1 Convolutional Dictionary Model

I will treat "convolution" and "circular convolution" as equivalent here. They are equivalent except at the boundaries. For boundary handling, see [1].

We model a signal as a sum of convolutions. $\boldsymbol{k}_{c,i}$ are the dictionary atom filters, c specifying channel, and i specifying the filter. $\boldsymbol{s}_{c,n}$ is the signal, c specifying channel, and n specifying sample. $\boldsymbol{x}_{i,n}$ are the coefficients, i specifying the filter, and n specifying the sample.

$$\boldsymbol{s}_{c,n} = \boldsymbol{\epsilon} + \sum_i \boldsymbol{x}_{i,n} * \boldsymbol{k}_{c,i} \tag{1}$$

We can construct a dictionary atom filter matrix by padding the filter, and then constructing a circulant matrix from the padded filter.

$$\boldsymbol{D}_{c,i} = \text{circ}(\text{pad}(\boldsymbol{k}_{c,i})) \tag{2}$$

This matrix operator performs circular convolution:

$$\boldsymbol{x}_{i,n} * \boldsymbol{k}_{c,i} = \boldsymbol{D}_{c,i}\boldsymbol{x}_{i,n} \tag{3}$$

So, we have the following.

$$\boldsymbol{D}_c = \begin{bmatrix} \boldsymbol{D}_{c,1} & \dots & \boldsymbol{D}_{c,K} \end{bmatrix} \tag{4}$$

$$\boldsymbol{x}_n = \begin{bmatrix} \boldsymbol{x}_{1,n} \\ \vdots \\ \boldsymbol{x}_{K,n} \end{bmatrix} \tag{5}$$

$$\boldsymbol{s}_{c,n} = \boldsymbol{\epsilon} + \boldsymbol{D}_c\boldsymbol{x}_n \tag{6}$$

C is the number of channels. N is the number of samples. $\boldsymbol{s}_{i,n}$ is the ith channel of the nth sample signal.

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{s}_{1,1} & \dots & \boldsymbol{s}_{1,N} \\ \vdots & \ddots & \vdots \\ \boldsymbol{s}_{C,1} & \dots & \boldsymbol{s}_{C,N} \end{bmatrix} \tag{7}$$

$\boldsymbol{D}_{c,i}$ is the cth channel of subdictionary corresponding to the ith dictionary atom filter.

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_{1,1} & \dots & \boldsymbol{D}_{1,K} \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C,1} & \dots & \boldsymbol{D}_{C,K} \end{bmatrix} \tag{8}$$

$\boldsymbol{x}_{i,n}$ is the coefficients corresponding to the ith dictionary atom filter for the nth sample.

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_{1,1} & \dots & \boldsymbol{x}_{1,N} \\ \vdots & \ddots & \vdots \\ \boldsymbol{x}_{K,1} & \dots & \boldsymbol{x}_{K,N} \end{bmatrix} \tag{9}$$

Finally, we have the full equation:

$$\boldsymbol{S} \approx \boldsymbol{D}\boldsymbol{X} \tag{10}$$

2 Dictionary Learning Overview

Using the dictionary $\boldsymbol{D}$ constructed from dictionary atom filters $\boldsymbol{k}_i$, dictionary learning can be thought of as the following minimization problem:

$$\begin{aligned} \min_{\boldsymbol{X},\boldsymbol{K}} & \frac{1}{2}\|\boldsymbol{S} - \boldsymbol{D}\boldsymbol{X}\|_F^2 + \lambda\|\boldsymbol{X}\|_1 \\ \text{subject to } & \|\boldsymbol{k}_{c,i}\|_2 \leq 1 \\ & \boldsymbol{D}_{c,i} = \text{circ}(\text{pad}(\boldsymbol{k}_{c,i})) \end{aligned} \tag{11}$$

Solving this jointly is just too hard. So, instead, we alternate updates of $\boldsymbol{D}$ for fixed $\boldsymbol{X}$ and updates for $\boldsymbol{X}$ for fixed $\boldsymbol{D}$.

3 The Coefficient Update

We can update coefficients independently for each sample:

$$\min_{\boldsymbol{x}_n} \frac{1}{2}\|\boldsymbol{s}_n - \boldsymbol{D}\boldsymbol{x}_n\|_F^2 + \lambda\|\boldsymbol{x}_n\|_1 \tag{12}$$

The standard approach to solving this problem is alternating direction method of multipliers (ADMM).

3.1 ADMM

ADMM works like this. To solve the problem

$$\min_x f(x) + g(x) \tag{13}$$

ADMM splits $\boldsymbol{x}$ into two variables $\boldsymbol{x}$ and $\boldsymbol{y}$, and then constains the two to be equal.

$$\begin{aligned} \min_{x,y} & f(x) + g(y) \\ \text{subject to } & y - x = 0 \end{aligned} \tag{14}$$

ADMM is an iterative algorithm with the following update equations:

$$x^{(k+1)} = \min_x f(x) + g(y^{(k)}) + \frac{\rho}{2}\|y^{(k)} - x + \frac{\gamma^{(k)}}{\rho}\|_2^2 \tag{15}$$

$$y^{(k+1)} = \min_y f(x^{(k+1)}) + g(y) + \frac{\rho}{2}\|y - x^(k+1) + \frac{\gamma^{(k)}}{\rho}\|_2^2 \tag{16}$$

$$\gamma^{(k+1)} = \gamma^{(k)} + \rho(y^{(k+1)} - x^{(k+1)}) \tag{17}$$

Obviously, a couple variables were added here. ρ is a scalar hyperparameter. γ is the dual variable corresponding to the constraint $y - x = 0$. If you want a surface level understanding without delving deep into convex optimization, I think my ADMM slides are your best bet. Here's another source on ADMM [2], but I don't think it's very accessable without a convex optimization background.

3.2 Applying ADMM to the Coefficient Update

$$f(\boldsymbol{x}_n) = \frac{1}{2}\|\boldsymbol{s}_n - \boldsymbol{D}\boldsymbol{x}_n\|_F^2 \tag{18}$$

$$g(\boldsymbol{y}_n) = \lambda\|\boldsymbol{y}_n\|_1 \tag{19}$$

$$\begin{aligned} &\min_{\boldsymbol{x}_n, \boldsymbol{y}_n} f(\boldsymbol{x}_n) + g(\boldsymbol{y}_n) \\ &\text{subject to } \boldsymbol{y}_n - \boldsymbol{x}_n = 0 \end{aligned} \tag{20}$$

3.2.1 First Update Equation

$$\boldsymbol{x}_n^{(k+1)} = \min_{\boldsymbol{x}} f(\boldsymbol{x}) + g(\boldsymbol{y}_n^{(k)}) + \frac{\rho}{2}\|\boldsymbol{y}_n^{(k)} - \boldsymbol{x} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho}\|_2^2 \tag{21}$$

$$\boldsymbol{x}_n^{(k+1)} = \min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{s}_n - \boldsymbol{D}\boldsymbol{x}\|_F^2 + \lambda\|\boldsymbol{y}_n^{(k)}\|_1 + \frac{\rho}{2}\|\boldsymbol{y}_n^{(k)} - \boldsymbol{x} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho}\|_2^2 \tag{22}$$

Take the derivative in respect to $\boldsymbol{x}$ and set equal to zero.

$$\mathbf{0} = \boldsymbol{D}^T\boldsymbol{D}\boldsymbol{x} - \boldsymbol{D}^T\boldsymbol{s}_n + \rho\boldsymbol{x} - \rho(\boldsymbol{y}_n^{(k)} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho}) \tag{23}$$

Shuffling things around, we have

$$(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})\boldsymbol{x} = \boldsymbol{D}^T\boldsymbol{s}_n + \rho(\boldsymbol{y}_n^{(k)} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho}) \tag{24}$$

$$\boldsymbol{x}_n^{(k+1)} = (\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1}(\boldsymbol{D}^T\boldsymbol{s}_n + \rho(\boldsymbol{y}_n^{(k)} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho})) \tag{25}$$

3.2.2 Second Update equation

$$\boldsymbol{y}_n^{(k+1)} = \min_{\boldsymbol{y}} f(\boldsymbol{x}_n^{(k+1)}) + g(\boldsymbol{y}) + \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{x}_n^{(k+1)} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho}\|_2^2 \tag{26}$$

$$\boldsymbol{y}_n^{(k+1)} = \min_{\boldsymbol{y}} \frac{1}{2}\|\boldsymbol{s}_n - \boldsymbol{D}\boldsymbol{x}_n^{(k+1)}\|_F^2 + \lambda\|\boldsymbol{y}\|_1 + \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{x}_n^{(k+1)} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho}\|_2^2 \tag{27}$$

Removing the irrelevant term.

$$\boldsymbol{y}_n^{(k+1)} = \min_{\boldsymbol{y}} \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{x}_n^{(k+1)} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho}\|_2^2 + \lambda\|\boldsymbol{y}\|_1 \tag{28}$$

This minimization problem is the proximal operator for the L1-norm. The solution is well-known, so I'll pass on deriving it. It's the elementwise shrinkage operator. (I included a brief explanation of proximal operators and FISTA at the end).

$$\mathrm{S}_\lambda(x) = \begin{cases} x - \lambda & x > \lambda \\ 0 & -\lambda < x < \lambda \\ x + \lambda & x < -\lambda \end{cases} \tag{29}$$

$$\boldsymbol{y}_n^{(k+1)} = S_\lambda(\boldsymbol{x}_n^{(k+1)} - \frac{\boldsymbol{\gamma}^{(k)}}{\rho}) \tag{30}$$

3.2.3 Third Update Equation

Nothing really changed here.

$$\boldsymbol{\gamma}^{(k+1)} = \boldsymbol{\gamma}^{(k)} + \rho(\boldsymbol{y}_n^{(k+1)} - \boldsymbol{x}_n^{(k+1)}) \tag{31}$$

3.3 Working with the Update Equations

The second and third update equations are simple and straightforward to implement.

The first is not.

$$\boldsymbol{x}_n^{(k+1)} = (\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1}(\boldsymbol{D}^T\boldsymbol{s}_n + \rho(\boldsymbol{y}_n^{(k)} + \frac{\boldsymbol{\gamma}^{(k)}}{\rho})) \tag{32}$$

$\boldsymbol{D}$ is a very large matrix. This inverse is impractical to compute directly. Let's recall what $\boldsymbol{D}$ is, though.

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_{1,1} & \dots & \boldsymbol{D}_{1,K} \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C,1} & \dots & \boldsymbol{D}_{C,K} \end{bmatrix} \tag{33}$$

where

$$\boldsymbol{D}_{c,i} = \text{circ}(\text{pad}(\boldsymbol{k}_{c,i})) \tag{34}$$

We're in luck here. The vectors of the Fourier basis are eigenvectors of circulant matricies, and circulant matrices are diagonalizable.

$$\boldsymbol{D} = \begin{bmatrix} \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{1,1}\mathcal{F} & \dots & \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{1,K}\mathcal{F} \\ \vdots & \ddots & \vdots \\ \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{C,1}\mathcal{F} & \dots & \mathcal{F}^{-1}\hat{\boldsymbol{D}}_{C,K}\mathcal{F} \end{bmatrix} \tag{35}$$

where $\hat{\boldsymbol{D}}_{c,i}$ is diagonal.

Now, let's pull the fourier transforms outside the dictionary matrix.

$$\hat{\boldsymbol{D}} = \begin{bmatrix} \hat{\boldsymbol{D}}_{1,1} & \dots & \hat{\boldsymbol{D}}_{1,K} \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{D}}_{C,1} & \dots & \hat{\boldsymbol{D}}_{C,K} \end{bmatrix} \tag{36}$$

$\hat{\boldsymbol{D}}$ is sparsely banded. Now, let's take a look at the matrix we are trying to invert.

$$\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D} = \rho\mathbf{I} + (\mathcal{F}^{-1}\hat{\boldsymbol{D}}\mathcal{F})^T(\mathcal{F}^{-1}\hat{\boldsymbol{D}}\mathcal{F}) \tag{37}$$

$$\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D} = \rho\mathbf{I} + \mathcal{F}^{-1}\hat{\boldsymbol{D}}^H\mathcal{F}\mathcal{F}^{-1}\hat{\boldsymbol{D}}\mathcal{F} \tag{38}$$

$$\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D} = \rho\mathbf{I} + \mathcal{F}^{-1}\hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}}\mathcal{F} \tag{39}$$

Remember that $\mathbf{I} = \mathcal{F}^{-1}\mathcal{F}$. So,

$$\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D} = \mathcal{F}^{-1}(\rho\mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}})\mathcal{F} \tag{40}$$

$$(\rho\mathbf{I} + \boldsymbol{D}^T\boldsymbol{D})^{-1} = \mathcal{F}^{-1}(\rho\mathbf{I} + \hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}})^{-1}\mathcal{F} \tag{41}$$

Looking closely at $\hat{\boldsymbol{D}}$ and $\hat{\boldsymbol{D}}^H\hat{\boldsymbol{D}}$ we can see that frequency bands do not interact. That is, if a row or column contains a DC (direct current aka zero-frequency) component of any filter, it contains no other frequency components from any filter. Therefore, we can split the inverse problem into a single $K \times K$ inverse computation for each pixel or element in the signal.

Let $\hat{\boldsymbol{D}}_{c,i}[j]$ be the jth diagonal element of $\hat{\boldsymbol{D}}_{c,i}$. Then, let $\hat{\boldsymbol{D}}[i]$ be the submatrix containing the jth diagonal elements for all channels and filters.

The inverse we want to compute is $(\rho\mathbf{I} + \hat{\boldsymbol{D}}^H[j]\hat{\boldsymbol{D}}[j])^{-1}$.

Note that the rank of $\hat{\boldsymbol{D}}^H[j]\hat{\boldsymbol{D}}[j]$ is less than or equal to the number of channels. If this number is small, the inverse can be computed efficiently using the Woodbury matrix identity.

If you are looking for more explanations or sources of what I've explained so far, [3] uses ADMM for convolutional sparse coding, [4] and [5] exploit the

sum-of-low-rank-and-identity nature of the matrix to be inverted, and [1] and [4] dicuss how to handle boundaries.

But what if the number of channels is not small?

3.4 Coefficient Update for Multi-Channel Signals

If the number of channels is large, standard approaches offer two options (see [6] (section VI)):

1. ADMM consensus:

   In the past I have used the subscript of the coefficients and signals to refer to the sample. Here, assume sample n, because I need the subscript to indicate channel. In the first update equation, estimate coefficients separately for each channel. For the second update equation, combine the results to get a single set of coefficients.

$$f_c(\boldsymbol{x}_c) = \frac{1}{2}\|\boldsymbol{s}_c - \boldsymbol{D}\boldsymbol{x}_c\|_F^2 \tag{42}$$

$$g(\boldsymbol{y}) = \lambda\|\boldsymbol{y}\|_1 \tag{43}$$

$$\begin{aligned} &\min_{\boldsymbol{x}_c, \boldsymbol{y}} \sum_{c=1}^{C} f_c(\boldsymbol{x}_c) + g(\boldsymbol{y}) \\ &\text{subject to } \boldsymbol{y} - \boldsymbol{x}_c = 0 \qquad \forall c \end{aligned} \tag{44}$$

   The update equations become the following:

   Obviously, C separate updates here.

$$\boldsymbol{x}_c^{(k+1)} = (\rho\mathbf{I} + \boldsymbol{D}_c^T\boldsymbol{D}_c)^{-1}(\boldsymbol{D}_c^T\boldsymbol{s}_c + \rho(\boldsymbol{y}^{(k)} + \frac{\boldsymbol{\gamma}_c^{(k)}}{\rho})) \tag{45}$$

$$\boldsymbol{y}^{(k+1)} = S_\lambda(\frac{1}{C}\sum_{c=1}^{C}\boldsymbol{x}_c^{(k+1)} - \frac{\boldsymbol{\gamma}_c^{(k)}}{\rho}) \tag{46}$$

   Obviously, C separate updates here.

$$\boldsymbol{\gamma}_c^{(k+1)} = \boldsymbol{\gamma}_c^{(k)} + \rho(\boldsymbol{y}^{(k+1)} - \boldsymbol{x}_c^{(k+1)}) \tag{47}$$

2. Alternatively, you can use the FISTA algorithm [7].

Now, here is my thought on all this. When learning a dictionary using an online method, the dictionary will be updated frequently. What if we could figure out a way to make the updates for each frequency low rank? Then we could use the Sherman-Morrison formula to efficiently update our inverses! Alternatively, a low-rank approximation could suffice, but I'm less enthused about that approach.

4 The Dictionary Update

Most dictionary update approaches are batch methods (they handle the entire dataset at once.)

Recall our minimization problem.

$$\begin{aligned} &\min_{\boldsymbol{X},\boldsymbol{K}} \frac{1}{2}\|\boldsymbol{S}-\boldsymbol{D}\boldsymbol{X}\|_F^2 + \lambda\|\boldsymbol{X}\|_1 \\ &\text{subject to } \|\boldsymbol{k}_i\|_2 \leq 1 \\ &\qquad \boldsymbol{D}_{c,i} = \text{circ}(\text{pad}(\boldsymbol{k}_{c,i})) \end{aligned} \tag{48}$$

We are going to keep $\boldsymbol{X}$ fixed, and update $\boldsymbol{D}$.

$$\begin{aligned} &\min_{\boldsymbol{K}} \frac{1}{2}\|\boldsymbol{S}-\boldsymbol{D}\boldsymbol{X}\|_F^2 \\ &\text{subject to } \|\boldsymbol{k}_i\|_2 \leq 1 \\ &\qquad \boldsymbol{D}_{c,i} = \text{circ}(\text{pad}(\boldsymbol{k}_{c,i})) \end{aligned} \tag{49}$$

Before, we used the dictionary atom filters to generate circulant matrices for the dictionary. Now, we can do the same with the coefficients. To avoid confusion, we can call these coefficients $\boldsymbol{Y}$.

I am flipping n and i for this.

$$\boldsymbol{Y}_{n,i} = \text{circ}(\boldsymbol{x}_{i,n}) \tag{50}$$

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y}_{1,1} & \dots & \boldsymbol{Y}_{1,K} \\ \vdots & \ddots & \vdots \\ \boldsymbol{Y}_{N,1} & \dots & \boldsymbol{Y}_{N,K} \end{bmatrix} \tag{51}$$

$$\boldsymbol{d}_c = \begin{bmatrix} \text{pad}(\boldsymbol{k}_{c,1}) \\ \vdots \\ \text{pad}(\boldsymbol{k}_{c,K}) \end{bmatrix} \tag{52}$$

$$\boldsymbol{s}_c = \begin{bmatrix} \text{pad}(\boldsymbol{s}_{c,1}) \\ \vdots \\ \text{pad}(\boldsymbol{s}_{c,N}) \end{bmatrix} \tag{53}$$

$$\begin{aligned} &\min_{\boldsymbol{d}_{c,i}} \frac{1}{2}\sum_{c=1}^{C}\|\boldsymbol{s}_c - \boldsymbol{Y}\boldsymbol{d}_c\|_2^2 \\ &\text{subject to } \|\boldsymbol{k}_{c,i}\|_2 \leq 1 \\ &\qquad \boldsymbol{d}_{c,i} = \text{pad}(\boldsymbol{k}_{c,i}) \end{aligned} \tag{54}$$

The two constraints

$$\|\boldsymbol{k}_{c,i}\|_2 \leq 1 \tag{55}$$

and

$$\boldsymbol{d}_{c,i} = \text{pad}(\boldsymbol{k}_{c,i}) \tag{56}$$

are convex constraints, the first normalizing the filter, and the second requiring the filters to be small/local. Let T be the convex set the constraints require $\boldsymbol{d}_{c,i}$ to be in.

The simplest approach to an online algorithm is projected gradient descent. That would look like this:

$$\boldsymbol{d}_c^{(k+1)} = \text{proj}_T(\boldsymbol{d}_c^{(k)} - \alpha \boldsymbol{Y}_n^T(\boldsymbol{Y}_n \boldsymbol{d}_c^{(k)} - \boldsymbol{s}_n)) \tag{57}$$

where α is the step size.

There are a couple variants of this, but ultimately things end up very similar. Add momentum, and you get the FISTA algorithm [7]. Include past coefficients in your gradient calculation before projection, and you get a second-order method [8].

My question is, can we find an update close to this one that is low-rank for each frequency, for efficient inverse updates? It feels like there has to be a better approach to coefficient updates than FISTA or consensus ADMM when doing online dictionary learning for multi-channel data.

5 FISTA

I've mentioned the fast iterative shrinkage algorithm a few times, so I probably should go ahead and explain it.

Suppose you have a two-term objective function $f(x) + g(x)$ and you want to minimize it.

f is easily differentiable, and g has a simple proximal operator.

Definition of proximal operator:

$$\text{prox}_g(x) = \arg\min_y \frac{1}{2}\|y - x\|_2^2 + g(y) \tag{58}$$

Then, the iterative shrinkage thresholding algorithm (ISTA) goes like this:

$$\boldsymbol{x}^{(k+1)} = \text{prox}_g(\boldsymbol{x}^{(k)} - \alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}) \tag{59}$$

Where α is step size for gradient steps.

The fast iterative shrinkage thresholding algorithm (FISTA) just adds momentum to the mix.

$$\boldsymbol{y}^{(k+1)} = \text{prox}_g(\boldsymbol{x}^{(k)} - \alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}) \tag{60}$$

$$t^{(k+1)} = \frac{1}{2}(1 + \sqrt{1 + 4(t^{(k)})^2}) \tag{61}$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{y}^{(k+1)} + \frac{t^{(k)} - 1}{t^{(k+1)}}(z^{(k+1)} - \boldsymbol{x}^{(k)}) \tag{62}$$

The concept of momentum (in case you aren't familiar) is a gradient descent concept. Instead of computing the gradient at the current location, you compute the gradient a little further ahead.

In the context of the equations above, think of $\boldsymbol{z}$ as your current location, but $\boldsymbol{x}$ as where you are looking for gradient information.

References

[1] B. Wohlberg, "Boundary handling for convolutional sparse representations," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 1833–1837, Sep. 2016.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[3] H. Bristow, A. Eriksson, and S. Lucey, "Fast convolutional sparse coding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 391–398, 2013.

[4] F. Heide, W. Heidrich, and G. Wetzstein, "Fast and flexible convolutional sparse coding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5135–5143, 2015.

[5] M. Šorel and F. Šroubek, "Fast convolutional sparse coding using matrix inversion lemma," *Digital Signal Processing*, vol. 55, pp. 44–51, 2016.

[6] C. Garcia-Cardona and B. Wohlberg, "Convolutional dictionary learning: A comparative review and new algorithms," *IEEE Transactions on Computational Imaging*, vol. 4, no. 3, pp. 366–381, 2018.

[7] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[8] J. Liu, C. Garcia-Cardona, B. Wohlberg, and W. Yin, "First-and second-order methods for online convolutional dictionary learning," *SIAM Journal on Imaging Sciences*, vol. 11, no. 2, pp. 1589–1628, 2018.