Continual-level Super-Resolution

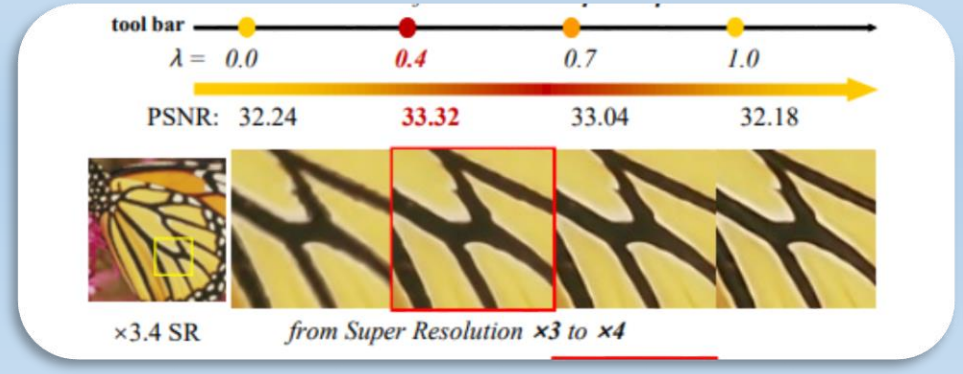
By

Eliran Elisha Harel Keller

Supervisor: Ofer Idan (Hauwei)

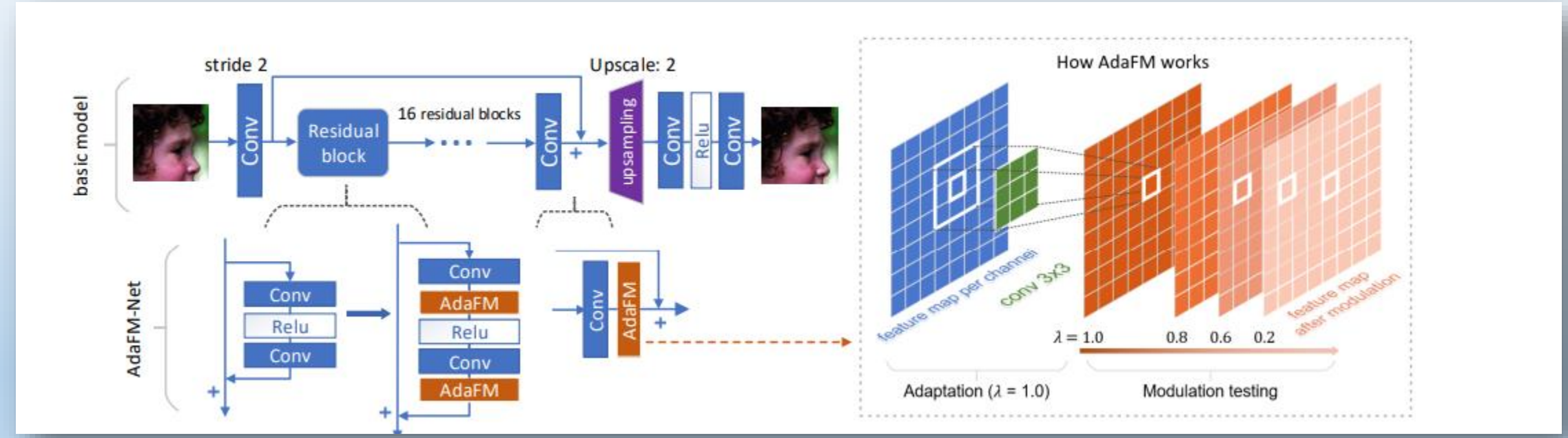
Introduction

- The main problem in image-restoration tasks - as denoising and super resolution, continual modulation of restoration level - is that transition from one step to another step is done discretely and not continuously.
- In practice, we have a difficulty modulating well-trained models with certain hyper-parameters.



Proposed solution

- Modulating Image Restoration with Continual Levels via Adaptive Feature Modification Layers [Jingwen He 2019]

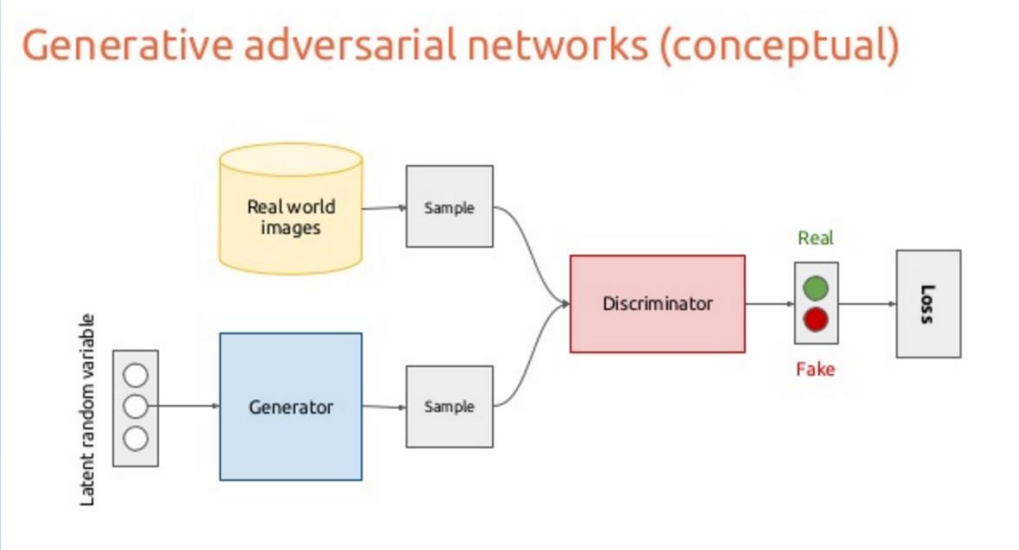


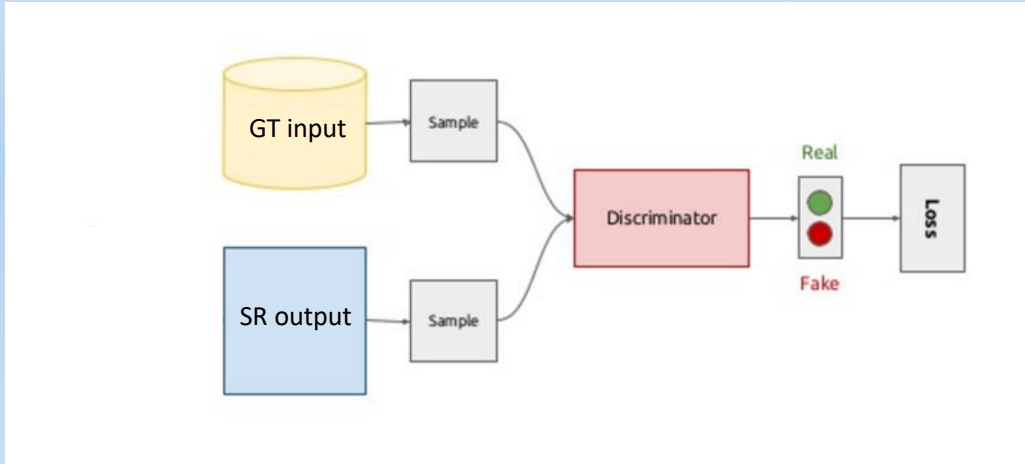
Our Innovation

- Implementation AdaFM code for SR
  - Both Super-Resolution and Denoising are problems at the pixel level.
    For solving the Denoising problem, we should use noise removal.
    For solving the SR problem, we need to reproduce textures along with high frequencies.
- Add a GAN ingredient to AdaFM

The process

- Implementation AdaFM code for SR
- Add a GAN ingredient to AdaFM
  - Add a GAN architecture to the model
  - Use the SR network output as the GAN generator





Dataset

- The original paper DIV2K dataset, DIV2K and Set5 dataset were used to evaluate the model.
- DIV2K is an high resolution dataset of 1000 images containing train, validation and test images.
- Set5 is a dataset of 5 images, used frequently for testing purpose only.
- The degraded samples of DIV2K and Set5 were created by a Matlab code.

evaluation

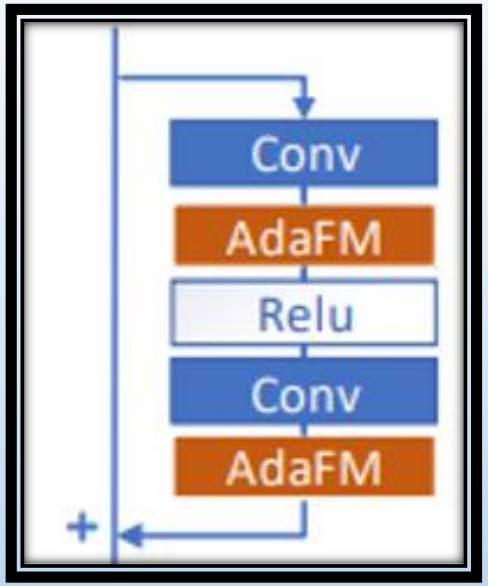
- Visual
- Numeric:
  - Reference image needed:
    - PSNR - Peak to Noise Ratio
    - SSIM - Structural similarity
  - No reference image needed:
    - Brisque
  - In our project we only used the PSNR method

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

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

PSNR

AdaFM SR implementation




```
(0): ResNetBlock(
  (res): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): AdaptiveFM(
      (transformer): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
    )
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): AdaptiveFM(
      (transformer): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
    )
  )
)
```

Results

Set5

Super resolution	1X1	3X3	5X5	7X7	Baseline
Original article	31.42	31.88	32.00	32.03	32.13
Our implementation	30.66	30.90	30.67	30.72	31.03

DIV2K test

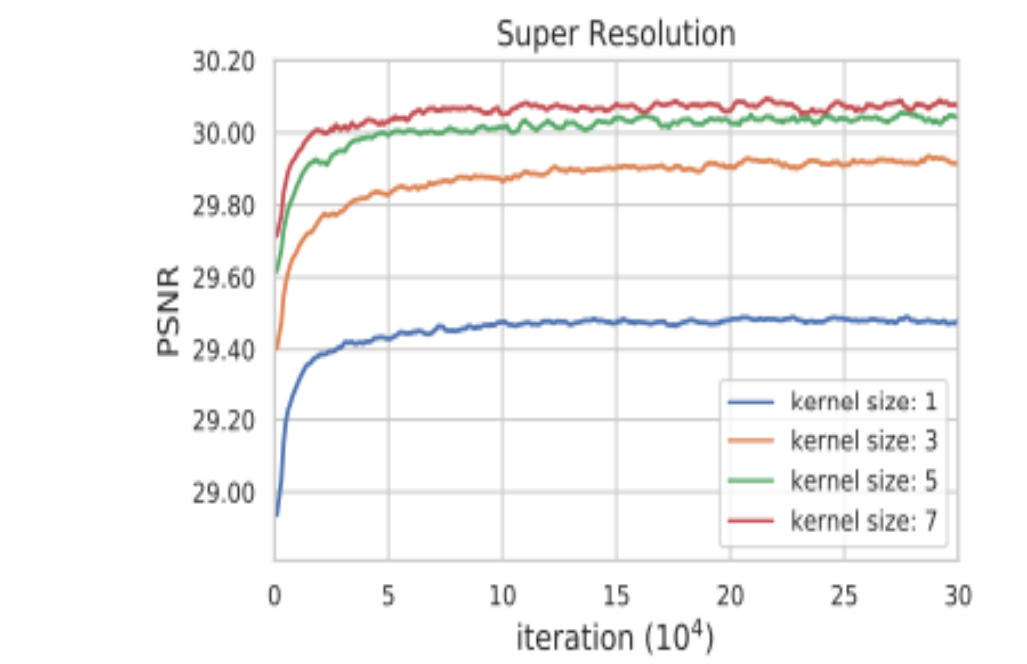
Super resolution	1X1	3X3	5X5	7X7	Baseline
Original article	29.89	30.20	30.28	30.30	30.37
Our implementation	28.86	29.35	29.54	29.6	29.38

- The results are a bit lower than the original article. This may have been caused by little changes in the code and lack of processing power.

Results

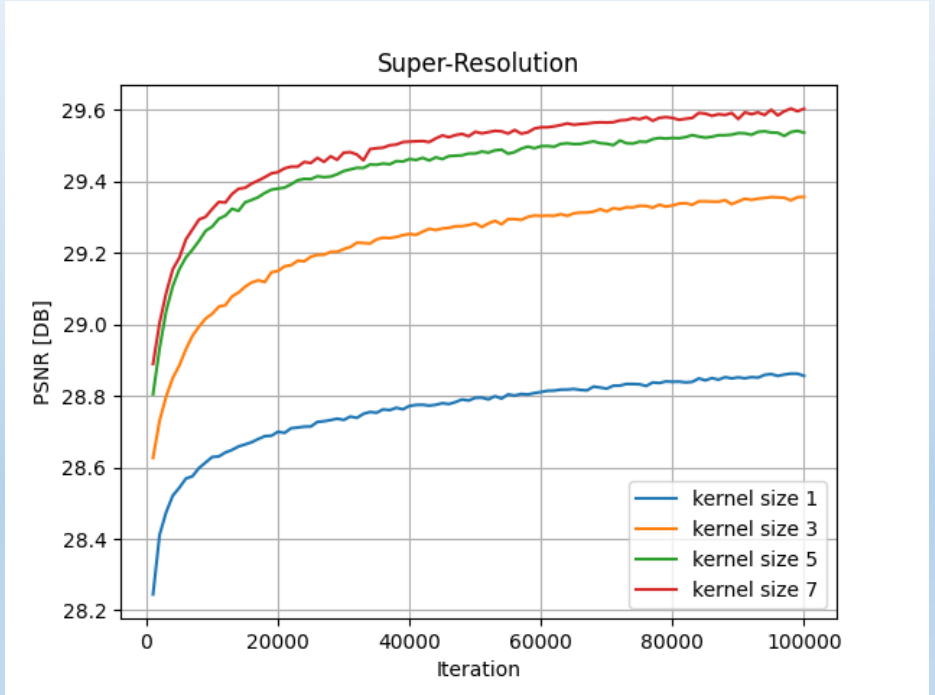
- PSNR over trains iterations for each Kernel size:

Original article



(Tested on DIV2K)

Our implementation



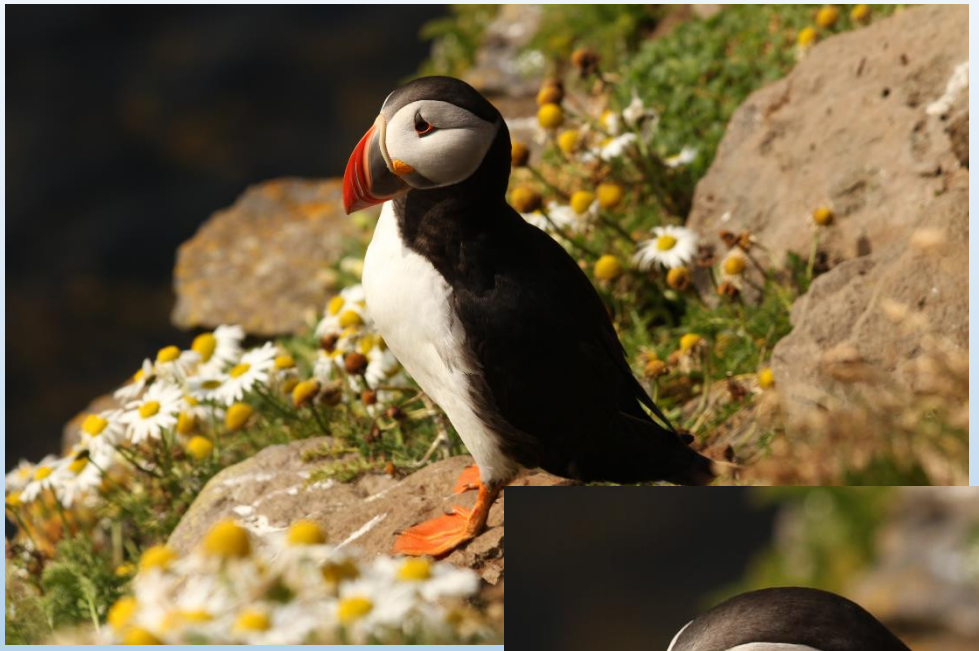
(Tested on DIV2K)

0

Open a File

Visual results

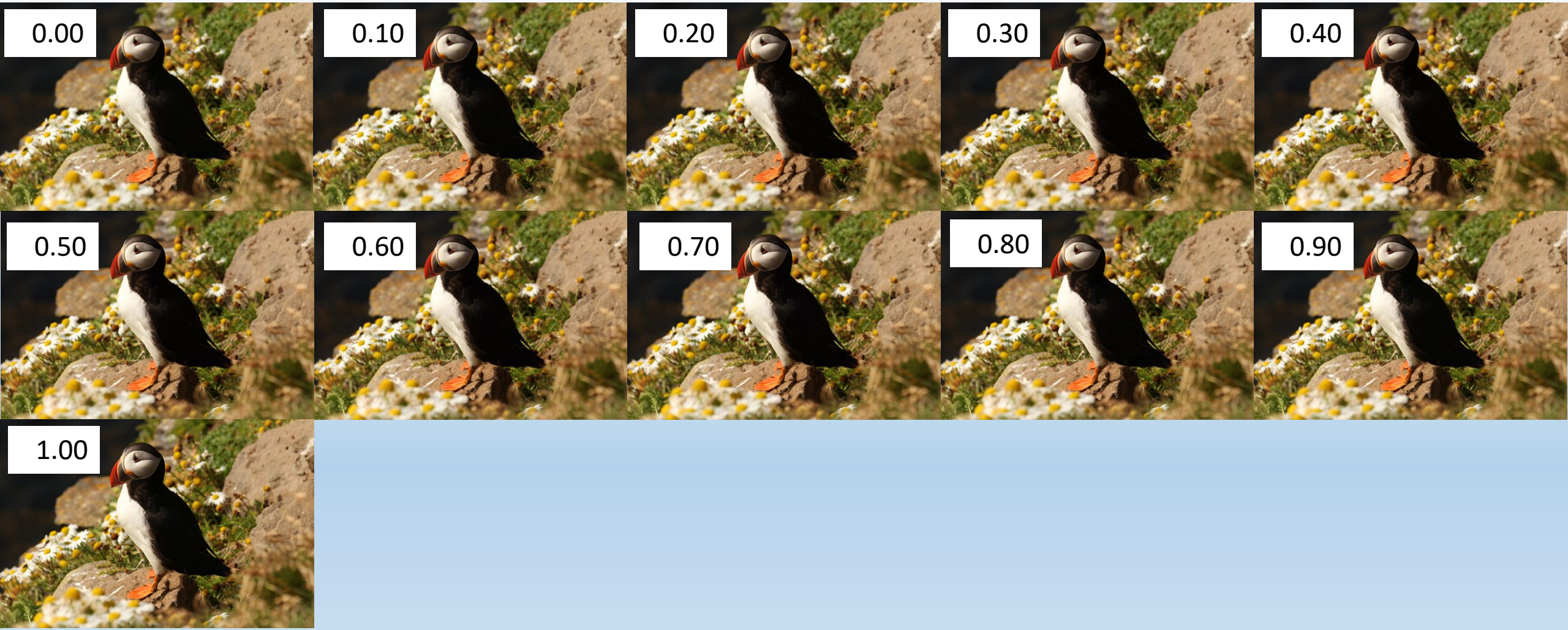
- For Kernel Size 5X5



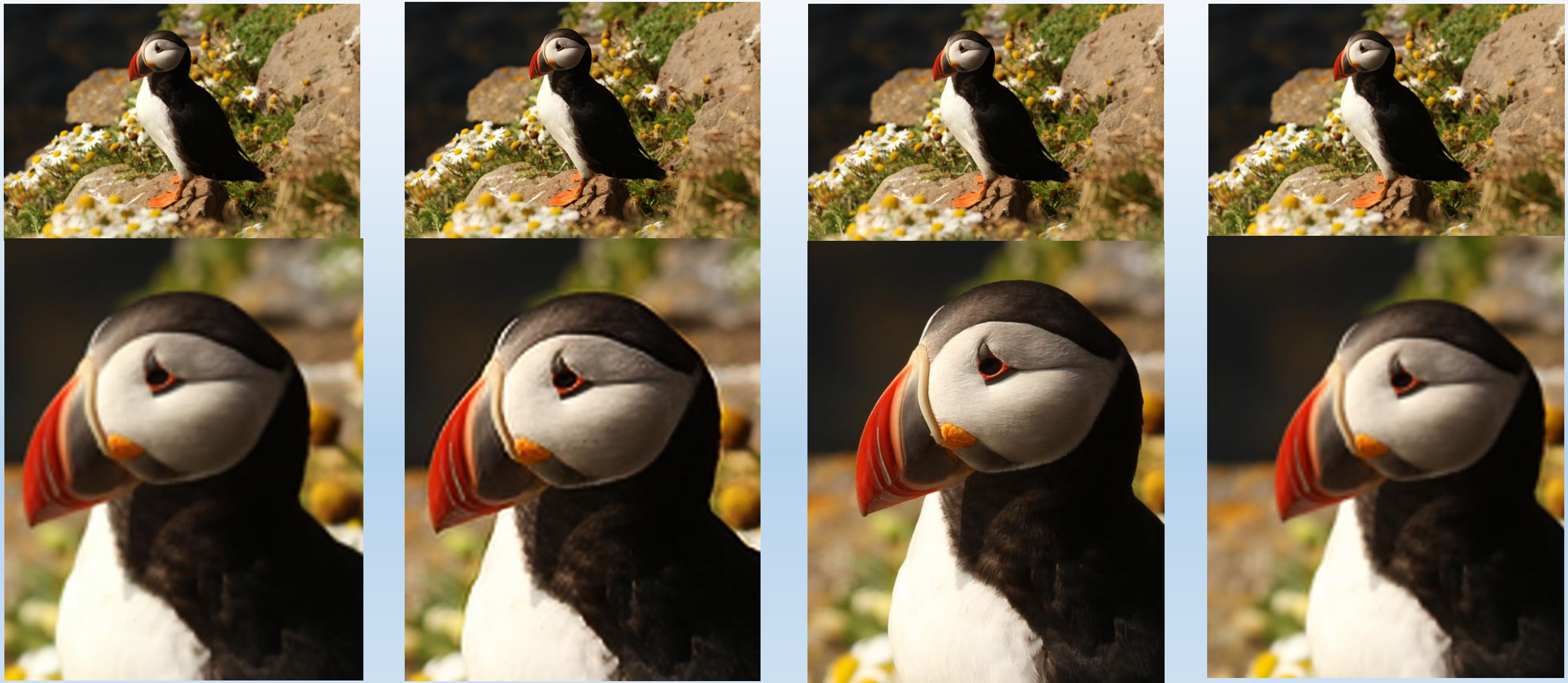
Original

X4

Visual results – x4



Coef 0.00 vs 1.00 vs Original HR image vs x4



Add a GAN ingredient to AdaFM

Results

Set5

Super resolution	1X1	3X3	5X5	7X7
ELGAN implementation	27.23	27.73	27.7	27.79

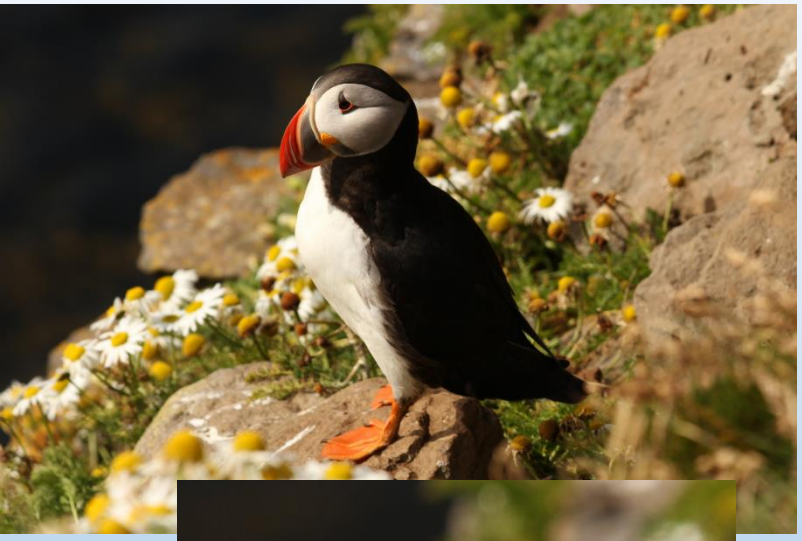
DIV2K test

Super resolution	1X1	3X3	5X5	7X7
ELGAN implementation	27.52	27.6	27.66	27.67

Visual results

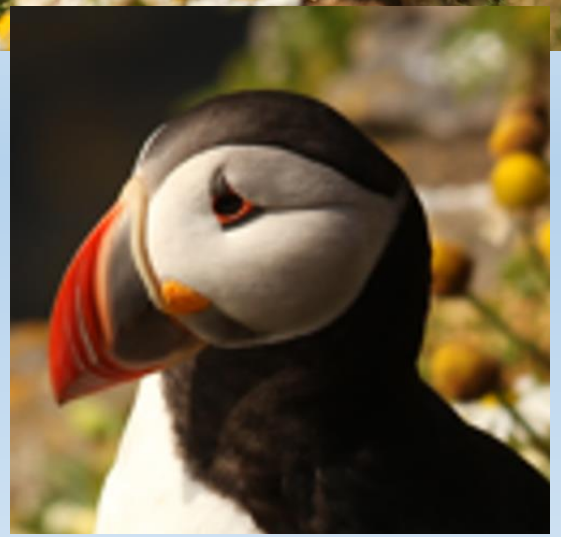
- For Kernel Size 5X5

Original

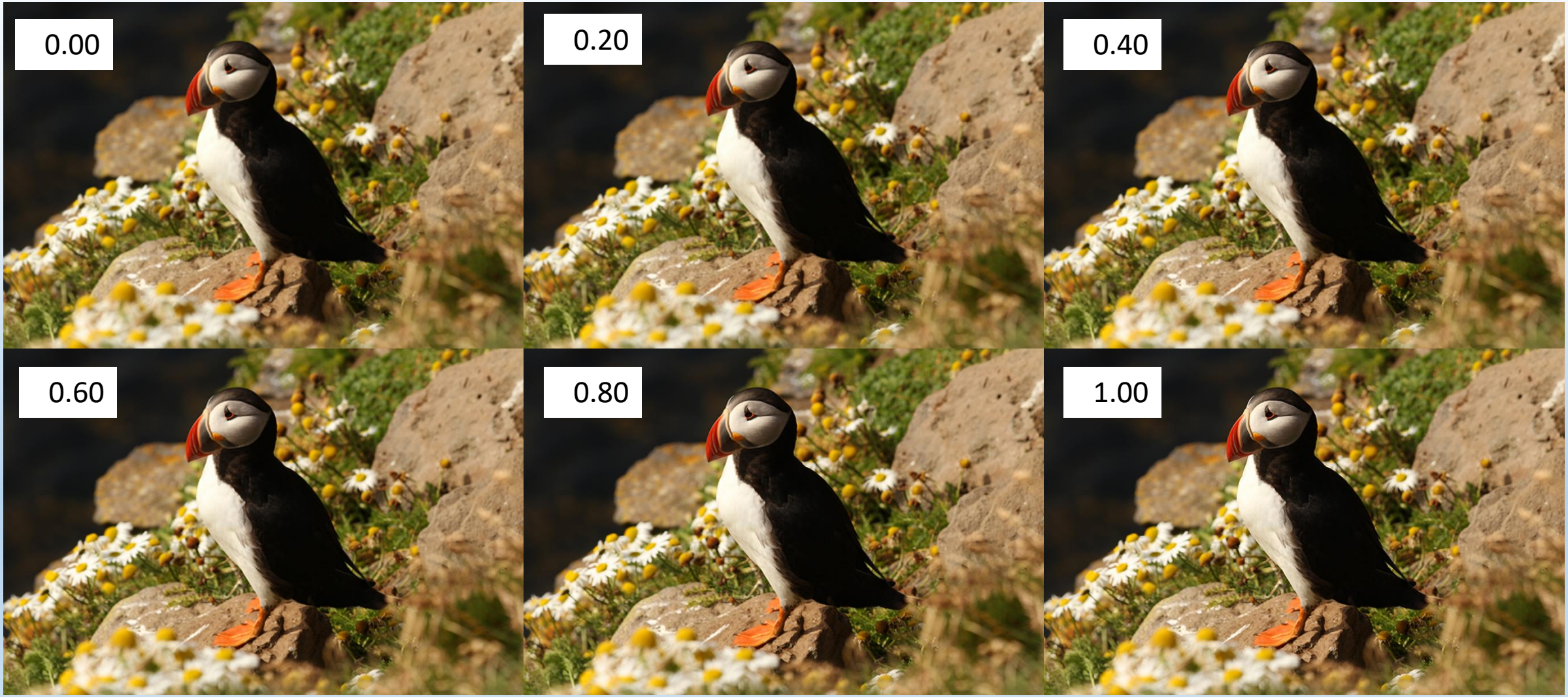


X3

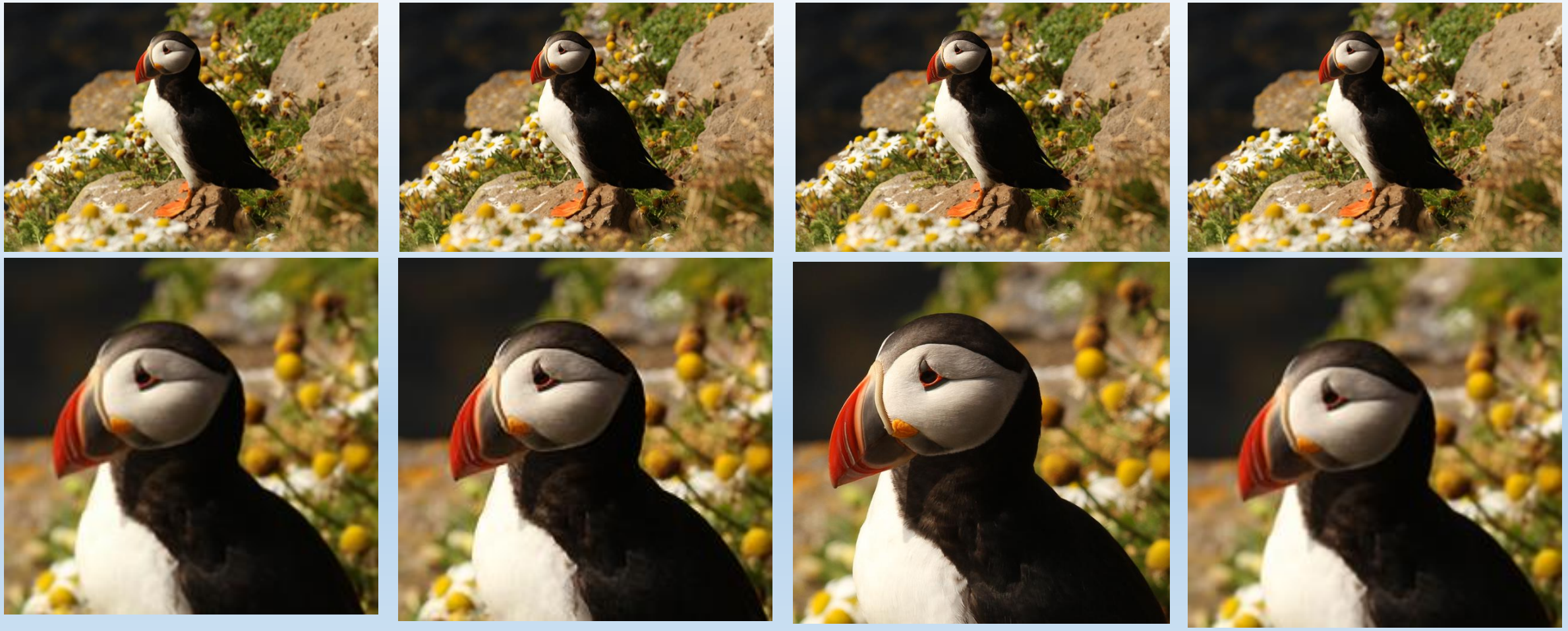
X4



Visual results – x4



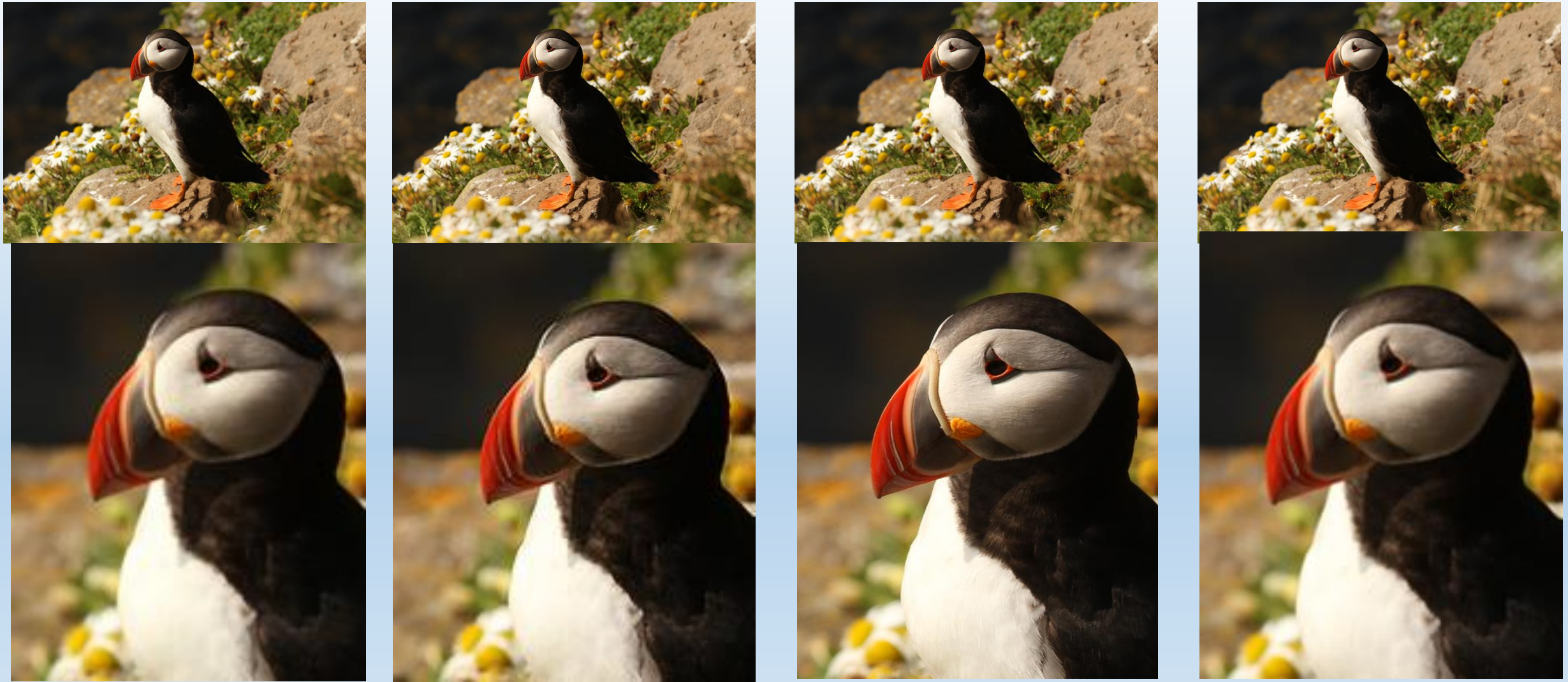
Coef 0.00 vs 1.00 vs Original HR image vs x4



Visual results – x3

	0.00	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00
PSNR	39.5	39.6	39.8	39.9	39.8	39.6	39.1	38.3	37.4	36.6	35.7

Coef 0.00 vs 0.7 vs Original HR image vs x3



AdaFM vs ELGAN

Results

Set5

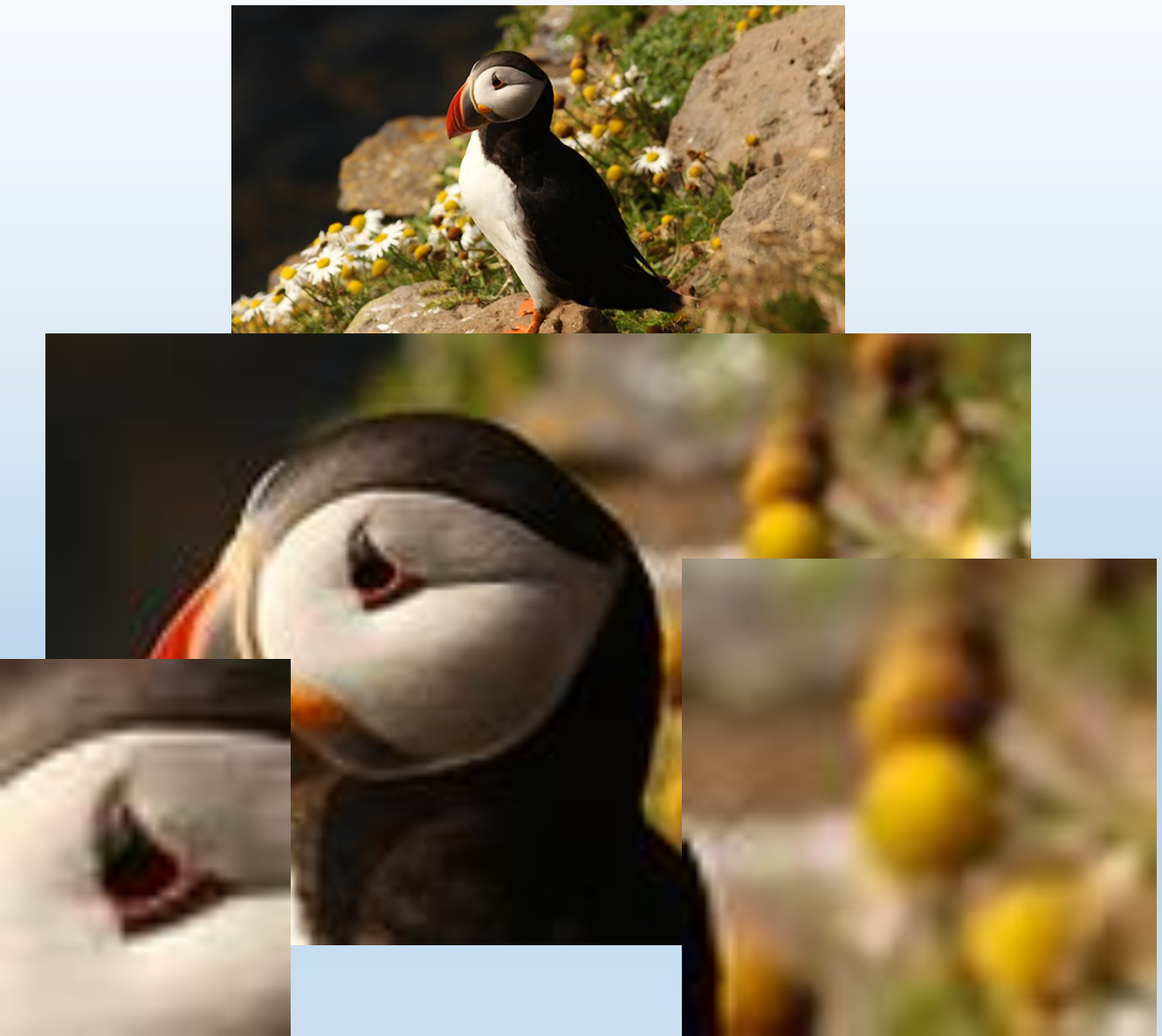
Super resolution	1X1	3X3	5X5	7X7
Our Adafm implementation	30.66	30.90	30.67	30.72
ELGAN implementation	27.23	27.73	27.7	27.79

DIV2K test

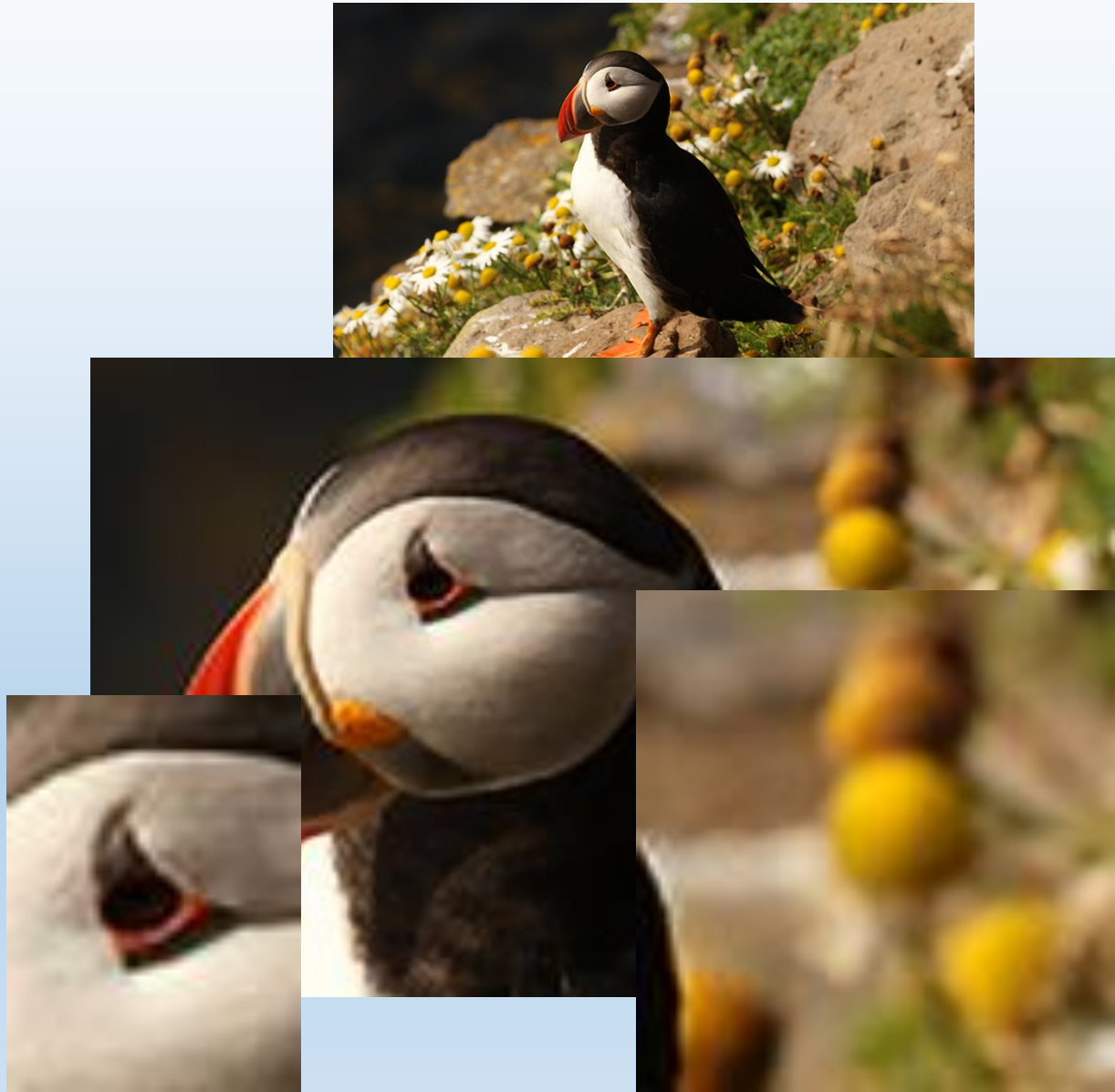
Super resolution	1X1	3X3	5X5	7X7
Our Adafm implementation	28.86	29.35	29.54	29.6
ELGAN implementation	27.52	27.6	27.66	27.67

Coef 1.00 vs 1.00 for X4 input

AdaFM



ELGAN



AdaFM
COEF 1.00

ELGAN
COEF 1.00

Conclusion

- As we can see, the PSNR of our model is lower than our AdaFM implementation.
- In the visual - as you can see - there is a little improvement.
- Next: Run the train model on bigger amount of train iterations. For now, ELGAN trained only on 1e4 iterations. The original AdaFM shows convergence after about 1e5 iterations.

Remember that image from the first slide?
This is how it look now.

Feel old yet?

Questions