ESRDGAN

Upscaling images using a Generative Adversarial Network

TDT4265 final project, spring 2019
Eirik Vesterkjær, MTTK

[Link to GitHub repo]

Project goals

1. Design a suitable GAN architecture for solving instances of the SISR problem based on recent research

2. Train the GAN to output magnified images from the input images

3. Observe and evaluate the results through established benchmarks through testing with new data

What I did

1. Researched existing implementations
2. Re-implemented an existing architecture in Pytorch (ESRGAN)
3. Modified it until I could achieve stable training
    1. This took *a lot of time.*
4. Trained it with Flickr2k
5. Observed and evaluated the results

Goal 1

Objective: SISR

Single Image Super Resolution

The key goal is to generate realistic looking high resolution images from low resolution input images.

Typical approach

x_{HR}

Downsampling (Bicubic)

$x_{LR} = f_D(x_{HR})$

x_{LR}

Upsampling (Solving the SR problem)

$x_{SR} = G(x_{LR}) \simeq f_D^{-1}(x_{LR})$

x_{SR}

Ideal scenario

$$G(x_{LR}) \equiv f_D^{-1}(x_{LR}) \quad \Longrightarrow \quad x_{SR} = G(x_{LR}) = f_D^{-1}(f_D(x_{HR})) = x_{HR}$$

But this is not possible, since f_D is an N:1 mapping (several HR images can give the same LR image)

So: We want to determine a G which is a good approximation of f_D^{-1}. But how?

HR image: “monarch” from Set14. SR image: output from my generator :)

Goal 1

Method: Deep Learning / cGAN

Generative Adversarial Network
In a GAN, two neural networks “compete”:
A discriminator (D), which classifies its input into 2 (or more) classes, and
A generator (D), which generates an output from an input (often gaussian noise).
In my project, I’m using a *conditional GAN* where the generator input is downsampled (LR) images.

How does the generator learn to generate sensible outputs?
The discriminator is trained to distinguish between real data and generated data.
The generator is trained to fool the discriminator, meaning that ……..
it is *trained to generate data that is perceived as real (by D at least..)*

Goal 1

Relevant literature and sources

- Deep Learning for SISR: A Brief Review, Yang et al. [arXiv, 2019]
    - Foundation
- Single-Image Super-Resolution: A Benchmark, Yang et al. [ECCV, 2014]
    - Foundation
- EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis, Sajjadi et al. [ICCV 2017]
    - State of the art Implementation
- SRGAN: Photo Realistic SISR Using a GAN, Ledig et al. [arXiv, 2017]
    - Near SOTA implementation
- ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks, Wang et al. [arXiv, 2018]
    - SOTA implementation
    - GitHub repo: BasicSR
- SRDGAN: Learning the noise prior for Super Resolution, Guan et al. [arXiv, 2019]
    - SOTA implementation focusing on SR applications on normal images (not bicubic downsampled)
- The Relativistic Discriminator, Jolicoeur-Martineau [arXiv, 2018]
    - For more stable training
- Which training methods for GANs do actually converge? Mescheder et al. [arXiv, 2018]
    - Discusses instance noise, among other things.
- Tricks of GANs (lanpartis.github.io)

Goal 1

GAN Architecture

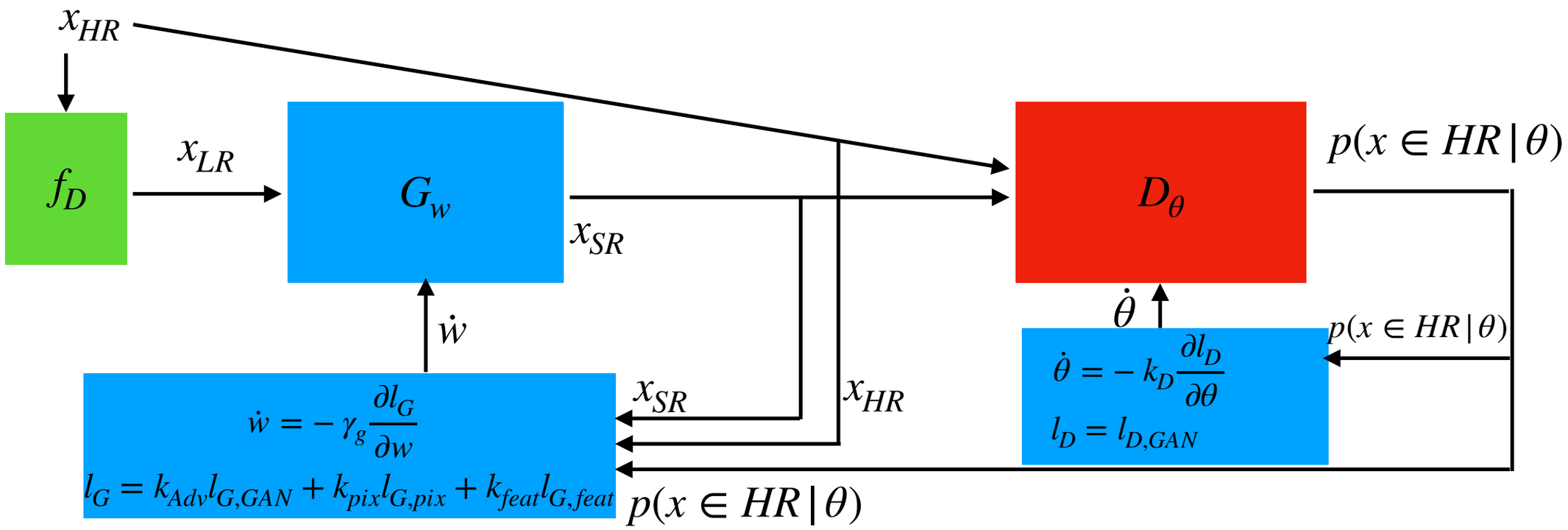
ESRGAN

I based my implementation on two existing implementations, ESRGAN and SRDGAN, which in turn is also based on ESRGAN. In order to understand what was going on, I coded the project from scratch (*mostly* based on ESRGAN's code).

Implementing this and achieving stable/working training was the bulk of my project. (Done in PyTorch)

"This code is not very easy to understand. Let's remake it from scratch - that shouldn't be too difficult" - me

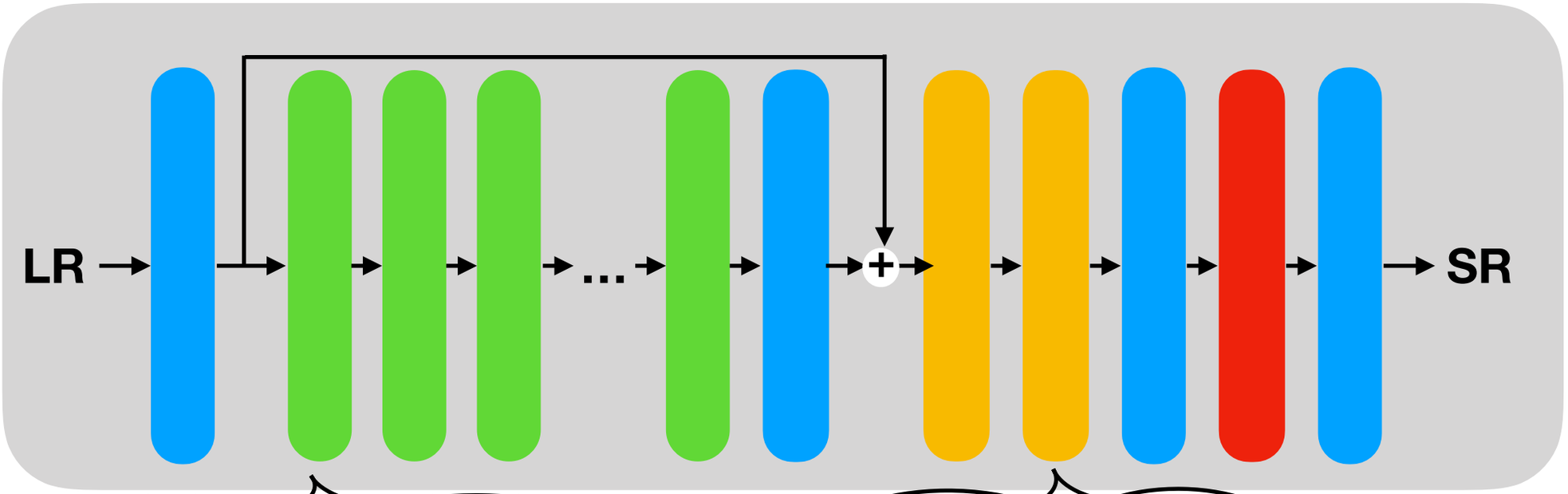
"It was, in fact, pretty difficult" - narrator

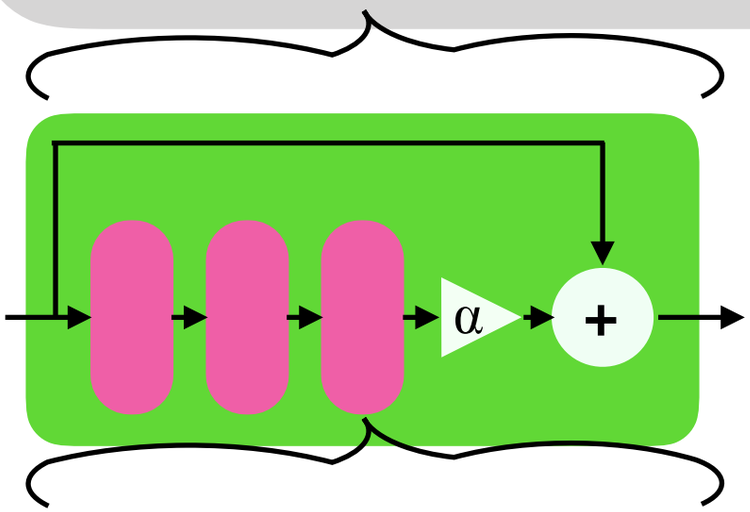


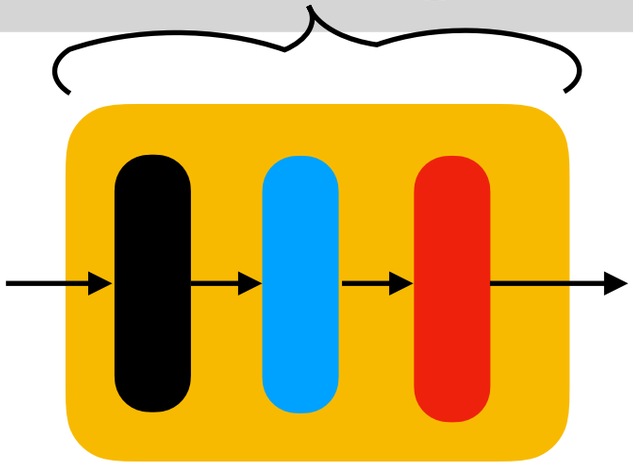
Note: I actually use the Adam optimizer, not SGD.

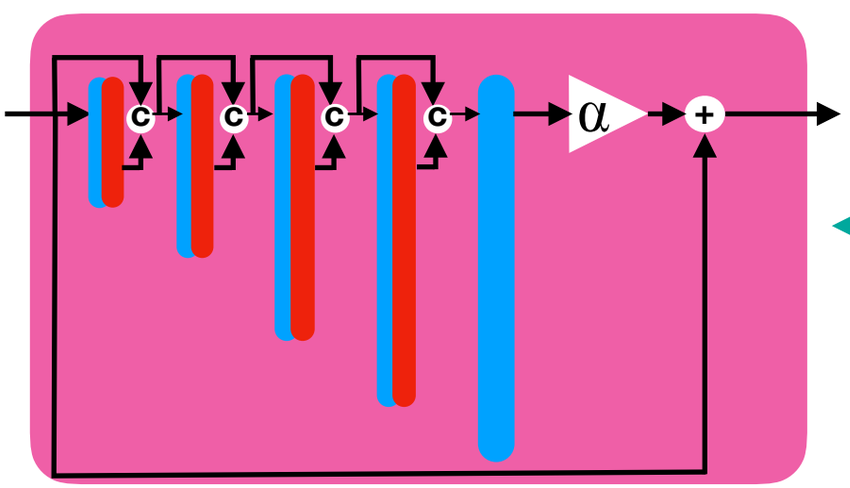
Goal 1

Generator Architecture









Legend (G)

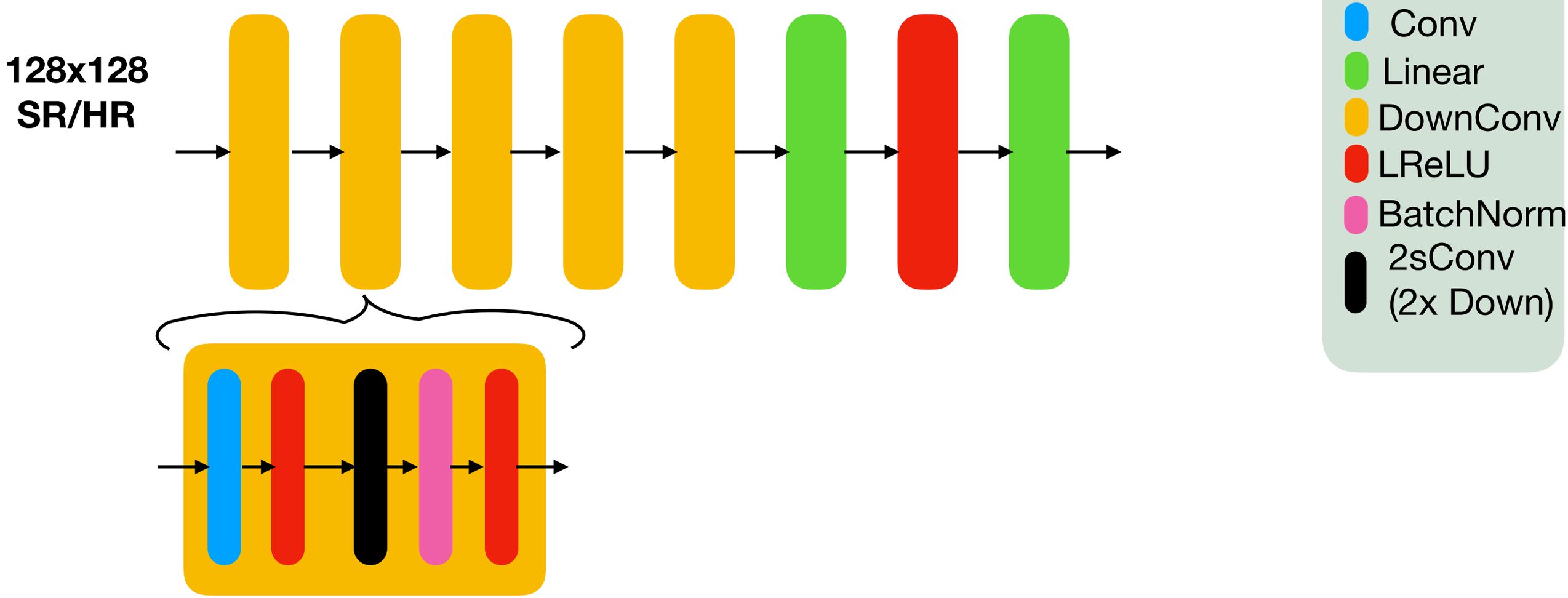
- Conv
- RRDB
- UpConv
- LReLU
- RDB
- 2x NN Up
- c Concat

Residual in Residual Dense block: “RRDB” (ESRGAN)

Residual Dense Block: “RDB” (ResNet)

Goal 1

Discriminator Architecture



Note: I also use a feature extractor: Pre-trained Pytorch VGG-19, layers 0 - 34. (Same as for ESRGAN)

Goal 2

Loss fn and hyperparameters

- **Loss function**
  - **BCE relativistic adversarial loss, L1 pixel loss, L1 feature loss**

$$L_D = L_{D,Adv}^{Re}$$

$$L_G = k_{Adv}L_{G,Adv}^{Re} + k_{Pix}L_{G,Pix} + k_{Feat}L_{G,feat}$$

$$L_{D,Adv}^{Re} = \frac{log[\sigma(D(x_{hr}) - D(x_{sr}))] + log[1 - \sigma(D(x_{sr}) - D(x_{hr}))]}{2}$$

$$L_{G,Adv}^{Re} = \frac{log[1 - \sigma(D(x_{hr}) - D(x_{sr}))] + log[\sigma(D(x_{sr}) - D(x_{hr}))]}{2}$$

Adversarial loss

$$L_{G,Pix} = ||x_{sr} - x_{hr}||_1$$

$$L_{G,Feat} = ||F(x_{sr}) - F(x_{hr})||_1$$

Pixel and feature loss

Goal 2

Objective: Train the architecture

- **Training dataset:** Flickr2k, subcrops (256 x 256) of the original images.
  - These are randomly cropped to 128x128 HR images during the run, in __getitem__
    - LR images are made during the run, using cv2.resize with inter_CUBIC (bicubic interpolation)
  - Data augmentation: Shuffle, flip
  - Total # of training data in Flickr2k: 2650
  - Total # of 256x256 crops: 143313 (interestingly: My dataloader shows len==17914)
  - Total # of 128x128 crops: 143313 * 128 * 128 = 2 348 040 192
  - Total possible images after shuffle & flip: 2 348 040 192 * 2 * 2 = ~10 billion

- **Validation dataset:** BSDS100
  - Without shuffle etc., but cropped
  - Total # of validation data: 100

- **A note on parameters:** *I did not achieve good results using ESRGAN's default parameters*
  - Implementation differences?
  - Training time?
    - ESRGAN's training time: 24h, starting with a G pretrained for high PSNR
    - I trained for ~24-14h max, on a non-pretrained G, on a *far* weaker GPU, with smaller b
  - I spent a lot of time finding parameters that worked even a bit.

Goal 2

Loss Fn and hyperparameters

- **There are more, but these are perhaps the most important hyperparams:**

Parameter	Value	Comment
Scale	4	SR upscale factor
Base # of Features (G)	128	In ESRGAN: 64
RRDB blocks (G)	12	In ESRGAN: 23
Residual scale α (G)	0.2	Same as ESRGAN
Base # of features (D)	96	In ESRGAN: 64
Learning rate (G and D)	10^-4	Same as ESRGAN
Optimizer	Adam	Same as ESRGAN
LR Scheduling	$\gamma \leftarrow 0{,}5 \cdot \gamma$ at iter [10k, 20k, 30k, 40k]	Sooner than ESRGAN
# training iterations	200k	~2/5 of ESRGAN
Pixel loss weight (G)	10^-2	Same as ESRGAN
Feature loss weight (G)	10^-1	In ESRGAN: 1.0
GAN loss weight (G)	5·10^-3	Same as ESRGAN
Batch size	8	In ESRGAN: 16
GAN type	Relativistic	In ESRGAN: Relativistic Avg
# of images trained on	200 000 * 8 = 1 600 000	500 000? * 16 ~= 8 000 000

Goal 2

Changes that helped

- Logging data and using Tensorboard (I didn't do this at the start)
- Instance noise (gaussian noise on HR/SR images fed to D)
  - Starts at σ = 1, decreases linearly, to σ = 0, at 60k it, then train until 200k it
- More features:
  - 64 -> 128 for G,
  - 64 -> 96 for D
- One sided label smoothing
- Training for a *long* time - much longer than anticipated
  - 10+ hours per run
- Shallower architecture (due to shorter required training time, possibly..)
- Updating G only every 2 iterations
- Relativistic GAN (honestly it only *might* have helped - I changed it just a short while before things started working, so I'm inclined to think that it contributed a bit..

Changes that didn't seem to help

- Noise on LR images (I used σ = 0.01 for [0,1] normalized images)
  - Maybe too much noise?
- Noisy labels (at least not noticeably)
- Changing the feature extractor (adding low level features output)
- NN downsampling for generating more realistic LR images
- A lot of other minor stuff

Goal 2

Resulting performance

NN Upsampled LR (4x) SR (4x) HR

Goal 2

Resulting performance

NN Upsampled LR (4x) **SR @ 100k (4x)** HR

Goal 2

Resulting performance

NN Upsampled LR (4x) **SR @ 200k (4x)** HR

Goal 2

Resulting performance

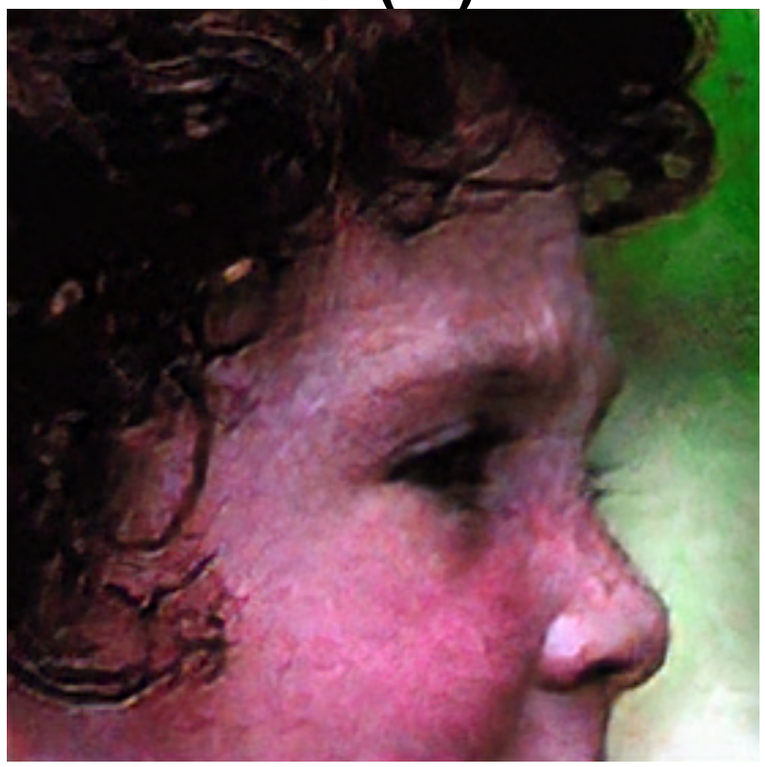
NN Upsampled LR (4x) SR (4x) **HR**

Goal 2

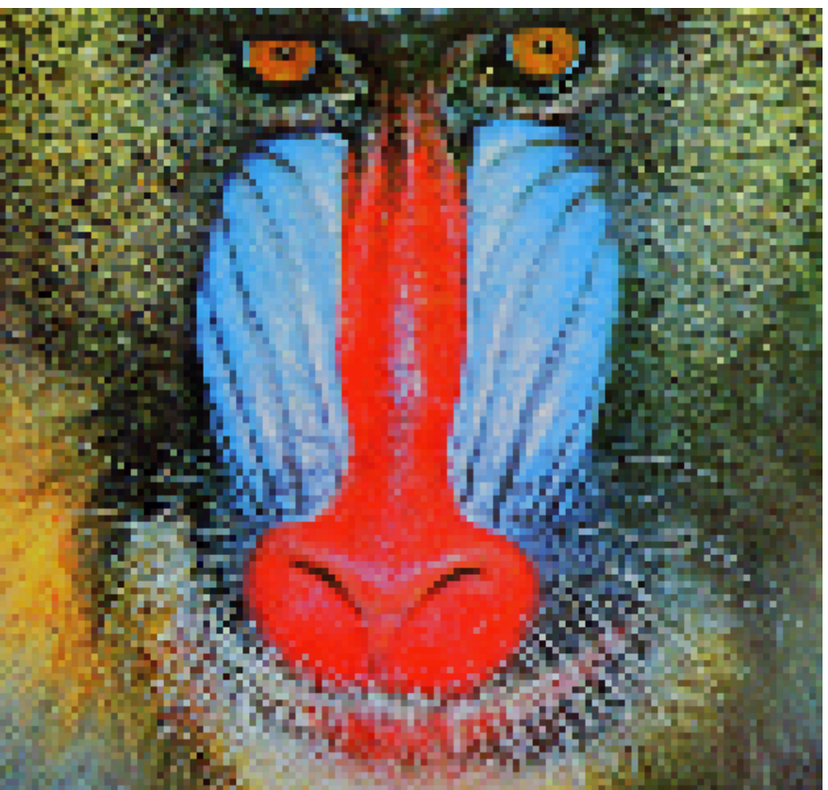
Resulting performance

NN Upsampled LR (4x)

SR (4x)



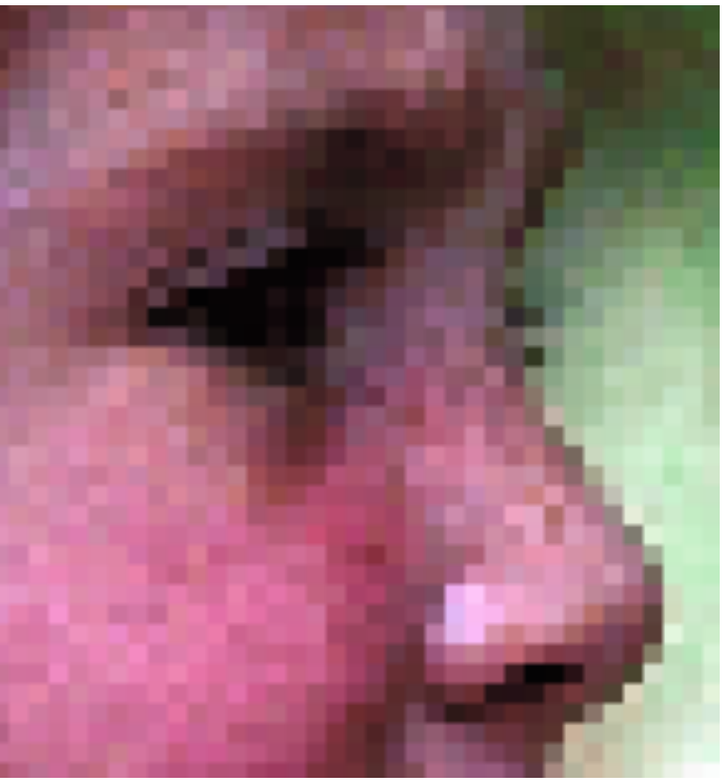
HR

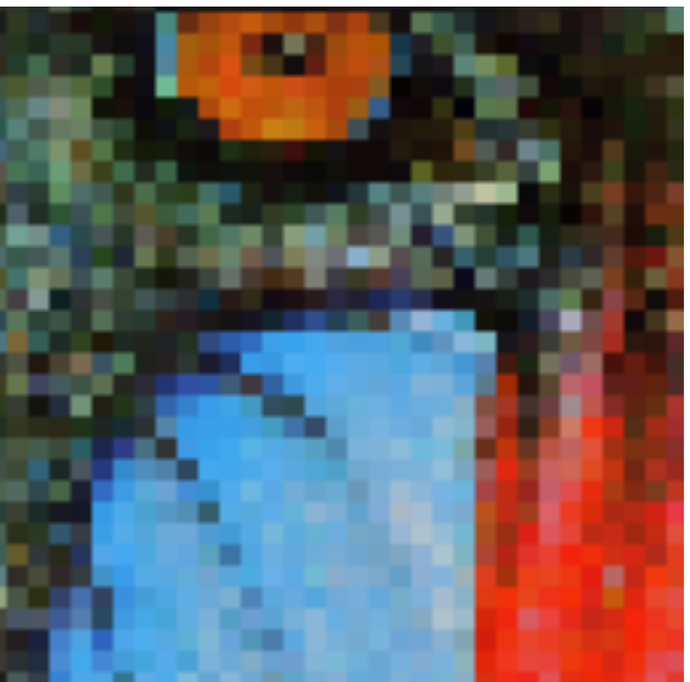


Goal 2

Resulting performance

NN Upsampled LR (4x) | **SR (4x)** | **HR**



Goal 2

Resulting performance

- The SR images look pretty decent, but slightly resemble oil paintings, I think.

- Seems to be better than bicubic interpolation in most cases at the very least!
  - Success!
  - But not as good as ESRGAN, or SRGAN

- Fine-grained features are not recreated. This is not surprising - *but* there are SR implementations which do just that .. like ESRGAN, EnhanceNet, etc.
  - Might be due to decreased depth
  - Or maybe training time.

SR (4x)

Bicubic upsampled LR (4x)

Goal 2

Resulting performance

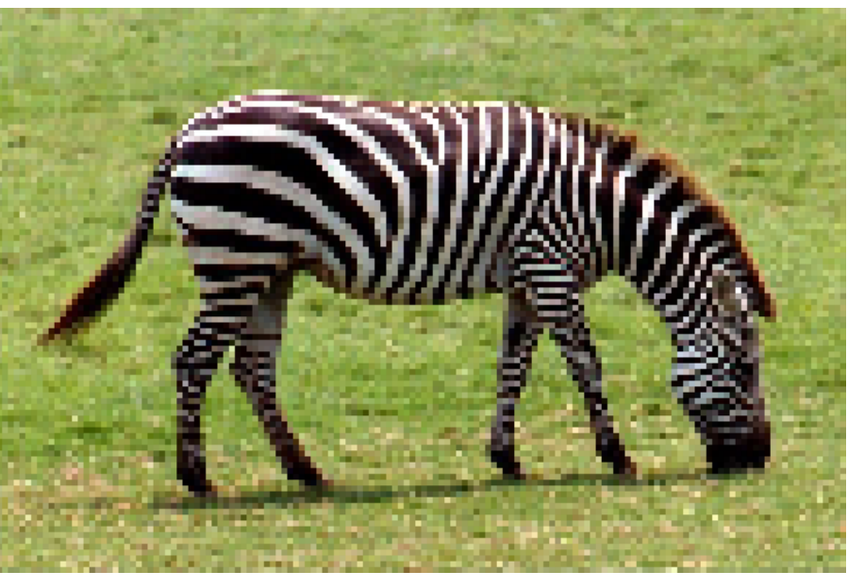
- **Here is an image which didn't turn out as good.** Often, this seems to be a consequence of the downsampling. In the lower images, you can see the patterns on the SR appear somewhat similar to the LR - but the LR is quite different from the HR image, which in turn causes the SR to be quite different from the HR!

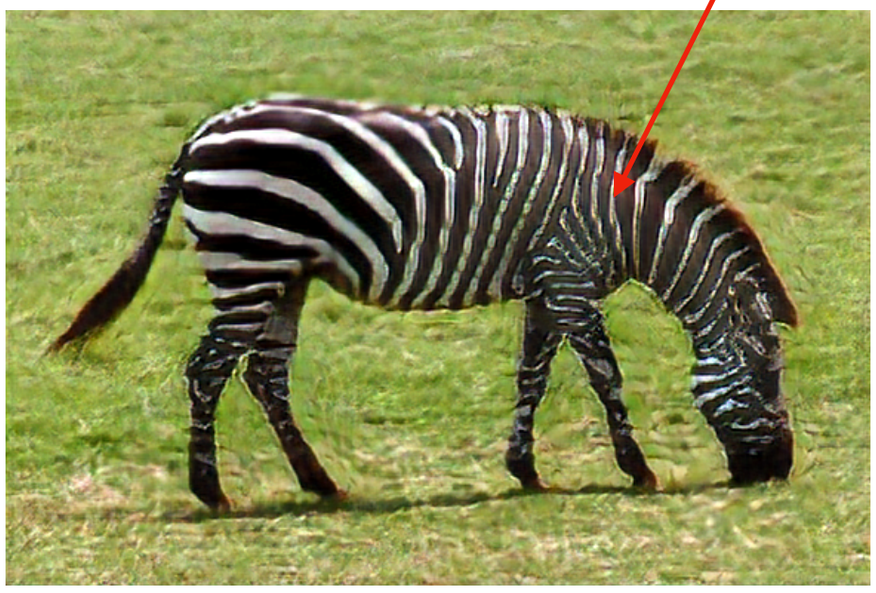
NN Upsampled LR (4x)

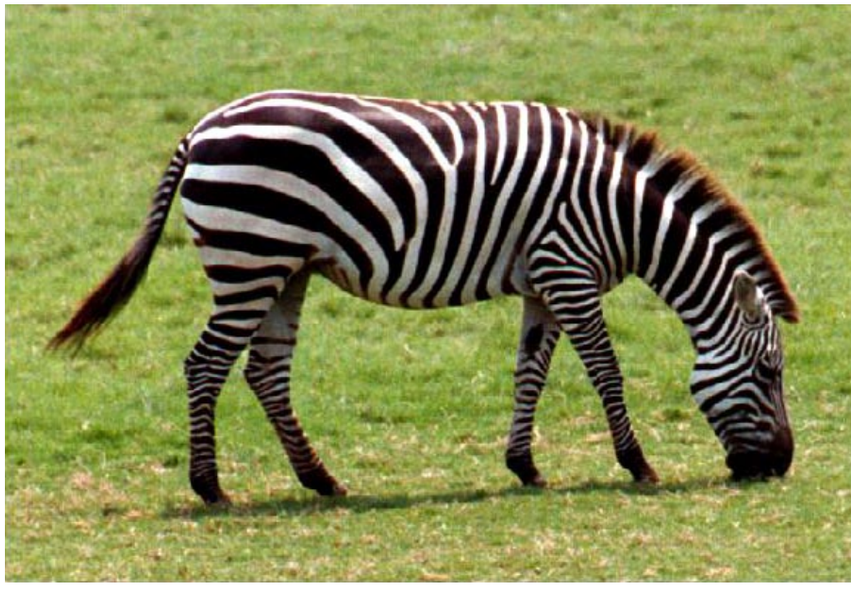
SR (4x)

Artefacts

HR







Goal 2

Resulting performance on my own image

- I could not upsample large images on my GPU (anything larger than ~200x200) due to running out of VRAM.
- So I did it using the CPU. A 400x700 image then takes about 6-8 minutes to upscale. :-(

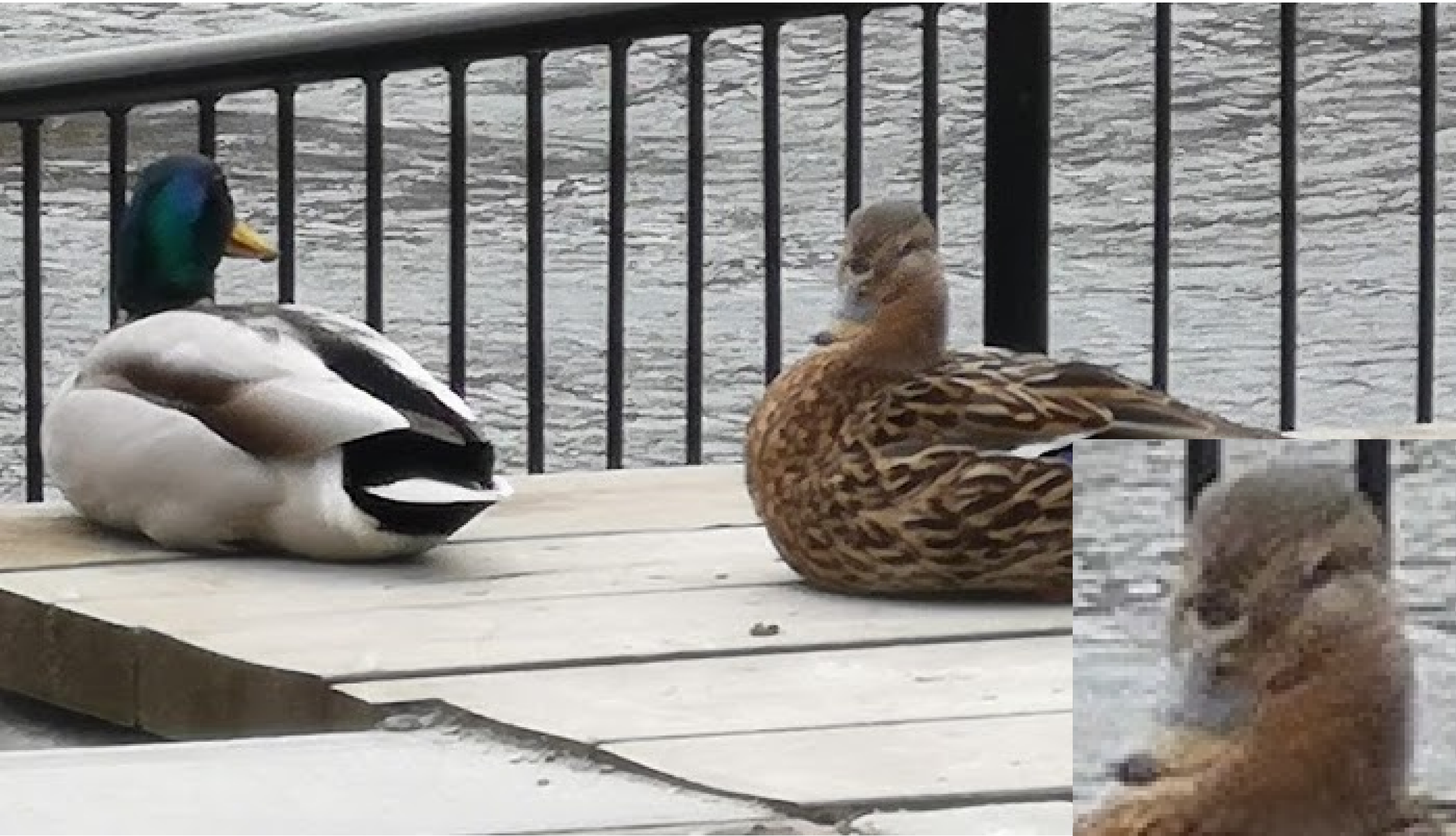
PID	COMMAND	%CPU	TIME	#TH	#WQ	#PORT	MEM	PURG	CMPRS
10959	python3.6	89.4	05:15.77	2/1	0	22	52G	0B	50G-
53	fseventsd	58.0	16:39.30	23/1	1	323+	8328K+	0B	5004K
0	kernel_task	39.0	12:30:15	154/4	0	0	798M	0B	0B

- Anyways: I think the result is pretty promising..!
  - *Much* better than NN interpolation
  - (Arguably) *notably* better than bicubic interpolation
  - Edges are kept, and appear pretty sharp.
  - One downside that I note, is that there is a slight change in the colour tint. This also happens during training, but the amount of colour shift falls over time.
    - This points to the model not being fully converged.
  - Also: *Artefacts!*

Goal 2

Resulting performance on my own images

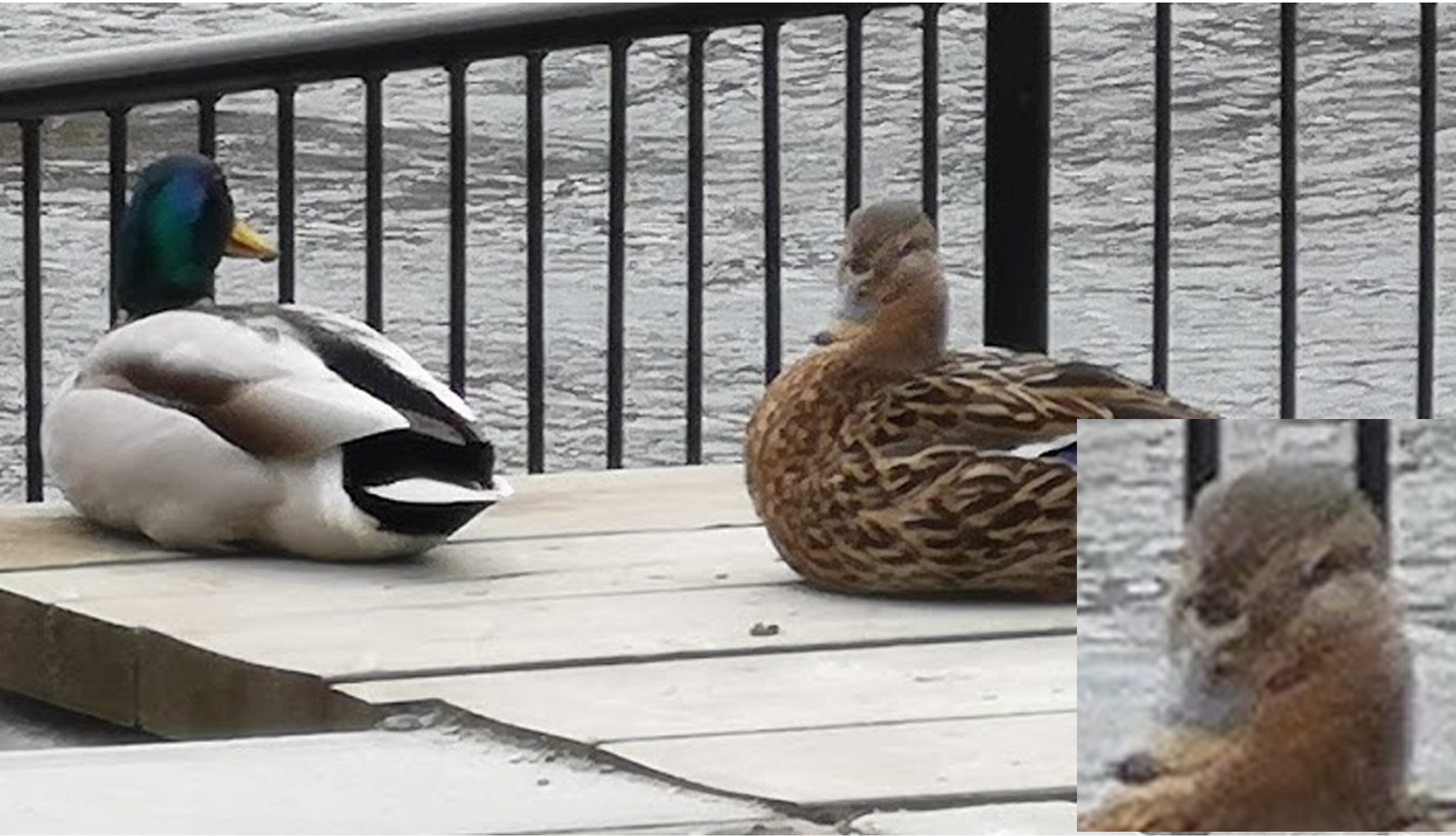
NN Upsampled (4x)



Goal 2

Resulting performance on my own image

Bicubic upsampled (4x)



Goal 2

Resulting performance on my own image

SR (4x)

Goal 3

Evaluation / metrics

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

- **Pixel space PSNR**
- **What? Are the Bicubic upsampled images all better?? Not necessarily!**
- **Why PSNR might not be the greatest measure…**
  - **Blur!**
  - **Instead:**
    - **Mean opinion score**
    - **Feature space PSNR**
  - **The amount of work I'll do is not certain - I've spent a lot of time working on this project already..!**

Set 14 Image	PSNR
baboon_hr_sr	17.229043530044972
baboon_hr_nearest	18.110912987687307
baboon_hr_bicubic	18.79858207417221
bridge_hr_sr	19.899253174835984
bridge_hr_nearest	20.599041548642575
bridge_hr_bicubic	21.699834285603078
coastguard_hr_sr	20.460025364432447
coastguard_hr_nearest	21.529752685426743
coastguard_hr_bicubic	22.031791840675115
foreman_hr_sr	24.30021950754444
foreman_hr_nearest	23.150601460245056
foreman_hr_bicubic	25.14909873084494
lenna_hr_sr	25.317115961771023
lenna_hr_nearest	25.387693700251326
lenna_hr_bicubic	27.396863325177844
man_hr_sr	21.554638975651276
man_hr_nearest	21.782458308182843
man_hr_bicubic	23.225622161674906
monarch_hr_sr	23.56851478474509
monarch_hr_nearest	22.927423468510515
monarch_hr_bicubic	25.388237003543217
pepper_hr_sr	24.784693537492867
pepper_hr_nearest	24.171995366756587
pepper_hr_bicubic	25.814109906291826

Conclusion

- This was a fun project, but it took a bit too much time (mostly because of my choice to implement the GAN myself, and the training time)
  - But I did learn a lot.

- Coding a GAN isn't terribly hard, but locating the source of bugs can be *very* challenging.
  - Implementing it myself showed some interesting things: like that ESRGAN's RDB differs from what is specified in the paper they refer to in their code.
    - It also forced me to actually understand the structure of the model

- Training a GAN is very difficult and takes a lot of time.

- Although the resulting generator is only "ok" when looking at the standard benchmark images, it *might* be more suited for upscaling real-world images (or more suited for images of pretty high resolution)

- It's very motivating to get a decent output from a model that I implemented myself