Data Representation, Reduction and Analysis

Project - Image Super Resolution

A theoretical and practical approach

Members

Andres Sánchez 0553771

Deniz Alp Atun 0552182

İsmail Göktuğ Serin 0552218

{ andres.sanchez , deniz.alp.atun , ismail.goktug.serin } @vub.be

Table of Contents

1. Introduction to Image Super Resolution

Recovering a High resolution image from a Low Resolution image is a classical problem in computer vision. This problem has multiple solutions, from a LR image can be obtained multiple HR images converting this problem in an ill-posed problem. This is because the processes that convert the HR image in a LR image is not reversible.

The problem of having multiple solution is mitigated constraining the solution space using prior information, to learn this prior the methods presented in this project uses an example-based strategy, learning mapping functions from a dataset of low- and high-resolution example pairs.

2. Background

2.1. Problem definition

The goal of SR Image model is to recover the HR image (SR image) from a LR one, the low resolution is the output of a degradation process, which is generally unknown and non-invertible. Therefore, the SR model is required to recover the HR image that is identical to the ground truth image. To assess the quality of the SR image generated by the model with respect to the HR image (ground truth), in this project PSNR is used for Image Quality Assessment.

2.2. Peak Signal-to-Noise Ratio

Peak Signal-to-Noise Ratio is used to measure the quality of the reconstruction of lossy transformation, in this case the image up sampling.

The definition is done using the maximum pixel value, and the MSE (Mean Squared Error) between images. Given SR Image and HR image, both with the same size (number of pixels), the MSE and PSNR (in dB) between the SR and HR images is given as shown below.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(I(i) - \hat{I}(i))^2,$$

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

In RGB with 8 bits representations the maximum value of a pixel is 255 and for YCbCr the maximum value is 1.

2.3. Datasets

For the problem of training a model to generate a SR image, a data set of images in HR and LR are needed. For this, there exists repositories such us DIV2K which provides both or BSD300 which provides only HR image in which case the LR image is generated using a resize function in python provided by PyTorch.

In this project, for all the repositories used only the HR image is used and the LR image is generated using python, by cropping a specified area in the center of the input image.

2.4. Operating Channels

In addition to the commonly used RGB color space, the YCbCr color space is also widely used for representing images and performing super-resolution. It is explained that with respect to the operating channels RGB and YCbCr used on training or evaluating, some difference may be observed (up to 4+ dB). [5]

2.5. Super Resolution Framework

The process of learning features from a dataset of low- and high-resolution example pairs can be done using multiple techniques and CNN framework, for the SR Image generation from a LR Image, at some point this image needs to be upscaled to meet the desired size. There are two alternatives, one is to upscale the LR image using an interpolation technique, then apply a CNN that will produce the SR Image. The other alternative is to feed a LR Image directly to a CNN, and this CNN will produce a SR Image with the desired size as its output.

To illustrate the difference, two framework implementation are referenced here, for the first approach SRCNN uses bicubic interpolation to upscale image and the method implemented in this project is ESPCN that learns the features from the LR image[4].

2.6. Convolution operator in a CNN architecture

The basic CNN unit has three elements an input, an output and a kernel, usually there is an element wise non-linear operation within the CNN unit, the figure below shows the basic unit [4].

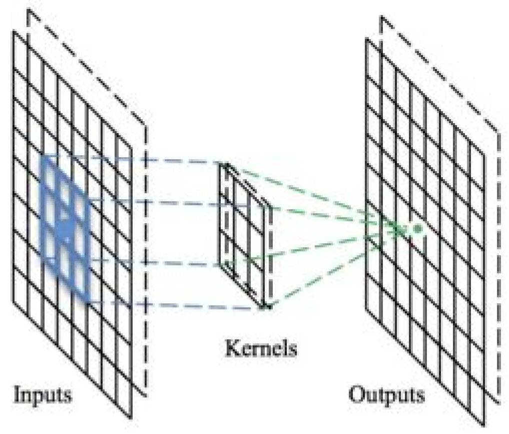


Figure 1: Basic CNN unit.

The notation used to describe the CNN unit is the following:

Input Feature

$$X(i, j, k),\ 0 \le i < Hin,\ 0 \le j < Win,\ 0 \le k < Cin,$$

Where Hin, Win and Cin denote the height, width and channel number of input.

Kernel Tensor

$$K(l, k, m, n),\ 0 \le m < M,\ 0 \le n < N$$

Where M and N is the height and width of convolutional kernel, l and k denote the channels of output and input.

Output Feature:

$$Y (i, j, l),\ 0 \le i < Hout,\ 0 \le j < Wout,\ 0 \le l < Cout.$$

If the input is sampled in intervals of s step in each direction and pad the input to maintain its size, the convolution operator is formulated as

$Y (i, j, l) = \text{Sum } (m,n,k)\ K(l, k, m, n)X((i-1)s + m,(j-1)s + n, k).$

Or with s= 1, simply

$Y (i, j, l) = \text{sum } K(l, k, m, n)X(i + m - 1, j + n - 1, k).$

2.7. SRCNN

The architecture proposed in SRCNN [3] is a three-layer CNN where the filter of each layer is presented below, according to the notation presented in the convolution operator:

First layer kernel:	64×1×9 ×**9**
Second layer kernel:	32×64×5×**5**
Third layer kernel :	1×32×5×**5**

The functions of these three nonlinear transformations are patch extraction, nonlinear mapping and reconstruction. The loss function for optimizing SRCNN is the mean square error (MSE), the architecture shown in the image below is the one that the authors refers as SRCNN 9-5-5, the numbers comes from the size of the kernels used at each layer.

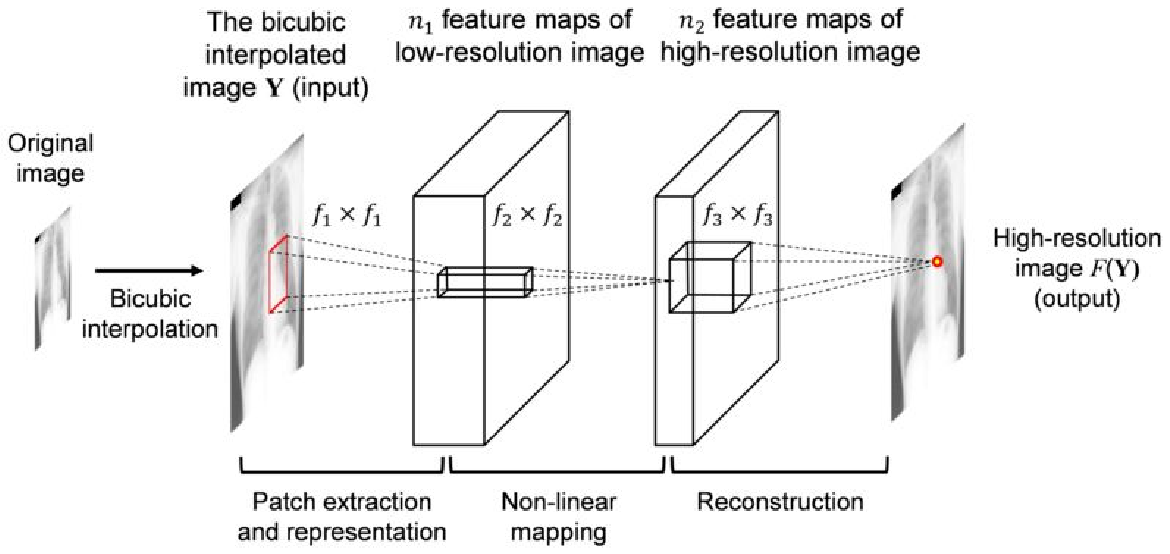

The bicubic interpolated image $\mathbf{Y}$ (input)
n_1 feature maps of low-resolution image
n_2 feature maps of high-resolution image
Original image
Bicubic interpolation
$f_1 \times f_1$
$f_2 \times f_2$
$f_3 \times f_3$
High-resolution image $F(\mathbf{Y})$ (output)
Patch extraction and representation
Non-linear mapping
Reconstruction

Figure 2: SRCNN (9-5-5) architecture.

Although, the simplicity of its three layer architecture, SRCNN has shown superiority over other methods because of the CNN strong capability of effective feature learning.

The main issue with this implementation is that the input of SRCNN is a version of LR Image that was upscaled using Bicubic interpolation. This can have three drawbacks [4]:

1. The image structure can be wrongly estimated due to the operation performed over the original input image.
2. With an image that already has the size of the SR image, the process of feature extraction will take more time.
3. When downsampling kernel is unknown, one specific interpolated input as raw estimation is not reasonable.

With these drawbacks in mind, one idea that arise to tackle them, is to design a CNN architecture that takes the LR image directly as its input, as discussed in the Super Resolution Framework section.

2.8. ESPCN

ESPCN [1] proposes a CNN with a LR Image as input with *L* layers, the first l-1 layers, the input LR image goes through fl×fl convolution and obtain nl-1 feature maps. At the last layer, an efficient sub-pixel convolution is performed to get back the HR image at the output with a r upscale factor.

Specifically, the authors propose a 3 three layer architecture described below:

First layer kernel:	64×1×5×5
Second layer kernel:	32×64×3x3
Third layer kernel :	1×32×3x3

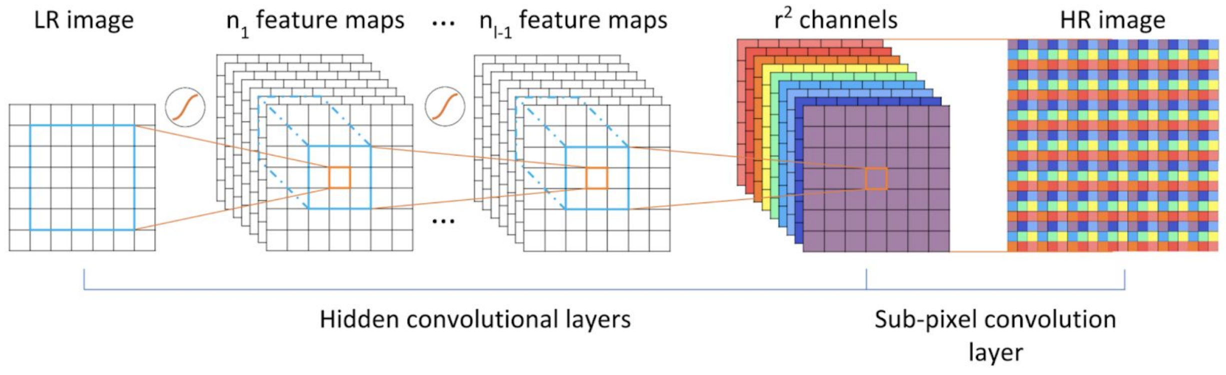


Figure 3 : Illustration of the ESCPN framework where r denotes the upscaling ratio

The input and the output of the model are images represented using YCbCr but only the Y component is used.

The last layer has r^2 feature maps that has the information to generate the SR Image, to do so the channels are reshaped using phase shift. Phase shit is a periodic shuffling operator that rearrange the elements of the last layer into a SR image.

For a better understanding of how the periodic operator works [3], refers to the image below where the last layer of the architecture has 4 channels (r=2), and the periodic shuffling operation is used to reshape the output channels to the SR image with a size of 8x8X1. It generalizes to any kernel shape of dimension (o, i, k, k) and rescale ratio r, in the case of the presented architecture the kernel shape is (1, r^2=9,3,3).

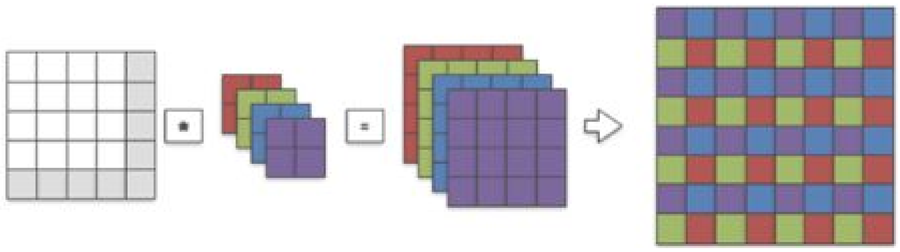


Figure 4 : Full view of the proposed subpixel convolution using just convolution

In the training process the last part instead of shuffling the output to generate the image, the training data can be pre-shuffled to match the output before PS.

ESPCN is the architecture implemented in this project using Pytorch, more about it in section 3.

2.9. Bicubic Interpolation

Bicubinc interpolation is used to interpolated data points in a two dimension regular grid, It is proven to be superior to other interpolation techniques, such as bilinear interpolation or nearest-neighbor interpolation, in terms of the smoothness of the generated surface. It is widely used in image processing when the speed is not an issue.

In the context of this project the HR Images are generated from LR Images using Bicubic interpolation to use this technique as benchmark.

3. Python Implementation of ESPCN

Before we begin, some clarifications are necessary to understand why our implementation code has similarities with the example that will be mentioned and why we decided on using PyTorch instead of TensorFlow.

3.1. Why Pytorch?

After our initial research, we found out there wasn’t much difference between PyTorch and TensorFlow implementations of image super resolution. The installation of TensorFlow was problematic and failed on our 3 uniquely different machines (CUDA supporting laptop, Intel iGPU laptop and a MacBook).

That initial experience led us to using PyTorch, since we’ve already been using Anaconda and it was a simpler process.

3.2. Similarities with Example

After installation and writing a small test code, it became apparent that libraries for small operations and functions provide more freedom in terms of implementation to a code; while libraries that do more complex operations and provide more complex functions tend to have stricter rules, definitions and inputs-outputs to follow.

This problem presented two solutions:

- Accepting most of the strictness of PyTorch whenever an error is presented and understanding the implementation.
- Trying to make everything unique.

At first we went with the second solution, but “loading dataset” into “DataLoader” quickly presented that it was a naive approach. After examining other repositories as well, we came into the realisation of making everything unique would take more time than we have and every other implementation of PyTorch also shows similarity with the example repository [7].

Thus we followed the first solution. While writing the code we investigated every line, understood the functionalities, and implementation. We added our value whenever possible.

3.3. Code implementation

We can generalise the process in two parts: Training and Super Resolution.
In terms of files there are 3 main files and 1 script folder, where scripts are being shared with training and SR parts.

Training (train.py)
Training starts with acquiring datasets. We’ve used to DIV2K datasets as suggested since it provided various datasets and their high resolution counterparts.

This acquisition process is composed of location, extraction, processing and finally loading.

- Location (dir_input function) is asked as an input and at that point the directory structure is also asked to the user.
- After that, extraction (scripts\file_process.py) extracts images to a temporary folder “extr” where images are separated into folders of training and validation.
- Now that files are extracted, they need to be prepared to be loaded.
  The preparation (scripts\dataset_maker.py) takes the arguments of crop and upscale that is provided by user. These arguments are used to crop and transform an image to be input (cropped/downscaled) and target (desired result).
  After that they are passed to compose to be formed as an acceptable class object.
- At loading part, PyTorch takes the class object and loads it to another acceptable object.

After being loaded to an appropriate object, last requirements of training are initialised. Such as determining the usage of cuda or CPU, which activator function/model to use, optimiser algorithm and paths/files to store and log results.

Finally, loaded training data is used to train the respective model. For every iteration of epoch, the loss is saved and the next step is trained.

After a training of epoch is done, the loaded validation data is used to provide an insight about how well is that epoch model behaving against the validation set. The results are also logged and saved.

Super-Resolution (super_res.py and batch_SR-PSNR_calculation.py)
Both mentioned super_res and batch_SR files are using the same basis of code. The only difference is super_res only takes one input and upscales it (depending on compare parameter also calculates PSNR and prints) while batch_SR takes two zip files as an input, one low resolution images to be upscaled and one high resolution images to be compared to.

Upscaling (super_res.py)
Compared to train.py, upscaling is very simple in nature.
First it takes an image as input and converts it into YCbCr format. (This format represents image colors in 3-dimensional space, which is represented in python as 2D array with 3 channels)
Then splits the image to use the y part for upscaling.

Before the upscaling process begins, the model path that was passed as an argument is used to load model, whether cuda is passed as parameter and if it can be used or not is checked.

Then the y part that is passed to the tensor is passed into the model, yielding y values that are transformed according to the model. All values are multiplied by 255 (for RGB reasons).
The values that pass 255 are clipped and final y results are yielded.

Last process is to increase the size of cb and cr parts to match with the new size, merge the components together and save the new image.

Batch_SR also follows the exact same route for upscaling, the only difference is that it takes two zip files and matches the name of image from HR zip file with the name of image from LR zip file to compare.

Comparison using PSNR

Below equations are used while calculating PSNR in our implementation [6].

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

4. Experiments for fine-tuning and selecting our model

4.1. Choosing of model parameters for experiments

We created different instances of the ESPCN model running with different input parameters of crop-size, activation function, image dataset and learning rate. By observing and comparing the output values (namely PSNR) for those different model instances trained on the training and validated on the validation set, we decided on the model instance that will be used for testing.

For each instance of the model, the upscaling factor was selected as 3, parallel to the model described in the paper by Shi. The number of epochs were selected as 30 since no major improvements were observed after the 30th object on the dataset (except for the experimental models which run with different learning rates - they were allowed more iterations to converge). Train batch size was selected as 4 in order to enhance performance (speed) improvements for the computer that runs the code (it has 4 cores, thus could run perform 4 operations at a time smoothly). The validation batch size was selected as 100 (which equals to total number of images in the validation set), by doing so we had only one PSNR value per epoch. By fixing the parameters as explained above, we were able to provide a basis for comparison among different instances of the model. We made our experiments using alternative values for the parameters: crop-size, activation function, image dataset and learning rate. PSNR results for each model instance was logged and model with the best PSNR value was chosen to be used in the testing dataset.

The parameters used in the model can be explained in two groups, as below:

A) Parameters that are fixed in all instances:
Upscaling factor: 3
Train_batch_size = 4,
Validation_batch_size = 100,
No_of_epochs = 30 (100 for models with different learning rates)

B) Parameters that vary in different instances:
Crop-size: 256, 128, 64
Activator Functions: tanh, relu, leaky_relu 0.01
Dataset:
DIV2K_train_HR. & valid_HR,
DIV2K_train_LR_unknown_X3 & DIV2K_valid_LR_unknown_X3
Learning rate: 0.001, 0.0001, 0.01
No of epochs: 30 for the models where default lr is used as in Shi paper (lr=0.001),
100 for the models where other lr values are experimented, factor of x10 bigger and lower vs default lr value (lr=0.01 and lr=0.0001 vs lr=0.001 default)

4.2. Comparison of obtained results of different models

The comparisons of the results will be based solely on the PSNR values in this assignment. Apart from PSNR values, very different run-times observed to train different models, but this will only be noted in detail below, and will not be investigated further as our primary focus is PSNR.

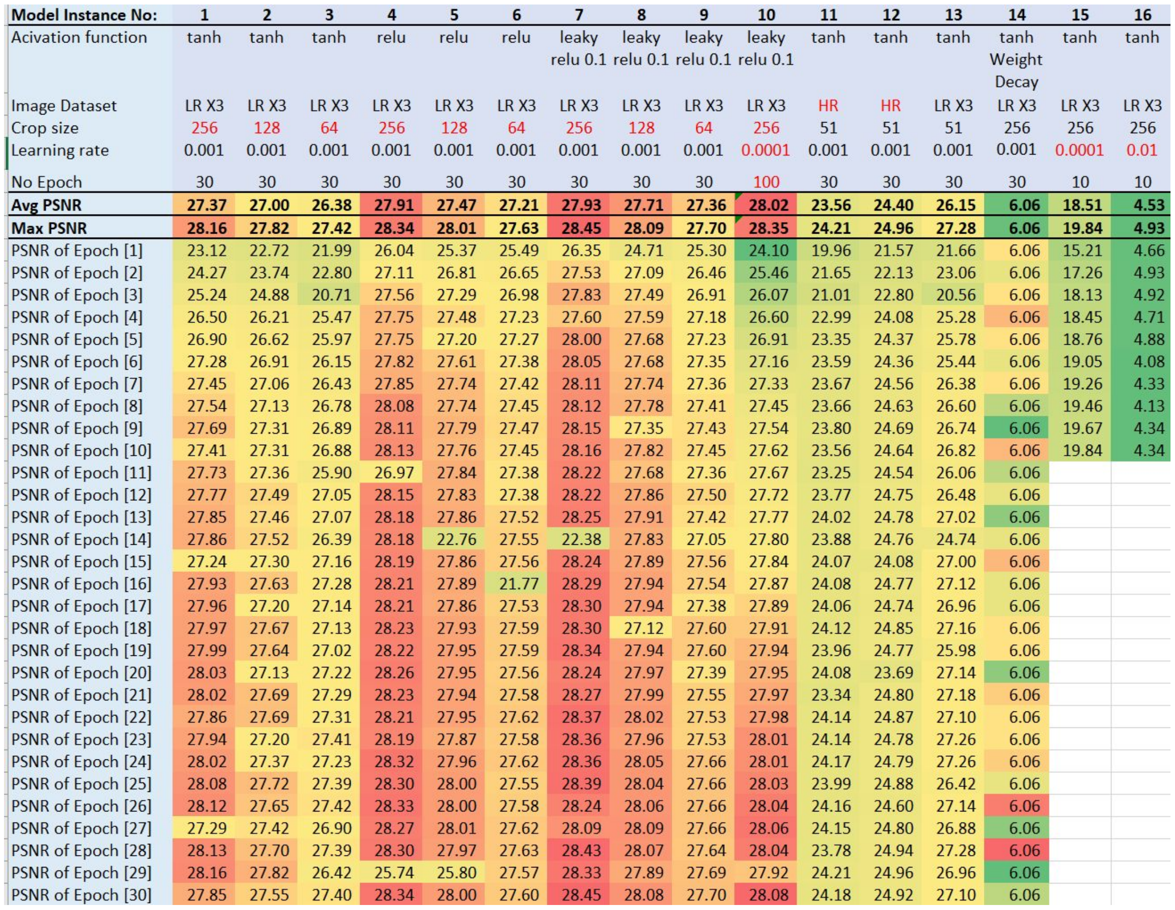
RunTimes

- Activity functions (models with tanh took notably longer to train vs models with relu and leaky relu activation functions) - as it requires much math-heavy computations.

- Models that take HR images as inputs took much longer - as their processing workload was much more.

- Models with smaller learning rates took longer as their gradient descent function needs to take many steps to reach local minima (number of epochs were much bigger)

Results obtained from different instances of the model can be summarized in the below table.



Model Instance No:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Acivation function	tanh	tanh	tanh	relu	relu	relu	leaky relu 0.1	leaky relu 0.1	leaky relu 0.1	leaky relu 0.1	tanh	tanh	tanh	tanh Weight Decay	tanh	tanh
Image Dataset	LR X3	LR X3	LR X3	LR X3	LR X3	LR X3	LR X3	LR X3	LR X3	LR X3	HR	HR	LR X3	LR X3	LR X3	LR X3
Crop size	256	128	64	256	128	64	256	128	64	256	51	51	51	256	256	256
Learning rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.0001	0.001	0.001	0.001	0.001	0.0001	0.01
No Epoch	30	30	30	30	30	30	30	30	30	100	30	30	30	30	10	10
Avg PSNR	**27.37**	**27.00**	**26.38**	**27.91**	**27.47**	**27.21**	**27.93**	**27.71**	**27.36**	**28.02**	**23.56**	**24.40**	**26.15**	**6.06**	**18.51**	**4.53**
Max PSNR	**28.16**	**27.82**	**27.42**	**28.34**	**28.01**	**27.63**	**28.45**	**28.09**	**27.70**	**28.35**	**24.21**	**24.96**	**27.28**	**6.06**	**19.84**	**4.93**
PSNR of Epoch [1]	23.12	22.72	21.99	26.04	25.37	25.49	26.35	24.71	25.30	24.10	19.96	21.57	21.66	6.06	15.21	4.66
PSNR of Epoch [2]	24.27	23.74	22.80	27.11	26.81	26.65	27.53	27.09	26.46	25.46	21.65	22.13	23.06	6.06	17.26	4.93
PSNR of Epoch [3]	25.24	24.88	20.71	27.56	27.29	26.98	27.83	27.49	26.91	26.07	21.01	22.80	20.56	6.06	18.13	4.92
PSNR of Epoch [4]	26.50	26.21	25.47	27.75	27.48	27.23	27.60	27.59	27.18	26.60	22.99	24.08	25.28	6.06	18.45	4.71
PSNR of Epoch [5]	26.90	26.62	25.97	27.75	27.20	27.27	28.00	27.68	27.23	26.91	23.35	24.37	25.78	6.06	18.76	4.88
PSNR of Epoch [6]	27.28	26.91	26.15	27.82	27.61	27.38	28.05	27.68	27.35	27.16	23.59	24.36	25.44	6.06	19.05	4.08
PSNR of Epoch [7]	27.45	27.06	26.43	27.85	27.74	27.42	28.11	27.74	27.36	27.33	23.67	24.56	26.38	6.06	19.26	4.33
PSNR of Epoch [8]	27.54	27.13	26.78	28.08	27.74	27.45	28.12	27.78	27.41	27.45	23.66	24.63	26.60	6.06	19.46	4.13
PSNR of Epoch [9]	27.69	27.31	26.89	28.11	27.79	27.47	28.15	27.35	27.43	27.54	23.80	24.69	26.74	6.06	19.67	4.34
PSNR of Epoch [10]	27.41	27.31	26.88	28.13	27.76	27.45	28.16	27.82	27.45	27.62	23.56	24.64	26.82	6.06	19.84	4.34
PSNR of Epoch [11]	27.73	27.36	25.90	26.97	27.84	27.38	28.22	27.68	27.36	27.67	23.25	24.54	26.06	6.06		
PSNR of Epoch [12]	27.77	27.49	27.05	28.15	27.83	27.38	28.22	27.86	27.50	27.72	23.77	24.75	26.48	6.06		
PSNR of Epoch [13]	27.85	27.46	27.07	28.18	27.86	27.52	28.25	27.91	27.42	27.77	24.02	24.78	27.02	6.06		
PSNR of Epoch [14]	27.86	27.52	26.39	28.18	22.76	27.55	22.38	27.83	27.05	27.80	23.88	24.76	24.74	6.06		
PSNR of Epoch [15]	27.24	27.30	27.16	28.19	27.86	27.56	28.24	27.89	27.56	27.84	24.07	24.08	27.00	6.06		
PSNR of Epoch [16]	27.93	27.63	27.28	28.21	27.89	21.77	28.29	27.94	27.54	27.87	24.08	24.77	27.12	6.06		
PSNR of Epoch [17]	27.96	27.20	27.14	28.21	27.86	27.53	28.30	27.94	27.38	27.89	24.06	24.74	26.96	6.06		
PSNR of Epoch [18]	27.97	27.67	27.13	28.23	27.93	27.59	28.30	27.12	27.60	27.91	24.12	24.85	27.16	6.06		
PSNR of Epoch [19]	27.99	27.64	27.02	28.22	27.95	27.59	28.34	27.94	27.60	27.94	23.96	24.77	25.98	6.06		
PSNR of Epoch [20]	28.03	27.13	27.22	28.26	27.95	27.56	28.24	27.97	27.39	27.95	24.08	23.69	27.14	6.06		
PSNR of Epoch [21]	28.02	27.69	27.29	28.23	27.94	27.58	28.27	27.99	27.55	27.97	23.34	24.80	27.18	6.06		
PSNR of Epoch [22]	27.86	27.69	27.31	28.21	27.95	27.62	28.37	28.02	27.53	27.98	24.14	24.87	27.10	6.06		
PSNR of Epoch [23]	27.94	27.20	27.41	28.19	27.87	27.58	28.36	27.96	27.53	28.01	24.14	24.78	27.26	6.06		
PSNR of Epoch [24]	28.02	27.37	27.23	28.32	27.96	27.62	28.36	28.05	27.66	28.01	24.17	24.79	27.26	6.06		
PSNR of Epoch [25]	28.08	27.72	27.39	28.30	28.00	27.55	28.39	28.04	27.66	28.03	23.99	24.88	26.42	6.06		
PSNR of Epoch [26]	28.12	27.65	27.42	28.33	28.00	27.58	28.24	28.06	27.66	28.04	24.16	24.60	27.14	6.06		
PSNR of Epoch [27]	27.29	27.42	26.90	28.27	28.01	27.62	28.09	28.09	27.66	28.06	24.15	24.80	26.88	6.06		
PSNR of Epoch [28]	28.13	27.70	27.39	28.30	27.97	27.63	28.43	28.07	27.64	28.04	23.78	24.94	27.28	6.06		
PSNR of Epoch [29]	28.16	27.82	26.42	25.74	25.80	27.57	28.33	27.89	27.69	27.92	24.21	24.96	26.96	6.06		
PSNR of Epoch [30]	27.85	27.55	27.40	28.34	28.00	27.60	28.45	28.08	27.70	28.08	24.18	24.92	27.10	6.06		

Figure 5 - Summary PSNR table for models trained with different parameters

Some conclusions we drew and some general tendencies we observed in the comparison table can be summarized as below:

- All other parameters being same, models with a bigger crop size (256 pixels) outperformed the ones with lower crop sizes. It may look contradictory to what was mentioned in Shi paper, but in our case the cropped image was just a small area in the center of the original image, whereas in the paper they used entire area of the image by chopping it into pieces defined by crop size. Our interpretation is that since larger crop size images enable training with more data, the models using those images were able to learn more features.

- All other parameters being the same, models using leaky relu 0.1 as activation function outperformed the models using tanh and relu (This was also observed in Shi paper). We do not know how to interpret this exactly, but an explanation could be that x value, in our f(x) may not have as many negative values as much as positive ones.

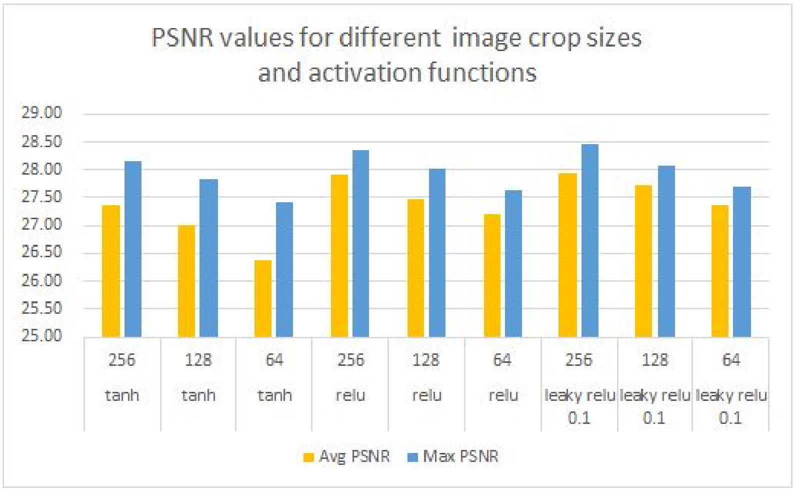


Figure 6 - PSNR results obtained from using different crop sizes and activation functions

- All other parameters being same, models using 0.001 learning rate (default lr value in Shi paper) outperformed the models with lr=0.01 and lr=0.0001 (x10 factor bigger and x10 factor smaller). It is obvious that 0.001 learning rate is optimal for the model. The model with a 10-factor smaller learning rate did not yield better results even with a 10 times increased epoch number. The gradient descent reached the local optima with learning rate of 0.001 (Max PSNR 28.45 vs 28.35, differences almost negligible).

- All other parameters being the same, using LRx3 images as dataset yielded in better PSNR values vs using HR images. This may be due to the fact that when using LR images (LRx3) as target image in our model, to create the input image (since no other image is fed to the model as the LR version), the target image is firstly downscaled by a factor of 3 (resulting in LRx9), the features are extracted on the downscaled image and then it is upscaled using pixel shuffling. Finally the resulting image (SR) is compared with the original input (target) image. The room for improvement from LRx9 to LRx3 image is much bigger compared to LRX3 to HR. Also it is possible to say that an image with a x3 lower resolution is more likely to include more features when a fixed crop size approach is applied independent of the target image resolution.

- An experimental model with weight decay of 1000 performed badly. During training of the model, we passed no arguments to nn.optimizer of PyTorch. We were curious about how the regularization was handled in the model and therefore decided to experiment with passing a weight decay parameter to the PyTorch model, to check if it would yield any improvements. It worsened our results dramatically, which means that the default PyTorch optimizer thanks into account the regularization parameter.

5. Testing ESPCN Implementation

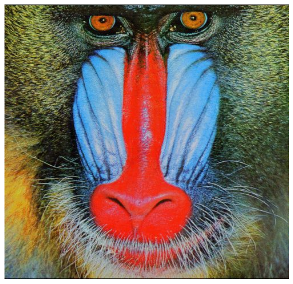
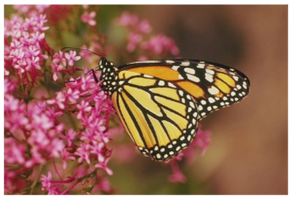
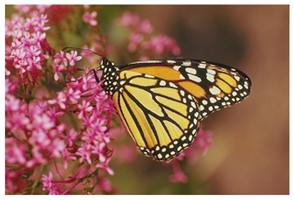
Original	**Tanh**	**Relu**	**Leaky Relu**
	PSNR: 22.3833	PSNR: 22.4409	PSNR: **22.54378**
	PSNR: 21.8976	PSNR: 21.9524	PSNR: **21.94637**
	PSNR: 29.1165	 PSNR: 29.3605	 PSNR: **29.7097**

Figure 7 - PSNR results obtained after running trained ESPCN model on some of the images from BSD300 dataset as used in [1] using the best performing parameters

Even though we obtained relatively good results, our results were on average 2-3 db below vs results obtained in the paper using BSD300 dataset [1]. Also our results were not able to outperform the bicubic interpolation results in Div2K dataset. Plain bicubic interpolation resulted in an average of 32.6db in DIV2K dataset, whereas our best performing model could score 28.45db. We have some predictions as to why this might be so. These are discussed in the remarks section.

6. Remarks

Comparing to the Shi paper, we are using a different approach of cropping the center of the images with a specified size, and using only that area for learning the features. That approach is used in order to limit the size and reduce the complexity of features in a single input. An entire image may be too complex for a model to learn and it will consist of too many features to learn at once. In Shi paper, the image is described to be divided into pieces and all parts of it is used separately. This on the other hand, will lead to a much heavier computational workload.

Another difference vs the reference paper [1], is that in the paper the images are low pass-filtered (blurred) with Gaussian distribution which helps the model to learn a smoother way to fill-in missing pixels while upscaling. Our implementation does not include such an approach. We think it may be deteriorating our PSNR results.

50.000 images are used in the Shi paper (model trained for a week), however in our case 800 images are used (selected model training lasts roughly 1 hour). The gap between the obtained PSNR result therefore is not very disappointing.

7. Conclusion

This project has been a very interesting and challenging assignment for us to work on and it helped us to facilitate our understanding of the concept of convolutional neural networks, and image super resolution both in theory and in practice. It familiarized us with a very powerful ML package PyTorch and helped us to develop our skills in Python. Nevertheless, thought us about methods of image super resolution ESPCN and SRCNN.

ESPCN presents a solution to the problem left by SRCNN where the LR image needs to be upscaled first, with ESPCN the features learning is performed directly from the LR Image, with this the problems generated by a HR size image, and the detail smoothing effects generated by the upscaling operation are solved.

In the last layer of the ESPCN architecture in order to generate the SR Image, the information from the r^2 learned feature maps is used with a Periodic Shuffling operation to rearrange the pixels. And the operation used to create r^2 feature maps is a convolution just as in the previous layers, meaning that the SR Image problem was solved without using neither an upscaling operation nor a deconvolution layer.

8. References

[1] Shi, Wenzhe, et al. "RealTime Single Image and Video SuperResolution Using an Efficient SubPixel Convolutional Neural Network." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[2] Shi, Wenzhe, et al. "Is the deconvolution layer the same as a convolutional layer?" arXiv preprint arXiv:1609.07009, 2016

[3] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2015

[4] Yang, Wenming, et al. "Deep Learning for Single Image Super-Resolution: A Brief Review" arXiv preprint arXiv:1808.03344v1, 2018

[5] Wang, Zhihao , et al. "Deep Learning for Image Super-resolution: A Survey" arXiv preprint arXiv:1808.03344v1, 2019

[6] Peak signal to noise ratio calculation
https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

[7] Pytorch Super Resolution Example
https://github.com/pytorch/examples/tree/master/super_resolution