

# ESRDGAN

Upscaling images using a Generative Adversarial Network

TDT4265 final project, spring 2019  
Eirik Vesterkjær, MTTK

Note: There are a lot of slides, but many of them are here mostly for reference to myself.

I doubt I'll use as many for the final presentation :)  
[ [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

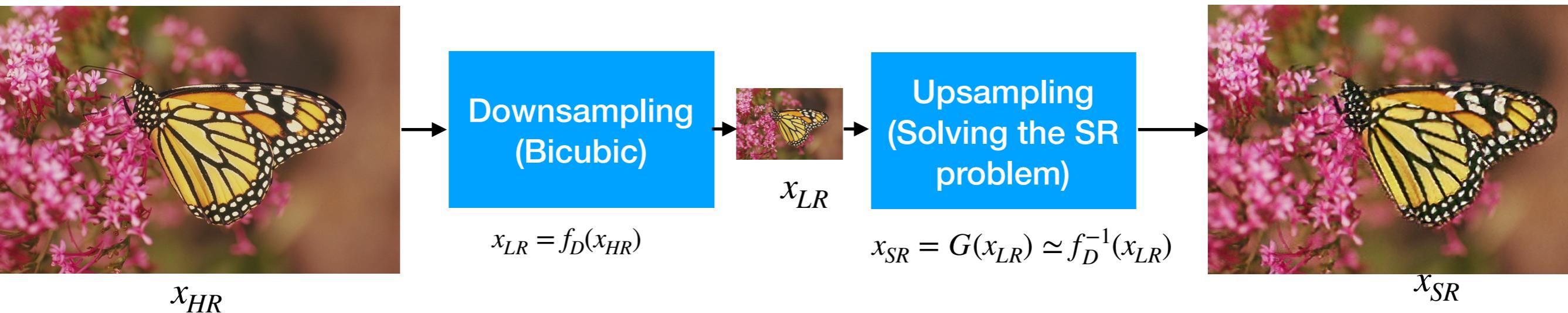
## Goal 1

# Objective: SISR

## Single Image Super Resolution

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

## Typical approach



## Ideal scenario

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

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?**

## 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 (G), 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](https://lanpartis.github.io))

## Goal 1

# GAN Architecture

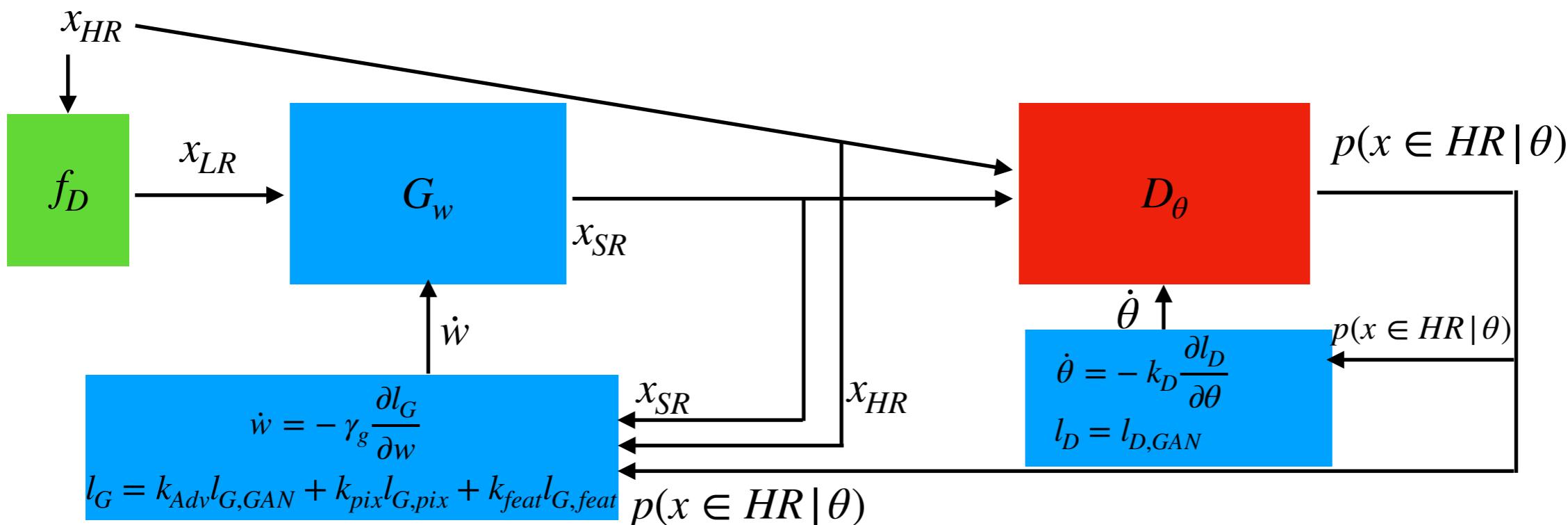
### ESRGAN

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

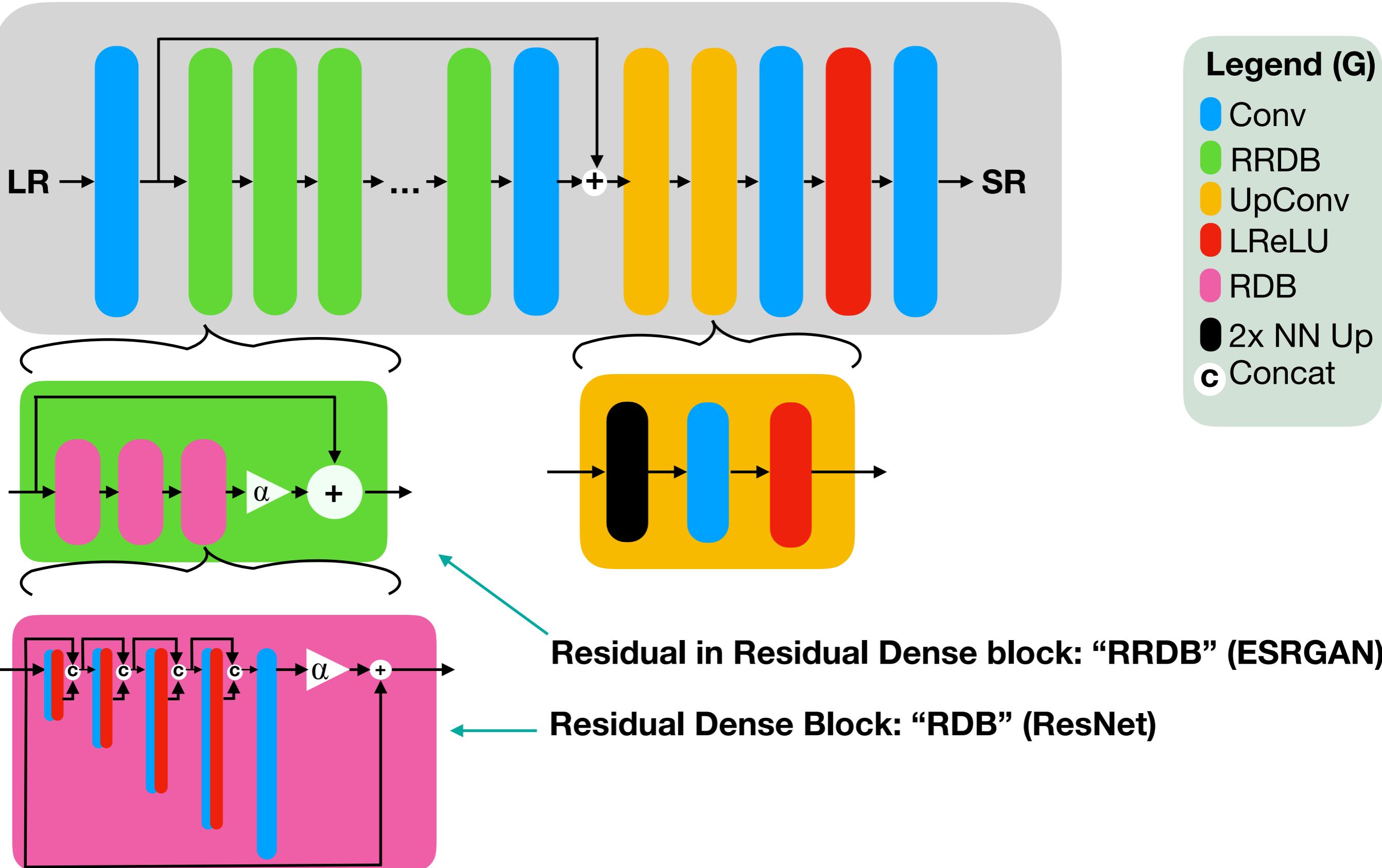
*"It was, in fact, pretty difficult"* - narrator



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

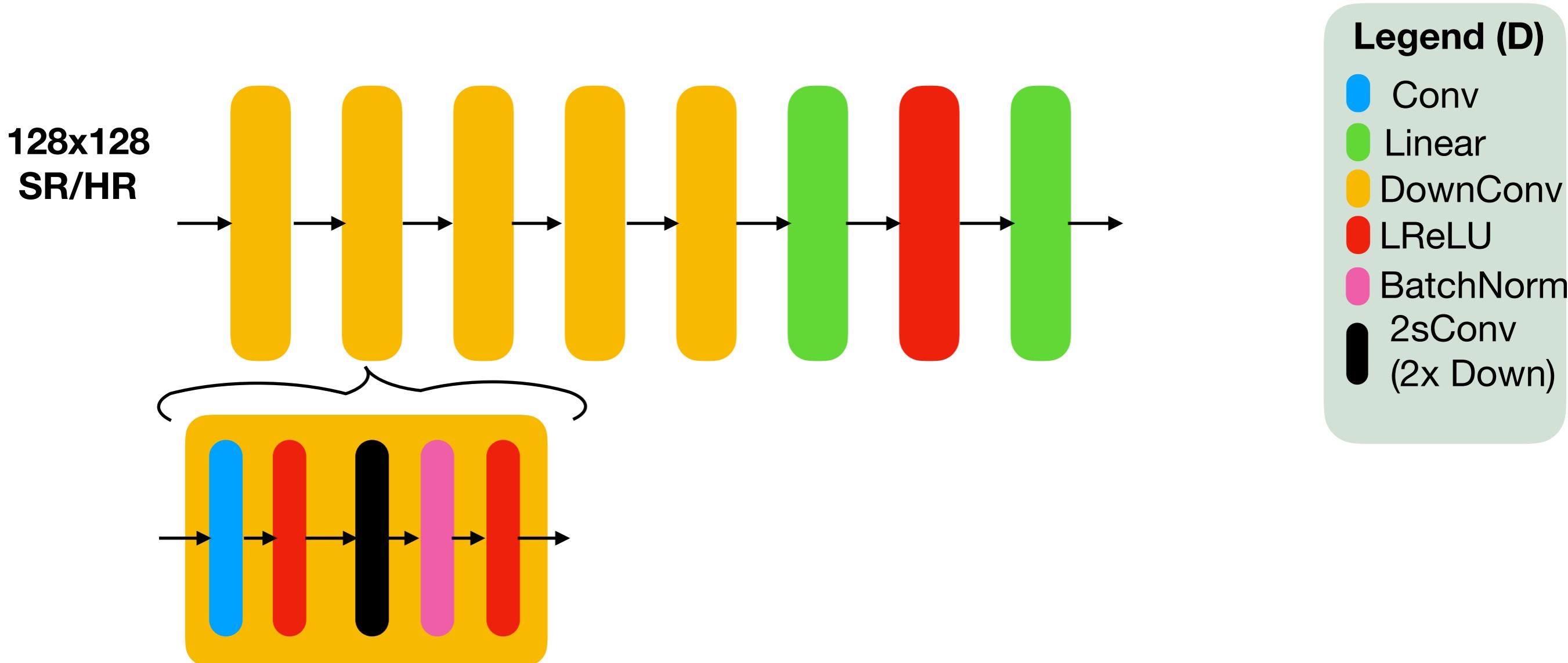
Goal 1

# Generator Architecture



Goal 1

# Discriminator Architecture



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

# 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
    - LR images are made during the run, using cv2.resize with inter\_CUBIC (bicubic interpolation)
  - Data augmentation: Shuffle, flip
  - Total # of training data: ~ 47k
- Validation dataset: Urban100
  - 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*
  - Perhaps due to some subtle differences in the implementations (things I've missed, maybe?)
  - .. or maybe I didn't give the GAN enough time to train.
    - ESRGAN's training time: 24h, starting with a G pretrained for high PSNR
    - I trained for ~12-14h max, on a non-pretrained G
      - And my GPU is weaker
  - I spent a lot of time finding parameters that worked even a bit.
  - Main take away:
    - High complexity -> long feedback time -> difficult to make decisions
      - it's wasn't always realistic to train with a set of parameters multiple times to verify performance, considering that it took hours to see the results.

Goal 2

# Loss fn and hyperparameters

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

$$L_{D,Adv}^{Re} = \frac{\log[\sigma(D(x_{hr}) - D(x_{sr}))] + \log[1 - \sigma(D(x_{sr}) - D(x_{hr}))]}{2}$$
$$L_{G,Adv}^{Re} = \frac{\log[1 - \sigma(D(x_{hr}) - D(x_{sr}))] + \log[\sigma(D(x_{sr}) - D(x_{hr}))]}{2}$$
$$L_{G,Pix} = ||x_{sr} - x_{hr}||_1$$
$$L_{G,Feat} = ||F(x_{sr}) - F(x_{hr})||_1$$

**Adversarial loss**

**Pixel and feature loss**

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

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

# Loss Fn and hyperparameters

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

Parameter	Value	Comment
<b>Scale</b>	4	SR upscale factor
<b>Base # of Features (G)</b>	128	In ESRGAN: 64
<b>RRDB blocks (G)</b>	12	In ESRGAN: 23
<b>Residual scale <math>\alpha</math> (G)</b>	0.2	Same as ESRGAN
<b>Base # of features (D)</b>	96	In ESRGAN: 64
<b>Learning rate (G and D)</b>	$10^{-4}$	Same as ESRGAN
<b>Optimizer</b>	Adam	Same as ESRGAN
<b>LR Scheduling</b>	$\gamma \leftarrow 0.5 \cdot \gamma$ at iter [10k, 20k, 30k, 40k]	Sooner than ESRGAN
<b># training iterations</b>	100k	~1/5 of ESRGAN
<b>Pixel loss weight (G)</b>	$10^{-2}$	Same as ESRGAN
<b>Feature loss weight (G)</b>	$10^{-1}$	In ESRGAN: 1.0
<b>GAN loss weight (G)</b>	$5 \cdot 10^{-3}$	Same as ESRGAN
<b>Batch size</b>	8	In ESRGAN: 16
<b>GAN type</b>	Relativistic	In ESRGAN: Relativistic Avg

## 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  $\sigma = 1$ , decreases linearly, to  $\sigma = 0$ , at 60k it, then train until 100k 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  $\sigma = 0.01$  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 (4x)

HR

Goal 2

# Resulting performance

NN Upsampled LR (4x)



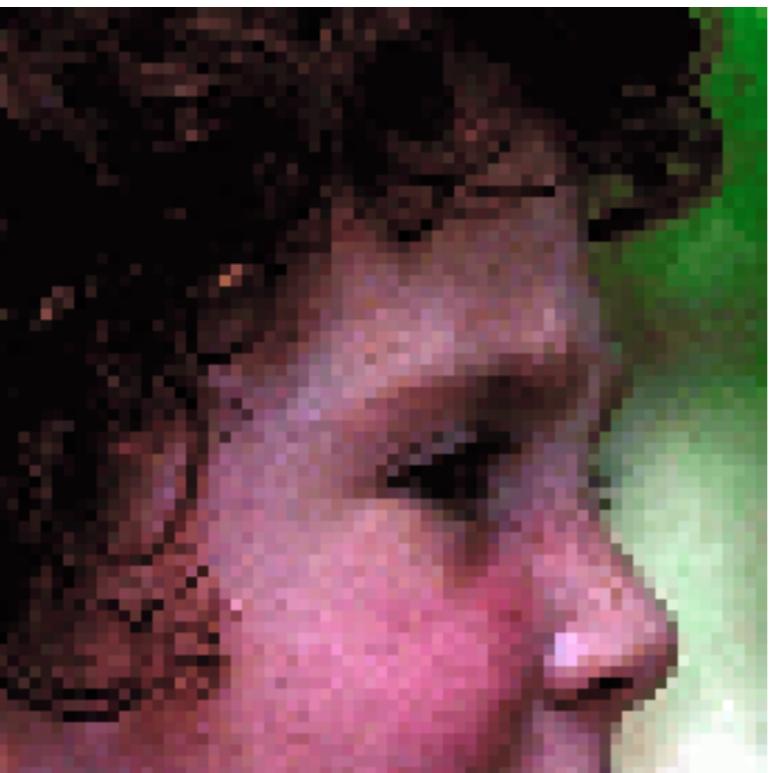
SR (4x)

HR

Goal 2

# Resulting performance

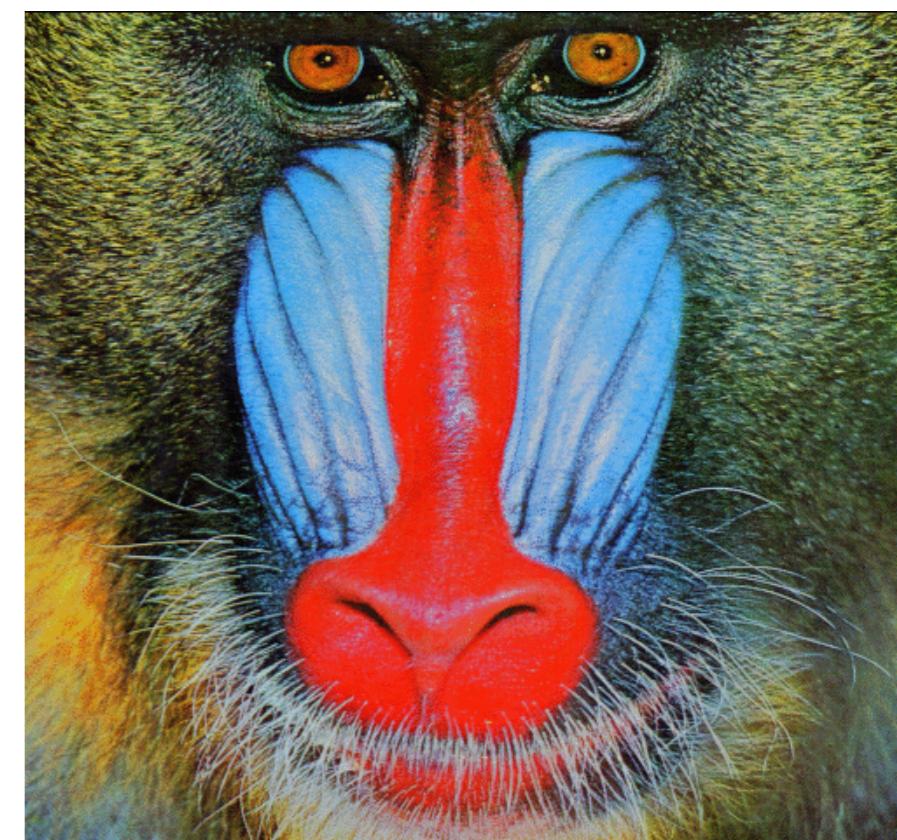
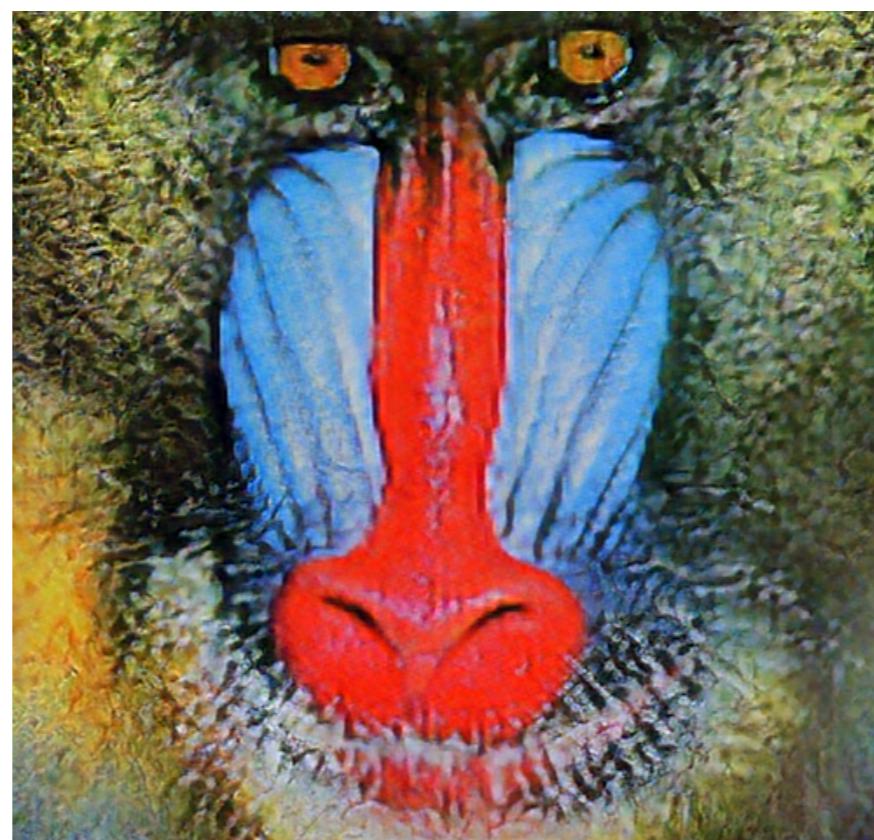
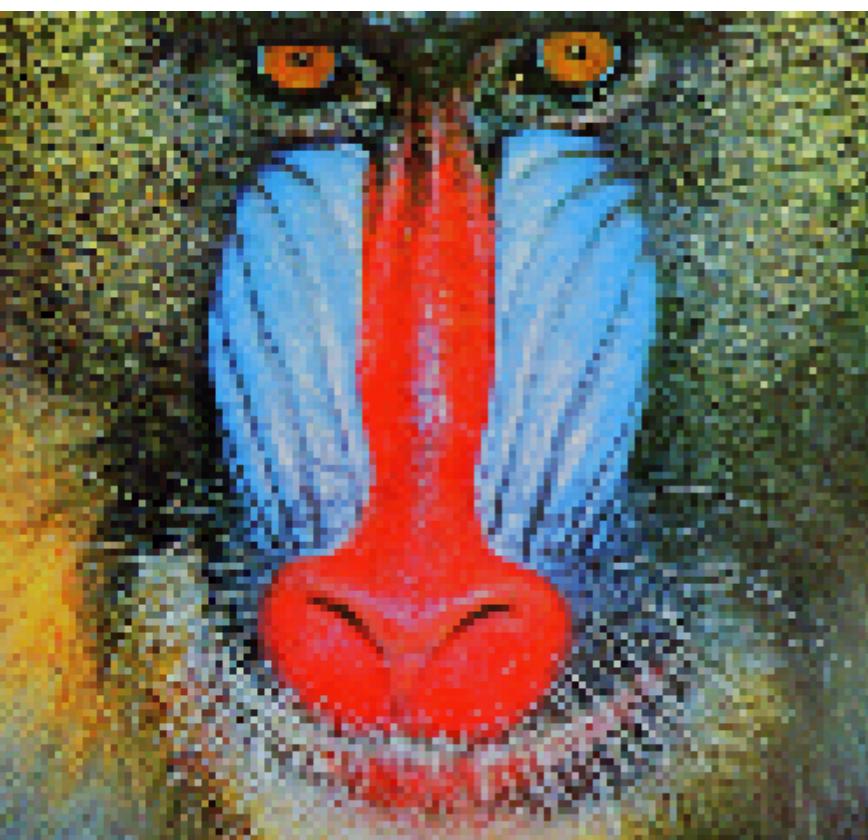
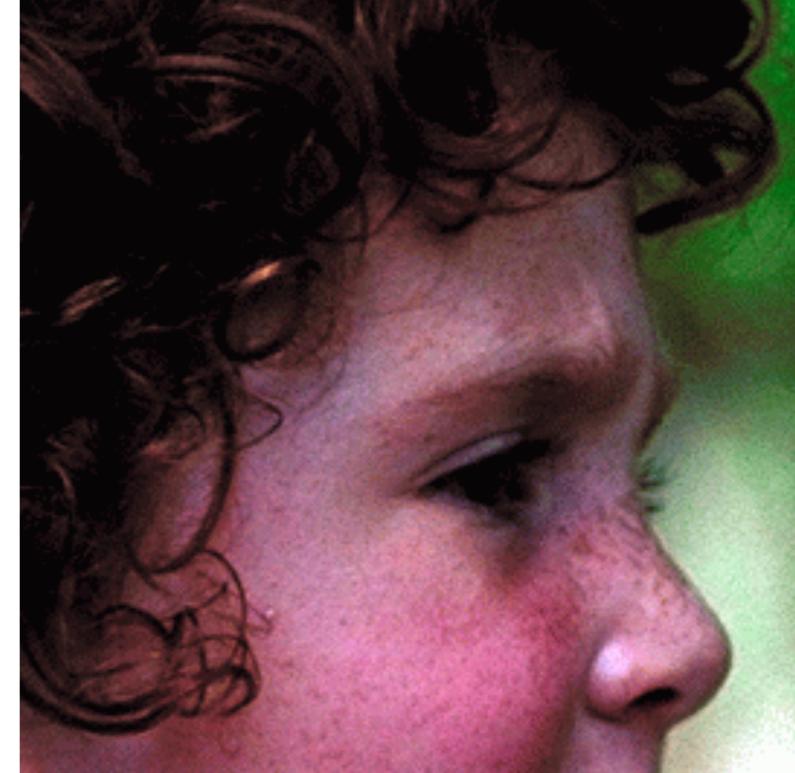
NN Upsampled LR (4x)



SR (4x)



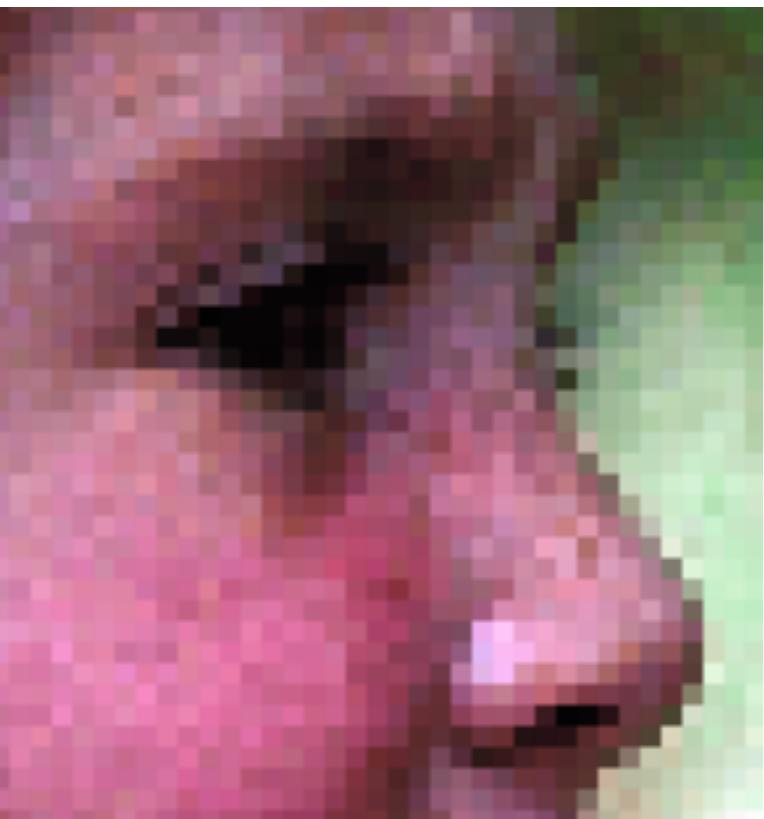
HR



Goal 2

# Resulting performance

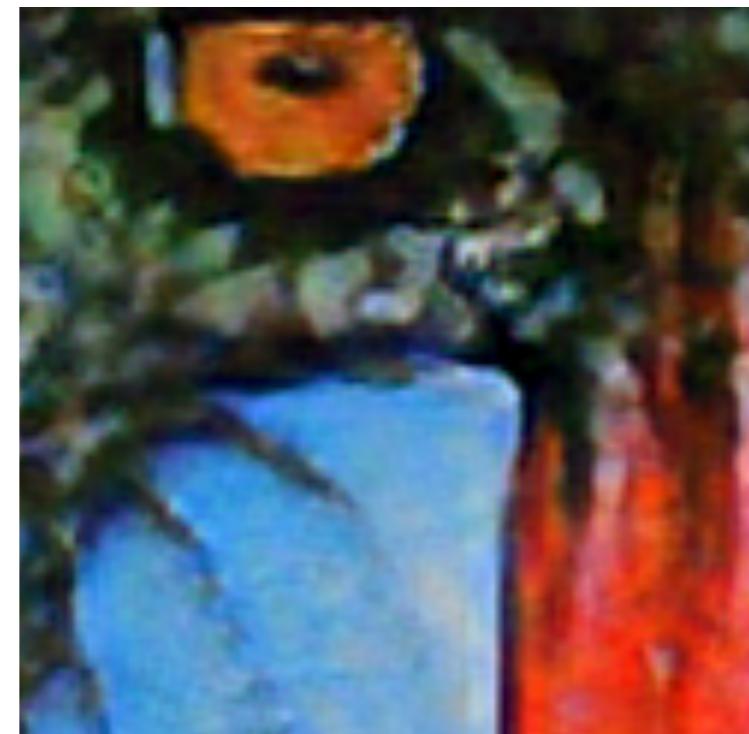
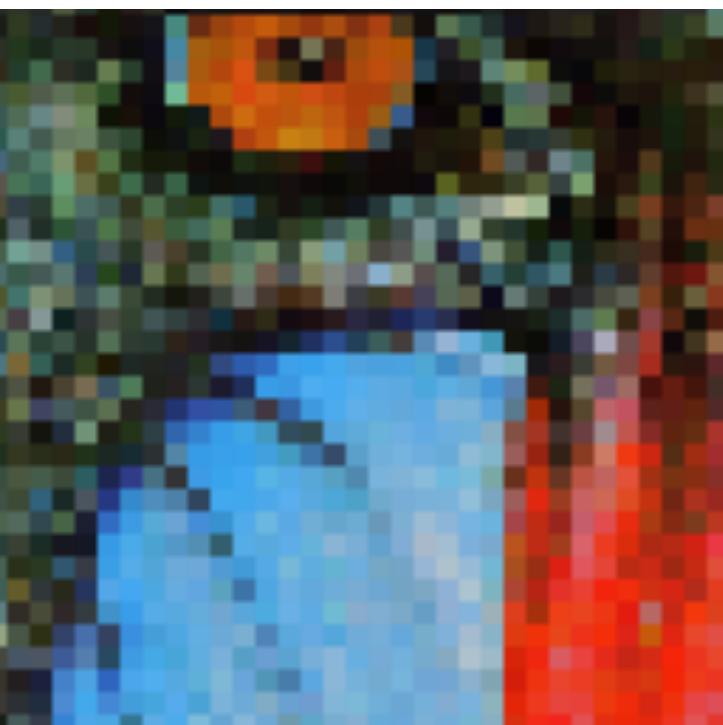
NN Upsampled LR (4x)



SR (4x)



HR



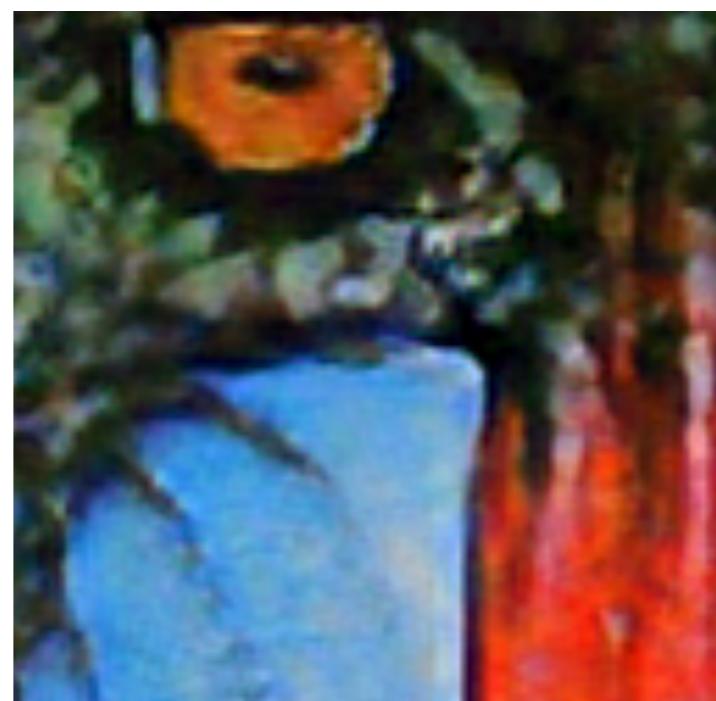
# Resulting performance

- 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)



# Resulting performance

- Here are some images 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 quite 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

Patterns change



HR



## Goal 2

# Resulting performance on my own image

- This is what happened when I tried running a ~2000x1800 image through G, using my laptop (no Nvidia GPU). Lots of caching, and the program was killed by the OS!

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	fsevents_sd	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
61759	Mail	22.6	00:10:26	10/2	6/1	6669	126M	0B	104M

- So I tried with a 400x700 crop of the image instead.
- 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.

Goal 2

# Resulting performance on my own image

NN Upsampled (4x)



Goal 2

# Resulting performance on my own image

Bicubic upsampled (4x)



Goal 2

# Resulting performance on my own image

SR (4x)



# Evaluation / metrics

- I have not yet finished this.
- What I will be working on before the final presentation:
  - Guaranteed: Metrics
    - PSNR
      - Need to crop the HR images so their height/widths are perfectly divisible by the scale!
      - Why PSNR might not be the greatest measure
      - Other alternatives (MOS for instance)
    - Possibly: Training the GAN more
      - Training the current model more
      - Possibly trying a deeper network
      - Possibly check performance difference when using kernel of size 1 in the local feature fusion layer of the residual dense blocks (that's what's specified in the paper where RDBs are presented - whereas ESRGAN uses kernel of size 3, which gives a far higher model complexity: ~16mill params vs 9mill params)
        - Did not try this yet because I prioritised achieving a working model before making smaller changes. :)
      - The amount of work I'll do is not certain - I've spent a lot of time working on this project already..!

# 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

