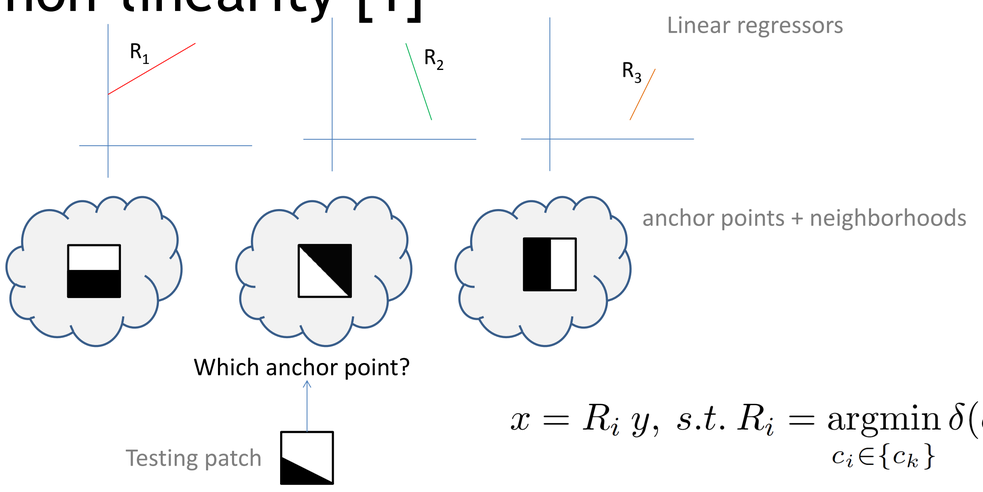
Half Hypersphere Confinement for Piecewise Linear Regression

Eduardo Pérez-Pellitero[1, 2], *Jordi Salvador*[1], *Javier Ruiz-Hidalgo*[3], *Bodo Rosenhahn*[2]

[1] Technicolor R&I, Hannover
[2] TNT Lab, Leibniz Universität Hannover
[3] Image Processing Group, Universitat Politècnica de Catalunya

1. Piecewise Linear Regresssion

- Mapping between low and high resolution patch manifolds is non-linear
  - Locally linear assumption
- Ensemble of linear regressors to model non-linearity [1]

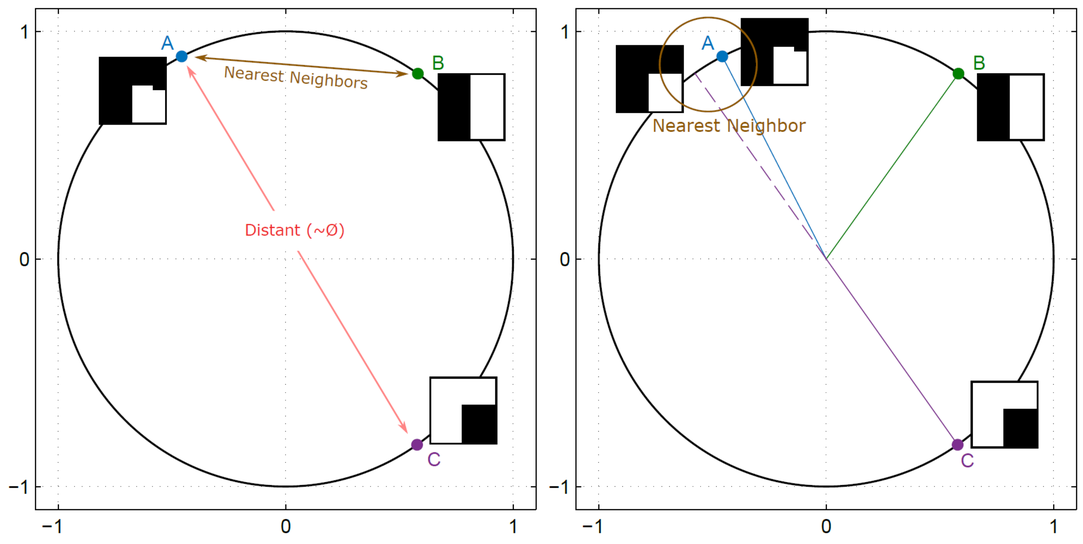


$$x = R_i\, y,\ s.t.\ R_i = \underset{c_i \in \{c_k\}}{\operatorname{argmin}}\, \delta(c_i,\, y)$$

2. Motivation

- What is the best metric for patch-based linear regression?
- Can this optimal metric be computed fast?

3. Antipodality



- Antipodal points: Two points diametrically opposite in the sphere.
- Antipodally invariant metric:

$$\delta(a,\, b) = \delta(-a,\, b) = \delta(a,\, -b) = \delta(-a,\, -b)$$

- Angular similarity:

$$\varsigma(c,\, y) = |\hat{c} \cdot \hat{y}| = |\cos\theta|$$

a) Finds a better regressor fit during the nearest neighbor search within the anchor points
b) Regressors training is improved thanks to tighter neighborhoods

References

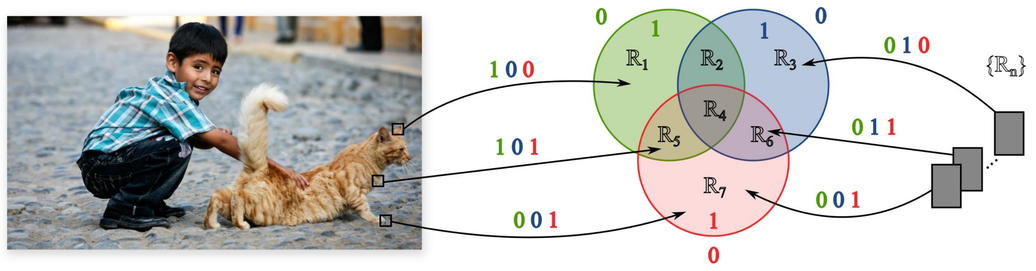
[1] Timofte, R., Smet, V.D., Goool, L.V.: Anchored neighborhood regression for fast example-based super-resolution. In: ICCV (2013)
[2] Heo, J.P., Lee, Y., He, J., Chang, S.F., Yoon, S.E.: Spherical hashing. In: CVPR. (2012)
[3] E. Pérez-Pellitero, J. Salvador, I. Torres, J. Ruiz-Hidalgo, B. Rosenhahn: Fast super-resolution via dense local training and inverse regressor search. In: ACCV (2014)
[4] C. Dong, C. Loy, K. He, and X. Tang: Learning a deep convolutional network for image super-resolution. In: ECCV (2014)
[5] S. Schulter, C. Leistner, and H. Bischof: Fast and accurate image upscaling with super-resolution forests. In: CVPR (2015)

4. Fast Search

- Builds on top of a non-hierarchical optimized dictionary of anchor points
- Spherical hashing (SpH) [2,3]:

$$h_k(y_F) = \begin{cases} 0 & when\ d(p_k, y_F) > t_k \\ 1 & when\ d(p_k, y_F) \leq t_k \end{cases}$$

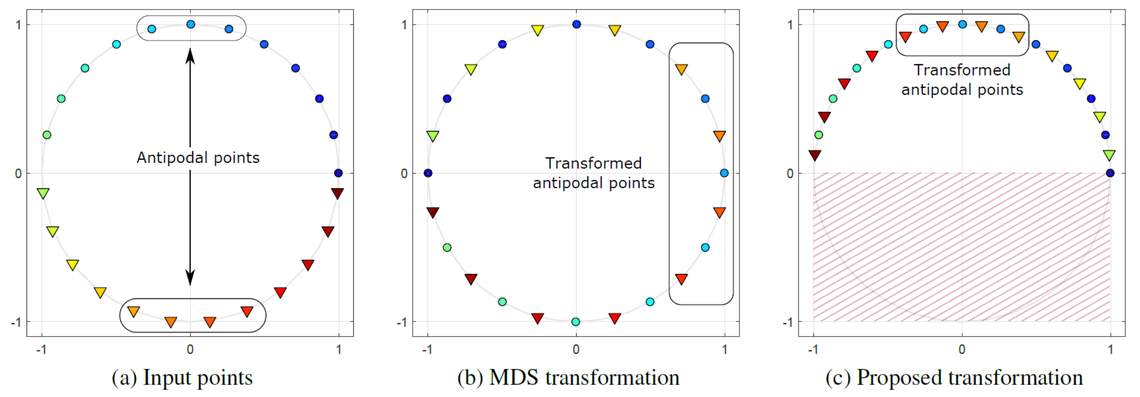
- Partitions obtained with natural image patches, leaves labelled with anchor points



5. Embedding in Euclidean Space

- MDS is computationally expensive and slow

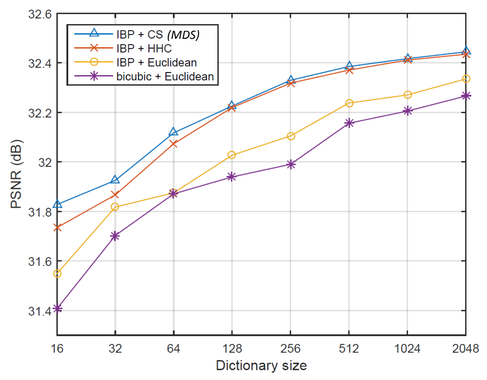
We propose the Half Hypersphere Confinement:



(a) Input points (b) MDS transformation (c) Proposed transformation

Simple yet effective

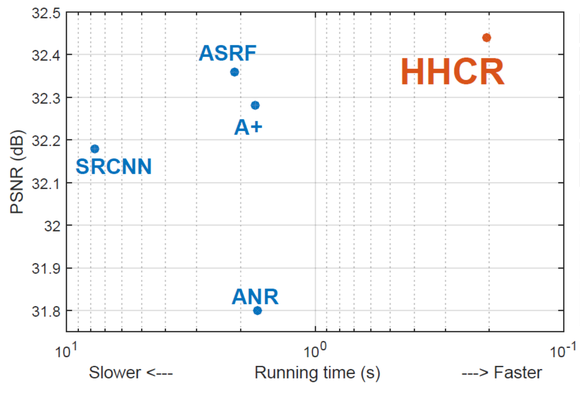
$$y_{TR} = -y, \qquad \text{if } y \cdot \mathbf{e_q} < 0$$



6. Results

- Tested for denoising and Super-Resolution:

Fastest (~x10) and best-performer (up to +0,3dB) within the benchmark !



	MF	Bicubic PSNR	Bicubic Time	ANR [20] PSNR	ANR [20] Time	SRCNN[6] PSNR	SRCNN[6] Time	ASRF [18] PSNR	ASRF [18] Time	A+ [21] PSNR	A+ [21] Time	HHCR, $s=1$ PSNR	HHCR, $s=1$ Time	HHCR, $s=7$ PSNR	HHCR, $s=7$ Time
Set5	2	33.66	0.002	35.83	0.712	36.34	3.953	36.69	1.264	36.55	0.761	**36.87**	1.167	36.80	**0.105**
	3	30.39	0.002	31.92	0.449	32.39	3.916	32.57	1.050	32.59	0.467	**32.79**	0.583	32.75	**0.080**
	4	28.42	0.002	29.69	0.348	30.09	4.031	30.20	1.059	30.29	0.346	**30.46**	0.382	30.45	**0.075**
Set14	2	30.23	0.002	31.80	1.717	32.18	7.695	32.36	2.114	32.28	1.739	**32.48**	2.223	32.44	**0.205**
	3	27.54	0.002	28.65	0.933	29.00	7.646	29.12	1.670	29.13	0.963	**29.26**	1.075	29.23	**0.153**
	4	26.00	0.002	26.85	0.696	27.20	7.944	27.31	1.371	27.32	0.714	**27.45**	0.716	27.42	**0.155**
Kodak	2	30.85	0.003	32.24	2.938	32.63	13.121	32.76	3.360	32.71	3.161	**32.89**	3.943	32.84	**0.339**
	3	28.43	0.003	29.21	1.615	29.43	12.805	29.63	2.555	29.57	1.678	**29.68**	1.771	29.65	**0.246**
	4	27.23	0.003	27.80	1.199	27.94	13.315	28.17	2.204	28.10	1.226	**28.17**	1.186	28.15	**0.245**

	MF	Bicubic PSNR	Bicubic Time	ANR [20] PSNR	ANR [20] Time	SRCNN[6] PSNR	SRCNN[6] Time	ASRF [18] PSNR	ASRF [18] Time	A+ [21] PSNR	A+ [21] Time	HHCR, $s=1$ PSNR	HHCR, $s=1$ Time	HHCR, $s=7$ PSNR	HHCR, $s=7$ Time
Bambo	2	31.60	0.003	33.06	3.373	33.84	15.135	34.56	3.608	34.31	3.640	**34.77**	4.259	34.71	**0.382**
	3	29.63	0.003	30.72	1.768	31.26	14.921	31.69	2.623	31.85	1.866	**32.05**	2.226	32.00	**0.278**
	4	28.40	0.003	29.33	1.292	29.75	15.285	30.13	2.221	30.25	1.331	**30.40**	1.392	30.38	**0.267**
Market	2	35.83	0.003	37.46	3.279	38.61	15.053	39.44	3.294	39.04	3.579	**39.59**	4.387	39.53	**0.351**
	3	33.82	0.003	35.11	1.889	35.92	14.643	36.58	2.540	36.73	1.990	**36.95**	1.998	36.91	**0.274**
	4	32.51	0.003	33.62	1.300	34.16	15.661	34.86	2.124	35.00	1.364	**35.21**	1.283	35.18	**0.270**

Go to http://perezpellitero.github.io/ for extended results and code!