

**TITLE OF YOUR THESIS**

A Dissertation  
Presented to  
The Academic Faculty

By

George P. Burdell

In Partial Fulfillment  
of the Requirements for the Degree  
Doctor of Philosophy in the  
School of Fiction

Georgia Institute of Technology

January 1927

Copyright © George P. Burdell 1927

# **TITLE OF YOUR THESIS**

Approved by:

Dr. Burdell, Advisor  
School of Myths  
*Georgia Institute of Technology*

Dr. Two  
School of Mechanical Engineering  
*Georgia Institute of Technology*

Dr. Three  
School of Electrical Engineering  
*Georgia Institute of Technology*

Dr. Four  
School of Computer Science  
*Georgia Institute of Technology*

Dr. Five  
School of Public Policy  
*Georgia Institute of Technology*

Dr. Six  
School of Nuclear Engineering  
*Georgia Institute of Technology*

Date Approved: January 11, 2000

A great quote to start the thesis

*George P. Burdell*

A great dedication goes here.

## **ACKNOWLEDGEMENTS**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

## TABLE OF CONTENTS

<b>Acknowledgments</b> . . . . .	v
<b>List of Tables</b> . . . . .	vii
<b>List of Figures</b> . . . . .	viii
<b>Chapter 1: Introduction</b> . . . . .	1
1.1 Dictionaries and Dictionary Learning . . . . .	1
1.1.1 Convolutional Dictionaries . . . . .	1
1.2 Convolutional Neural Networks . . . . .	1
1.3 Multi-Layer Dictionaries . . . . .	1
1.4 Contributions and Organization of Dissertation . . . . .	1
<b>Chapter 2: Learning Dictionaries for Multi-Channel Signals</b> . . . . .	3
2.1 Introduction . . . . .	3
2.2 Dictionary Types . . . . .	3
2.3 Pursuit and Sparse Coding . . . . .	4
2.4 ADMM . . . . .	5
2.5 Applying ADMM to the Sparse Coding Problem . . . . .	7
2.5.1 Exploiting Dictionary Structure for the Inverse Problem . . . . .	9

2.6	Literature Review . . . . .	11
2.6.1	Convolutional Sparse Coding . . . . .	11
2.7	ADMM with Low-Rank Updates . . . . .	11
2.8	Conclusion . . . . .	11
<b>Chapter 3: Learning Multi-Layer Dictionaries . . . . .</b>		<b>13</b>
3.1	Introduction . . . . .	13
3.2	Literature Review . . . . .	13
3.3	Multi-Layer ADMM with Low-Rank Updates . . . . .	14
3.4	Summary . . . . .	14
<b>Chapter 4: JPEG Artifact Removal . . . . .</b>		<b>15</b>
4.1	Introduction . . . . .	15
4.2	JPEG Algorithm . . . . .	15
4.3	Literature Review . . . . .	16
4.4	Modelling Compressed JPEG Images . . . . .	16
4.5	Handling Quantization . . . . .	16
4.6	Experiments . . . . .	16
4.6.1	Experiment Setup . . . . .	16
4.6.2	Results . . . . .	16
4.7	Conclusion . . . . .	16
<b>Chapter 5: Practical Considerations Concerning Tensorflow . . . . .</b>		<b>17</b>
5.1	Boundary Handling . . . . .	17

5.2	Removing Low-Frequency Signal Content . . . . .	17
5.2.1	JPEG Artifact Removal . . . . .	17
5.3	Tensorflow and Keras . . . . .	17
5.3.1	Why Not Use Gradient Tape and TensorFlow-1-Style Code? . . . . .	17
5.3.2	Shared Weights Between Layers . . . . .	18
5.3.3	Custom Partial Gradients . . . . .	18
5.3.4	Updating TensorFlow Variables After Applying Gradients . . . . .	19
5.3.5	The Perils of Using Built-In Functions for Complex Tensors and Arrays . . . . .	21
<b>Appendix A: Experimental Equipment . . . . .</b>		<b>23</b>
<b>Appendix B: Data Processing . . . . .</b>		<b>24</b>
<b>References . . . . .</b>		<b>26</b>
<b>Vita . . . . .</b>		<b>27</b>



## LIST OF TABLES

1.1	This is an example Table. . . . .	1
-----	-----------------------------------	---

## LIST OF FIGURES

1.1	This is an example Figure. . . . .	2
2.1	This is another example Figure, rotated to landscape orientation. . . . .	12

## SUMMARY

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

# CHAPTER 1

## INTRODUCTION

### Dictionaries and Dictionary Learning

#### Convolutional Dictionaries

### Convolutional Neural Networks

### Multi-Layer Dictionaries

### Contributions and Organization of Dissertation

Table 1.1: This is an example Table.

x	f(x)	g(x)
1	6	4
2	6	3
3	6	2
4	6	2

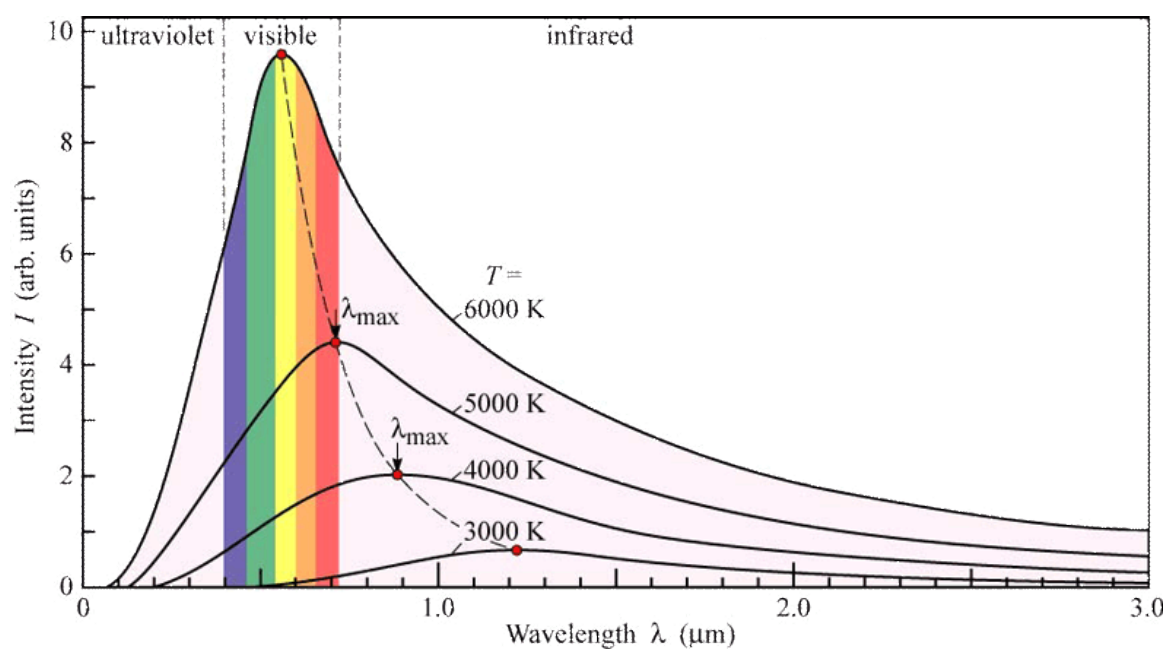


Figure 1.1: This is an example Figure.

## **CHAPTER 2**

### **LEARNING DICTIONARIES FOR MULTI-CHANNEL SIGNALS**

#### **Introduction**

When using a multi-layer dictionary model, the coefficients corresponding to a dictionary from one layer become the "signal" for the subsequent layer. The number of channels for this "signal" is the number of dictionary filters from the previous layer. Much of the literature on learning convolutional dictionaries is tailored to applications with signals that only have a small number of channels. This chapter presents a novel method for learning convolutional dictionaries from and for multi-channel signals.

#### **Dictionary Types**

There are many ways to construct a convolutional sparse representation of a multi-channel signal, but broadly the distinctions reduce down to if and how signal channels share dictionaries and coefficients, and if and how those non-shared entities interact across channels.

It is common in many applications for dictionary models to share dictionaries across channels, which requires the use multi-channel coefficients. If such models were used in a multi-layer dictionary model, the tensor rank would increase with each subsequent layer.

For this work, I focus instead on the multi-channel dictionary with shared coefficients. This structure matches that of convolutional neural networks, and the number of channels for a subsequent dictionary is the number of filters for the dictionary from the previous layer.

## Pursuit and Sparse Coding

The dictionary model decomposes the signal  $s_i$  into a dictionary  $D$  (which generalizes to other signals) and the coefficients  $x_i$  (which are specific to the signal  $s_i$ ):

$$s_i \approx Dx_i \quad (2.1)$$

(Here the subscript  $i$  specifies a particular signal and its corresponding coefficients.) A pursuit algorithm finds the coefficients  $x_i$  corresponding to a particular signal  $s_i$  for known dictionary  $D$ . If the number of dictionary atoms (columns) is larger than the dimension of the signal, then the number of unknowns is larger than the number of equations, and many solutions for  $x_i$  represent  $s_i$  equally well (at least in an L2 sense). Researchers and practitioners commonly either impose a sparsity constraint on the coefficients or add a coefficient L1 penalty to the objective function, which removes the ambiguity from the problem construction. When such a penalty or constraint is added, pursuit is sometimes called sparse coding. With the added coefficient L1 penalty, the pursuit optimization problem looks like this:

$$x_i = \arg \min_x \frac{1}{2} \|s_i - Dx\|_2^2 + \lambda \|x\|_1 \quad (2.2)$$

where  $\lambda$  is a hyperparameter greater than zero controlling how much the L1 norm of the coefficients is penalized. Researchers have proposed many ways to solve this problem. If the dictionary is convolutional and the number of channels is low, a standard approach is to use the Alternating direction Method of Multipliers (ADMM) algorithm.

## ADMM

ADMM is a convex-optimization algorithm used to solve the optimization problem:

$$\begin{aligned} & \underset{\mathbf{x}, \mathbf{y}}{\text{minimize}} f(\mathbf{x}) + g(\mathbf{y}) \\ & \text{subject to } \mathbf{Ax} + \mathbf{By} + \mathbf{c} = \mathbf{0} \end{aligned} \quad (2.3)$$

where  $f$  and  $g$  are convex functions [1]. (I will address how to put the sparse coding problem in this form in the next section.)

The ADMM algorithm makes use of the augmented Lagrangian, a particular expression that has a saddle point at the solution to the constrained optimization problem:

$$\mathcal{L}_\rho(\mathbf{x}, \mathbf{y}, \mathbf{u}) = f(\mathbf{x}) + g(\mathbf{y}) + \mathbf{u}^H(\mathbf{Ax} + \mathbf{By} + \mathbf{c}) + \frac{\rho}{2} \|\mathbf{Ax} + \mathbf{By} + \mathbf{c}\|_2^2 \quad (2.4)$$

where  $\rho$  is a hyperparameter greater than zero and  $\mathbf{u}$  is the dual variable for the constraints.

At the saddle-point solution, the augmented Lagrangian is at a minimum in respect to  $\mathbf{x}$  and  $\mathbf{y}$ , but at a maximum in respect to  $\mathbf{u}$ .

The ADMM algorithm is an iterative search for the saddle point of the augmented Lagrangian. Each iteration consists of a primal update for  $\mathbf{x}$ , a primal update for  $\mathbf{y}$ , and a dual update for  $\mathbf{u}$ :

$$\mathbf{x}^{(k+1)} = \arg \min_{\mathbf{x}} \mathcal{L}_\rho(\mathbf{x}, \mathbf{y}^{(k)}, \mathbf{u}^{(k)}) \quad (2.5)$$

$$\mathbf{y}^{(k+1)} = \arg \min_{\mathbf{y}} \mathcal{L}_\rho(\mathbf{x}^{(k+1)}, \mathbf{y}, \mathbf{u}^{(k)}) \quad (2.6)$$



$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \rho(\mathbf{Ax}^{(k+1)} + \mathbf{By}^{(k+1)} + \mathbf{c}) \quad (2.7)$$

The primal updates serve to move towards the minimum of the augmented Lagrangian in respect to  $\mathbf{x}$  and  $\mathbf{y}$  with  $\mathbf{u}$  fixed, and the dual update fixes  $\mathbf{x}$  and  $\mathbf{y}$ , and performs gradient ascent on  $\mathbf{u}$  with stepsize  $\rho$ . Under very mild assumptions, this process converges to the saddle point of the augmented Lagrangian, which matches the solution to the constrained optimization problem.

There are two common variations of the ADMM algorithm that this work will make use of. The first is the scaled form, which comes from completing the square for the augmented lagrangian function:

$$L_\rho(\mathbf{x}, \mathbf{y}, \mathbf{u}) = f(\mathbf{x}) + g(\mathbf{y}) + \frac{\rho}{2} \|\mathbf{Ax} + \mathbf{By} + \mathbf{c} + \frac{\mathbf{u}}{\rho}\|_2^2 - \frac{1}{2\rho} \|\mathbf{u}\|_2^2 \quad (2.8)$$

The term  $-\frac{1}{2\rho} \|\mathbf{u}\|_2^2$  can be ignored for the primal updates because it has no dependence on the primal variables. For this reason, it is sometimes more convenient to keep track of  $\frac{\mathbf{u}}{\rho}$  instead of  $\mathbf{u}$ , since that is the form that appears in the augmented Lagrangian after completing the square.

$$\frac{\mathbf{u}^{(k+1)}}{\rho} = \frac{\mathbf{u}^{(k)}}{\rho} + \mathbf{Ax}^{(k+1)} + \mathbf{By}^{(k+1)} + \mathbf{c} \quad (2.9)$$

This form is known as scaled ADMM.

The other common variation of ADMM updates the dual variable more frequently.

$$\mathbf{x}^{(k+1)} = \arg \min_{\mathbf{x}} L_\rho(\mathbf{x}, \mathbf{y}^{(k)}, \mathbf{u}^{(k)}) \quad (2.10)$$

$$\mathbf{u}^{(k+\frac{1}{2})} = \mathbf{u}^{(k)} + (\alpha - 1)\rho(\mathbf{Ax}^{(k+1)} + \mathbf{By}^{(k)} + \mathbf{c}) \quad (2.11)$$

$$\mathbf{y}^{(k+1)} = \arg \min_{\mathbf{y}} L_{\rho}(\mathbf{x}^{(k+1)}, \mathbf{y}, \mathbf{u}^{(k+\frac{1}{2})}) \quad (2.12)$$

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k+\frac{1}{2})} + \rho(\mathbf{Ax}^{(k+1)} + \mathbf{By}^{(k+1)} + \mathbf{c}) \quad (2.13)$$

When  $\alpha > 1$ , this is known as overrelaxation, and if  $\alpha < 1$ , this is known as underrelaxation.<sup>1</sup>  $\alpha$  is always chosen to be greater than zero. In some applications, researchers have found using over-relaxation converges faster than without overrelaxation [2], but optimal choice of  $\alpha$  is problem-dependent [3].

### Applying ADMM to the Sparse Coding Problem

Recall from section 2.3, equation 2.2 for sparse coding.

$$\mathbf{x}_i = \arg \min_{\mathbf{x}} \frac{1}{2} \|\mathbf{s}_i - \mathbf{Dx}\|_2^2 + \lambda \|\mathbf{x}\|_1 \quad (2.14)$$

This can be rewritten to match the ADMM form from equation 2.3:

$$\begin{aligned} & \underset{\mathbf{x}, \mathbf{y}}{\text{minimize}} \quad \frac{1}{2} \|\mathbf{s}_i - \mathbf{Dx}\|_2^2 + \lambda \|\mathbf{y}\|_1 \\ & \text{subject to} \quad \mathbf{y} - \mathbf{x} = \mathbf{0} \end{aligned} \quad (2.15)$$

Given sufficient iterations,  $\mathbf{x}$  and  $\mathbf{y}$  will both be close to the optimal, but they may not be equal. Either can be used as an approximate solution to the sparse coding problem.

Computing the augmented Lagrangian of convex optimization problem in expression

---

<sup>1</sup>I have chosen to notate over/under relaxation differently than what is standard, but the  $\alpha$  is the same, and the notations are mathematically equivalent. The standard notation instead adds the term  $-(1-\alpha)(\mathbf{Ax}^{(k+1)} + \mathbf{By}^{(k)} + \mathbf{c})$  to  $\mathbf{Ax}^{(k+1)}$  and substitutes that expression for  $\mathbf{Ax}^{(k+1)}$  in subsequent equations.

2.15 yields the following equation:

$$L_\rho(\mathbf{x}, \mathbf{y}, \mathbf{u}) = \frac{1}{2} \|\mathbf{s}_i - \mathbf{D}\mathbf{x}\|_2^2 + \lambda \|\mathbf{y}\|_1 + \frac{\rho}{2} \|\mathbf{y} - \mathbf{x} + \frac{\mathbf{u}}{\rho}\|_2^2 - \frac{1}{2\rho} \|\mathbf{u}\|_2^2 \quad (2.16)$$

Starting with the  $\mathbf{x}$ -update:

$$\mathbf{x}^{(k+1)} = \arg \min_{\mathbf{x}} L_\rho(\mathbf{x}, \mathbf{y}^{(k)}, \mathbf{u}^{(k)}) \quad (2.17)$$

Since the minimum is desired, setting the gradient to zero will produce the solution.

$$\nabla_{\mathbf{x}^{(k+1)}} L_\rho(\mathbf{x}^{(k+1)}, \mathbf{y}^{(k)}, \mathbf{u}^{(k)}) = \mathbf{0} \quad (2.18)$$

$$\mathbf{0} = \mathbf{D}^T \mathbf{D} \mathbf{x}^{(k+1)} - \mathbf{D}^T \mathbf{s}_i + \rho \mathbf{x}^{(k+1)} - \rho(\mathbf{y}^{(k)} + \frac{\mathbf{u}^{(k)}}{\rho}) \quad (2.19)$$

$$(\rho \mathbf{I} + \mathbf{D}^T \mathbf{D}) \mathbf{x}^{(k+1)} = \mathbf{D}^T \mathbf{s}_i + \rho(\mathbf{y}^{(k)} + \frac{\mathbf{u}^{(k)}}{\rho}) \quad (2.20)$$

$$\mathbf{x}^{(k+1)} = (\rho \mathbf{I} + \mathbf{D}^T \mathbf{D})^{-1} (\mathbf{D}^T \mathbf{s}_i + \rho(\mathbf{y}^{(k)} + \frac{\mathbf{u}^{(k)}}{\rho})) \quad (2.21)$$

In subsection 2.5.1, there is a discussion of the implications of this update equation, how to compute it for cases in which the signal has a low number of channels, and the challenges it poses for signals with many channels.

If using over-relaxation, there is a dual update:

$$\frac{\mathbf{u}^{(k+\frac{1}{2})}}{\rho} = \frac{\mathbf{u}^{(k)}}{\rho} + (\alpha - 1)(\mathbf{y}^{(k)} - \mathbf{x}^{(k+1)}) \quad (2.22)$$

Moving on to the  $\mathbf{y}$ -update:

$$\mathbf{y}^{(k+1)} = \arg \min_{\mathbf{y}} L_{\rho}(\mathbf{x}^{(k+1)}, \mathbf{y}, \mathbf{u}^{(k+\frac{1}{2})}) \quad (2.23)$$

Excluding the terms that don't include  $\mathbf{y}$ , I have

$$\mathbf{y}^{(k+1)} = \arg \min_{\mathbf{y}} \lambda \|\mathbf{y}\|_1 + \frac{\rho}{2} \left\| \mathbf{y} - \mathbf{x}^{(k+1)} + \frac{\mathbf{u}^{(k+\frac{1}{2})}}{\rho} \right\|_2^2 \quad (2.24)$$

This is a well-known problem, whose solution is

$$\mathbf{y}^{(k+1)} = S_{\frac{\lambda}{\rho}} \left( \mathbf{x}^{(k+1)} - \frac{\mathbf{u}^{(k+\frac{1}{2})}}{\rho} \right) \quad (2.25)$$

where  $S$  is the shrinkage operator:

$$S_b(x) = \begin{cases} x - b & x > b \\ 0 & -b < x < b \\ x + b & x < -b \end{cases} \quad (2.26)$$

In the case of a vector, matrix, or tensor input, the shrinkage operator is applied element by element.

Finally, the last update equation for the dual variable:

$$\frac{\mathbf{u}^{(k+1)}}{\rho} = \frac{\mathbf{u}^{(k+\frac{1}{2})}}{\rho} + \mathbf{y}^{(k+1)} - \mathbf{x}^{(k+1)} \quad (2.27)$$

### Exploiting Dictionary Structure for the Inverse Problem

Returning to the  $\mathbf{x}$  update:

$$\mathbf{x}^{(k+1)} = (\rho \mathbf{I} + \mathbf{D}^T \mathbf{D})^{-1} (\mathbf{D}^T \mathbf{s}_i + \rho (\mathbf{y}^{(k)} + \frac{\mathbf{u}^{(k)}}{\rho})) \quad (2.28)$$

For problems using a dictionary with convolutional structure, this inverse for the convolutional sparse coding problem is very structured. Exploiting this structure is important for efficient computation, because the matrix  $\rho\mathbf{I} + \mathbf{D}^T\mathbf{D}$  is a large matrix.

Writing  $\mathbf{D}$  in a block structure, I have

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_{1,1}, & \dots, & \mathbf{D}_{1,M} \\ \vdots & \ddots & \vdots \\ \mathbf{D}_{C,1}, & \dots, & \mathbf{D}_{C,M} \end{bmatrix} \quad (2.29)$$

where  $\mathbf{D}_{c,m}$  is a toplitz matrix capturing channel  $c$  of the  $m$ th filter of the dictionary.

Toplitz matrices are diagonalizable with Fourier eigenvectors:

$$\mathbf{D} = \begin{bmatrix} \mathcal{F}^{-1}\hat{\mathbf{D}}_{1,1}\mathcal{F}, & \dots, & \mathcal{F}^{-1}\hat{\mathbf{D}}_{1,M}\mathcal{F} \\ \vdots & \ddots & \vdots \\ \mathcal{F}^{-1}\hat{\mathbf{D}}_{C,1}\mathcal{F}, & \dots, & \mathcal{F}^{-1}\hat{\mathbf{D}}_{C,M}\mathcal{F} \end{bmatrix} \quad (2.30)$$

where  $\hat{\mathbf{D}}_{c,m}$  is a diagonal matrix whose elements are the discrete Fourier transform (FFT) of channel  $c$  of the  $m$ th dictionary filter.

This sparsely banded structure is a useful form in analyzing the structure of the inverse problem:

$$(\rho\mathbf{I} + \mathbf{D}^T\mathbf{D})^{-1} = \mathcal{F}^{-1}(\rho\mathbf{I} + \hat{\mathbf{D}}^H\hat{\mathbf{D}})^{-1}\mathcal{F} \quad (2.31)$$

where

$$\hat{\mathbf{D}} = \begin{bmatrix} \hat{\mathbf{D}}_{1,1}, & \dots, & \hat{\mathbf{D}}_{1,M} \\ \vdots & \ddots & \vdots \\ \hat{\mathbf{D}}_{C,1}, & \dots, & \hat{\mathbf{D}}_{C,M} \end{bmatrix} \quad (2.32)$$

and in a slight abuse of notation,  $\mathcal{F}$  computes the FFT separately on the coefficients for each filter. In [4], Bristow et al. observe the matrix  $\rho\mathbf{I} + \hat{\mathbf{D}}^H\hat{\mathbf{D}}$  is sparsely banded, so the inverse can be broken down into much smaller inverse problems, and one only needs to compute the inverse of an  $M \times M$  matrix for every pixel in the image.  $(\rho\mathbf{I} + \hat{\mathbf{D}}^H\hat{\mathbf{D}}$  is

an  $M \times M$  block matrix, whose blocks are diagonal. Each submatrix collects one element from the diagonal of each of the blocks.)

Furthermore, the maximum rank of these submatrices is  $C$ , so if  $C$  is small, these inverses can be computed even more efficiently using the Woodbury matrix identity [5] [6] [7].

According to the Woodbury matrix identity, for any invertible matrix  $U$  and any matrix  $V$ :

$$(U + V^H V)^{-1} = U^{-1} - U^{-1} V^H (I + V U^{-1} V^H)^{-1} V U^{-1} \quad (2.33)$$

So,

$$(\rho I + \hat{D}^H \hat{D})^{-1} = \frac{1}{\rho} I - \frac{1}{\rho} \hat{D}^H (\rho I + \hat{D} \hat{D}^H)^{-1} \hat{D} \quad (2.34)$$

This means that instead of computing the inverse of an  $M \times M$  matrix for every pixel in the image, one could instead choose to compute the inverse of a  $C \times C$  matrix for each pixel in the image.<sup>2</sup>

## Literature Review

### Convolutional Sparse Coding

### ADMM with Low-Rank Updates

## Conclusion

---

<sup>2</sup>Generally, Cholesky or LDLT decomposition would be preferable to explicitly computing the inverse, and the efficiency gains due to the Woodbury matrix identity are relevant regardless of the chosen representation.

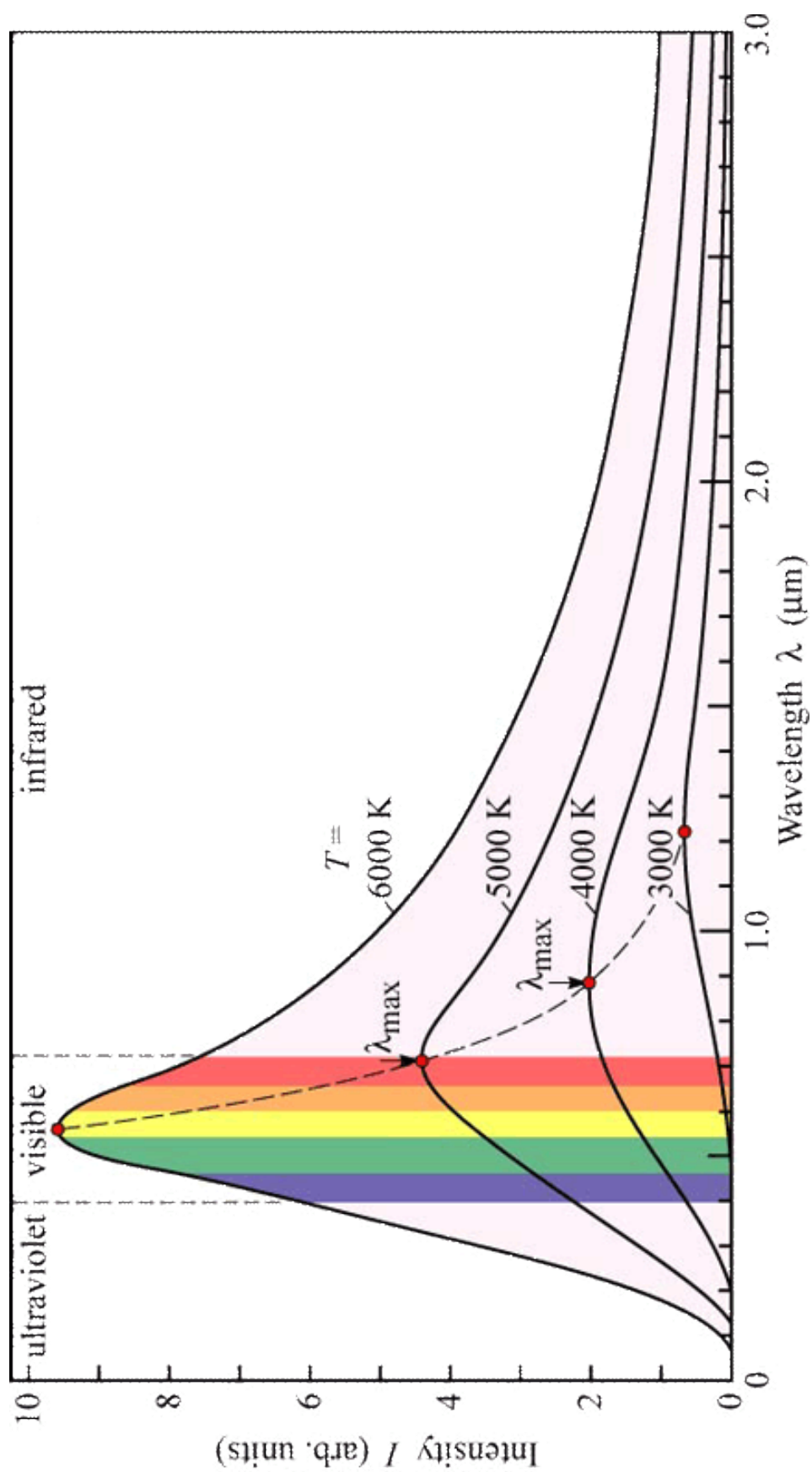


Figure 2.1: This is another example Figure, rotated to landscape orientation.

## **CHAPTER 3**

### **LEARNING MULTI-LAYER DICTIONARIES**

#### **Introduction**

A multi-layer dictionary model is composed of multiple dictionaries; the model treats the dictionary coefficients of a previous layer as the signal for the subsequent layer. This model dates back to Zeiler’s Deconvolutional Neural Networks [8] and can be thought of as a deep autoencoder [9, Chapter 14][10]. Some researchers have interpreted convolutional neural networks as multi-layer dictionary models, the convolution and its corresponding rectified linear units serving as a crude pursuit algorithm [11]. In this chapter, I explain how to apply the novel dictionary learning algorithm from the prior chapter to the multi-layer dictionary learning problem.

#### **Literature Review**

In 2010, Zeiler et al. proposed a multi-layer dictionary model termed a deconvolutional network. The learning process for dictionary filters is entirely unsupervised, and they learn their filters layer-by-layer. Their algorithm is greedy in the sense that there is no feedback from subsequent layers to influence the learning process on the previous layer. This approach was tested both on the task of removing added gaussian noise to images, and also as a feature extraction method for object recognition on the Caltech-101 dataset [12]. While this research drew a lot of attention at the time, as the success of alternative models like convolutional neural networks grew, the popularity of deconvolutional networks decreased.

Multi-layer dictionaries also appear in Bayesian models, going by names such as hierarchical convolutional factor analysis [13][14] and deep deconvolutional learning [15]. These networks use probabilistic models to prune network architecture and provide interpretable



dictionaries. Inference can be slow.

## **Multi-Layer ADMM with Low-Rank Updates**

### **Summary**

## **CHAPTER 4**

### **JPEG ARTIFACT REMOVAL**

#### **Introduction**

Despite the existence of better compression algorithms, use of the JPEG compression algorithm is ubiquitous: it is the most commonly used image compression algorithm. Overzealous JPEG compression can produce visible distortions, and image restoration from these distortions is a challenging problem. There are two aspects of JPEG compression which make the restoration process more challenging than simpler restoration problems like deblurring or removing salt-and-pepper noise: JPEG's block-based approach is not spatially invariant, and the quantization is nonlinear. This chapter describes a novel approach to address the challenges of JPEG image restoration using the ADMM-based convolutional sparse coding for a multi-layer dictionary model.

#### **JPEG Algorithm**

The JPEG compression process begins with an RGB image input, and consists of five steps. The first is a color transformation, transitioning from RGB to YUV. Then, the U and V color channels are downsampled. The DCT for each  $8 \times 8$  block is computed (separately for each channel). The DCT coefficients are then quantized using a quantization matrix determined by a user-chosen JPEG quality factor. Finally, these quantized coefficients are reordered and encoded using a lossless variable length coding process.

The standard reconstruction process reverses the lossless encoding, computes the IDCT of the blocks, upsamples the color channels, and reverses the color transform.

**Literature Review**

**Modelling Compressed JPEG Images**

**Handling Quantization**

**Experiments**

Experiment Setup

Results

**Conclusion**

## **CHAPTER 5**

### **PRACTICAL CONSIDERATIONS CONCERNING TENSORFLOW**

#### **Boundary Handling**

#### **Removing Low-Frequency Signal Content**

##### JPEG Artifact Removal

#### **Tensorflow and Keras**

Most of the computations for my research rely on TensorFlow version 2.3.1 [16], a Python library for machine learning specializing in building models with differentiable, parameterizable composite functions and learning model parameters using gradient descent or other gradient-based optimization methods. TensorFlow is a common platform for researchers and developers working on artificial neural networks, and there are many tutorials and examples freely available online, so I will not replicate that work here. This chapter section the reader already has some familiarity with TensorFlow and Keras [17] (a high-level library inside TensorFlow). The goal of this section is to provide the reader with the tools and workarounds to be able to replicate my work without resorting to hacking things together with gradient tape and/or TensorFlow-1-style code.

##### Why Not Use Gradient Tape and TensorFlow-1-Style Code?

Keras offers a high-level environment. Code written in Keras's framework is easier to integrate with other work. Gradient tape is great for hacking something together or debugging, but promotes styles of coding that are less readable, less maintainable, and less portable. Keras also has a lower learning curve than the broader TensorFlow library.

## Shared Weights Between Layers

Trainable TensorFlow variables declared outside of any Keras layer will not be automatically added to a Keras model's list of trainable variables. In most cases, this limitation is not a problem; it is intuitive to declare a layer's weights inside that layer. However, sometimes the same variable is needed in multiple distinct layers. To be include a variable in the model's trainable variables, it is sufficient to declare the variable in one layer and pass the variable (or the layer it was initialized in) as an input argument to the `__init__` function of the other layers that share that variable. This will work even if the Keras model does not use the layer that declared the variable.<sup>1</sup>

## Custom Partial Gradients

TensorFlow offers a well-documented means of replacing TensorFlow's gradient computations of an operation with specified custom gradient computations. However, if the operation involves multiple tensors that are inputs or trainable variables, the standard approach replaces all the gradients with custom gradients. If TensorFlow's gradient computations are sufficient for some tensors but not others, a workaround is necessary. This workaround is best explained by example.

Suppose the operation is the following:

$$z = f(x, y)$$

for which the standard TensorFlow gradient computations of  $f$  are desired in respect to  $x$ , but the custom gradient computations desired in respect to  $y$  are specified in function  $g(\nabla_z \mathcal{L})$ . This can be rewritten as the following:

---

<sup>1</sup>One could instead declare the variable outside any layers, pass it into the `__init__` functions of all the variables that depend on it, and then manually add the variable to the model's list of trainable variables, but I do not recommend this approach. The resulting code will be less readable and much less maintainable.

```

@tf.custom_gradient
def h(z,y):
    def grad_fun(grad):
        return (tf.identity(grad),g(grad))
    return z,grad_fun
z = f(x,tf.stop_gradient(y))
z = h(z,y)

```

The function  $h$  does nothing on the forward pass, but in the backward pass computes the custom gradient in respect to  $y$  as intended.

### Updating TensorFlow Variables After Applying Gradients

To update TensorFlow Variables after applying gradients, it is necessary to track which variables are affected and what their corresponding update functions are. To accomplish this, I store the update functions in a Python dictionary using variable names as the dictionary keys. This Python dictionary needs to be widely accessible so that layers can add update functions when they are initialized; a simple way to do this is to make the update function Python dictionary a class attribute. The keys need to be unique, but TensorFlow variable names can conflict. It is easy to avoid this problem by checking for conflicts before adding a new update function.

```

class PostProcess:
    update = {}
    def add_update(varName,update_fun):
        assert varName not in PostProcess.update
        PostProcess.update[varName] = update_fun

```

In the standard Keras training paradigm, models are trained using the fit function, a method in the Keras model object. The fit function calls the function `train_step`, where gradients are applied. To update TensorFlow Variables after gradients are applied, `train_step`

is the function to modify. The only change that needs to be made is adding a function call to all update functions that correspond to the model's list of trainable variables.

```
class Model_subclass(tf.keras.Model):  
    def train_step(self, data):  
        trainStepOutputs =  
            tf.keras.Model.train_step(self, data)  
        update_ops = []  
        for tv in self.trainable_variables:  
            if tv.name in PostProcess.update:  
                PostProcess.update[tv.name]()  
        return trainStepOutputs
```

Changes to Tensorflow variables in the update function must use the assign command (or its variants: assign\_add, assign\_sub, ect). Otherwise, TensorFlow will detect that computations lie outside of its computational graph and throw an error. Note that using the assign command on Python variables that are not TensorFlow variables will produce some very cryptic error messages, so be sure to use the assign command correctly. If the value change of one TensorFlow variable depends on the value of another TensorFlow variable value pre-update, it may be necessary to use the TensorFlow control\_dependencies command to get TensorFlow to track that dependency. TensorFlow has a useful tool called TensorBoard that helps visualize TensorFlow's dependencies, but a workaround is required to use TensorBoard on update functions that are called after applying gradients. To use TensorBoard to visualize dependencies in an update function, temporarily call the update function in the layer's call method, use TensorBoard to verify all necessary dependencies are being tracked, then remove the update function call from the layer's call method.

## The Perils of Using Built-In Functions for Complex Tensors and Arrays

The TensorFlow Probability version 0.11.1 [18] is an extension of TensorFlow mostly used for probabilistic models. The library contains a Cholesky update function, but the function does not properly handle complex inputs. To compute Cholesky updates for complex inputs, users should either write their own implementation or use my code (included in supplementary material). Similarly, the Randomized SVD algorithm in the Python scikit-learn library does not properly handle complex inputs.

Errors like these are fairly common, so when dealing with complex data, researchers and practitioners should carefully verify that the function libraries they rely on are properly handling complex numbers.



# **Appendices**

## **APPENDIX A**

### **EXPERIMENTAL EQUIPMENT**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

## **APPENDIX B**

### **DATA PROCESSING**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

## REFERENCES

- [1] 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.
- [2] J. Eckstein, “Parallel alternating direction multiplier decomposition of convex programs,” *Journal of Optimization Theory and Applications*, vol. 80, no. 1, pp. 39–62, 1994.
- [3] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. Jordan, “A general analysis of the convergence of admm,” in *International Conference on Machine Learning*, PMLR, 2015, pp. 343–352.
- [4] H. Bristow, A. Eriksson, and S. Lucey, “Fast convolutional sparse coding,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 391–398.
- [5] M. Šorel and F. Šroubek, “Fast convolutional sparse coding using matrix inversion lemma,” *Digital Signal Processing*, vol. 55, pp. 44–51, 2016.
- [6] 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*, 2015, pp. 5135–5143.
- [7] B. Wohlberg, “Efficient algorithms for convolutional sparse representations,” *IEEE Transactions on Image Processing*, vol. 25, no. 1, pp. 301–315, 2015.
- [8] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, “Deconvolutional networks,” in *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, IEEE, 2010, pp. 2528–2535.
- [9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [10] A. Rangamani, A. Mukherjee, A. Basu, A. Arora, T. Ganapathi, S. Chin, and T. D. Tran, “Sparse coding and autoencoders,” in *2018 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2018, pp. 36–40.
- [11] V. Pappas, Y. Romano, and M. Elad, “Convolutional neural networks analyzed via convolutional sparse coding,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2887–2938, 2017.

- [12] L. Fei-Fei, R. Fergus, and P. Perona, “Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories,” in *2004 conference on computer vision and pattern recognition workshop*, IEEE, 2004, pp. 178–178.
- [13] B. Chen, G. Polatkan, G. Sapiro, L. Carin, and D. B. Dunson, “The hierarchical beta process for convolutional factor analysis and deep learning,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 361–368.
- [14] ar, “Deep learning with hierarchical convolutional factor analysis,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1887–1901, 2013.
- [15] Y. Pu, X. Yuan, and L. Carin, “Generative deep deconvolutional learning,” *ArXiv preprint arXiv:1412.6039*, 2014.
- [16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.
- [17] F. Chollet *et al.*, *Keras*, <https://keras.io>, 2015.
- [18] J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous, “Tensorflow distributions,” *ArXiv preprint arXiv:1711.10604*, 2017.

## **VITA**

Vita may be provided by doctoral students only. The length of the vita is preferably one page. It may include the place of birth and should be written in third person. This vita is similar to the author biography found on book jackets.