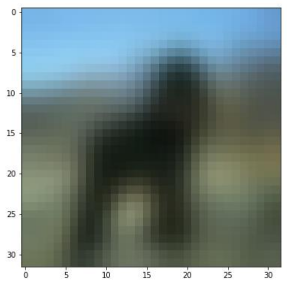
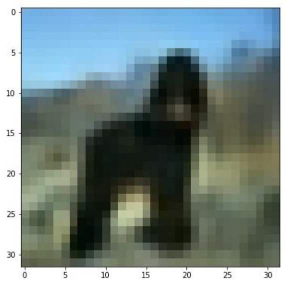
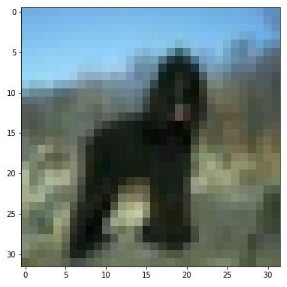
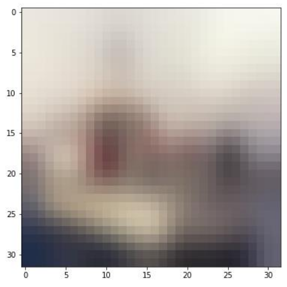
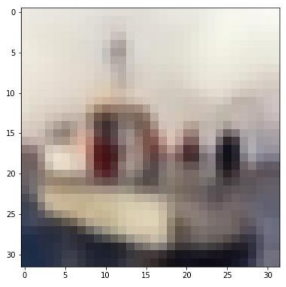
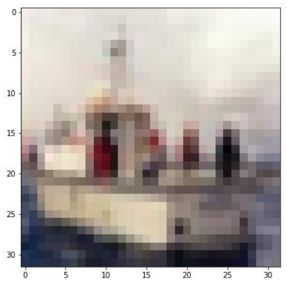
blurred	***predicted***	***sharp***
		
		
		
		

Table 3. Output examples of our model

References

[1] Tsitsulin, Anton & Munkhoeva, Marina & Mottin, Davide & Karras, Panagiotis & Bronstein, Alex & Oseledets, Ivan & Müller, Emmanuel. (2019). Intrinsic Multi-scale Evaluation of Generative Models.
[2] Li, Linyi & Weber, Maurice & Xu, Xiaojun & Rimanic, Luka & Xie, Tao & Zhang, Ce & Li, Bo. (2020). Provable Robust Learning Based on Transformation-Specific Smoothing.
[3] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.
[4] Zhou Wang et al. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: IEEE Transactions on Image Processing (2004).
[5] A. Hore and D. Ziou, "Image quality metrics: PSNR vs. SSIM," in Proc.Int. Conf. on Pattern Recognition (ICPR), 2010, pp. 2366–2369
[6] Estimation lemma. Estimation lemma | Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio, 2021.
[7] Olaf Ronneberger et al. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. arXiv: 1505.04597.