Super-resolution with PyTorch

Marie-Hélène Burle
training@westgrid.ca

November 24, 2021

WEST GRID

Definitions

LR: low resolution

HR: high resolution

SR: super-resolution = reconstruction of HR images from LR images

SISR: single-image super-resolution = SR using a single input image

History of super-resolution

Can be broken down into 2 main periods:

- A rather slow history with various interpolation algorithms of increasing complexity before deep neural networks
- An incredibly fast evolution since the advent of deep learning (DL)

SR history Pre-DL

Pixel-wise interpolation prior to DL

Various methods ranging from simple (e.g. nearest-neighbour , bicubic) to complex (e.g. Gaussian process regression , iterative FIR Wiener filter) algorithms

SR history Pre-DL

Nearest-neighbour interpolation

Simplest method of interpolation
Simply uses the value of the nearest pixel

Bicubic interpolation

Consists of determining the 16 coefficients a_{ij} in:

$$p(x,y) = \sum_{i=0}^{3} \sum_{i=0}^{3} a_{ij} x^i y^j$$

SR history with DL

Deep learning has seen a fast evolution marked by the successive emergence of various frameworks and architectures over the past 10 years

Some key network architectures and frameworks:

- CNN
- GAN
- Transformers

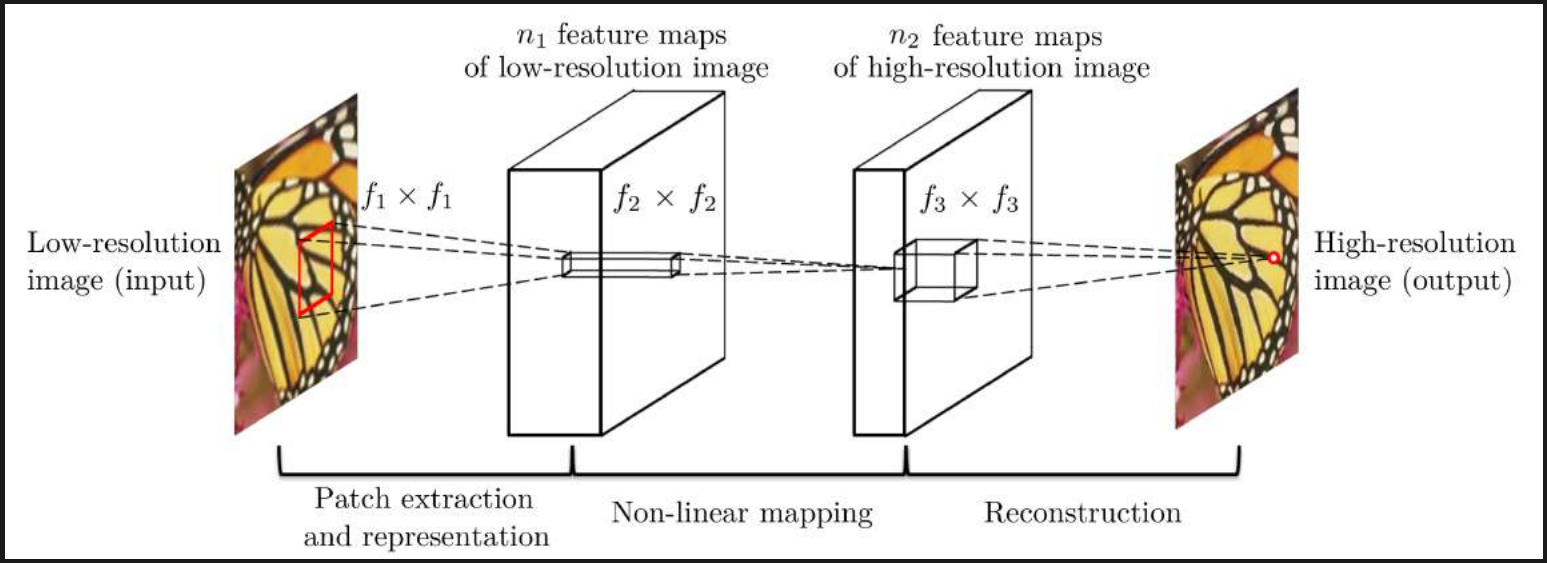
These have all been applied to SR

SR history with DL

SR using (amongst others):

Convolutional Neural Networks (SRCNN) — 2014
Random Forests — 2015
Perceptual loss — 2016
Sub-pixel CNN — 2016
ResNet (SRResNet) & Generative Adversarial Network (SRGAN) — 2017
Enhanced SRGAN (ESRGAN) — 2018
Predictive Filter Flow (PFF) — 2018
Densely Residual Laplacian attention Network (DRLN) — 2019
Second-order Attention Network (SAN) — 2019
Learned downscaling with Content Adaptive Resampler (CAR) — 2019
Holistic Attention Network (HAN) — 2020
Swin Transformer — 2021

SRCNN

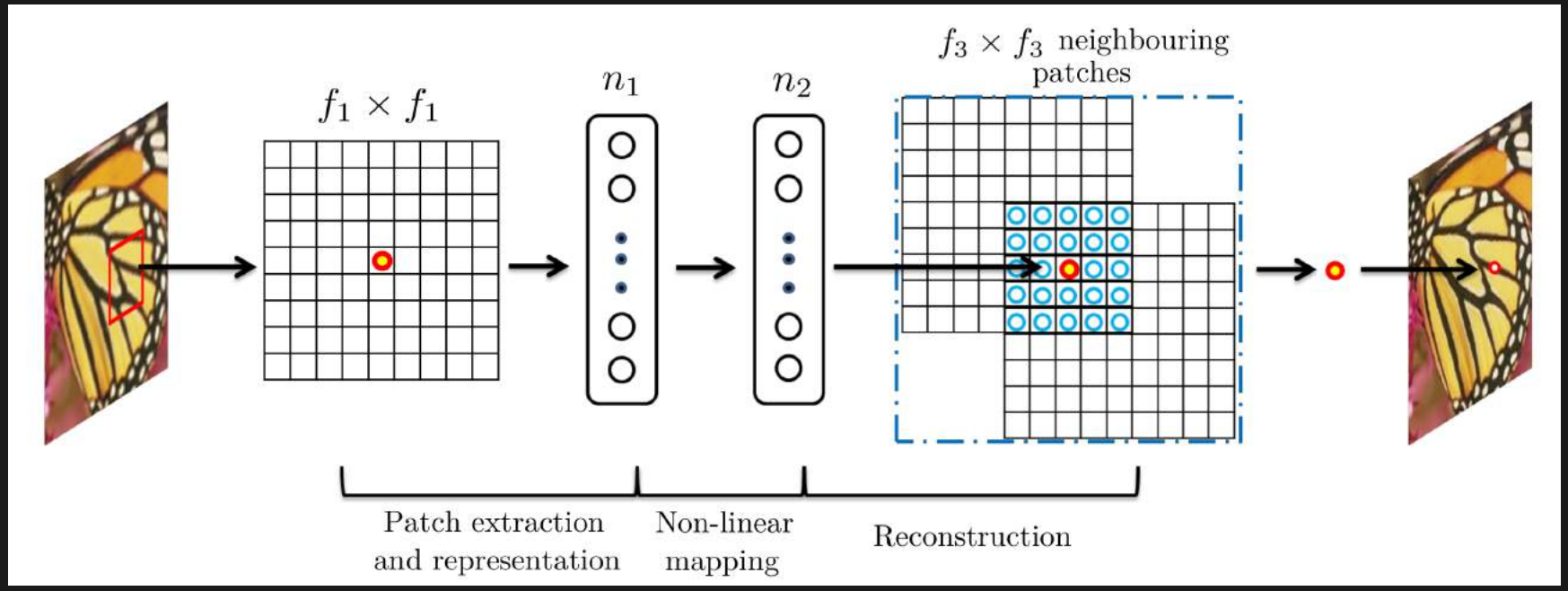


Dong, C., Loy, C. C., He, K., & Tang, X. (2015). Image super-resolution using deep convolutional networks. IEEE transactions on pattern analysis and machine intelligence, 38(2), 295-307

> *Given a low-resolution image Y, the first convolutional layer of the SRCNN extracts a set of feature maps. The second layer maps these feature maps nonlinearly to high-resolution patch representations. The last layer combines the predictions within a spatial neighbourhood to produce the final high-resolution image F(Y)*

SRCNN

Can use sparse-coding-based methods

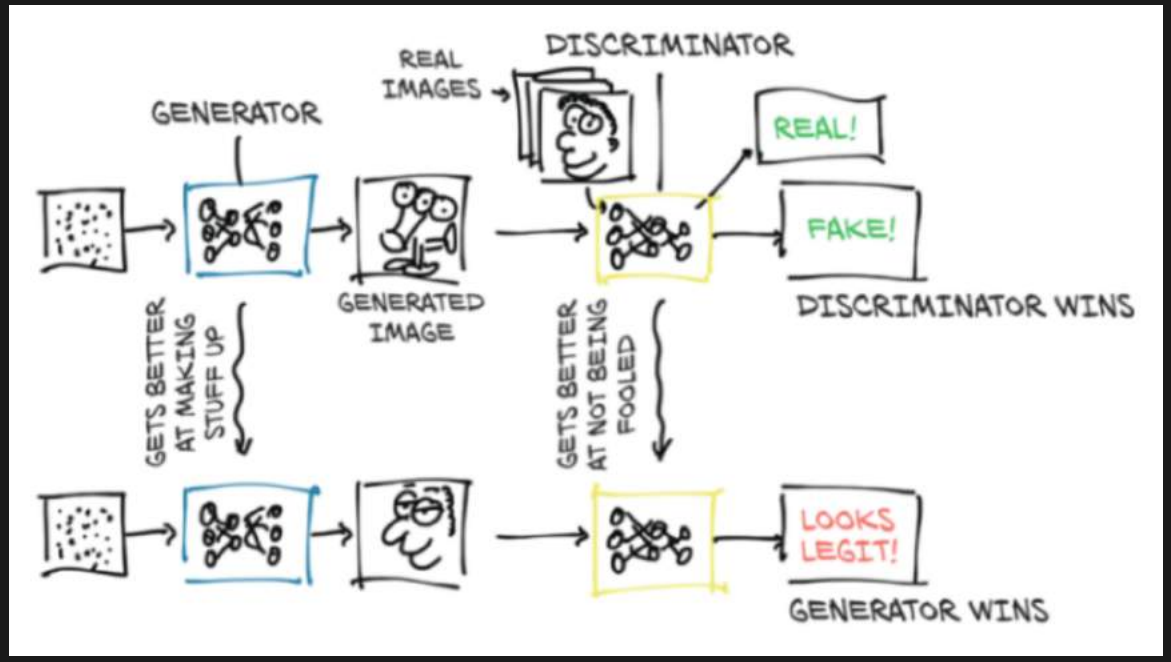


Dong, C., Loy, C. C., He, K., & Tang, X. (2015). Image super-resolution using deep convolutional networks. IEEE transactions on pattern analysis and machine intelligence, 38(2), 295-307

SRGAN

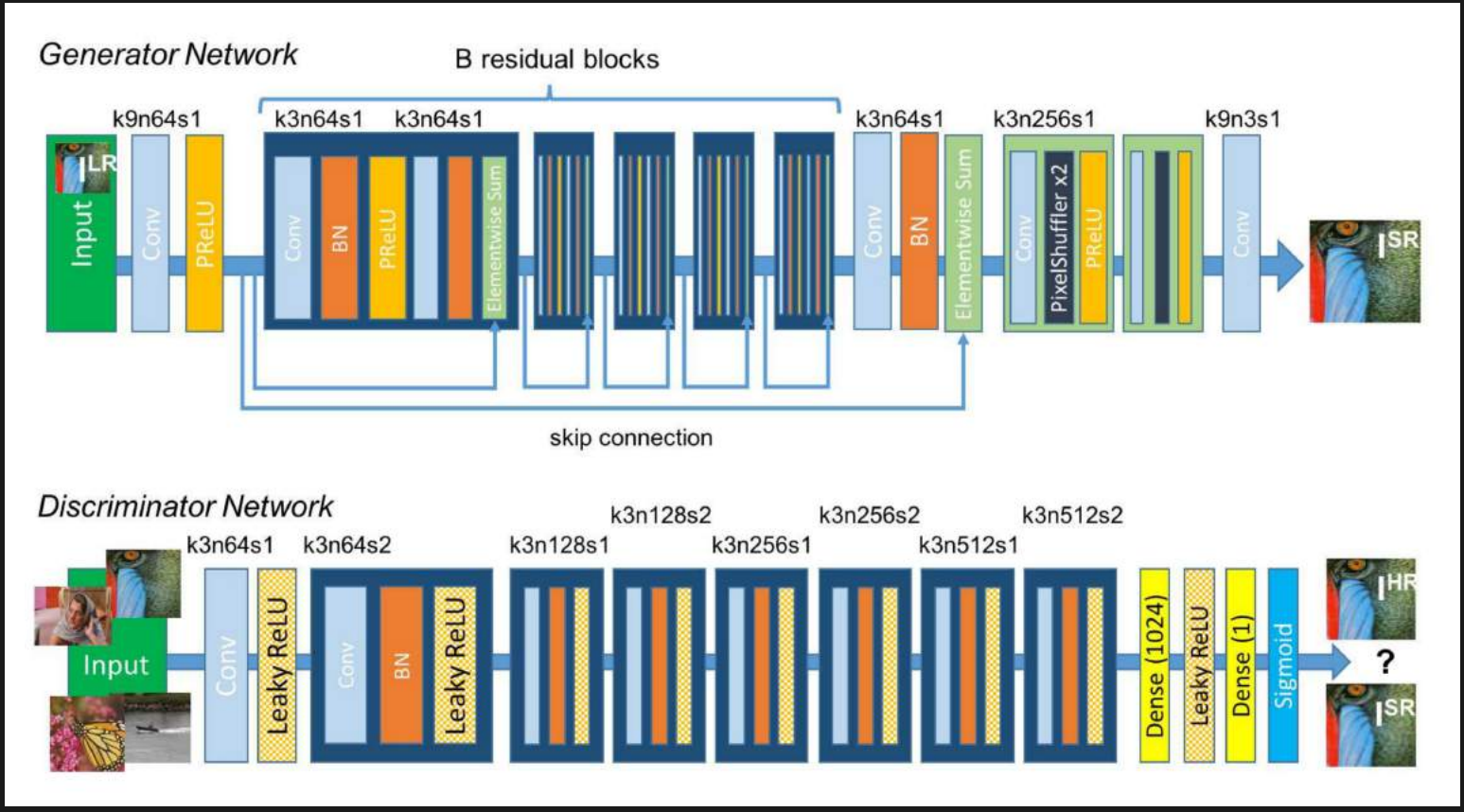
Do not provide the best PSNR, but can give more realistic results by providing more texture (less smoothing)

GAN



Stevens E., Antiga L., & Viehmann T. (2020). Deep Learning with PyTorch

SRGAN



Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., ... & Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4681-4690)

SRGAN

Followed by the ESRGAN and many other flavours of SRGANs

SwinIR

Attention

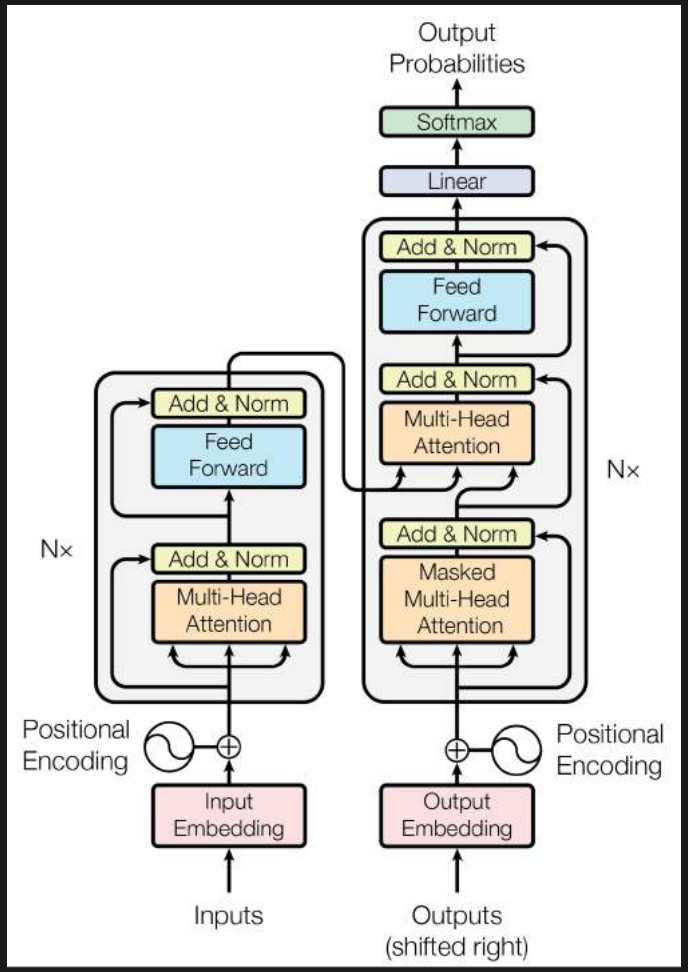
> Mnih, V., Heess, N., & Graves, A. (2014). Recurrent models of visual attention. In Advances in neural information processing systems (pp. 2204-2212)

(cited 2769 times)

> Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008)

(cited 30999 times...)

Transformers



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008)

Transformers

Initially used for NLP to replace RNN as they allow parallelization

Now entering the domain of vision and others

Very performant with relatively few parameters

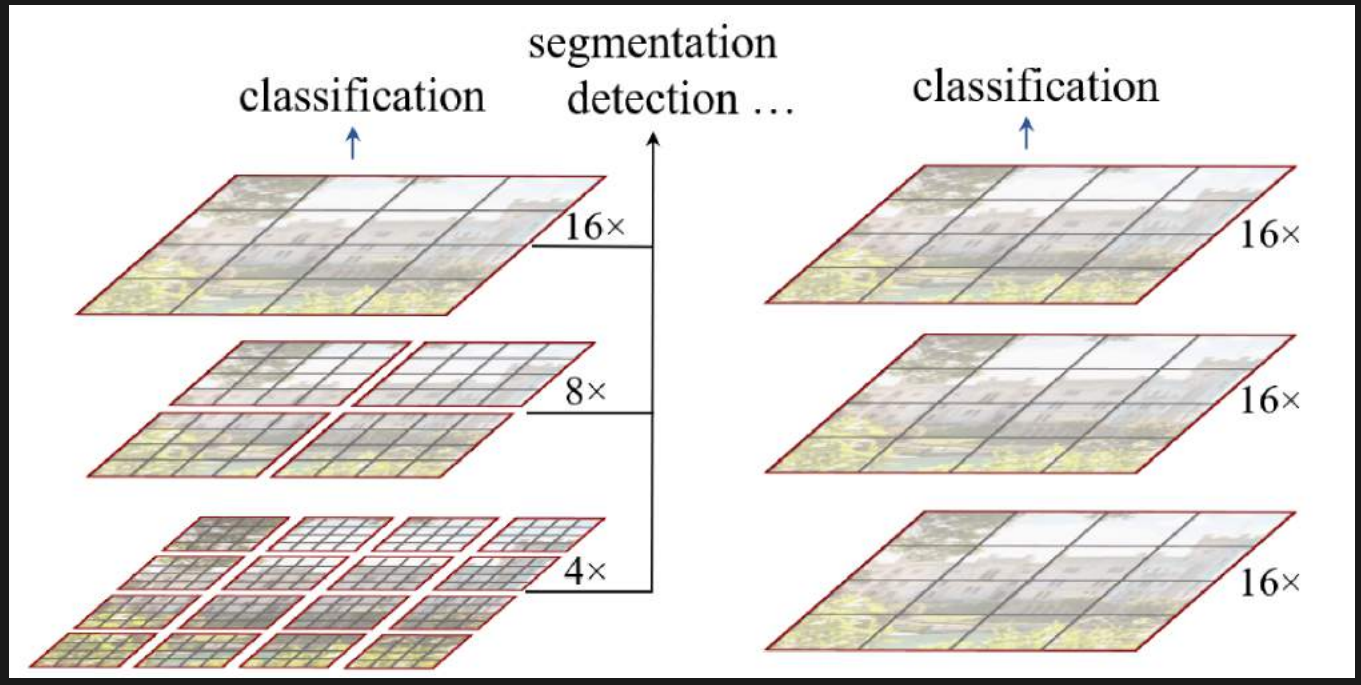
Swin Transformer

The Swin Transformer improved the use of transformers to the vision domain

Swin = Shifted WINdows

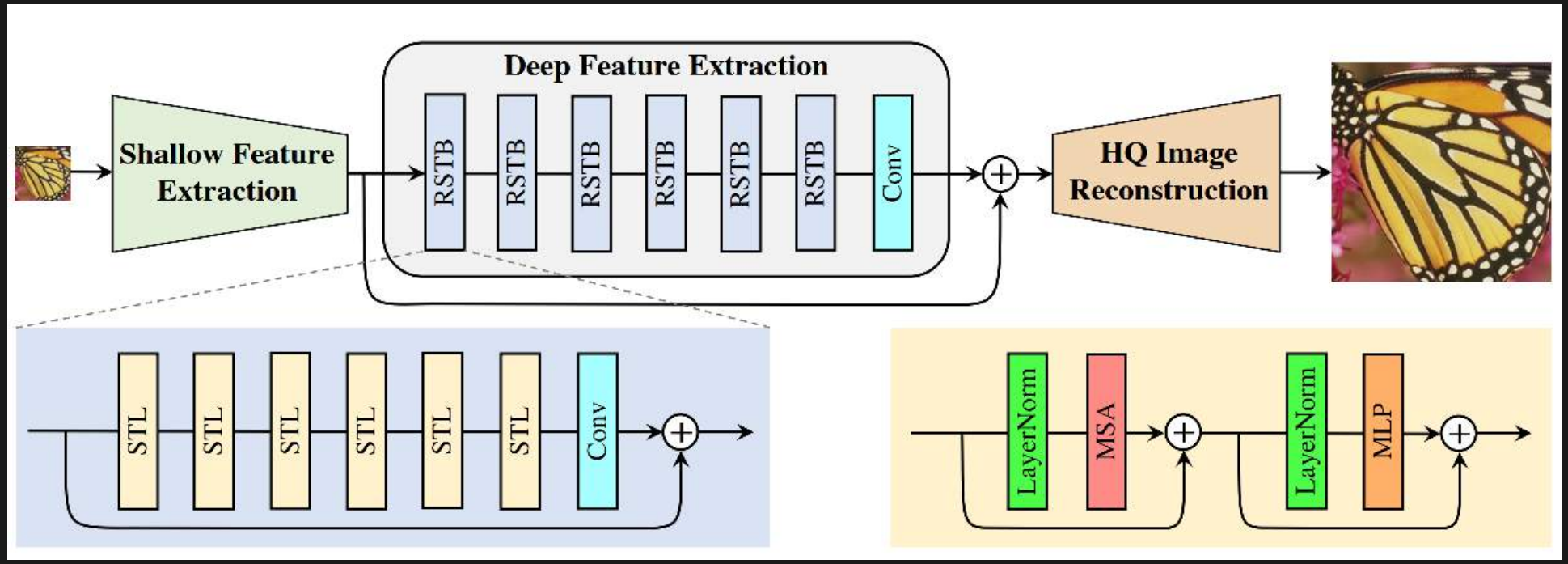
Swin Transformer

Swin transformer (left) vs transformer as initially applied to vision (right):



Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. arXiv preprint arXiv:2103.14030

SwinIR



Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., & Timofte, R. (2021). SwinIR: Image restoration using swin transformer. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1833-1844)

Training sets used

DIV2K , Flickr2K, and other datasets

Models assessment

3 metrics commonly used:

Peak sign-to-noise ratio (PSNR) measured in dB

$$\frac{\text{Maximum possible power of signal}}{\text{Power of noise (calculated as the mean squared error)}}$$

Calculated at the pixel level

Structural similarity index measure (SSIM)

Prediction of perceived image quality based on a “perfect” reference image

Mean opinion score (MOS)

Mean of subjective quality ratings

Models assessment

Peak sign-to-noise ratio (PSNR) measured in dB

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

Structural similarity index measure (SSIM)

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1) + (2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Mean opinion score (MOS)

$$MOS = \frac{\sum_{n=1}^{N} R_n}{N}$$

Metrics implementation

- Implement them yourself (using `torch.log10` , etc.)
- Use some library that implements them (e.g. kornia)
- Use code of open source project with good implementation (e.g. SwinIR)
- Use some higher level library that provides them (e.g. ignite)

Metrics implementation

- Implement them yourself (using `torch.log10` , etc.)
- Use some library that implements them (e.g. kornia)
- Use code of open source project with good implementation (e.g. SwinIR)
- Use some higher level library that provides them (e.g. with ignite)

Metrics implementation

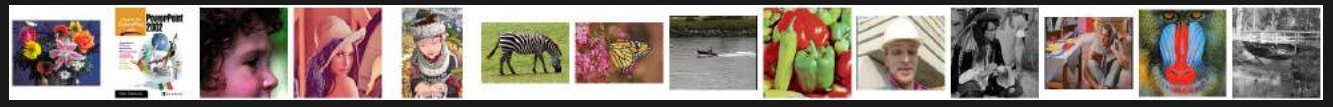
```
import kornia

psnr_value = kornia.metrics.psnr(input, target, max_val)
ssim_value = kornia.metrics.ssim(img1, img2, window_size, max_val=1.0,
```

See the Kornia documentation for more info on kornia.metrics.psnr & kornia.metrics.ssim

Benchmark datasets

Set5

Set14

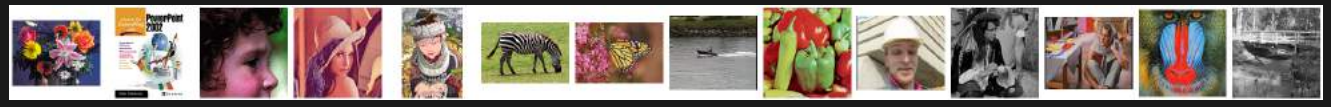


BSD100 (Berkeley Segmentation Dataset)

Benchmark datasets

Set5

Set14



BSD100 (Berkeley Segmentation Dataset)

The Set5 dataset

A dataset consisting of 5 images which has been used for at least 18 years to assess SR methods

How to get the dataset?

From the HuggingFace Datasets Hub with the HuggingFace datasets package:

```
from datasets import load_dataset

set5 = load_dataset('eugenesiow/Set5', 'bicubic_x4', split='validation
```

Dataset exploration

```
print(set5)
len(set5)
set5[0]
set5.shape
set5.column_names
set5.features
set5.set_format('torch', columns=['hr', 'lr'])
set5.format
```

Benchmarks

A 2012 review of interpolation methods for SR gives the metrics for a series of interpolation methods (using other datasets)

TABLE I
PSNR (dB) of the different image interpolation algorithms

Images	Bicubic [4]	LMMSE [15]	NEDI [19]	SAI [24]	BSAI [17]	RSAI [25]
Kodim01	25.208	25.069	24.827	25.223	25.339	25.422
Kodim02	33.069	33.113	33.042	33.327	33.482	33.533
Kodim03	33.913	34.173	34.157	34.711	34.663	34.839
Kodim04	34.019	33.868	33.754	34.276	34.421	34.419
Kodim05	26.06	26.16	26.045	27.109	26.98	27.062
Kodim06	26.593	26.588	26.597	26.788	26.778	26.966
Kodim07	33.347	33.417	33.017	34.377	34.28	34.326
Kodim08	22.403	22.45	22.074	22.51	22.517	22.692
Kodim09	31.652	31.676	31.417	32.225	32.171	32.177
Kodim10	31.696	31.72	32.208	32.768	32.415	32.863
Kodim11	28.218	28.183	28.194	28.465	28.547	28.617
Kodim12	32.499	32.654	32.371	32.868	33.057	33.01
Kodim13	22.931	22.824	22.822	23.03	22.975	23.17
Kodim14	28.596	28.584	28.445	29.023	28.982	29.073
Kodim15	32.654	32.719	32.751	33.116	33.207	33.293
Kodim16	30.073	30.106	30.055	30.166	30.286	30.355
Kodim17	31.59	31.681	31.716	32.181	32.108	32.289
Kodim18	27.311	27.2	26.976	27.491	27.416	27.528
Kodim19	27.001	27.024	26.088	26.727	27.287	27.364
Kodim20	30.834	31.25	31.264	31.729	31.697	31.809
Kodim21	27.594	27.483	27.306	27.756	27.729	27.851
Kodim22	29.74	29.604	29.564	29.92	29.908	30.047
Kodim23	35.041	34.892	35.283	36.197	35.91	36.236
Kodim24	26.06	25.918	25.831	26.133	26.06	26.277
Average	29.504	29.514	29.408	29.921	29.925	**30.051**

TABLE II
SSIM [79] of the different image interpolation algorithms

Images	Bicubic [4]	LMMSE [15]	NEDI [19]	SAI [24]	BSAI [17]	RSAI [25]
Kodim01	0.7659	0.7509	0.7393	0.7647	0.7673	0.769
Kodim02	0.8842	0.8746	0.8802	0.8867	0.889	0.8907
Kodim03	0.9223	0.9173	0.923	0.9279	0.9279	0.93
Kodim04	0.9035	0.8894	0.8926	0.904	0.9052	0.9051
Kodim05	0.8616	0.8549	0.8475	0.8782	0.8758	0.8754
Kodim06	0.8041	0.7965	0.799	0.8085	0.8087	0.8139
Kodim07	0.949	0.9423	0.9447	0.9567	0.9559	0.9563
Kodim08	0.7771	0.7751	0.7572	0.7804	0.7838	0.7841
Kodim09	0.9045	0.8979	0.9021	0.9093	0.9137	0.9119
Kodim10	0.9046	0.8982	0.9065	0.9139	0.9145	0.9159
Kodim11	0.8344	0.8261	0.8276	0.8379	0.8408	0.8422
Kodim12	0.8856	0.8787	0.8812	0.8876	0.8911	0.8911
Kodim13	0.729	0.7126	0.7073	0.7273	0.7265	0.7305
Kodim14	0.8495	0.8386	0.8354	0.8537	0.8536	0.8547
Kodim15	0.9088	0.9028	0.9078	0.9117	0.9137	0.9147
Kodim16	0.8426	0.8343	0.8394	0.8441	0.8456	0.8482
Kodim17	0.9162	0.9096	0.9155	0.9249	0.9245	0.9259
Kodim18	0.8502	0.837	0.8321	0.8522	0.8522	0.8523
Kodim19	0.8454	0.8351	0.8378	0.8465	0.8509	0.8518
Kodim20	0.9121	0.904	0.9118	0.9175	0.9198	0.9196
Kodim21	0.8685	0.8569	0.8595	0.8707	0.8729	0.873
Kodim22	0.8598	0.8432	0.8453	0.8596	0.8598	0.8612
Kodim23	0.9507	0.9407	0.9528	0.9556	0.9554	0.9569
Kodim24	0.8528	0.8419	0.8384	0.8565	0.855	0.8572
Average	0.8659	0.8566	0.8577	0.8698	0.8710	**0.8722**

TABLE IV
PSNR (dB) of the estimated images using different algorithms

Images	IBP [59]	AWF [65]	GPR [37]	ASDS [43]	IWF [47]
Bike	22.808	22.876	22.312	23.6	**23.784**
House	20.353	20.328	20.019	20.48	**20.682**
Game	25.546	25.574	25.076	**25.975**	25.966
Statue	28.614	28.678	28.151	29.269	**29.27**
Woman	25.717	25.765	25.223	26.083	**26.238**
Lighthouse	23.76	23.717	23.442	23.885	**23.948**
Parrot	29.439	29.639	28.69	30.387	**30.467**
Lena	31.334	31.5	30.774	32.696	**32.721**
Average	25.946	26.01	25.461	26.547	**26.635**

TABLE V
SSIM [79] of the estimated images using different algorithms

Images	IBP [59]	AWF [65]	GPR [37]	ASDS [43]	IWF [47]
Bike	0.7161	0.7199	0.6686	0.7582	**0.7597**
House	0.6446	0.6409	0.6096	0.6572	**0.6668**
Game	0.7402	0.7421	0.7039	**0.7569**	0.7548
Statue	0.8379	0.8407	0.817	0.8551	**0.857**
Woman	0.7629	0.7669	0.7172	0.7762	**0.7765**
Lighthouse	0.7559	0.753	0.732	0.7665	**0.7671**
Parrot	0.9084	0.9089	0.8963	0.917	**0.9182**
Lena	0.8792	0.8807	0.8636	**0.8923**	0.892
Average	0.7807	0.7816	0.751	0.7974	**0.799**

Interpolation methods

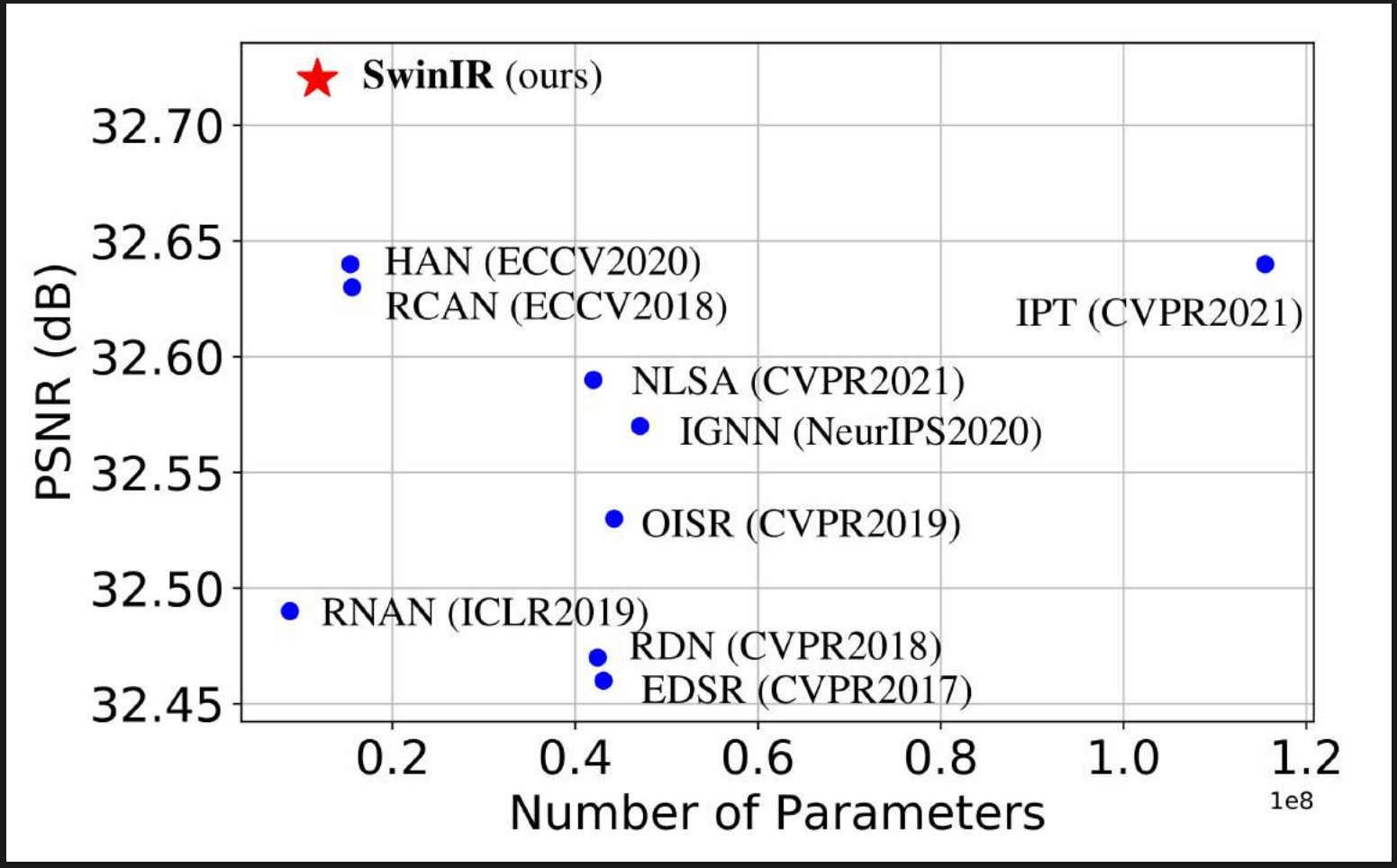
Average	29.504	29.514	29.408	29.921	29.925	**30.051**
Average	25.946	26.01	25.461	26.547		**26.635**
Average	0.8659	0.8566	0.8577	0.8698	0.8710	**0.8722**
Average	0.7807	0.7816	0.751	0.7974		**0.799**

DL methods

The Papers with Code website lists available benchmarks on Set5

Rank	Model	PSNR ↑	SSIM	MOS	LPIPS	Perceptual Index	Paper	Code	Result	Year
1	SwinIR	32.93	0.9043				SwinIR: Image Restoration Using Swin Transformer			2021
2	HAN+	32.75	0.9016				Single Image Super-Resolution via a Holistic Attention Network			2020
3	PFF	32.74	0.9021				Image Reconstruction with Predictive Filter Flow			2018
4	DRLN+	32.74	0.9013				Densely Residual Laplacian Super-Resolution			2019
5	SRGAN + Residual-in-Residual Dense Block	32.73	0.9011				ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks			2018
6	RFN	32.71					Progressive Perception-Oriented Network for Single Image Super-Resolution			2019
7	SAN	32.70	0.9013				Second-Order Attention Network for Single Image Super-Resolution			2019
8	ABPN	32.69	0.9				Image Super-Resolution via Attention based Back Projection Networks			2019
9	CSNLN	32.68	0.9004				Image Super-Resolution with Cross-Scale Non-Local Attention and Exhaustive Self-Exemplars Mining			2020
10	DBPN-RES-MR64-3	32.65	0.899				Deep Back-Projection Networks for Single Image Super-resolution			2019
11	RCAN	32.63	0.9002				Image Super-Resolution Using Very Deep Residual Channel Attention Networks			2018
12	HBPN	32.55	0.9				Hierarchical Back Projection Network for Image Super-Resolution			2019

PSNR vs number of parameters for different methods on Set5x4



Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., & Timofte, R. (2021). SwinIR: Image restoration using swin transformer. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1833-1844)

Comparison between SwinIR & a representative CNN-based model (RCAN) on classical SR images x4

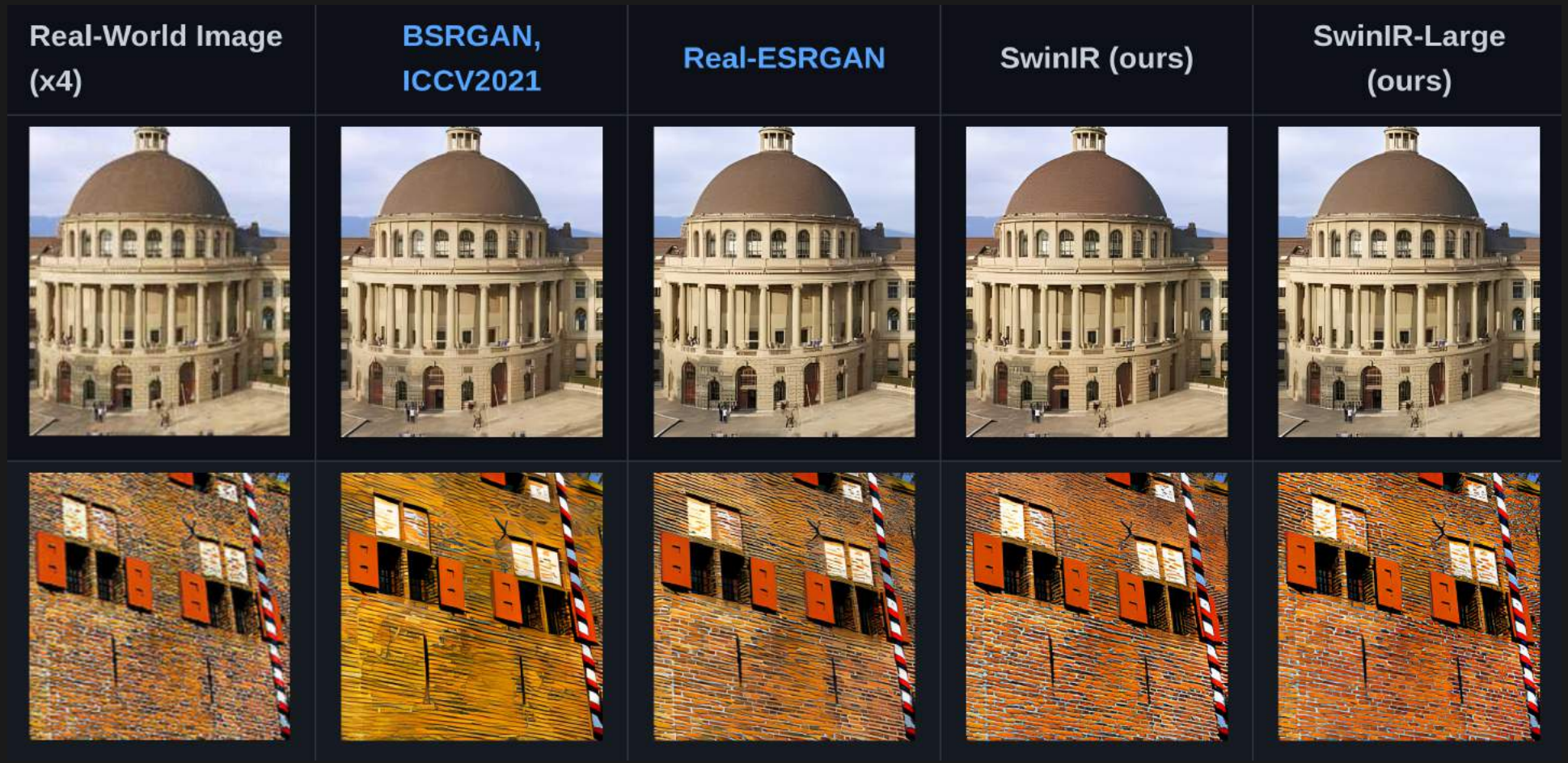
Method	Training Set	Training time (8GeForceRTX2080Ti batch=32, iter=500k)	Y-PSNR/Y-SSIM on Manga109	Run time (1GeForceRTX2080Ti, on 256x256 LR image)*	#Params
RCAN	DIV2K	1.6 days	31.22/0.9173	0.180s	15.6M
SwinIR	DIV2K	1.8 days	31.67/0.9226	0.539s	11.9M

Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., & Timofte, R. (2021). SwinIR: Image restoration using swin transformer. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1833-1844)

Comparison between SwinIR & a representative CNN-based model (RCAN) on classical SR images x4

#FLOPs	Testing memory
850.6G	593.1M
788.6G	986.8M

Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., & Timofte, R. (2021). SwinIR: Image restoration using swin transformer. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1833-1844)



Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., & Timofte, R. (2021). SwinIR: Image restoration using swin transformer. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1833-1844)

Let's use SwinIR

```
# Get the model
git clone git@github.com:JingyunLiang/SwinIR.git
cd SwinIR

# Copy our test images in the repo
cp -r <some/path>/my_tests /testsets/my_tests

# Run the model on our images
python main_test_swinir.py --tile 400 --task real_sr --scale 4 --large
```

Ran in 9 min on my machine with one GPU and 32GB of RAM

Results

Results

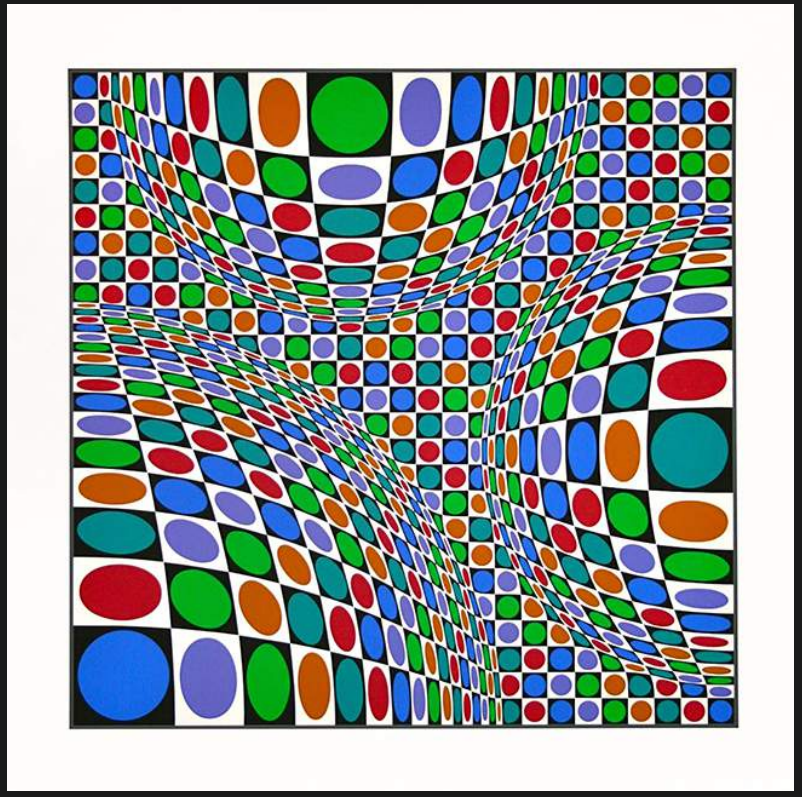
Results

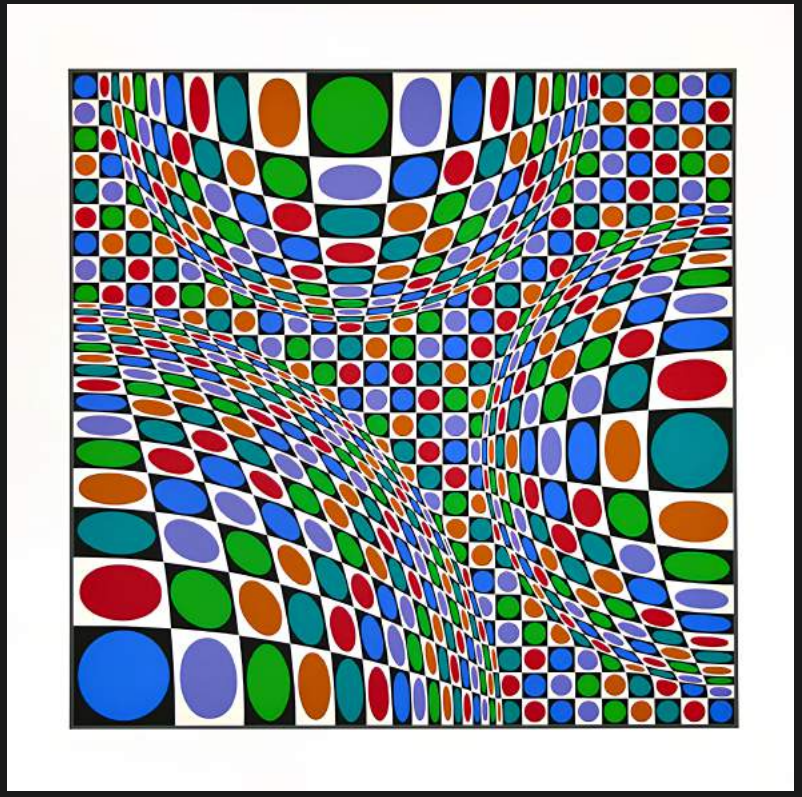
Results

Results

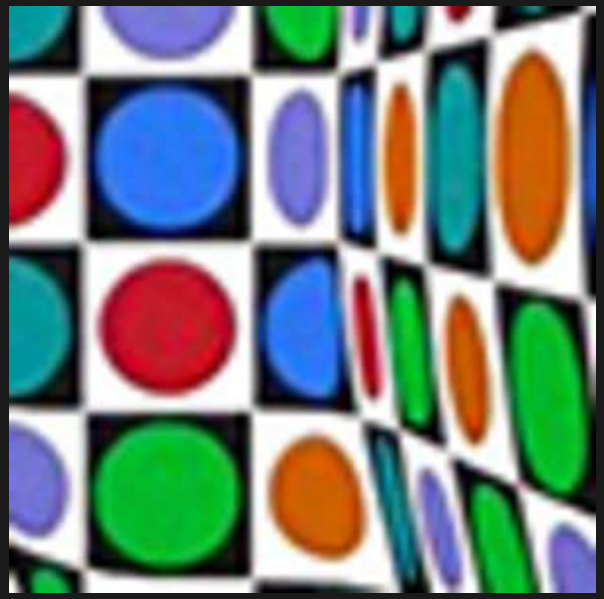
Results

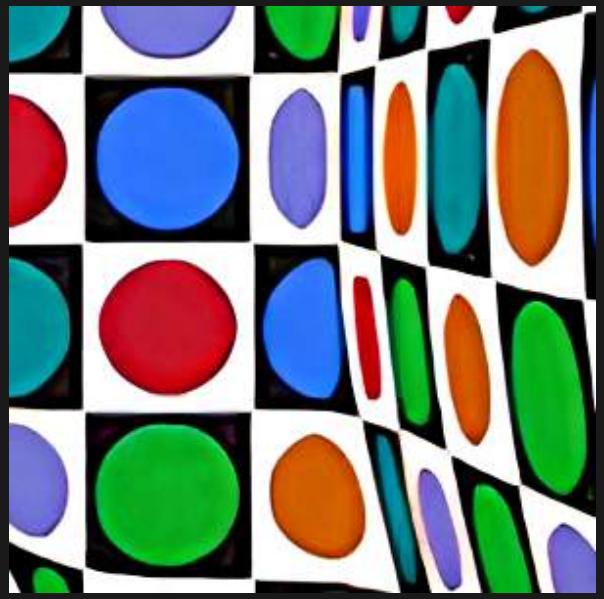
Results





Results





Results

Results

Metrics

We could use the PSNR and SSIM implementations from SwinIR , but let's try the Kornia functions we mentioned earlier:

- kornia.metrics.psnr
- kornia.metrics.ssim

Metrics

Let's load the libraries we need:

```
import kornia
from PIL import Image
import torch
from torchvision import transforms
```

Metrics

Then, we load one pair images (LR and HR):

```
berlin1_lr = Image.open("<some/path>/lr/berlin_1945_1.jpg")
berlin1_hr = Image.open("<some/path>/hr/berlin_1945_1.png")
```

We can display these images with:

```
berlin1_lr.show()
berlin1_hr.show()
```

Metrics

Now, we need to resize them so that they have identical dimensions and turn them into tensors:

```
preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.ToTensor()
        ])

berlin1_lr_t = preprocess(berlin1_lr)
berlin1_hr_t = preprocess(berlin1_hr)
```

Metrics

[In]

```
berlin1_lr_t.shape
berlin1_hr_t.shape
```

[Out]

```
torch.Size([3, 267, 256])
torch.Size([3, 267, 256])
```

We now have tensors with 3 dimensions:

- the channels (RGB)
- the height of the image (in pixels)
- the width of the image (in pixels)

Metrics

As data processing is done in batch in ML, we need to add a 4th dimension: the **batch size**

(It will be equal to `1` since we have a batch size of a single image)

```
batch_berlin1_lr_t = torch.unsqueeze(berlin1_lr_t, 0)
batch_berlin1_hr_t = torch.unsqueeze(berlin1_hr_t, 0)
```

Metrics

Our new tensors are now ready:

[In]

```
batch_berlin1_lr_t.shape
batch_berlin1_hr_t.shape
```

[Out]

```
torch.Size([1, 3, 267, 256])
torch.Size([1, 3, 267, 256])
```

PSNR

[In]

```
psnr_value = kornia.metrics.psnr(batch_berlin1_lr_t, batch_berlin1_hr_
psnr_value.item()
```

[Out]

```
33.379642486572266
```

SSIM

[In]

```
ssim_map = kornia.metrics.ssim(batch_berlin1_lr_t, batch_berlin1_hr_t,
ssim_map.mean().item()
```

[Out]

```
0.9868119359016418
```


Questions?