Dictionary learning for real images encoding: benchmark of different solutions

François Devrainne, Sebastian Boie, Piotr Swierczynski

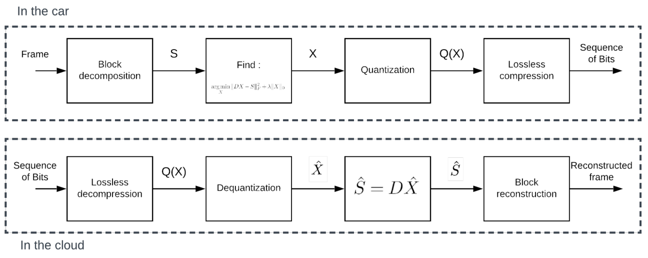


Fig. 1. The car sends frames to the cloud through dictionary learning, those entities have to share the same D

Abstract — **Over the past two decades, there have been various studies on the ability of dictionaries to sum up the information of an image. Transfer images in new basis more suitable to their specificities has been a well-known workaround for compressing images. JPEG widely uses frequency domain space to express images in a space where they are sparse. Because the transformation from the time-space to the frequency-space is bijective, it makes the coding and decoding really straightforward in spite of low computational ressources. DCT also has been choosen among a various range of basis for its good result among a wide-range of different images.**

Jack of all trades and Master of none, one can want to design the dictionary for its own problem. Also the redundant nature of the dictionary that leads to a non-reversible transformation, could be a good clue for a sparser representation.

Dictionary learning | Sparse representation | Image encoding| Image analysis | Image quality assessment | Autonomous driving

Correspondence: ***francois.devrainne@ensae.fr***

Introduction. Orthogonal Matching Pursuit (OMP) is known to offer the best answer to the sparse representation problem). Nevertheless its greediness that implies a very high latency remains its huge drawback that makes it useless for any real life applications.

On the other hand, Alternating Direction Method of Multipliers (ADMM) that attempts to solve a supposed equivalent problem, is much faster to converge, but also returns a much less sparse code.
The first part of this document will be dedicated to a comparison between those two algorithms. The second part will focus on an idea to decrease the latency of OMP using ADMM as a preprocessing step. Finally, different ways of speeding up the solving will be presented and compared.

Global overview and notations. Firstable, and it is here a basic step in any compression algorithm, the image is divided in blocks. Empirically it has been found that 8x8[1] blocks gives the best results. To simplify, height and width are considered multiple of 8, so cropping or handling overlaps are avoided.

Let S be the numerical representation of an image, S is then an array of shape $(height, width, 3)$ where 3 stands for the three channels (RGB, YUV, ...). In the following, S will rather denote the block decomposition of the original image such that :

$$(\frac{height \cdot width}{8 \cdot 8}, 8 \cdot 8 \cdot 3) = (\frac{height \cdot width}{64}, 192)$$
$$= (nb\_blocks, 192)$$

is the new shape of S.
Given a certain signal S, the general form of the problem can be written :

$$\underset{D,X}{\arg\min} \underbrace{\|XD - S\|_F^2}_{\text{Data fidelity}} + \lambda \overbrace{\|X\|_0}^{\text{Sparsity}} \quad \textbf{(1)}$$

where $D \in \mathcal{M}_{\text{basis\_size},192}$ is the dictionary and $X \in \mathcal{M}_{\text{nb\_blocks},\text{basis\_size}}$ is the code of each block of the image.
$\|.\|_F$ denotes the Froebenius norm over $\mathcal{M}_{nb\_blocks,192}$
$\|X\|_0$ denotes the L_0 norm of X ie the number of non-zero value in X.
λ is the sparsity parameter with $\lambda > 0$ because the quantity $\|X\|_0$ is also wanted to be minimized.

L_0 norm is non-convex, and cannot be derivated. It is assumed that solutions to 1 can be approximated by the solutions to :

$$\underset{D,X}{\arg\min} \underbrace{\|XD - S\|_F^2}_{\text{Data fidelity}} + \lambda \overbrace{\|X\|_1}^{\text{Sparsity}} \quad \textbf{(2)}$$

To avoid any scale effects, we also impose that the norm of each atom is equal to 1. The signal is also centralized ; the mean of each block is substracted[2]. This vector of size nb_blocks would have to be sent additionaly to the code X.

[1]It could be interesting to reconsider this argument. Especially, because the mean of each block has to be stored and sent. A bigger size of block could then lead to less means to send

[2]Also it could be relevant to substract the mean quantized, to make the mean addition-substraction a losslessw step

The classic way to solve 2 is alternate between the two following convex sub-problems :

$$\arg\min_{X} \|XD - S\|_F^2 + \lambda \|X\|_1 \tag{2a}$$

$$\arg\min_{D} \|XD - S\|_F^2 + \lambda \|X\|_1 \tag{2b}$$

Updating the value of D and X at each iterations.
Because X is fixed in 2b, the equation is equivalent to :

$$\arg\min_{D} \|XD - S\|_F^2 \tag{2b*}$$

It is then clear that the optimal dictionaries for the sub-problem 2b and its equivalent in terms of L_0 norm, are the same. In the following, D is supposed to be already found, and the problems that will be rather analysed are 2a and its equivalent in terms of L_0 norm :

$$\arg\min_{X} \|XD - S\|_F^2 + \lambda \|X\|_0 \tag{2a0}$$

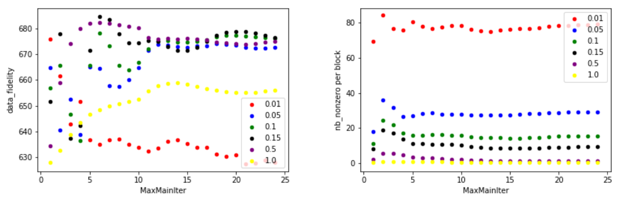
Also basis_size is choosen equal to 500, which, regarding to the 192, clearly makes the basis over-completed.

OMP and ADMM. The basic mechanisms of those two algorithms will not be detailed in here .
The conscientious reader could look at the following articles (1), (2) for further explanations.
First, and it is here the main difference between OMP and ADMM, the first one attempt to solve (2a0) while ADMM (2a). Outputs are then naturally expected to be different, but maybe still share some properties.
OMP has been implemented with a custom package that mostly uses sklearn. ADMM uses the package sporco and has been externally modified to include a reconstruction guarantee.
ADMM and OMP outputs will be analysed to see what they can offer in terms of reconstruction, running time, and sparsity of the code.
For a greater relevance, the sparsity of code has been analysed in terms of number of non zero per block. Of course, it is an average over all blocks, and some needs a much longer linear combination of atom than others.
The quality reconstruction metric between a compressed reconstruction and the original image is the following one :

$$DQ(X,S) = \frac{\|XD - S\|_F^2}{max(S)}$$

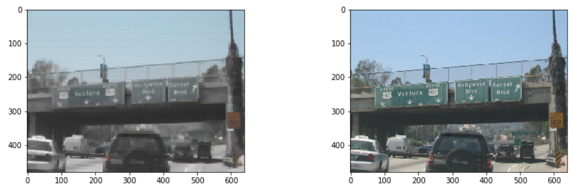
So the smaller DQ, the better the reconstruction.

ADMM. The results obtained for different values of λ can be seen in **Fig.** 2, **Fig.** 3 The 24 iterations last around 7 seconds. A greater λ leads to a better sparsity but also to a worse reconstruction. Also after the fifth iteration, the sparsity seems to be almost constant, and the following iterations are only meant to increase the quality of the reconstruction.



(a) **(b)**

Fig. 2. Evolution of metrics through iterations



(a) $\lambda = 0.5$ **(b)** $\lambda = 0.01$

Fig. 3. ADMM outputs for different λ values

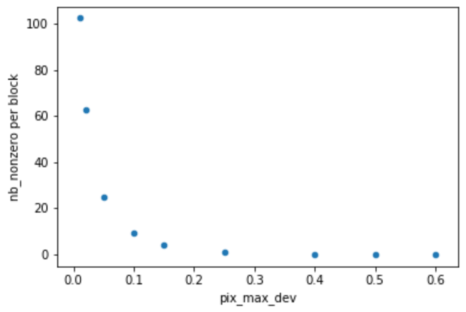


Fig. 4. Sparsity of the code for different reconstruction guarantees

OMP. For OMP[3] it is not relevant to speak about a number of iteration. Indeed OMP solves for each block the following lasso problem :

$$\arg\min_{X\_block \in \mathbb{R}^{500}} \|X\_blockD - S\_block\|_F^2 + \lambda \|X\_block\|_0 \tag{3}$$

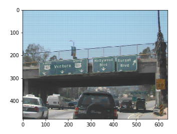
It starts with an X_block full of zero, and at each iteration, it adds a coefficient to the sparse code until a certain reconstruction guarantee is reached. It is the pixel max deviation that has been taken here as the reconstruction guarantee metric.
It takes :

- 320 seconds for pix_max_dev = 0.01
- 24 seconds for pix_max_dev = 0.1
- 3 seconds for pix_max_dev = 0.5

The results for the sparsity w.r.t. the guarantee reconstruction can be seen in **Fig.** 5

[3]In a nutshell : OMP solves block-wise sub-problems in a greedy way, it does not need to access the gradient of the loss function. Unlike ADMM, that alternates between three gradient descents to globally solve the problem.

(a) pix_max_dev = 0.01


(b) pix_max_dev = 0.1

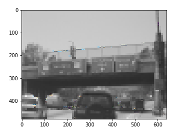
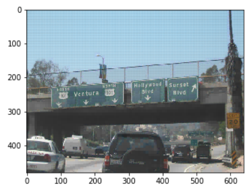

(c) pix_max_dev = 0.5

Fig. 5. OMP outputs for different value of pix_max_dev


(a) OMP output pix_max_dev = 0.1

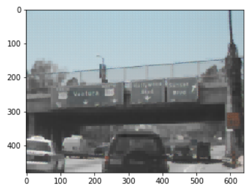

(b) ADMM output : $\lambda = 0.1$, 4 iterations

Fig. 6. OMP ad ADMM outputs

Similarities between OMP and ADMM outputs. The two outputs that are going to be analysed give the two following reconstructions. The first one is computed in 1.2 seconds whereas the OMP is done in 24 seconds.

By looking at those two reconstructions **Fig.** 6, it is hard to believe that similarities between sparsecodes do exist.

Firstable, the code of OMP has 44716 non zero values[4]. It is roughly 9 atoms per 8x8 blocks. the code from ADMM is much sparser (23398 : around 5), and of course with a worse quality reconstruction (679.9 versus 652.6).

For instance for the first blocks, that constitute the sky. The sparsecode from OMP contains only one non zero coefficient, so does ADMM sparsecode. Moreover the index of this coefficient is the same for both codes, but amplitudes are really different : -0.47 for ADMM, while -1.45 for OMP.

That can be explained by the fact L_1 norm promotes coefficients to be close to 0 in terms of absolute value, regardless of the coefficient relevance. More globally, there are 1030 blocks for which indexes of non zero coefficients are exactly the same [5]. Among those that are not exactly the same, there are similarities such as "the index of the max coefficient is the same" etc. But there will not be detailed in here.

ADMM preprocesses OMP. Looking back at 3, it is easy to understand that the slowness of OMP is mostly due to the fact X_block is looked for in $\mathbb{R}^{500}$. If one is able for a block to restrict the research of X_block in a small subspace of $\mathbb{R}^{500}$, that would increase the speed of induced OMP. Let's say, one knows that for a certain block, only the first n atoms are relevant to be considered, then 3 becomes :

[4]here, it is in fact the quantized code that is analysed. It has to be kept in mind that the quantization induces a natural threshold on coefficients.
[5]On 4800 blocks

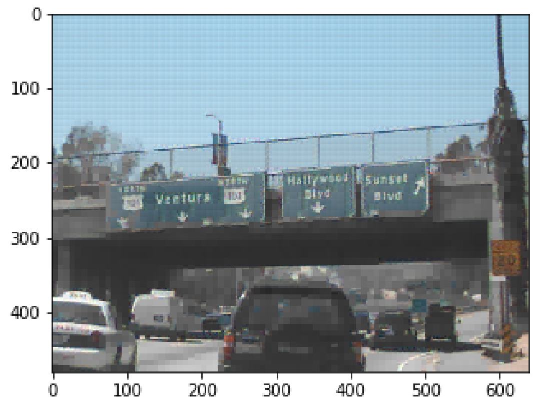


Fig. 7. Output of Algorithm 1

$$\underset{X\_block \in \mathbb{R}^n}{\arg\min} \|X\_blockD' - S\_block\|_F^2 + \lambda \|X\_block\|_0 \tag{3*}$$

where D' is the n first rows (/atoms) of D. Such that $D' \in \mathcal{M}_{n,192}$. It is what is proposed to be done with the following algorithm 1:

Algorithm 1 Encode S with a dictionary D

```
Require: $S \in (nb\_blocks, 192)$, $D \in \mathcal{M}_{\text{basis\_size},192}$
  Compute $X_{admm} \in \mathcal{M}_{\text{nb\_blocks,basis\_size}}$ with ADMM
  for all $X\_block \in X_{admm}$ do
    $I\_block \leftarrow$ indexes of the non-zero coefficients of $X\_block$ sorted from the greater absolute value to the lower
    Add $I\_block$ to $I$
    return $I$
  end for
Require: $S, D, I$
  for all $S\_block \in, I\_block \in S, I$ do
    Solve 3* for $S\_block$ and $D$ restricted to $I\_block$ rows
    Add $X\_block$ to $X$
  end for
  return $X$
```

We use the output that gives the reconstruction of **Fig.** 5b for the first step and in order to get I. Then each restricted subproblem is solved through OMP with pix_max_dev = 0.1. The whole encoding run lasts 2.5 seconds. The sparse code finally obtained contains 12283 (≈ 2.5 atoms per block) non zero values, and the quality of the reconstruction is 653.8 (to be compared to the 652.6 of the OMP alone)
The reconstruction is the **Fig.** 7.

Alternative to the OMP post-processing step. As we can see in Algorithm 1, OMP requires S, D, I ; the original image, the dictionary, and for each block the relevant subspace to be considered. It returns X_{omp}, the code for each block.

One can ask themselves : Is OMP the best algorithm to achieve this post-processing task ?
If it is assumed, that for each block I_block gives a relevant subspace (by selecting the relevant rows/atoms to pick in D) to approximate S_block. It is natural to consider :

$$\underset{X\_block \in \mathbb{R}^{Card(I_{block})}}{\arg\min} \|X\_block D' - S\_block\|_F^2 \tag{4}$$

where D' is the selection of the row from D picked with I_block as the list of indexes. It is assumed here, that ADMM offers a good enough sparsity but fails to find the best code for such a sparsity. It is rather a refining step that is needed after ADMM.
Problem 4 is a well-known problem : project a point (S) on a subspace (D') and get the coefficients of the linear combination (X).
We proposed to solve this thanks to linalg.lstsq provided by numpy. The algorithm 1 is just slightly modified :

Algorithm 2 Encode S with a dictionary D

```
Require: $S \in (nb\_blocks, 192)$, $D \in \mathcal{M}_{\text{basis\_size},192}$
  Compute $X_{admm} \in \mathcal{M}_{\text{nb\_blocks,basis\_size}}$ with ADMM
  for all $X\_block \in X_{admm}$ do
    $I\_block \leftarrow$ indexes of the non-zero coefficients of $X\_block$ sorted from the greater absolute value to the lower
    Add $I\_block$ to $I$
    return $I$
  end for
Require: $S$, $D$, $I$
  for all $S\_block \in, I\_block \in S, I$ do
    Project $S\_block$ on $D$ restricted to $I\_block$ rows and get $X\_block$
    Add $X\_block$ to $X$
  end for
  return $X$
```

Keeping the parameter of the preprocessing ADMM step unchanged, the output gives the following results :

- Encoding time : 1.78 seconds
- Sparsity : 23016 non-zero coefficients
- Data quality : 653.4

The sparsity degree is here greater, but we get faster a slightly better reconstruction.

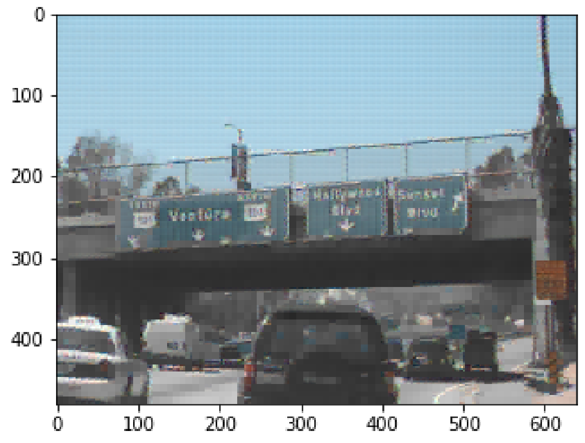


Fig. 8. Output of algorithm 2

Assuming that maybe all atoms in I_block are not relevant to be considered to project S_block. It could be relevant to threshold I_block, by keeping only the m first indexes of I_block, (cf (3), (4)). It would also get smaller subspace for the projection subproblem, so one can expect a smaller latency.

For $m = 3$[6], we get the following results with the reconstructed image in **Fig.** 8 :

- Encoding time : 1.68 seconds
- Sparsity : 9035 non-zero coefficients
- Data quality : 648.6

[6]For each block, only indexes corresponding to the three greatest coefficient are kept

Conclusion. Based on dictionary learning well-known technics, we have proposed an algorithm that attempts to approximate the solution of 1, with a satisfying latency, sparsity and quality of reconstruction. A lot of parameters remain to be optimized (parameters of the preprocessing and post-processing step). Also D has been choosen to be best for 2b*, and maybe there is a better D for our modified problem. Also D could have a different size. The projection could be somehow more efficient.
The dictionary has been choosen to encode a whole 3-channel block. But it is also possible to encode each channel independently[7]. Maybe even use a different dictionary for each channel. The benefit of this method, is reducing the size of the dictionary, which leads to a faster computation time. D would be in $\mathcal{M}_{\text{basis\_size},64}$ instead of $\mathcal{M}_{\text{basis\_size},192}$. Also the basis size could be smaller, because less atoms are needed to make an over-completed basis.

L1 norm has been used to approximate the L0 norm. The need for ADMM to access gradient of the loss function makes it the natural choice. Also the pain point of this method has been adressed with the post-processing step. ADMM output is trusted in terms of indexes of non-zero value, but with no prior on the amplitude, that is inexorably pushed to small values by the L1 norm gradient.
Nevertheless different choices for this approximation can be relevant, one is mentioned in the following article (3).

I think there are also possible improvements with the data quality, and the metric to use. With a view to being compared to JPEG encoding, I think it would be better to analyse the distribution of the pixel deviations within the image rather than using only the max of the distribution.
For a given home-made reconstruction, Kullback-Leibler-Divergence can be used to find the "closest" JPEG reconstruction among all different quality JPEG reconstruction.
Then filesizes are relevant to be compared.

In here, only the lossy part of the codec has been detailed. Our goal was a good sparsity, because we assumed a good sparsity leads to a small filesize if an efficient lossless encoding[8] is achieved. This could be the next subject to focus on. Also the parallelization has not been exploited.

We do not pretend to have found the best pre-processing and post-processing steps. For instance the modified ADMM proposed by this article (2) could increase our performances.
But it is rather a method that is proposed in here : using a preprocessor to get the relevant subspaces, then try to find the best representation in those subspaces.

[7]as it would be done for a grayscale image

[8]This lossless part could also take more benefits from a three-channel encoding, exploiting the redundancies between channels.

Software Availability. Sporco is the package that is mostly and widely used. It proposes a dedicated class for solving dictionary learning or convolutional dictionary learning with ADMM.

- https://github.com/bwohlberg/sporco

ACKNOWLEDGEMENTS
We would like to thank Ecatherina for some interesting discussions that generated some good research clues.

Bibliography

1. Stephane G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12), 1993.
2. Aaditya Ramdas and Ryan J. Tibshirani. Fast and flexible admm algorithms for trend filtering. 2015.
3. Massoud Babaie-Zadeh Hadi Zayyani. Thresholded smoothed-l0 (sl0) dictionary learning for sparse representations. *Nature methods*, 2009.
4. Cheng-wei Sang Hong Sun and Didier Le Ruyet. Sparse signal subspace decomposition based on adaptive over-complete dictionary. *EURASIP*, 2017.