Universiteit Leiden
The Netherlands

Opleiding Informatica

High-resolution SRCNN

hyperparameter optimisation

Mark Kompier

Supervisors:
D.M. Pelt & Serban Vadineanu

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

31/1/2023

Abstract

This thesis will focus on hyperparameter optimisation of SRCNN on high resolution images. SRCNN is an existing well-established upscaling neural network. In the thesis a pytorch implementation of this network will be changed and for each change results will be put out. The results are upscales of downscaled samples of an image set. These results will be compared to the original, unaltered image set using image metrics. Lastly all the image metric results from each change will be put into tables to compare. The results will be discussed and notable observations will be pointed out.

Contents

1 Introduction **1**
- 1.1 Image upscaling techniques . . . 1
  - 1.1.1 Nearest neighbour . . . 1
  - 1.1.2 Bilinear . . . 1
  - 1.1.3 Bicubic . . . 2
- 1.2 Big Buck Bunny . . . 2
- 1.3 Pytorch implementation . . . 2
- 1.4 Thesis overview . . . 2

2 Definitions **3**
- 2.1 Neural Networks . . . 3
- 2.2 Convolutional Neural Networks . . . 3
- 2.3 Foundation paper - SRCNN . . . 4
- 2.4 Hyperparameters . . . 4
  - 2.4.1 Additional layer . . . 4
  - 2.4.2 Activation functions . . . 5
- 2.5 Image Metrics . . . 8
  - 2.5.1 PSNR . . . 8
  - 2.5.2 SSIM . . . 8
- 2.6 Hardware . . . 9

3 Related Work **9**
- 3.1 SRCNN . . . 9
- 3.2 Other upscaling neural networks . . . 10

4 Experiments **11**
- 4.1 Initial training . . . 12
- 4.2 Hyperparameters . . . 12
  - 4.2.1 Additional convolutional layer . . . 12
  - 4.2.2 Alternative activation functions . . . 12
- 4.3 Comparison . . . 13
- 4.4 Source resolution . . . 13

4.4.1 Scale 3 . . . 14
4.4.2 Scale 4 . . . 14
4.4.3 Scale 6 . . . 14
4.4.4 Scale 8 . . . 15
4.5 Visual Results . . . 16
4.5.1 Default SRCNN . . . 16
4.5.2 Tuned SRCNN . . . 16
4.5.3 Tuned over resolution . . . 17

5 Discussion **20**
5.1 Blurred frames and point of focus . . . 20
5.2 Data bias . . . 21
5.3 Overfitting . . . 22

6 Further Research and Conclusion **22**
6.1 Further research . . . 22
6.2 Conclusion . . . 23

References **25**
.1 Appendix . . . 25

1 Introduction

This research is going to focus on hyperparameter optimisation of upscaling neural networks. An existing upscaling network will be used and the parameters that build up the network will be adjusted in our experiments. The goal will be to specifically improve the upscaling of higher resolution images from this network. The research question of this paper is:
"Can the results of the neural network SRCNN be further improved by adjusting the hyperparameters for different resolutions?"
We will start the paper by explaining the field of image upscaling, a few core image upscaling techniques and outlining the image source and pytorch implementation we have used.
Image upscaling is stretching a lower resolution image over a display that is larger than said image. [SAL20] With how many new upscaling techniques and papers that are being published regularly, image upscaling is an evolving field of research.

1.1 Image upscaling techniques

It's important to explain these basic upscaling algorithms as we use bicubic upscaling as comparison later on. Some very basic standard upscaling algorithms are the following:

1.1.1 Nearest neighbour

The nearest neighbour upscaling algorithm in essence is based on finding the 'nearest neighbour'. When interpolating a new pixel during upscaling, meaning that a new pixel inbetween the existing pixels of a lower resolution image needs to be generated, the algorithm simply copies the value of the nearest pixel. [OH12] This would be the nearest neighbouring pixel that originated from the lower resolution source image, the image that is to be upscaled.

1.1.2 Bilinear

Bilinear upscaling doesn't just simply copy the pixels from the source image, but actually uses a linear equation to calculate the new interpolated pixels. Depending on how far the interpolated pixel is relative to the original pixels, the upscaling algorithm will generate values for the new pixel relative to the four closest original surrounding pixels using linear polynomials. It is called bilinear because the operation has to be performed for both the pixel rows and the columns in a picture.
The equation for bilinear interpolation is:

$$H = L_1(1 - \Delta r)(1 - \Delta c) + L_2\Delta r(1 - \Delta c) + L_3(1 - \Delta r)\Delta c + L_4\Delta r\Delta c$$

[Key81] [Ruk18]
The left hand-side of the equation, H, represents the high resolution output of the bilinear interpolation. The right-hand side components L1, L2, L3, L4 represent the low pixel values of the image. Delta r and Delta c are numerical variables for the distances between pixels. Each high resolution interpolated pixel uses 4 low resolution pixels in it's calculation. The interpolated pixels are pixels positioned somewhere in-between the low resolution pixels.

1.1.3 Bicubic

Bicubic is important for this research as the images will first be downsampled using bicubic upscaling. The bicubic image will then be used as base to train the network. The image will also be used to compare with the SRCNN upscaled image in the image metric comparison afterwards. In our case bicubic produces better results than bilinear and is prefered over bilinear interpolation.
The equation for the interpolated surface from bicubic interpolation is the following third-order polynomial:

$$\sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} \times x^i x^j \qquad \text{[Key81]}$$

Unlike bilinear, bicubic is uses the 16 nearest low resolution pixel values instead of the 4 nearest pixel values to interpolate a new high resolution pixel.

1.2 Big Buck Bunny

The image set that will be used for the research will be frames from the Big Buck Bunny video.
(c) copyright 2008, Blender Foundation / www.bigbuckbunny.org
This video is openly licensed [Fou08] and it is free to remix, transform, and build upon the material for any purpose as long as credit is given. We have cut this video into individual frames and we will pick a small selection of these frames. The video is in 2160p.

1.3 Pytorch implementation

To start using the SRCNN network, we need to have an implementation that the computer will be able to run. For this we have decided to use the implementation written in python by Jeffrey Yeo on github. [Yeo19] The implementation uses the pytorch python machine learning framework. We will adjust the SRCNN network by editing the code of this implementation.

1.4 Thesis overview

After this introduction the thesis will explain definitions that are vital to understand the the rest of the paper. Following will be a section outlining some related work and their possible relevance, after which the experiments will be explained and displayed. The The image metric results from the experiments will prove quite similar across the different changes in the network implementation. Visual comparisons will be shown alongside the experiments to put these results into perspective. The results will be discussed afterwards and potential observations will be noted such as how frames that are meant to be blurry by design more easily yield high image metric scores after being upscaled by the network. Lastly the conclusion will end the thesis followed by an appendix with additional visual results from the experiments.

2 Definitions

Firstly it's important to understand some of the definitions and concepts that are going to be used in this research. Without knowing what certain terms entice it would be difficult to follow what's to come.
Starting with the basics, machine learning [Nie19b] is the broad field where machines are able to improve themselves depending on the data they are fed. A computer is able to approach being able to learn as it iteratively improves depending on the data. Deep learning is a subset of machine learning where neural networks are used as learning tools.
[SS18]

2.1 Neural Networks

A neural network [RN] is a network of neurons where each neuron is a basic unit that can activate depending on the weighed sum of their inputs and the activation function. Neural networks are very roughly based on an analogy with the human brain.
The multilayered neural network [RN] is a neural network where all neurons are connected to the neurons in the next and previous layer. In between the input and output layer are one or multiple 'hidden layers' that connect the the layers to each other. All neurons are connected therefore there is a very large amount of connections in multilayered neural networks. This makes it computationally intensive to train this network. Because this type of neural network is rather inefficient in computer vision we have opted not look at the standard multilayered neural network.

2.2 Convolutional Neural Networks

A special type of neural network is the convolutional network. Unlike a default neural network, a convolutional neural network uses convolutional layers to apply a filter or kernel over the input. The process where the kernel moves over the input is also called convoluting. The kernel is then able to reduce the dimension of the input and filter out certain features from the input. The result after the input has passed through a convolutional layer is called a feature map.

Convolutional neural networks are especially useful for appliances within computer vision, as images consist of a grid of pixels where features can be found from how groups of pixels neighbour eachother. In contrast to just inputting the pixel values of all the pixels in the image and letting the network adjust it's weights based on that. The kernel, for example a 3 by 3 matrix, can filter out edges based on how pixels group together. Afterwards it's then possible to pass the input through another convolutional layer and get another feature map. Eventually it's possible for feature maps to outline very sophisticated features in an image like the face of a cat or even entire objects like cellphones.
[QYLC18]

2.3 Foundation paper - SRCNN

"Image super resolution using deep convolutional neural networks" [DLHT16] is the paper where the network we are using originates from. From now on, the paper will be abbreviated to SRCNN and will be referenced as such.

The reason we choose this network is that it's an established network known for its good performance in many situations and with many sources. Additionally it's a simple network so that makes it more suited to change it and adjust its parameters.

2.4 Hyperparameters

Neural networks inherently adjust their models parameters during training to produce a more desired outcome. In the context of this paper parameters will therefore equate to the models internal parameters that are adjusted during the training using the Big Buck Bunny data. In of itself we will not be touching any parameters directly for the research, we will only be training the network so it can adjust its parameters to its best ability.
In contrast, hyperparameters are different. Hyperparameters are essentially variables that control the training of the network. In other words, hyperparameters are the parameters that dictate how the neural network adjusts its inherent parameters. The hyperparameters define the architecture of the neural network itself rather than individual neurons. [Pan19]

2.4.1 Additional layer

When talking about additional layers in this paper, we are referring to additional convolutional layers in the python implementation of SRCNN. We edited the architecture of SRCNN in python such that only an additional convolutional layer is added.

2.4.2 Activation functions

Activation functions [KO11] are used in a neural network to determine whether a neuron will fire or not. The input of the neuron is summed up and using this weighted sum the function calculates whether or not the it fires. It is common for activation functions to be differential and nonlinear. This is because differential nonlinear activation function are able to learn more complex problems than other activation functions.

- ReLU
  By default, the python implementation uses the ReLU activation function. The Rectified Linear Unit function has the following equation for input x:

$$ReLU(x) = max(0, x) \quad [Ped18]$$

  ReLU will output zero when the input x is negative, otherwise it will output x.

  The graph for ReLU:

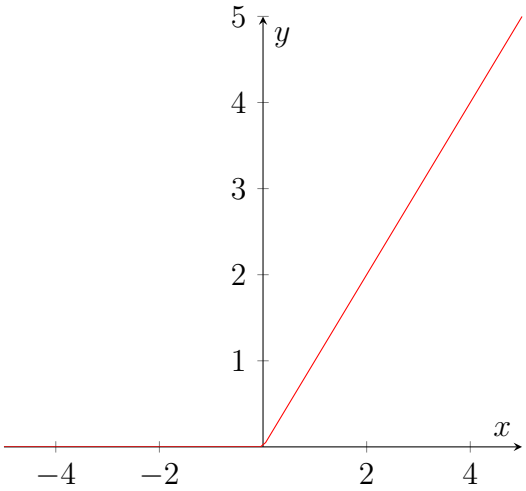

Figure 1: ReLU shown in a graph. One of the activation functions used as hyperparemeter that is consistently zero when x is negative and follows the function y=x when x is positive as depicted

- ELU
  The ELU function is very similar to the ReLU function. However, unlike the rectified linear unit, the exponential linear unit doesn't output 0 when the input x is below 0. Instead the output will be equal to:

$$\alpha(e^x - 1)$$

Just like the ReLU function, the output will be equal to the input x when said input is not below zero.
The equation for ELU is then:

$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases} \quad [\text{Ped18}]$$

The graph for ELU:

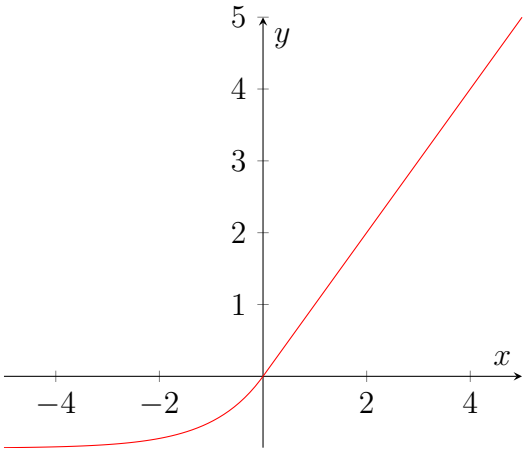

Figure 2: ELU shown in a graph. One of the activation functions used as hyperparameter that is equal $\alpha(e^x - 1)$ when x is smaller than zero and is equal to y=x when x is bigger than zero.

- SIGM
  The sigmoid function is a little different. The sigmoid function is a logistic function that has an output between 0 and 1. The equation is:

$$sigmoid(x) = 1/(1 + e^{-x}) \quad [\text{KO11}]$$

The graph for sigmoid:

- TANH
  The TANH function is similar to the sigmoid function and has a very similar shape. However, unlike the sigmoid function it outputs a value between -1 and 1. The equation:

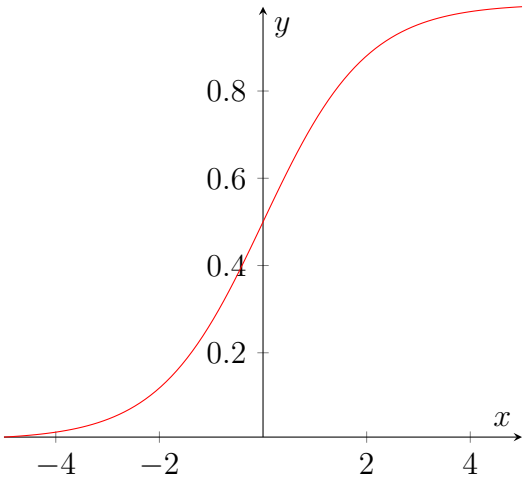

Figure 3: Sigmoid shown in a graph. One of the activation that is used as hyperparameter. The sigmoid activation function uses the same equation for all values of x

$$TANH(x) = (e^x - e^{-x})/(e^x + e^{-x}) \quad [\text{KO11}]$$

The graph for TANH:

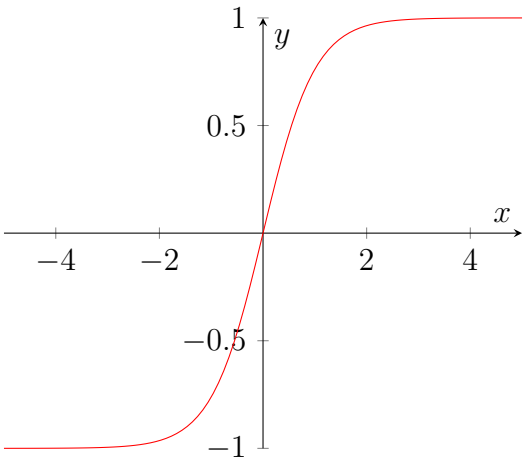

Figure 4: TANH shown in a graph. One of the activation functions used as hyperparameter. The TANH activation function uses the same equation for all values of x.

2.5 Image Metrics

For the the research we have decided it would be too arbitrary and unreliable to compare results using the naked eye. Deciding whether an upscaled image looks better than another depending solely on how good it looks at first glance leaves too much room for human error as well as being inconsistent. This is where image metrics are useful. Image metrics are used to evaluate the quality of an image. [HZ10] To produce consistent, measurable data, we have opted to use image metrics to do comparisons in the experiments.
The image metrics are calculated over a reference image A where it is compared to a processed image B to search for any noise and degradation in comparison to image A. The processed image in our case will be the upscaled output images from the neural network.

2.5.1 PSNR

Peak signal to noise ratio is an image metric where noise between two images is calculated in dB. PSNR compares image A and B of the same size by using the mean squared error MSE. The MSE over A and B is defined as:

$$MSE(A,B) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(A_{ij} - B_{ij})^2 \qquad \text{[HZ10]}$$

Using the MSE, the PSNR for 8-bit images is then calculated as:

$$PSNR(A,B) = 10\log_{10}(\frac{255^2}{MSE(A,B)}) \qquad \text{[HZ10]}$$

The higher the PSNR, the higher the quality of image B is in comparison to A. In our case this means the higher the PSNR, the better the upscaled image approaches the original image. In general, a PSNR of 40 or higher is regarded as being an almost unnoticable difference. [YTA09]

2.5.2 SSIM

SSIM, structural similarity index, is an image metric which uses 3 key features. Using these features, the comparison between two images is made. These features are luminance, contrast and structure. SSIM is defined as:

$$SSIM(A,B) = l(A,B)c(f,g)s(f,g) \qquad \text{[HZ10]}$$

Afterwards, a structure similarity index is calculated. This index is between -1 and 1, An index of 1 means the two pictures are very similar or even identical, while an index of -1 means the two pictures are very different.

2.6 Hardware

To get good results, we are going to have to utilize recent computer hardware. Computer hardware is a complete field of study in of itself, but we are going to be using some terminology refering to computer hardware. Following is an explanation of some useful terms for this research and machine learning in general:

GPU: Graphics processing unit. [Sch15] Hardware device that is build for parallel processing. The gpu is able to simultaneously perform simple computations on a lot of data. Because of this, matrix multiplication and therefore convolutional neural networks work well with GPUs. The GPU is an important tool to be able to train convolutional neural networks in reasonable time.

VRAM: Video random access memory. [TF09] Because we want to make use of the increased performance of the gpu, we will have to make sure the network will fit within the VRAM capacity during its training.

The specific GPU we will be using for this research is the gigabyte rtx 3070 windforce with 8 gigabytes of VRAM.

3 Related Work

3.1 SRCNN

As explained above, the research is based on a specific, well-established neural network from a foundation paper. We use the abbreviation SRCNN. This will be the core related work as well as the basis for the hyper-parameter optimization research.

3.2 Other upscaling neural networks

It's important to give a brief comparison of alternative deep-learning upscaling research that are established in computer science research. At the start of the research the neural networks from these were discussed and compared to SRCNN for potential hyper-parameter optimization research. These networks could have been used as basis instead of SRCNN. Note that there are a lot of upscaling neural networks in the field of computer science research and we will only discuss a selection of these four.

1. Deep Learning for Image Super-resolution: A Survey

   This paper [WCH20] gives a general overview of recent advances in image upscaling deep learning. It describes common challenges deep learning super resolution faces and explains some of the new trends in this field. In the paper various image metrics are explained including PSNR and SSIM. It also mentions some public image upscaling benchmark datasets.

   The paper references several super-resolution neural networks which could have been used as basis for this hyperparameter optimisation research.

2. NTIRE 2020 Challenge on Real-World Image Super-Resolution: Methods and Results

   The second paper [LDT20] portrays an image upscaling challenge where participants compare results after they have trained their neural networks on a given challenge dataset. Unlike this paper, the images being used in this paper originate from the real world. In this paper our images are frames from a computer animated video.

3. Accurate Image Super-Resolution Using Very Deep Convolutional Networks

   In this research [KLL16] image upscaling is done with the focus on a very deep convolutional neural network. For the neural network the training procedure utilizes extremely high learning rates while the neural model itself is build with up to 20 weight layers.

4. On-the-Fly Machine Learning for Improving Image Resolution in Tomography
   This research paper [HPP+19] addresses image upscaling in the field of tomography. Image upscaling has useful applications in tomography because tomography is inherently limited by the resolution of the detector. In the paper machine learning is used to improve the resolution of tomographic reconstructions. A neural network is trained with complete and partial tomographic reconstructions as its input. From the results the paper shows how they were able to achieve improvements in the upscaling of tomographic reconstructions by using machine learning.

4 Experiments

In the research we are going to experiment by training the network and comparing the upscaled results the trained network is going to put out. We are going to call one session of training the network a 'training run'.
For the comparison we are going to use a python script metrics.py. The content of the script is the following python code:

```
import argparse
import skimage.metrics as sm
import imageio

parser = argparse.ArgumentParser(description='Compute metrics between images.')
parser.add_argument('highres', type=str)
parser.add_argument('upscaled', type=str)
args = parser.parse_args()

highres = imageio.imread(args.highres)
upscaled = imageio.imread(args.upscaled)

psnr = sm.peak_signal_noise_ratio(highres, upscaled)
print('PSNR: ', psnr)

ssim = sm.structural_similarity(highres, upscaled, multichannel=True)
print('SSIM: ', ssim)
```

This script is run 30 times with a different picture. The script will give a PSNR and SSIM value. This is done over the pictures resulted from a training run that was trained with a tweaked hyperparameter. The average over these 30 image metrics is then calculated to determine the score of a training run. The average is then rounded to 3 decimals. (outdated, no longer 10)

4.1 Initial training

For the initial training the network was trained without adjustment. This means we have used the default values from the github repository to train the network. This training run will function as the baseline for the following experiments. The output of this run will be compared to the output of each hyperparameters corresponding run. After that a second series of experiments will involve changing the source resolution and repeating the experiments for the hyperparameters on this changed resolution. The aim of the experiments is to look where the output of the network can be improved by adjusting the hyperparameters and the source resolution. For the evaluation of the metrics we will use image 155 to 445. Image 155 is the 155th image we have extracted from the Big Buck Bunny video. The image metrics PSNR and SSIM are calculated using with he python language using a script metrics.py. After the network was trained with default parameters the network was used to upscale image 155 to 445. Afterwards, the PSNR and SSIM for the images 155 to 445 is calculated. The average PSNR over these 30 images is then used as comparison.

4.2 Hyperparameters

For the first series of experiments the hyperparameters have been adjusted for a couple of training runs. Subsequently the results are going to be compared using image metrics. Each of training runs is ran by running train.py with the following arguments:

```
python3 train.py --train-file "./2160p_scene2.h5" --eval-file "./2160p_scene3.h5"
--outputs-dir "./Results/hyperparameter_name" --scale 2 --lr 1e-4 --batch-size 16
--num-epochs 40 --num-workers 8 --seed 123 --num-epochs 40 --num-workers 8 --seed 123
```

4.2.1 Additional convolutional layer

Adding into model.py an extra convolutional layer on top 64 channels led to out of memory exceptions at first. This is when doubling the amount of channels produced by each convolution so this is to be expected. To solve this issue the output channels were simply halved to 64. The consequence of this is that the last feature map was reduced from 32 channels to 16 channels.
A slight improvement in PSNR with a small increase over the default model was gained. SSIM did not significantly improve and stayed almost identical.

4.2.2 Alternative activation functions

Unlike the other activation functions, ELU was able to run without reducing the amount of channels. The PSNR results were almost identical with only a small difference with ELU as activation function for the network instead of ReLU. SSIM did not significantly improve and stayed almost identical.

For sigmoid it was again necessary to reduce initial output layers to 32 because of VRAM constraints. Sigmoid performed slightly worse with a lower PSNR and SSIM result.

For TANH it was once again necessary to reduce initial output layers to 32 because of VRAM constraints. TanH performed slightly worse in both PSNR and SSIM.

4.3 Comparison

For the initial experiments the network is trained to upscale from the resolution 1920 by 1080. Important to note that for the following table the default parameter SRCNN network metrics from the experiments are compared to each of the image metrics of the changed hyperparameter results. Each row represents a different hyperparameter training run with its own image metrics for comparison.

Hyperparameter	PSNR	SSIM
Default	42.910	0.985
Additional layer	43.095	0.985
Elu function	42.855	0.985
Sigmoid function	42.742	0.984
TANH function	42.769	0.984

It is important to note that for the sigmoid and tanh activation functions the initial output layers were reduced to 32. To compensate for this in the comparison, a second default parameter training run was done with 32 initial output layers. Next the image metrics from this training run are again compared to the results from the sigmoid and tanh training runs to have a comparison with equal initial output layers. This is done to rule out that any difference in the image metrics will not be from a the different output layers, which were only changed due to not having enough vram. In the comparison table the sigmoid and tanh results are exact copies from the last experiment, as the goal is to compare these on a baseline with the same amount of input channels. The comparison table for this is shown below:

Hyperparameter	PSNR	SSIM
Low channels	42.984	0.985
Sigmoid function	42.742	0.984
TANH function	42.769	0.984

4.4 Source resolution

For the second type of experiments we are going to change the source resolution of the image. In the implementation of the network the images are scaled down before being fed to the network. This means that when given a native full HD picture, the implementation will will first downscale this depending on a scale value and will then use the downscaled image for training. With this in mind the results of a series of training runs with each different scale values will be compared.
The goal of these experiments will be to investigate whether changing the hyperparameters when upscaling from a different source resolution will affect the results. The same hyperparameters will be adjusted to do the comparisons, only this time with a different source resolution. Each source resolution will have its own comparison tables. Note that the table for the adjusted comparison in regarding to the initial output channels is added as well for each source resolution.

4.4.1 Scale 3

Not surprisingly, the image metrics get worse when the network is upscaling lower source resolution pictures. This is because the lower the source resolution, the more data of the original image is lost and this is data that the network needs to fill in. The difference in image metrics between the hyperparameters is still very small. Here we scale from the source resolution 1280 by 720.

Hyperparameter	PSNR	SSIM
Default	40.832	0.970
Additional layer	40.783	0.970
Elu function	40.649	0.969
Sigmoid function	40.679	0.968
TANH function	40.485	0.968

Hyperparameter	PSNR	SSIM
Low channels	40.578	0.969
Sigmoid function	40.679	0.968
TANH function	40.485	0.968

4.4.2 Scale 4

Again a small difference, but this time we can see the additional layer perform the best like we did at the first series of experiments. Here we scale from the source resolution 960 by 540.

Hyperparameter	PSNR	SSIM
Default	39.186	0.953
Additional layer	39.198	0.954
Elu function	39.191	0.952
Sigmoid function	39.013	0.951
TANH function	38.863	0.951

Hyperparameter	PSNR	SSIM
Low channels	39.071	0.952
Sigmoid function	39.013	0.951
TANH function	38.863	0.951

4.4.3 Scale 6

Image metrics dropping as expected, with small differences once again. Additional layer is performing extremely close to the default training run, barely getting lower image metrics. Take note that

SSIM and PSNR differ here. The additional layer performs worse PSNR-wise, whereas it performs better SSIM-wise. Here we scale from the source resolution 640 by 360.

Hyperparameter	PSNR	SSIM
Default	36.856	0.921
Additional layer	36.850	0.922
Elu function	36.747	0.919
Sigmoid function	36.686	0.919
TANH function	36.576	0.918

Hyperparameter	PSNR	SSIM
Low channels	36.724	0.920
Sigmoid function	36.686	0.919
TANH function	36.576	0.918

4.4.4 Scale 8

Lastly the additional layer training run still performs the best. Aside from the additional layer, the difference in SSIM have become indistinguishable with only 3 decimals. Here we scale from the source resolution of 480 by 270.

Hyperparameter	PSNR	SSIM
Default	35.010	0.896
Additional layer	35.124	0.898
Elu function	34.980	0.896
Sigmoid function	34.847	0.896
TANH function	34.878	0.896

Hyperparameter	PSNR	SSIM
Low channels	35.059	0.896
Sigmoid function	34.847	0.896
TANH function	34.878	0.896

4.5 Visual Results

This section will bring some perspective by viewing tangible results. These concrete tangible images result from the neural networks trained in the experiments. The images The images will be compared to each other, the original high resolution image as well as the bicubic upscaled image.

4.5.1 Default SRCNN

The initial experiment used the scale 2 to upscale from 1920 by 1080 to 3840 by 2160. A zoomed in picture of a squirells whiskers in the 155th image of Big Buck Bunny is used for comparison. This squirell will be shown in the figures accompanying the experiments. In the appendix alternative different zoomed-in croppings of different pictures from the results can be viewed later on.
Here we compare the original groundtruth image, the bicubic image and the default SRCNN models image:

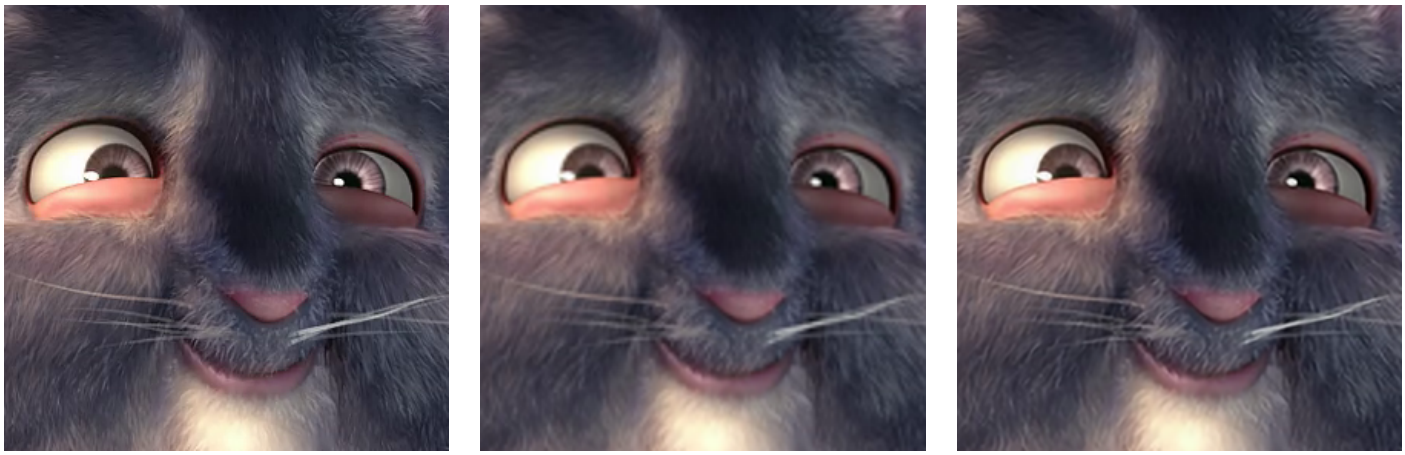

Figure 5: Comparison of the original image, the bicubic image and an image generated by the network. From left to right: groundtruth, bicubic and default SRCNN

4.5.2 Tuned SRCNN

When changing and optimising the hyperparameters of the network, the results are not as visible as the comparison above. In this section we will compare a cropped image of the squirrel for every hyperparameter.

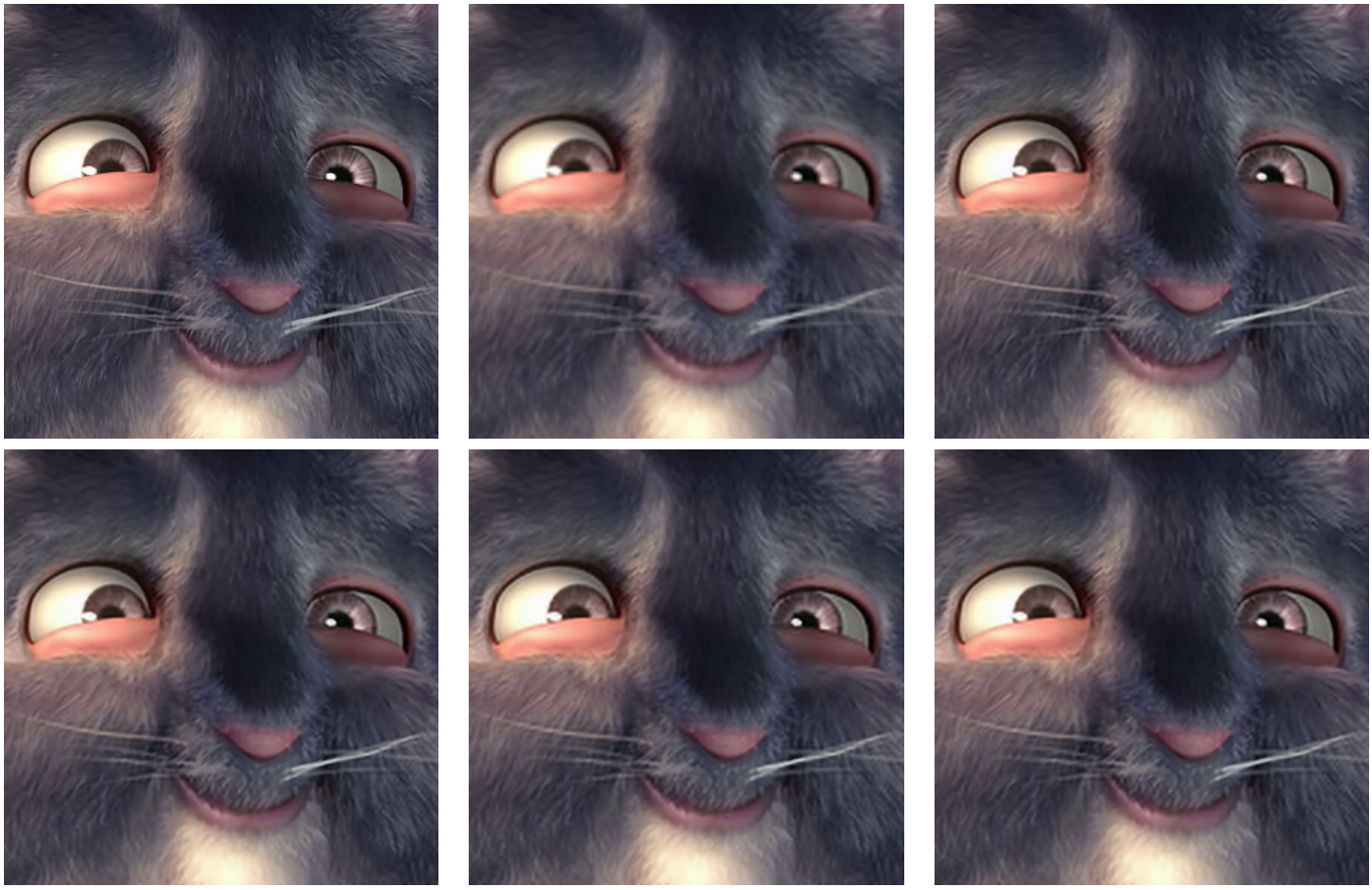

Figure 6: Comparison of the bicubic upscaled image and the images generated by the network for each hyperparameter. From left to right: groundtruth, bicubic and SRCNN with an extra convolutional layer. On the second row first the ELU activation function, then the sigmoid activation function and on the rightmost picture the TANH activation function.

Note that the visual differences seem very hard to spot. This is to be expected since the PSNR values of these upscaled images are all above 40 and when above 40 PSNR it becomes almost impossible to spot differences.

4.5.3 Tuned over resolution

Besides using the network to upscale the resolution 2 times over the source image, we have also conducted experiments over upscaling 3, 4, 6 and 8 times. The upscaled output image is always in 4k, but we have downscaled the image from Big Buck Bunny with the higher factor. In these experiments we will look at how the end result in 4k compares to the best hyperparameter from each series of experiments. The best scoring hyperparameter when it comes to image metrics is consistently the extra convolutional layer and therefore each comparison is showing a picture of the output with this hyperparametert next to a bicubic upscale of the same factor. The groundtruth image has been left out as it naturally remains unchanged.

Times 3 - 1280×720 to 3840×2160

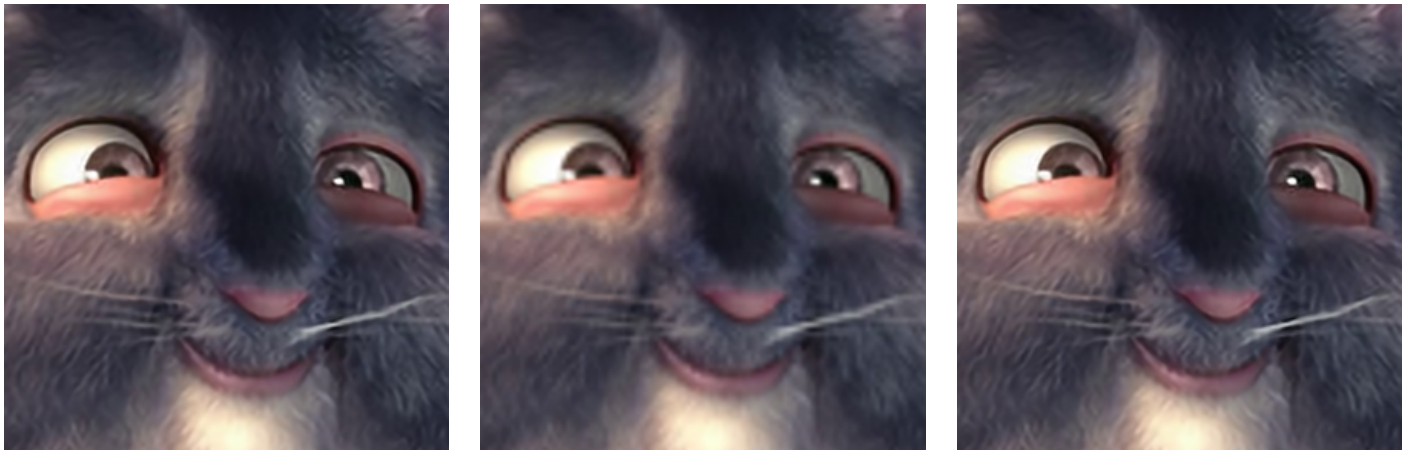

Figure 7: Comparison of the default configuration of SRCNN to the bicubic upscaled image and the images generated by the network when adding an extra convolutional layer. The scale of upscaling is 3.

Times 4 - 960×540 to 3840×2160

Figure 8: Comparison of the default configuration of SRCNN to the bicubic upscaled image and the images generated by the network when adding an extra convolutional layer. The scale of upscaling is 4.

Times 6 - 640×360 to 3840×2160

Figure 9: Comparison of the default configuration of SRCNN to the bicubic upscaled image and the images generated by the network when adding an extra convolutional layer. The scale of upscaling is 6.

Times 8 - 480×270 to 3840×2160

Figure 10: Comparison of the default configuration of SRCNN to the bicubic upscaled image and the images generated by the network when adding an extra convolutional layer. The scale of upscaling is 8.

5 Discussion

5.1 Blurred frames and point of focus

Some frames are already inherently blurry. An intentionally blurry groundtruth image can be thought of as an image where less upscaling has to be done to produce a desirable upscaled image. The blurry frames in the image-set perform well in the network and have gotten a high PSNR compared to the other images. This is notable because this likely means the network does not oversharpen intentionally blurry images and does well in upscaling these to a higher resolution.

Some frames are blurry as a whole, while others are partially blurry with a sharper part of the images in focus. These frames focus more on one point of interest, like an animal in the foreground. In this case it the network has to performs differently on the animal and the blurry background, where it has to produce a sharper image for the point of interest and a relatively unsharp section for the background.

To visualize the difference in PSNR within one image, we have cut up a single image entirely in cropped sections. Each cropped section will be evaluated for their PSNR with the image metric script. These cropped sections will be color-coded to their respective PSNR. By doing this it will become visible that the PSNR differs when the parts of the image gets blurry or out of focus.

Figure 11: Frame of the Big Buck Bunny video that has been separated in 115 different cropped segments.

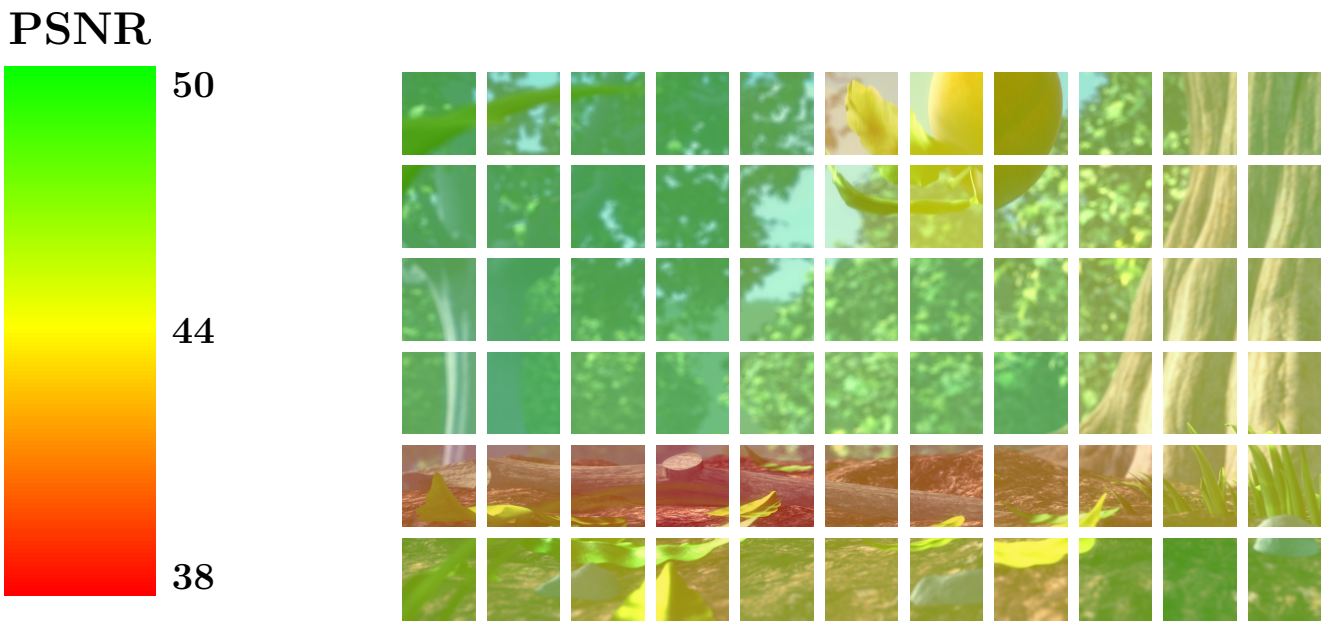

Figure 12: Frame of the Big Buck Bunny video that has been separated in 115 different cropped segments.

PSNR	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11
Row0	47.33	46.46	47.10	47.38	46.66	42.65	43.90	46.40	45.02	45.37	45.09	45.83
Row1	47.20	47.17	47.44	47.24	46.50	44.63	44.24	46.62	45.05	44.65	44.36	45.43
Row2	46.78	47.47	47.12	47.10	46.09	45.90	46.15	46.31	45.31	44.75	44.73	44.85
Row3	46.65	47.73	46.65	47.12	45.79	45.57	46.06	46.50	46.72	44.39	44.43	44.37
Row4	41.37	41.75	40.76	38.74	39.77	41.77	41.44	42.19	42.28	42.53	43.65	44.29
Row5	45.53	45.27	44.19	43.35	44.69	44.27	44.52	45.11	43.52	45.87	46.82	46.11

As you can see in figure 12, the cropped parts of the frame that are already sharp than the blurry parts yield a lower PSNR when upscaled. Therefore the network performs better on parts that are blurred by design before upscaling.
What was interesting to see is that the network performed very differently depending on the picture that the network was given to upscale. This would imply quite a large margin of error on a picture by picture basis. This could be explained by the blurryness in the pictures but there also could be different reasons for why the network would upscale certain frames better over others.

5.2 Data bias

A consideration of interest in the research is that we have only used pictures from one video, the Big Buck Bunny video, to train the neural network. This could a cause potential data bias [KKK+19] where the upscaling network only performs well on these specific pictures. The risk doesn't necessarily stop there either, because it can also be considered that the network is only performs well on pictures of similar animated content. The network might not work will with

different content such as urban scenes or real life data. In other words, the training data has a data bias on grasslands, forests and some of their wildlife. We have accepted this data bias as is. For this research we have only examined changing the hyperparameters.

5.3 Overfitting

Overfitting [RN] might be a possible problem when it comes to training the network. What we mean by this is that the network might have been overtrained and is trying to produce results too similar to the training data. However, within the scope of this research we have considered this not to be a problem.

6 Further Research and Conclusion

6.1 Further research

When taking the average of the image metrics we have taken the average over thirty images from the video source. Thirty is not a large amount of images for this purpose and this might have impacted the results slightly. To reduce the margin of error more images could be used in future research to lessen this impact and compensate more for the margin of error. [Nie19a] Furthermore, other than just taking the average, more elaborate statistical analysis can be performed on this image set.

For future research more comparisons could be done on images with more varying content, such as urban scenes. This might reduce the potential data bias of this trained network on the Big Buck Bunny video. [KKK+19]

To further avoid overfitting and overlapping test sets, for future research k-fold crossvalidation [Nie19a] might be a consideration. In this research we have not opted to use k-fold crossvalidation because it would be very computationally expensive and take too long to train the network. In fact, it would take k times as long. This is because when using k-fold crossvalidation, the data that is used is split into k subsets. Afterwards, k times 1 of the subsets is taken as the testset while the rest is used for training.

6.2 Conclusion

The research in this paper has been based on the established upscaling network of SRCNN, standing for Super-Resolution Convolutional Neural Network. In the paper we have first made steps to explain some background information in the field of upscaling including relevant algorithms and definitions. To start working and experimenting with the SRCNN network, we have used an implementation of the network in pytorch. We then continued the research by producing experiments with the neural network by tweaking various hyperparameters of the network, which included an additional convolutional layer and trying out different activation functions within the implementation. For the input of the implementation we have opted to use the Big Buck Bunny video [Fou08] cut up into frames. The experiments produced upscaled images and we have compared the upscaled frames to the original frames resulting in image metrics. These image metrics we have put into tables in both PSNR and SSIM for different hyperparameters and starting resolutions.

When considering this pytorch implementation of SRCNN as representation of the neural network, tweaking the hyperparameters from the default implementation still results in similar image metrics over the image output. When upscaling from a lower resolution image to 3840 by 2160 pixels, the network is expectantly performing worse in general. The network has to perform more work when the image has to be upscaled from a lower resolution after all. However, the network architectures with different hyperparameters still perform closely in the same regard as to when upscaling from a relatively higher source resolution. SRCNN is therefore performing well consistently for different hyperparameters over different resolutions. It is robust. Changing to a selection of different activation functions doesn't affect results much if at all. This likely means that the network already has selected optimal hyperparameters that work well even to upscale to higher resolutions.
[Nie19a]

References

[DLHT16] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[Fou08] Blender Foundation. Big buck bunny, 2008.

[HPP+19] Allard A. Hendriksen, Daniël M. Pelt, Willem Jan Palenstijn, Sophia B. Coban, and Kees Joost Batenburg. On-the-fly machine learning for improving image resolution in tomography. *Applied Sciences*, 9(12), 2019.

[HZ10] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.

[Key81] R. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160, 1981.

[KKK+19] Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, and Junmo Kim. Learning not to learn: Training deep neural networks with biased data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[KLL16] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654, 2016.

[KO11] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[LDT20] Andreas Lugmayr, Martin Danelljan, and Radu Timofte. Ntire 2020 challenge on real-world image super-resolution: Methods and results. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 494–495, 2020.

[Nie19a] Michael Nielsen. Data mining: Practical machine learning tools and techniques. 2019.

[Nie19b] Michael Nielsen. *Neural Networks and Deep Learning*. 2019.

[OH12] Rukundo Olivier and Cao Hanqiang. Nearest neighbor value interpolation. *International Journal of Advanced Computer Science and Applications*, 3(4), 2012.

[Pan19] Balaram Panda. Hyperparameter tuning. 10 2019.

[Ped18] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task, 2018.

[QYLC18] Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. How convolutional neural network see the world - a survey of convolutional neural network visualization methods, 2018.

[RN] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach. Third edition.* PEARSON.

[Ruk18] Olivier Rukundo. Half-unit weighted bilinear algorithm for image contrast enhancement in capsule endoscopy. In Hui Yu and Junyu Dong, editors, *Ninth International Conference on Graphic and Image Processing (ICGIP 2017)*, volume 10615, pages 775 – 783. International Society for Optics and Photonics, SPIE, 2018.

[SAL20] ISHA SALIAN. What is ai upscaling? 2020.

[Sch15] Daniel Schlegel. Deep machine learning on gpu. *University of Heidelber-Ziti*, 12, 2015.

[SS18] Pramila P. Shinde and Seema Shah. A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, 2018.

[TF09] Shigeyoshi Tsutsui and Noriyuki Fujimoto. Solving quadratic assignment problems by genetic algorithms with gpu computation: A case study. GECCO '09, page 2523–2530, New York, NY, USA, 2009. Association for Computing Machinery.

[WCH20] Zhihao Wang, Jian Chen, and Steven CH Hoi. Deep learning for image super-resolution: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3365–3387, 2020.

[Yeo19] Jeffrey Yeo. Pytorch implementation of image super-resolution using deep convolutional networks (eccv 2014), 2019.

[YTA09] Qingxiong Yang, Kar-Han Tan, and Narendra Ahuja. Real-time o(1) bilateral filtering. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 557–564, 2009.

.1 Appendix

In this appendix additional results from the experiments will be shown that were excluded from the previous sections. More pictures can be added without bloating the paper by instead placing them in this section.

Figure 13: Comparison of the original image, the bicubic image and an image generated by the network. From left to right: groundtruth, bicubic and default SRCNN

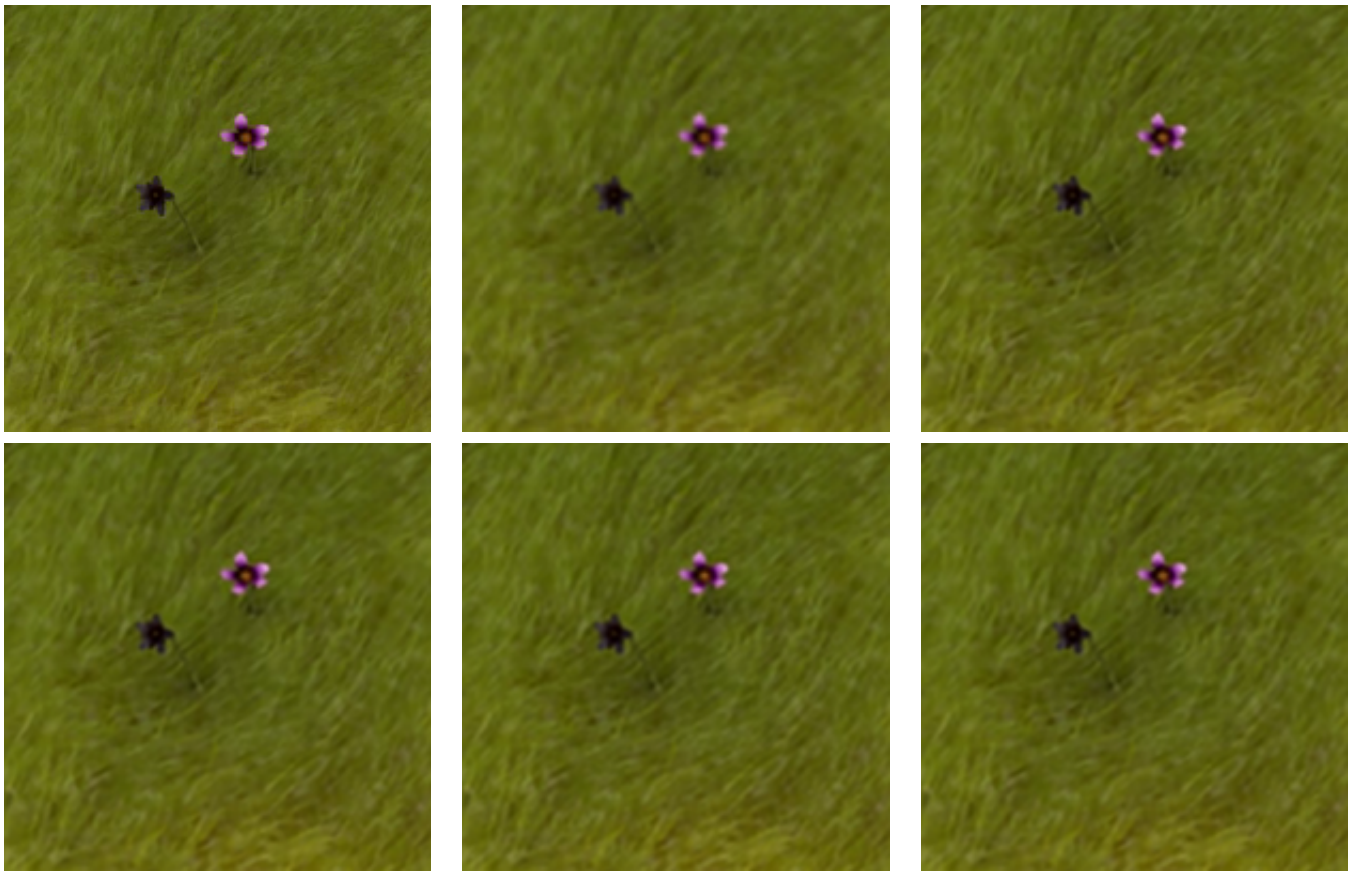

Figure 14: Comparison of the bicubic image and the images generated by the network for each hyperparameter. From left to right: groundtruth, bicubic and SRCNN with an extra convolutional layer

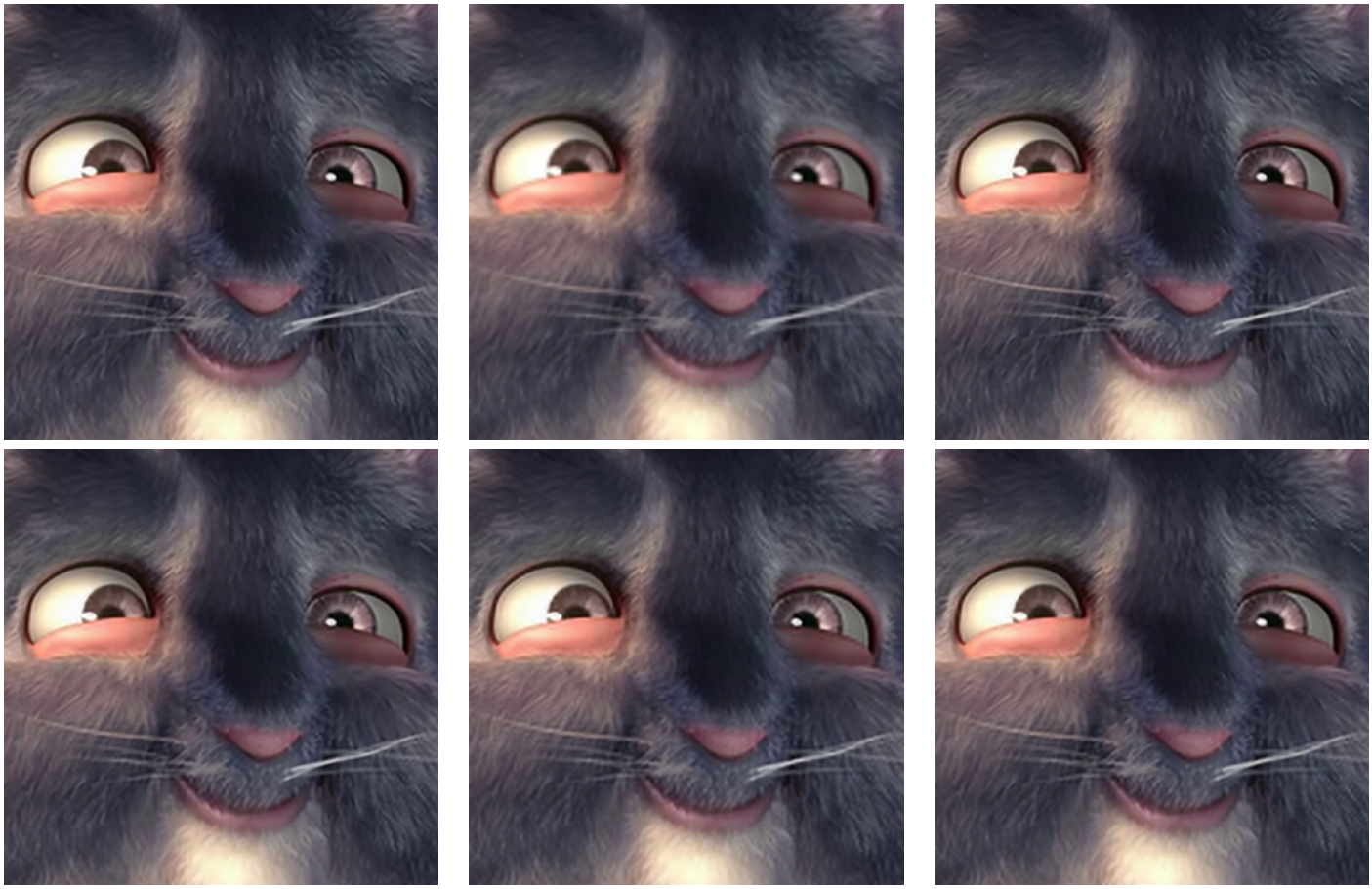

Figure 15: Whiskers scale 2 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

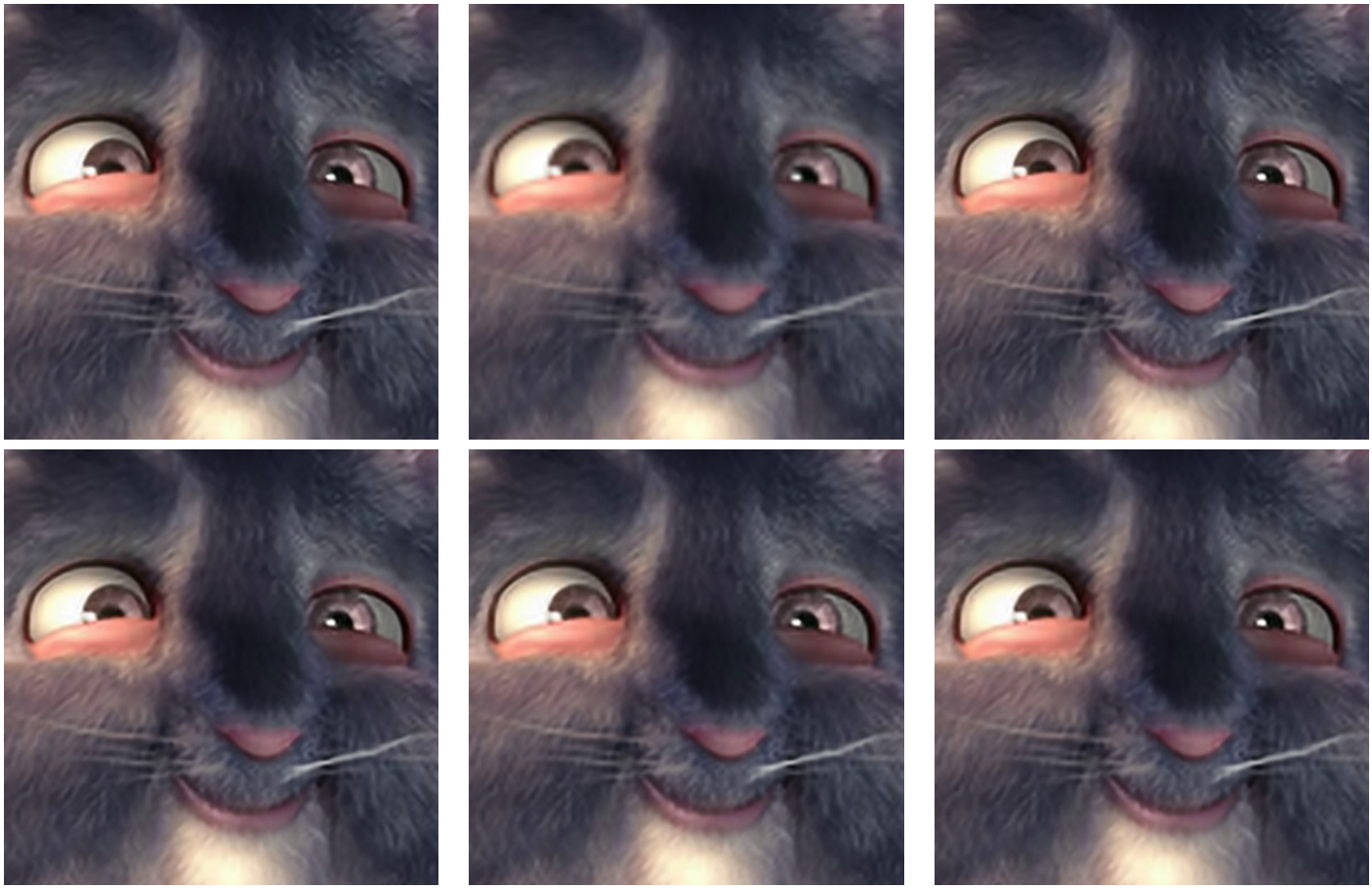

Figure 16: Whiskers scale 3 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

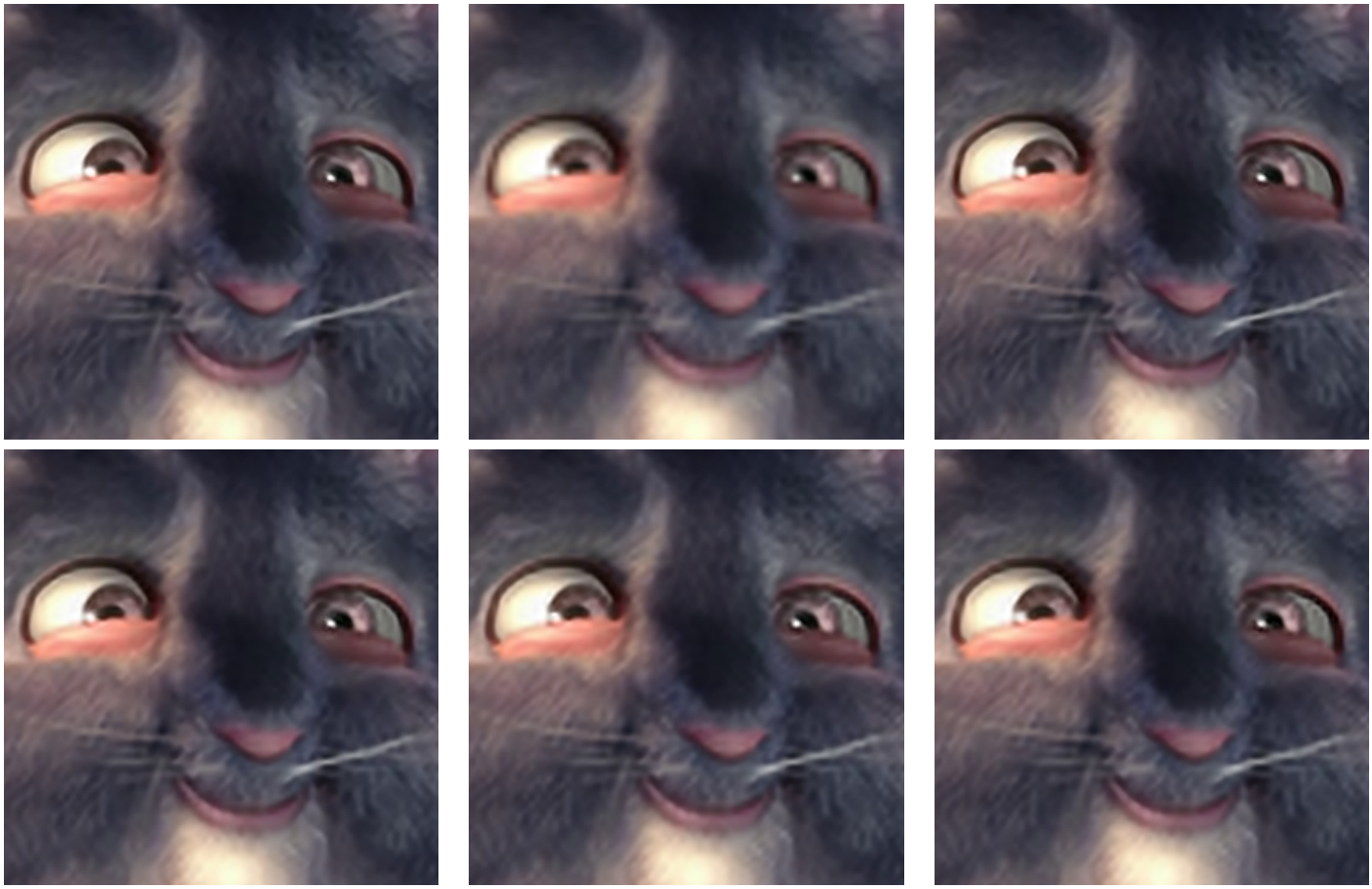

Figure 17: Whiskers scale 4 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

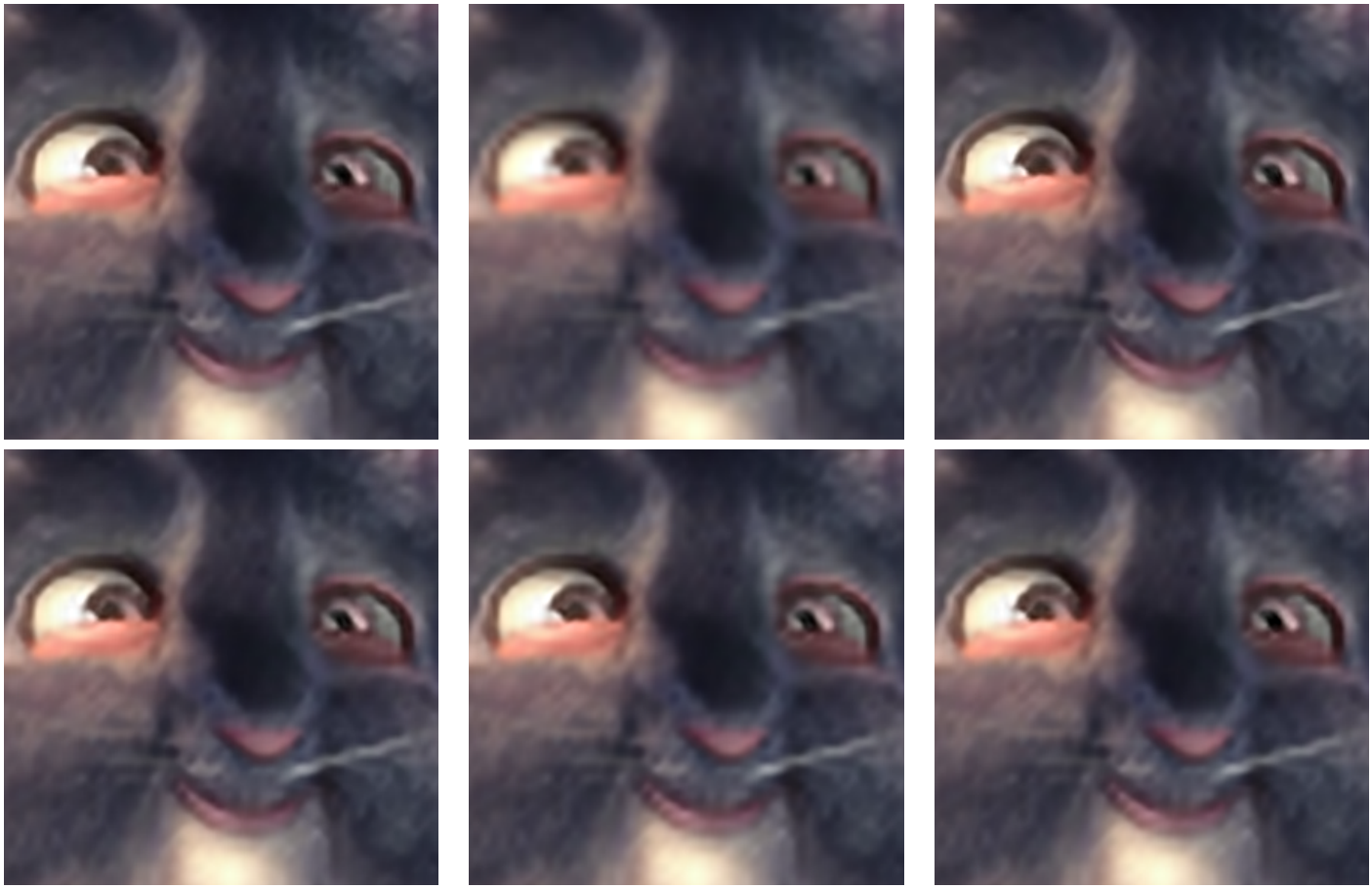

Figure 18: Whiskers scale 6 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

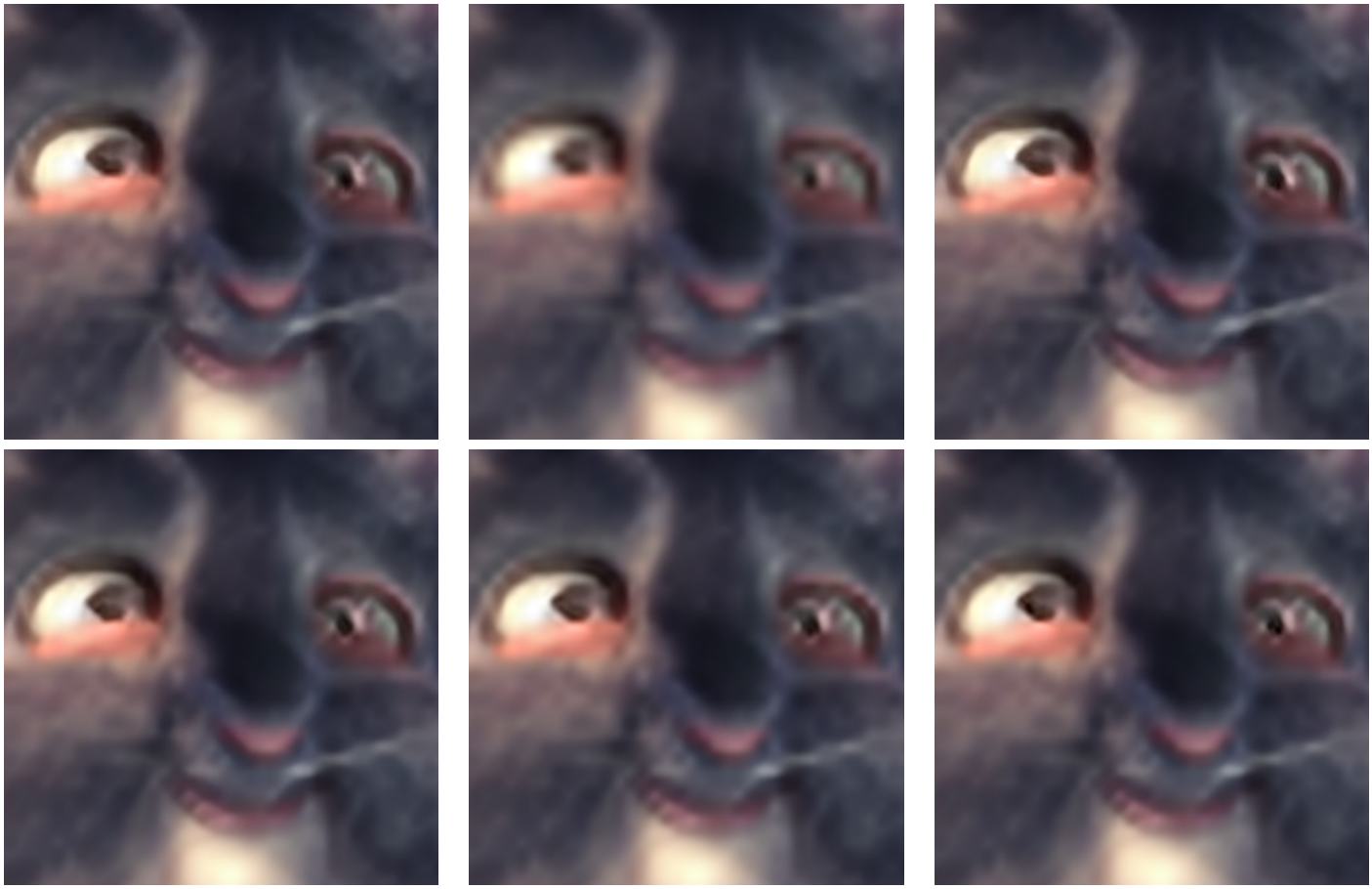

Figure 19: Whiskers scale 8 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

Figure 20: Butterfly scale 2 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

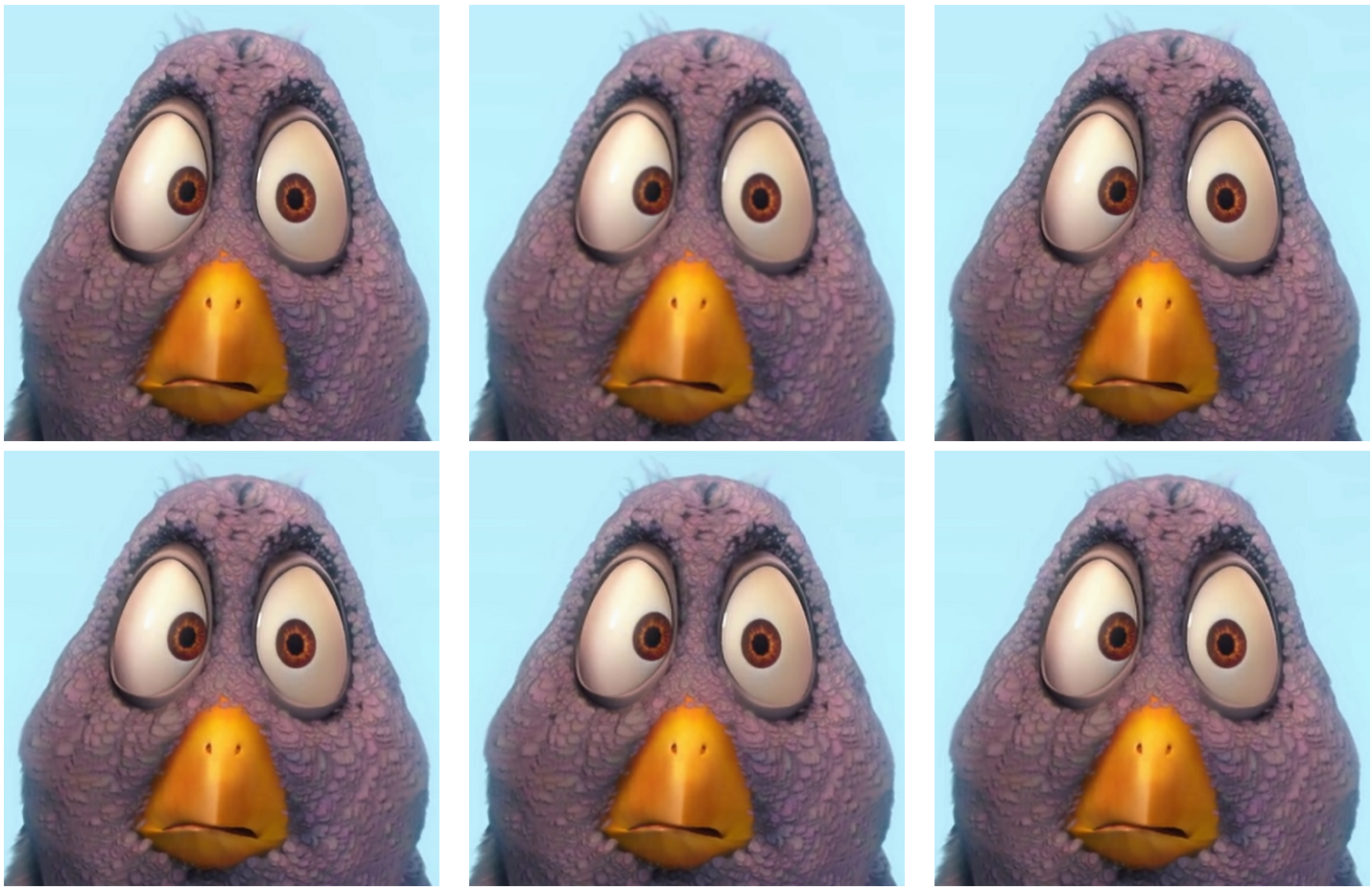

Figure 21: Bird scale 2 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

Figure 22: Grassland scale 2 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

Figure 23: Squirrel tail scale 2 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH

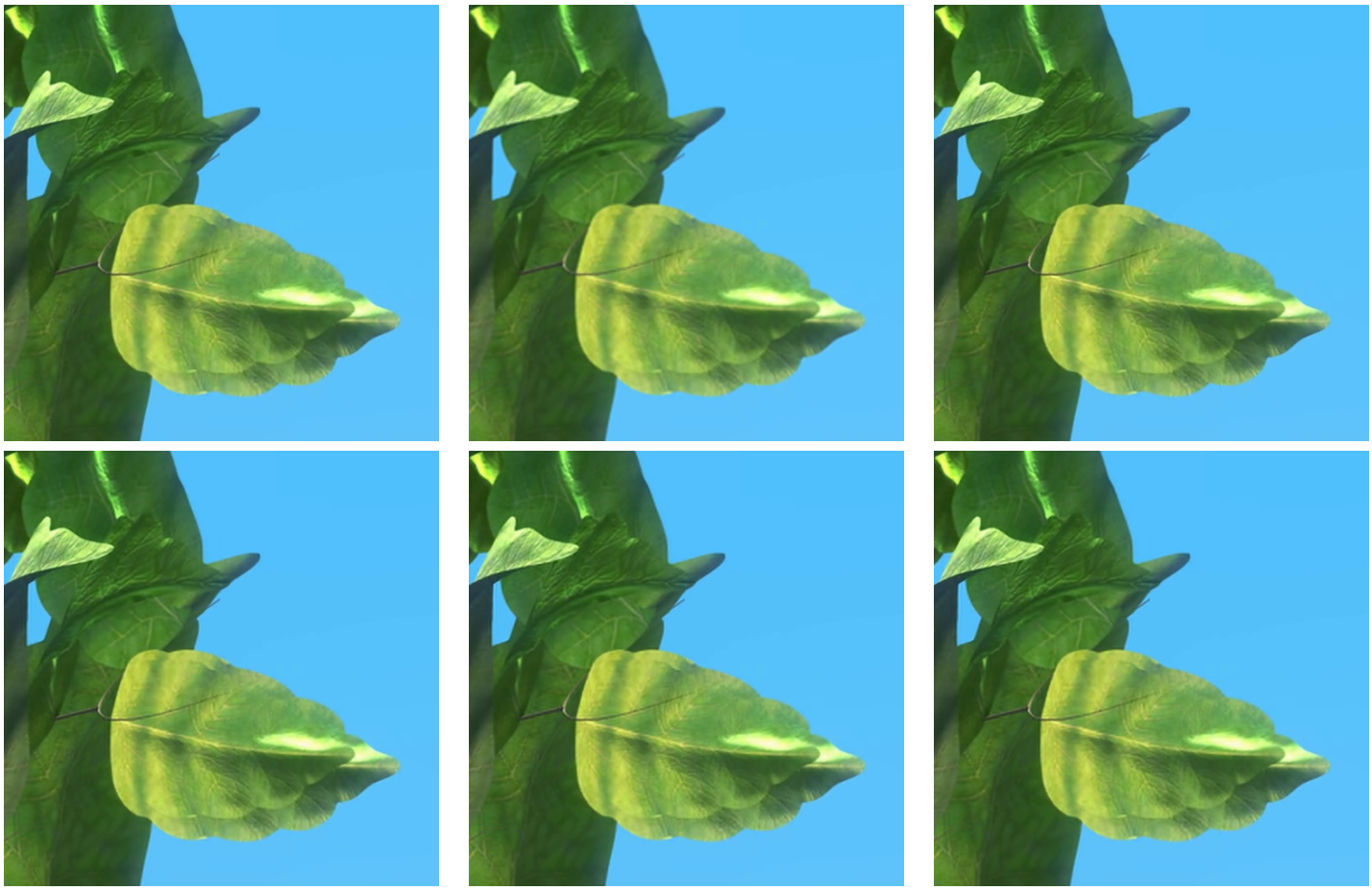

Figure 24: Leaves scale 2 - Baseline : Bicubic : ExtraL : ELU : SIGM : TANH