Image Resolution Enhancement using Multi-Step Reinforcement Learning

Budapest University of Technology and Economics

Author:
Máté Nyikovics

Supervisor:
Khalid Kahloot

Abstract

This paper tackles the problem of image resolution enhancement using reinforcement learning with pixel-wise rewards. After the introduction of the deep Q-network, deep reinforcement learning has been achieving great success. However, the applications of deep reinforcement learning for image processing are still limited. Therefore, I try to address this issue by implementing a multi-agent reinforcement learning network which in theory could outperform traditional image upsampling algorithms. This work is largely inspired by a similarly innovative solution called pixelRL. Similarly to the aforementioned paper I treat each pixel as a separate agent, and the agent changes the pixel value by taking an action. I try the proposed method on BSD68 dataset and evaluate the prediction results against bicubic interpolation. My initial results demonstrate that the proposed method achieves comparable or better performance, using PSNR quality measurement.

Introduction

Today, deep learning is a thriving filed with many practical applications such as understanding speech or images, making medical diagnoses, or automating routine labor. These are the problems, that are generally easy for humans to perform but hard to formally describe. The motivation for this paper was to take one such problem, specifically image resolution enhancement, and try to tackle it using novel deep learning methods to get better quality images than by using the traditional algorithms. If achieved, this could have a wide range of real-word applications, from medical image processing to the field of self-driving cars. Image resolution enhancement is the process of recovering high-resolution images from low-resolution ones. In general, this problem is very challenging and inherently ill-posed since there are always multiple high-resolution images corresponding to a single low-resolution one. In literature, there are many deep learning approaches described to solve this problem (prediction-based methods (Keys, 1981) (Duchon, 1979), edge-based methods (Fattal, 2011), statistical methods (Kwon, 2010), etc.). These methods generally fall under the supervised machine learning category. Supervised learning (Supervised learning, 2019) is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. In supervised learning, each example is a pair consisting of an input object and a desired output value. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. In case of image upsampling, the family of these algorithms differ from each other in two major aspects: different types of network architectures and different types of loss functions. The drawback of supervised learning in image processing is that for training you have to have both the high-resolution and the low-resolution image pairs, which is usually achieved by downsampling your high-resolution image prior to training. In this way the downsampling operation effectively plays a role in the achievable outcome. This effect is represented by the following equations:

$$I_x = D(I_y; \delta)$$

$$D(I_y; \delta) = I_y \downarrow_s, \{s\} \subset \delta$$

where I_y is the corresponding high-resolution image and D represents the degradation mapping function with δ as its parameters (e.g. the scaling factor). To combat this effect, I decided to use another machine learning paradigm, namely reinforcement learning. Reinforcement learning (Reinforcement learning, 2019) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. After the introduction of the deep Q-network (Mnih V., 2013), which can play Atari games on the human level, much attention has been focused on deep reinforcement learning, therefore some algorithms also emerged in the image processing field. However, these algorithms threat images as single agents, which means that they cannot apply sophisticated, pixel-wise manipulations to it.

My solution differs from this approach by using the pixels of an image as the agents in the model. During training the agents learn the optimal behavior to maximize the mean of the expected total rewards at all pixels. Each pixel value is regarded as the current state and is iteratively updated by the agent's action. Applying the existing techniques of the multi-agent reinforcement learning is impractical in terms of computational cost because the number of agents is extremely large. I solve this problem by employing the fully convolutional network (FCN). The merit of using FCN is that all the agents can share the parameters, which in terms greatly reduces computational cost. To train the agents I have used the asynchronous advantage

actor-critic (A3C) algorithm. A3C is one of the actor-critic methods, which has two networks: the actor (A) and the critic (C) network. I denote the parameters of each network as θ_A and θ_C respectively. Both networks use the current state s^t as the input, where s^t is the state of the image at time step t. The critic network outputs the value $C(s^t)$, the expected total rewards from state s^t, which shows how good the current state is. The gradient for $\theta_C$is computed as follows:

$$R^t = r^t + \gamma^1 r^{t+1} + \gamma^2 r^{t+2} + \cdots + \gamma^{n-1} r^{t+n-1} + \gamma^n r^{t+n}$$

$$d\theta_C = \nabla_{\theta_C}(R^t - C(s^t))^2$$

where γ^i is the i-th power of the discount factor γ. The actor network outputs the policy $\pi(a^t|s^t)$ of taking action $a^t \in \mathcal{A}$ at time step t. $\mathcal{A}$ is the pre-defined action set. The gradient for the actor network is computed as follows:

$$A(a^t, s^t) = R^t - C(s^t)$$

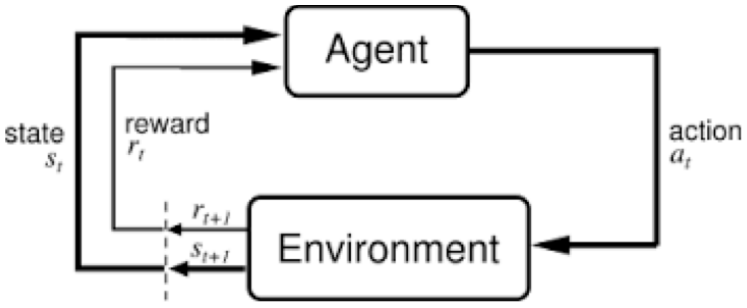
$$d\theta_A = -\nabla_{\theta_A} \log \pi(a^t|s^t) A(a^t, s^t)$$

where $A(a^t, s^t)$ is called advantage. In summary pixel-wise agent training can be summed up in the following process. Let I_i be the i-th pixel of the input image I that has N pixels $i = (1, \cdots, N)$. Each pixel has an agent and its policy is denoted as $\pi_i({a_i}^t|{s_i}^t)$. The starting state is the input image itself ${s_i}^0 = I_i$. The agents obtain the next states $s^{t+1} = ({s_i}^{t+1}, \cdots, {s_N}^{t+1})$ and rewards $r^t = ({r_i}^t, \cdots, {r_N}^t)$ from the environment by taking the actions $a^t = ({a_1}^t, \cdots, {a_N}^t)$. The objective is to learn the optimal policies $\pi = (\pi_1, \cdots, \pi_N)$ that maximize the mean of the total expected rewards at all pixels:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E_{\pi_i}\left(\sum_{t=0}^{\infty} \gamma^t \bar{r}^t\right)$$

$$\bar{r}^t = \frac{1}{N}\sum_{i=1}^{N} {r_i}^t$$

where $\bar{r}^t$ is the mean of the rewards ${r_i}^t$ at all pixels.



1. Figure Reinforcement learning process.

A naive solution for solving this problem would be to train a network that output policies for all possible set of actions a^t. However, it is computationally impractical because the dimension

of the last fully connected layer must be $|\mathcal{A}|^N$, which is too large. Another solution would be to divide the problem into N independent subproblems and train N networks in such way that the i-th agent to maximize the expected total reward at the i-th pixel. However, training N networks is also computationally impractical when N (number of pixels) is large. In addition, it threats only fixed size images. To solve this problem, I employed a fully convolutional network instead of N networks.

To measure the effectiveness of the trained network, I measured the quality of the output images. Image quality in this context refers to visually significant attributes of images and focuses on the perceptual assessments of human viewers. The process of determining image quality is called image quality assessment (IQA). IQA methods (Zhihao Wang, 2019) range from human observer's perceptual evaluation to computational models automatically predicting image quality. These methods aren't usually consistent between each other, because the latter ones are often unable to capture the human visual perception very accurately, which may lead to large difference in IQA results. The IQA metric I decided on using is called peak signal-noise ratio (PSNR). This is a commonly used metric to measure the reconstruction quality of lossy transformations. PSNR is defined via the maximum possible pixel value L and the mean squared error (MSE) between low- and high-resolution image pairs.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(I_i - \hat{I}_i\right)^2$$

$$PSNR = 10 \cdot \log_{10}\left(\frac{L^2}{MSE}\right)$$

In general cases using 8-bit image representations, L equals to 255 and the typical values for the PSNR vary from 20 to 40, where higher is better. When L is fixed, the PSNR is only related to the pixel-level MSE between images, only caring about the difference between the pixel values at the same positions instead of human visual perception. Therein lies its weakness, but due to the necessity to compare performance with literature works, PSNR is currently widely used evaluation criteria.

Method

The first step of implementing the project was finding a suitable image dataset. Luckily nowadays there are a variety of datasets available for image manipulation, which greatly differ in image amounts, quality, resolution and diversity. I settled on using BSD68 grayscale dataset, which contained approximately 500 mid- to high-resolution images. 500 is a relatively small number of images for this kind of task, but my main purpose was to implement a proof of concept model rather than focusing on result quality.

Next, I implemented my image input pipeline in Python. I chose Python language for its ease of use and the fact, that most machine learning libraries support it. This resulted in implementing tree classes:

- *DirectoryStructureManager*, which was responsible for creating/reinstating the necessary directory structure for the preprocessed images to be placed in,

- *ImageDatasetHandler* class, which basically is an interface with the goal of providing consistent image loading from different image datasets, this way ensuring further dataset extendibility,
- *ImageDatasetManager* class, which ties the input process all together, it contains methods from image noise addition to loading batches of preprocessed images.

The algorithm for preprocessing images was the following:

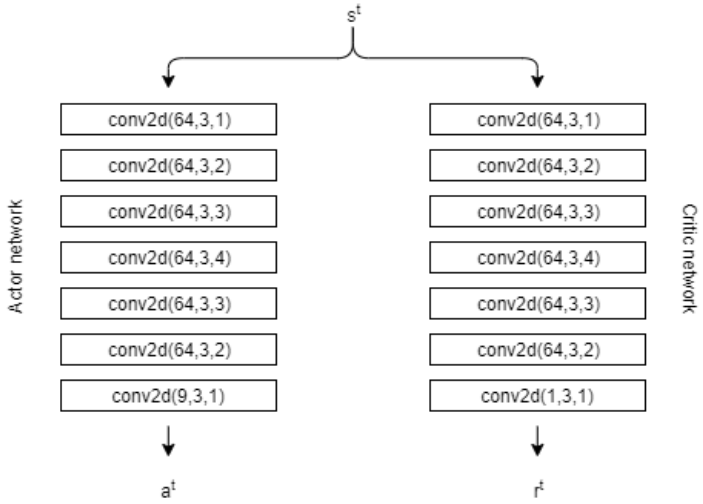
Algorithm 1. Image preprocessing. Preprocesses images by grayscaling them and cutting an arbitrary sized chunk (w, h) out of them.

```
Require: d, preprocessed image directory structure
Require: ih, the image handlers
while ih do
    while i = read_image(d) do
        grayscale(i)
        cut_to_size(i, (w,h))
        write_to_disk(i)
    end while
end while
```

I chose to implement the image processing with Pythons OpenCV (OpenCV, 2019) library, because it showed great performance compared to similar libraries. I would like to note here, that despite of using only grayscale images for training, one still can use this solution to enhance the resolution of images by running each color channel through the network sequentially. After I've got the preprocessed images, I could hand them to a generator function, which in turn took an arbitrary number of them, calculated their upsampled representations and returned the original and upsampled image pairs. My method for imitating upsampling was to replace every 2^{nd} pixel with white values. This would essentially result in doubling the original image size, however by refining this algorithm, the network could work with arbitrary resolution changes.

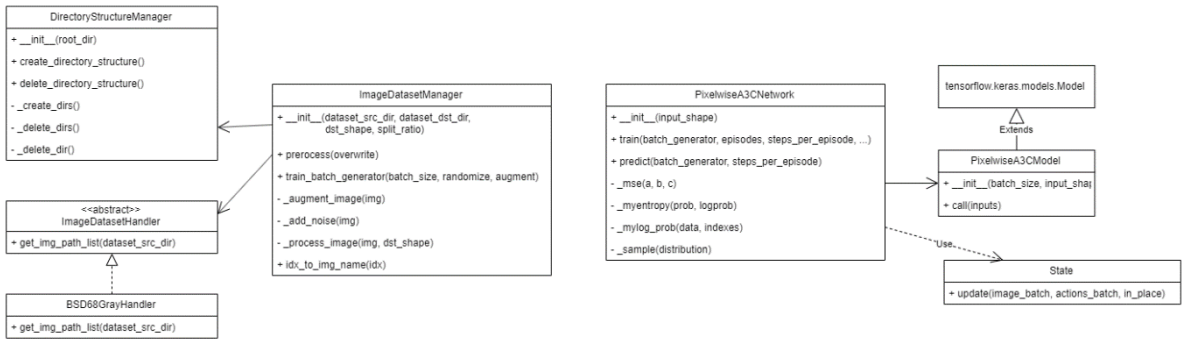
2. Figure The output of the generator function. The prerocessed original image (left) and the upsample imitated (right).

After I was done with implementing the input pipeline, I turned to implementing the neural network. For this part I was largely inspired by another model called pixelRL (Ryosuke Furuta, 2019), a similar reinforcement learning based multi-agent model for image denoising and restoration. Based on the findings detailed in the corresponding paper, I decided not to deviate from the network structure that has been used.



3. Figure Actor-critic model.

Both the actor and critic network are comprised of sequential convolutional layers. The numbers besides the layers are the number of filters, the size of the convolutional kernels, and the dilation rate. On closer observation one can see that they differ only in their parametrization of their last layers. While the actor model has 9 output filters (which precisely equals to the number of possible actions), the critic network has one. This is not surprising, after all the purpose of the actor network is to produce policies (probability distributions of the possible actions), while the critic network judges the current state with one number. A small problem with implementing the pixelRL actor-critic model was that it used chainer (Chainer - A Powerful, Flexible, and Intuitive Framework for Neural Networks, 2019) as a deep learning library. I felt that using Google backed Tensorflow 2.0 (Tensorflow - An end-to-end open source machine learning platform , 2019) would be a better choice for future compatibility and improvability, not to mention the features it brings (ability to run models on mobile platforms like iOS or Android, cross platform, GPU and TPU accelerated computing, etc.). So, I had to take a migration step prior to continuing with the implementation of the training logic, which meant that I had to basically learn not one but two deep learning frameworks. After the migrating step I ended up with the following class diagram:



4. Figure Class diagram.

Now the model was ready to train. For training I've used pictures of size 70 by 70. I found this setting a good compromise between image quality and memory footprint.

5. Figure Preprocessed image sample used for training.

Furthermore, I defined the 9 possible actions in the following way:

	action	filter size	parameter
1	box filter	5x5	-
2	bilateral filter	5x5	$\sigma_c = 1.0, \sigma_S = 5.0$
3	bilateral filter	5x5	$\sigma_c = 0.1, \sigma_S = 5.0$
4	median filter	5x5	-
5	Gaussian filter	5x5	$\sigma = 1.5$
6	Gaussian filter	5x5	$\sigma = 0.5$
7	pixel value += 1	-	-
8	pixel value -= 1	-	-
9	do nothing	-	-

6. Figure Set of actions.

For the optimization process I've used Adam optimizer. The detailed training algorithm is shown below.

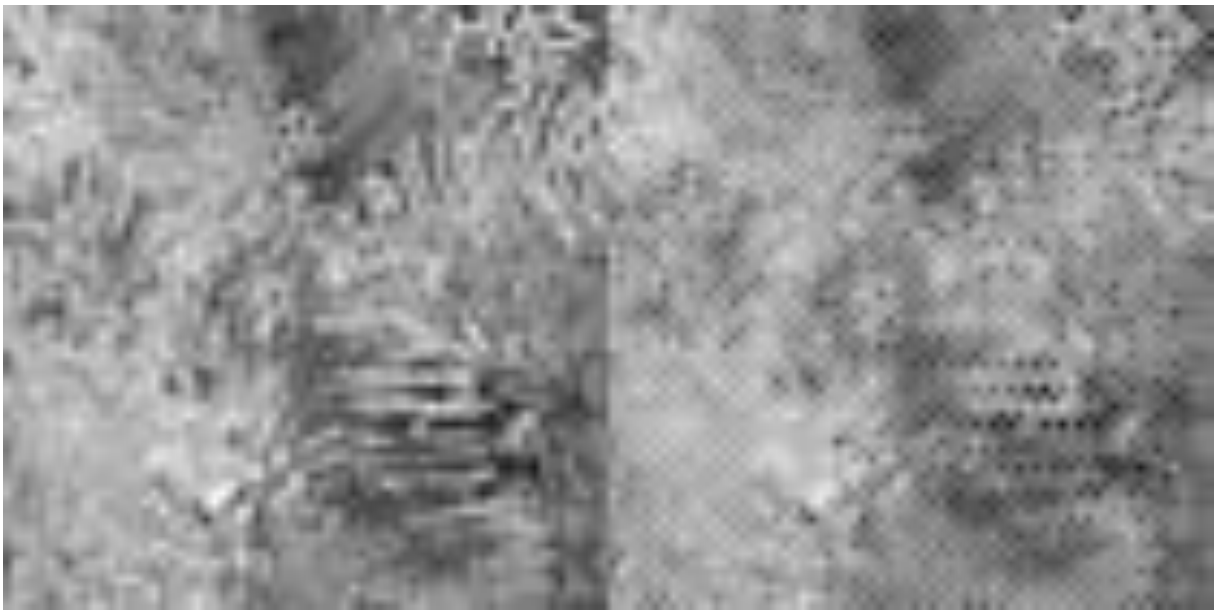
Algorithm 2. Training the model using e number of episodes, e_s exploration steps per episode.

Require: A, actor model
Require: C, critic model
Require: io, original image
Require: ia, augmented image
Require: $\boldsymbol{s^t}$, status
Require: $\boldsymbol{\gamma}$, discount factor
for k = 0,1,..,e **do**
  for i = 0,1,...e_s **do**
    $\boldsymbol{s^i}$ = **ia**
    a = **A**($\boldsymbol{s^i}$)
    c^i = **C**($\boldsymbol{s^i}$)
    $\boldsymbol{s^{i+1}}$ = update($\boldsymbol{s^i}$, sample(**a**))
    r^i = (**io**-$\boldsymbol{s^i}$) · (**io**-$\boldsymbol{s^{i+1}}$)
  end for
  for i = e_s,..,1,0 **do**
    R += $\boldsymbol{\gamma^i} r^i$
    A = R-c^i
  end for
$d\theta_C = \nabla_{\theta_C}(R - C(s^t))^2$
$d\theta_A = -\nabla_{\theta_A} \log \pi(a^t|s^t) A(a^t, s^t)$
backpropagate(**A**, $d\theta_A$)
backpropagate(**C**, $d\theta_C$)
end for

Results and Analysis

After training the network for 1000 episodes (with 4 steps per episode) the network produced the following results.

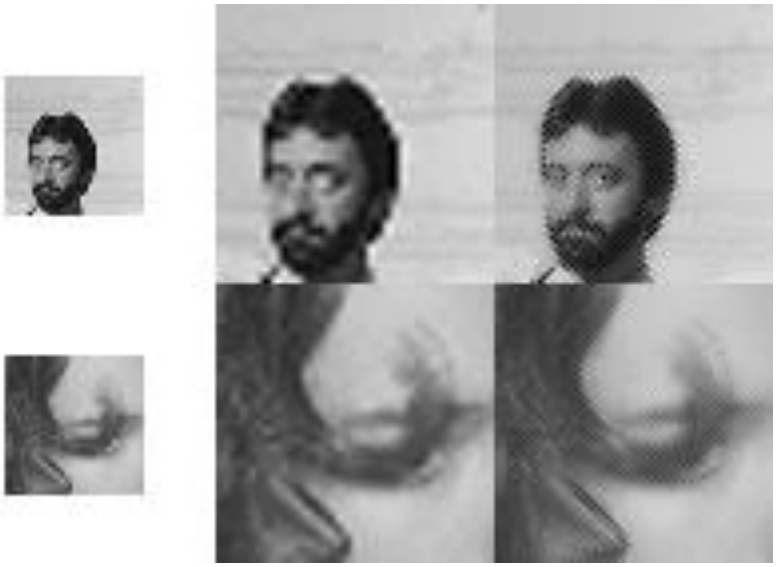
7. Figure Prediction results 1. High-resolution image (l), predicted (r).



8. Figure Prediction results 2. High-resolution image (l), predicted (r).

I was very happy with these, as it showed that the theory is working, although it's easy to see that there is room for improvement, because the images did not become crystal clear. To further investigate the performance of the network, I was also curious how these results compare to an algorithmic approach. As mentioned in introduction, I've picked bicubic interpolation as my baseline test algorithm. As for my next step, I gathered 20 original images o, downsampled them using basic nearest neighbor interpolation, these became my small image batch s. Then I fed the original images o (after preprocessing) into the network, the output became my predicted image batch p. Meanwhile I also used bicubic interpolation on o to upsample them

u. After executing all these manipulations, I measured PSNR using $o|u$ and $o|p$ image pairs. The result of this calculation can be found in 1. Table PSNR values.1. Table.



9. Figure The low-resolution image(s), the bicubic interpolated (u), network predicted (p).

On the images themselves it's easy to see that the interpolated image u has smoother edges, but the predicted image p seems to have preserved more of the original quality of the image. This observation is proved by looking into the PSNR values of the predictions shown in 1. Table. The predicted image quality values are mostly higher than the interpolated ones, but the outcome of the two is definitely comparable.

Conclusion

The goal of this project was to implement a proof of concept image resolution enhancing method using reinforcement learning. With the proposed multi-agent model, I managed to achieve this goal. I admit that there is still room for improvement, for example I would like to train the model on a larger dataset, investigate different image augmenting methods and action sets, and also implement a new way of calculating the rewards during training called reward map convolution. Despite these, the model has its advantages (e.g. interpretability, which can be extremely important in critical applications like medical image processing), and it managed to keep the same level of image quality during upsampling as a traditional algorithm.

References

Chainer - A Powerful, Flexible, and Intuitive Framework for Neural Networks. (2019. 12). Forrás: chainer.org: https://chainer.org/

Duchon, C. E. (1979). *Lanczos filtering in one and two dimensions* (Vol. 18). Journal of Applied Meteorology.

Fattal, G. F. (2011). *Image and video upscaling from local self-examples* (Vol. 30). TOG.

Keys, R. (1981). *Cubic convolution interpolation for digital image processing* (Vol. 29). IEEE Transactions on Acoustics, Speech, and Signal Processing.

Kwon, K. I. (2010). *Single-image super-resolution using sparse regression and natural image prior* (Vol. 32). TPAMI.

Mnih V., K. K. (2013). *Playing atari with deep reinforcement learning.* NIPS Deep Learning Workshop.

OpenCV. (2019, 12). Retrieved from https://opencv.org/

Reinforcement learning. (2019, 12). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Reinforcement_learning

Ryosuke Furuta, N. I. (2019. 12). *Fully Convolutional Network with Multi-Step Reinforcement Learning for Image Processing.* Forrás: arxiv.org: https://arxiv.org/abs/1811.04323

Supervised learning. (2019, 12). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Supervised_learning

Tensorflow - An end-to-end open source machine learning platform . (2019. 12). Forrás: tensorflow.org: https://www.tensorflow.org/

Zhihao Wang, J. C. (2019). Deep Learning for Image Super-resolution: A Survey. *IEEE*.

Appendices

1. Table PSNR values.

image nr.	$PSNR_{bicubic\ interpolation}$	$PSNR_{\text{network prediction}}$
1	22.47798177213141	24.958952802279562
2	25.480251280719965	26.736054424046344
3	25.326720402383245	28.105546305433123
4	27.336362541891887	28.73304458296839
5	24.476852197842124	26.901553407422895
6	18.25316721600761	21.1848304036375
7	25.679692384417155	27.183276706357322
8	35.15802344639418	32.60007657018241
9	21.827023050151237	24.780935065609746
10	20.81296452846209	24.10112607054638
11	22.873424002222336	24.990535470273777
12	23.403149589634864	25.764490285389577
13	32.90460496789871	32.020510566747156
14	29.39821461419417	30.380349068751382
15	30.131409066139142	32.01571641836855
16	37.290661442275066	36.36032934670061
17	26.32341405825224	28.869532456139538
18	21.608480353210517	24.807293045548263
19	18.242838745868976	21.086792607706602
20	22.04986347912229	24.294342335004814
mean:	25.55275495696096	27.29376439695569