Rochester Institute of Technology

RIT Digital Institutional Repository

Theses

8-2019

# Deep Grassmann Manifold Optimization for Computer Vision

Breton Lawrence Minnehan
blm2144@rit.edu

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Minnehan, Breton Lawrence, "Deep Grassmann Manifold Optimization for Computer Vision" (2019). Thesis. Rochester Institute of Technology. Accessed from

This Dissertation is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

Deep Grassmann Manifold Optimization for Computer Vision

by

Breton Lawrence Minnehan

A dissertation submitted in partial fulfillment of the
requirements for the degree of
**Doctor of Philosophy**
**in Engineering**

Kate Gleason College of Engineering
Rochester Institute of Technology

August 2019

Signature of the Author ________________________________________

Certified by ________________________________________
Ph.D. Program Director Date

Ph.D. IN ENGINEERING PROGRAM
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

---

Ph.D. DEGREE DISSERTATION

---

The Ph.D. degree dissertation of Breton Lawrence Minnehan
has been examined and approved by the
dissertation committee as satisfactory for the
dissertation required for the
Ph.D. degree in Engineering

Dr. Andreas Savakis, Dissertation Advisor Date

Dr. Christopher Kanan, Committee Member Date

Dr. Andres Kwasinski, Committee Member Date

Dr. Panos P. Markopoulos, Committee Member Date

*This dissertation is dedicated to my Papou, Nicholas Papatones. Whenever I'm in need of encouragement I think of you and hear you saying:*

*"Be strong, like bull!"*

Deep Grassmann Manifold Optimization for Computer Vision

by

Breton Lawrence Minnehan

Submitted to the
Kate Gleason College of Engineering Ph.D. in Engineering Program
in partial fulfillment of the requirements for the
**Doctor of Philosophy Degree**
at the Rochester Institute of Technology

## Technical Abstract

In this work, we propose methods that advance four areas in the field of computer vision: dimensionality reduction, deep feature embeddings, visual domain adaptation, and deep neural network compression. We combine concepts from the fields of manifold geometry and deep learning to develop cutting edge methods in each of these areas. Each of the methods proposed in this work achieves state-of-the-art results in our experiments. We propose the Proxy Matrix Optimization (PMO) method for optimization over orthogonal matrix manifolds, such as the Grassmann manifold. This optimization technique is designed to be highly flexible enabling it to be leveraged in many situations where traditional manifold optimization methods cannot be used.

We first use PMO in the field of dimensionality reduction, where we propose an iterative optimization approach to Principal Component Analysis (PCA) in a framework called Proxy Matrix optimization based PCA (PM-PCA). We also demonstrate how PM-PCA can be used to solve the general $L_p$-PCA problem, a variant of PCA that uses arbitrary fractional norms, which can be more robust to outliers. We then present Cascaded Projection (CaP), a method which uses tensor compression based on PMO, to reduce the number of filters in deep neural networks. This, in turn, reduces the number of computational operations required to process each image with the network. Cascaded Projection is the first end-to-end trainable method for network compression that uses standard backpropagation to learn the optimal tensor compression. In the area of deep feature embeddings, we introduce Deep Euclidean Feature Representations through Adaptation on the Grassmann manifold (DEFRAG), that leverages PMO. The DEFRAG method improves the feature embeddings learned by deep neural networks through the use of auxiliary loss functions and Grassmann manifold optimization. Lastly, in the area of visual domain adaptation, we propose the Manifold-Aligned Label Transfer for Domain Adaptation (MALT-DA) to transfer knowledge from samples in a known domain to an unknown domain based on cross-domain cluster correspondences.

# Deep Grassmann Manifold Optimization for Computer Vision

by

Breton Lawrence Minnehan

Submitted to the
Kate Gleason College of Engineering Ph.D. in Engineering Program
in partial fulfillment of the requirements for the
**Doctor of Philosophy Degree**
at the Rochester Institute of Technology

## Outreach Abstract

As digital imaging sensors have become ubiquitous in modern society, there has been increasing interest in extracting information from the data the sensors collect. The study of extracting information from visual data is known as computer vision. The field of computer vision has made significant progress in recent years thanks in large part to the emergence of deep learning. In this work, we present novel methods that advance five areas of computer vision and deep learning. The first area of contribution is Grassmann manifold optimization, a constrained optimization technique that has application in many fields, including: physics, communications, deep learning, and computer vision. In the second area of research, we propose methods for dimensionality reduction based on our manifold optimization method. Dimensionality reduction is used in many fields to reduce high dimensional data to a more compact representation. In the third area of research we propose a method for compressing deep neural networks in order to reduce the number of computations required and reduce the processing time. In the fourth area of research we propose a method that encourages deep networks to learn better representations of data by grouping similar classes closer than dissimilar classes. The fifth area of research is in the field of domain adaptation, that aims to solve a problem known as domain shift encountered due to the difference between the data that a classifier is trained on and data the classifier encounters when deployed. The domain adaptation method developed in this work mitigates the domain shift by finding correspondences between the data in two domains and uses the correspondences to transfer the labels from the labeled source samples to the unlabeled target samples. Our proposed work in each of these areas has many applications such as pedestrian detection in autonomous vehicles, classifying remote sensing images and object recognition on mobile devices.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The contributions of this dissertation can roughly be separated into four distinct areas: dimensionality reduction, network compression, deep feature embeddings, and visual domain adaptation. However, all four areas are closely related and improvements in one can prove useful in the others. There are two core conceptual threads that tie all of these areas together: deep learning and manifold optimization. Chapter 2 provides the reader with the theoretical background on the former while Chapter 3 focuses on the latter.

Deep Convolutional Neural Networks (CNNs) and their variants have emerged as the architecture of choice for computer vision. Deep networks have achieved state-of-the-art results in object class recognition [73], [130], [48], face recognition [123], semantic segmentation [90], pose estimation [149], and visual tracking [106] among other applications.

There is a large body of work studying Grassmann manifolds and their applications in the field of computer vision. Grassmann manifolds have been used in distance metric learning and subspace analysis for problems such as domain adaptation and visual object tracking. In this work we demonstrate how Grassmann manifold optimization can be used to perform robust Principal Component Analysis in a computationally efficient manner. We demonstrate how by restricting fully-connected layers of Artificial Neural Networks we can ensure the feature embedding is a Euclidean space. This provides multiple benefits including that the resulting features can be accurately compared using the Euclidean distance. Additionally, we demonstrate how this framework can be adopted to obtain an end-to-end method for visual domain adaptation.

Dimensionality reduction methods have an incredibly vast array of applications in signal processing, such as signal compression or classification applications in computer vision. Most computer visions tasks require extremely high dimensional data to be reduced to only the pertinent information in the form of features. In this work we propose new methods for dimensionality reduction using iterative optimization methods originally developed in the field of deep learning.

The field of feature embedding has received some attention recently from the standpoint of improving feature clustering [101] or for zero/one-shot learning. In each of these cases, attempts have been made to learn image embeddings using deep neural networks where images of similar subjects/classes have features that are close together and images of different subjects/classes are further apart. However, many of these works ignore a key problem, the feature spaces they learn lack an orthonormal basis, and thus are not Euclidean spaces. Yet these methods continue to use the Euclidean distance as a similarity metric for the feature embeddings. Furthermore, there are many examples of methods which use the features extracted from the last layer of a deep neural network, specifically after a rectified linear activation function, as their feature embedding. This is even more problematic as the feature representation is inherently sparse. We will go into more detail as to why this is a problem in Chapter 6. We will also discuss how Grassmann manifold optimization can be used during the training process to generate proper Euclidean space embeddings such that the resulting features can be accurately compared using the Euclidean metric.

There are many potential applications for learning deep Euclidean embedding spaces from direct use in deep networks to reduce overfitting and improve the clustering behavior. For applications where there is only a limited number of samples available, such as person re-identification, a k-Nearest Neighbor (kNN) classifier is typically used which often relies on the Euclidean distance metric between features. In a different context, deep Euclidean embeddings could be used in applications such as visual object tracking.

A common problem that is faced when a classification method is developed for application to a real-world problem is a reduction in accuracy when applied to new data. This problem arises because the data used to train the classifier and the data the classifier processes when deployed can be quite different. This difference, known as domain shift or dataset bias, can result from changes in illumination conditions (daylight vs. night, sunny vs. cloudy), environmental conditions (desert vs. forest, urban vs. residential), or changes in sensor modalities (low resolution vs. high resolution). Each of these presents an instance where a classifier trained in one domain, called the source, would not perform well in a different domain, called the target, unless somehow adapted. This adaptation process is known as domain adaptation and is a quickly developing area of research. As is evident from the multiple examples there are many applications that would greatly benefit from advances in domain adaptation. We propose the development of domain adaptation techniques that leverage our work in the two other fields to provide a method that is better suited for the situations commonly faced in modern computer vision applications.

The remainder of this work is organized as follows: Chapter 2 discusses related work and other background knowledge in the field of deep learning. Chapter 3 provides background knowledge on manifold optimization and presents a new method developed in this work. Chapter 4 outlines our work in the field of dimensionality reduction. Chapter 5 applies our manifold optimization method to the field of deep neural network compression for improved efficiency and reduced size. Chapter 6 introduces new methods for learning better features representations though the combination of deep learning and manifold optimization. Chapter 7 presents our work in the field of domain adaptation. Finally, Chapter 8 provides concluding remarks.

## 1.1 Contributions

The contributions of this dissertation are as follows:

- Develop optimization on Grassmann Manifolds using recent advancements in gradient-based optimization to accelerate convergence and improve robustness to non-convex loss functions. We call our new Grassmann manifold optimization technique Proxy Matrix Optimization.

- Develop a novel dimensionality reduction framework that uses an artificial neural network approach for Grassmann manifold optimization. The proposed framework can solve for various dimensionality reduction techniques through simple changes in the network's training loss function.

- Develop a new method for compression of deep neural networks using a decomposition method and Proxy Matrix Optimization, that is trained in an end-to-end manner.

- Develop a deep feature embedding method that learns semantically meaningful Euclidean embedding space. The deep embeddings can both reduce the potential for overfitting and ensure the network is more adaptable to different problem-types and image-domains.

- Advance the field of domain adaptation so that a trained classifier can be adapted to new domains with minimal loss in classification accuracy.

# Chapter 2

# Deep Learning

Deep learning is one of the fastest growing areas of research in the field of computer science. In recent years there have been countless breakthroughs across a wide array of problem areas that have been achieved through use of deep learning methods. In this chapter we provide a general overview of the field of Artificial Neural Networks (ANNs) and Deep Learning.

## 2.1 Artificial Neural Networks

The fast-growing field of deep learning traces its origins back to the artificial perceptron [115] first proposed in the 1950's. The design of the perceptron is said to be inspired by the connections on neurons. Where, similar to a biological neuron, the perceptron's activation is determined based on a weighted summation of its input.

The basic building block of ANNs is the artificial perceptron, shown in Figure 2.1. At its core the perceptron is a weighted summation of an input vector that is passed through a nonlinearity. This operation consists of an inner product of the input vector, $\mathbf{x}$, and the weight vector, $\mathbf{w}$. A bias, $b$ is added to the weighted sum, and then a nonlinear function $g(\cdot)$ is applied as:

$$f(\mathbf{x}) = g(\mathbf{x}^T\mathbf{w} + b) \tag{2.1}$$

The perception can easily process multiple inputs simultaneously, known as batch processing. Additionally, perceptrons can be used in parallel to generate what is known as a layer of perceptrons with multiple outputs, as shown in Figure 2.2. This doesn't

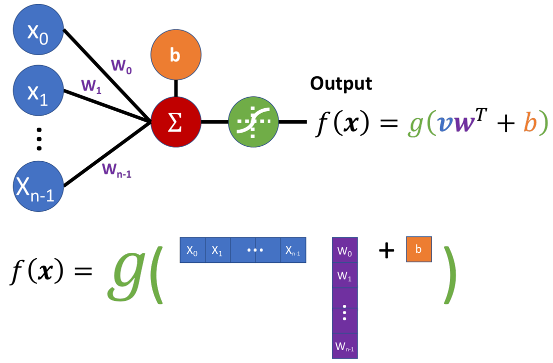


Figure 2.1: Visual depiction of the Perceptron.

require any change to the linear algebra except that the weight vector, $\mathbf{w}$, is replaced with a $m \times n$ weight matrix, generated by stacking the $m$ different $n$-dimensional weight vectors where $m$ is the number of perceptrons (output dimension).

The power of the artificial perceptron becomes more apparent when $d$ layers of perceptrons are stacked on top of each other, where the output of one layer is used as the input to the next layer, as shown in Figure 2.3. A non-linear function is used as the activation function for each perceptron. Again, the mathematics for the Multi-Layer Perceptron (MLP) do not change besides stacking the operations of each perceptron as:

$$f(\mathbf{x}) = g(...g(g(\mathbf{x}(\mathbf{W^0})^T + \mathbf{b^0})(\mathbf{W^1})^T + \mathbf{b^1})...(\mathbf{W^{d-1}})^T + \mathbf{b^{d-1}}) \tag{2.2}$$

To this point we only discussed the forward pass of ANNs. In ANNs the weight and bias values are adjusted with back propagation [118] using an optimization method, e.g. stochastic gradient decent. The weights and biases are adjusted by first calculating the outputs, $f(\mathbf{x})$, generated by the input, $\mathbf{x}$. The output $f(\mathbf{x})$ is compared to a desired output, $\mathbf{t}$, such as a one-hot encoding of the samples class, and a loss $L$ is calculated using one of many commonly used loss functions such as the $L_2$-norm or cross entropy. No matter which loss is used, the core concept behind backpropagation is the application of the chain rule to calculate the partial derivatives of the loss with respect to each sets of weights. The partial derivative of the error with respect to each of the weights, $\frac{\partial L}{\partial w}$, is then used to adjust the weights in the direction that minimizes the error. For networks with multiple layers, this process is performed through use of the chain rule. For output nodes the partial derivative of loss, $L$ with respect to the

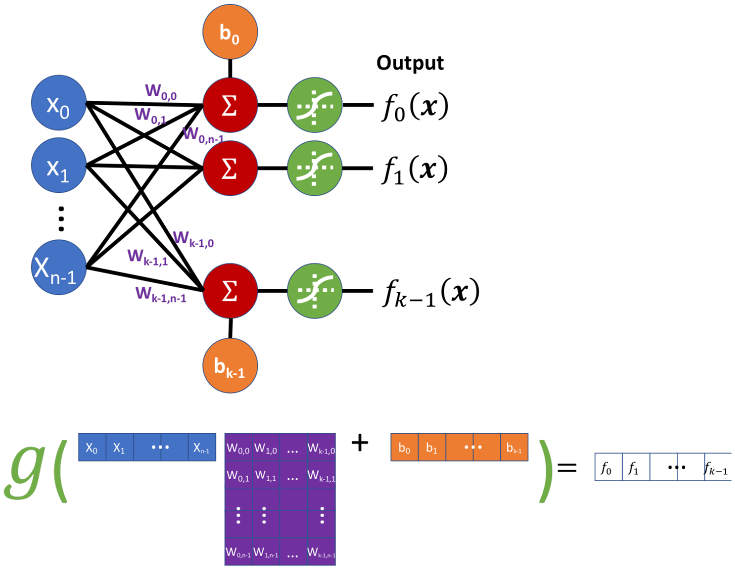


Figure 2.2: Visual depiction of using multiple perceptrons to generate multiple outputs for a single input.

weight connecting the output node $o_j$ with the node $o_i$ in the previous layer is given as:

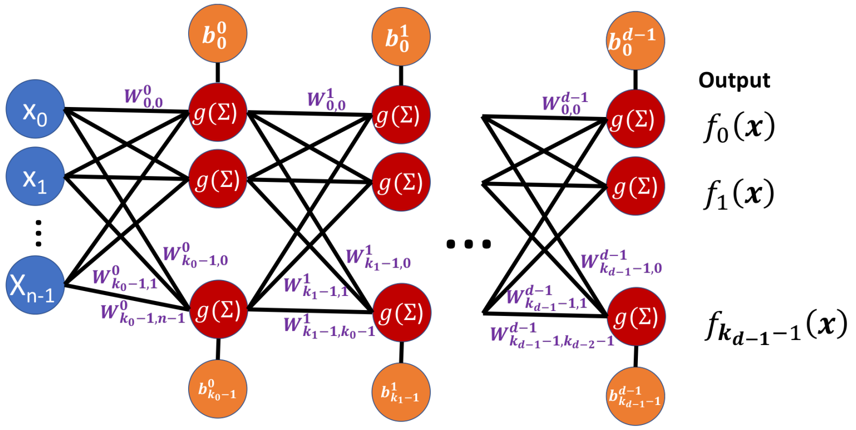
$$\frac{\partial L}{\partial w_{i,j}} = o_i(o_j - t_j)o_j(1 - o_j) \tag{2.3}$$

where $t_j$ is the ground truth output value for the node $o_j$. If the node $o_j$ is not an output node the partial derivative of the loss function with respect to the weight connecting it to the node $o_i$ in the previous layer is $\frac{\partial L}{\partial w_{i,j}} = o_i\delta_j$ where $\delta_j$ is given as:

$$\delta_j = (\sum_{l \in L} w_{i,l}\delta_l)o_j(1 - o_j) \tag{2.4}$$

The loss produced by the perceptron for each input is used to update the weight and bias vectors.

The field of ANNs stalled for many years for three primary reasons: Firstly, the fully-connected ANN architecture required a large number of parameters and thus was prone to overfitting smaller datasets. Secondly, ANNs require many passes of forward and backward propagation the computational resources were not sufficient to



$$f(\boldsymbol{x}) = g(\ldots g(g(\boldsymbol{v}(\boldsymbol{w}^0)^T + b^0)(\boldsymbol{w}^1)^T + b^1) \ldots (\boldsymbol{w}^{d-1})^T + b^{d-1})$$

Figure 2.3: Visual depiction of a Multiple Layer Perceptron (MLP).

train deep networks on large datasets. Third, ANNs were viewed as theoretical black boxes with little intuition or framework.

Convolutional Neural Networks [82] were revolutionary because they provided a way for ANNs leverage the spatial information available in imagery. Another benefit to CNNs over MLPs is that they required a much smaller set of trained-parameters. With traditional fully-connected MLPs the number of parameters for each layer was $n^m$ where $n$ is the input dimension $(h \times w)$ of the image and $m$ is the number of outputs for the layer is the number of outputs for the layer. For example, a two layer layer fully-connected network with 2516 hidden nodes and 10 output nodes and an input image of size (28 × 28), requires 203,530 parameters. However, with CNNs the number of parameters for each layer is reduced to $mk^2$ where $k$ is the size of the convolution kernel, usual 3 to 7 pixels. Therefor a similar convolutional network with $(3 \times 3)$ kernels would require just 33,280 parameters, assuming a global pooling layer is on the last layer to reduce each feature map to a scalar value, as is commonly done. A visual representation of a CNN is given in Figure 2.4. The mathematical formulation for each layer of a CNN is given as:

$$f(\mathbf{X}) = g(\mathbf{XW} + b) \tag{2.5}$$

where $\mathbf{X}$ is the Two-Dimensional input to the layer and $\mathbf{W}$ is the weight kernel and

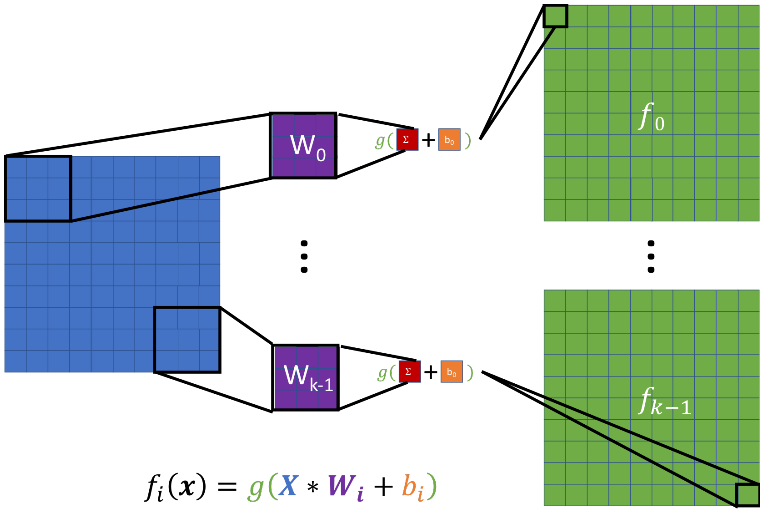

W0
g( Σ + b0 )
f0
Wk-1
g( Σ + b0 )
fk−1
$f_i(\boldsymbol{x}) = g(\boldsymbol{X} * \boldsymbol{W}_i + b_i)$

Figure 2.4: Visual depiction of a single layer of a Convolutional Neural Network (CNN).

$*$ is the convolution operation. This can be extended for cases where the input has multiple channels by performing the 2D convolution on the $H \times W \times C$ input with a $k \times k \times C$ kernel.

Although it is more efficient to implement a CNN using convolution operations it is important to recognize that this operation can be performed using the MLP formulation. The only change required is that the input image must be decomposed into separate $k \times k$ blocks that are vectorized from an input matrix $\mathbf{X}$ that has dimensions $WH \times k^2$. A visual example of this decomposition method is given in Figure 2.5.

The field of deep learning did not fully take off until AlexNet [73] which demonstrated the ability of deep CNNs to classify a large number of classes in ImageNet by leveraging their parallel nature and using Graphic Processing Units (GPUs) during training. The advances in the field of deep learning are too numerous to review here. However, we will outline some of the particular advances that are specifically pertinent to this work.

Batch Normalization was originally introduced as a method to improve the training speed and reduce the tendency of deep neural networks to overfit the training set. This was done by normalizing inputs to each neuron, $x_i$, in the network across all the samples in the current mini-batch. The whitened input, $\hat{x}_i$, of each input to neurons is

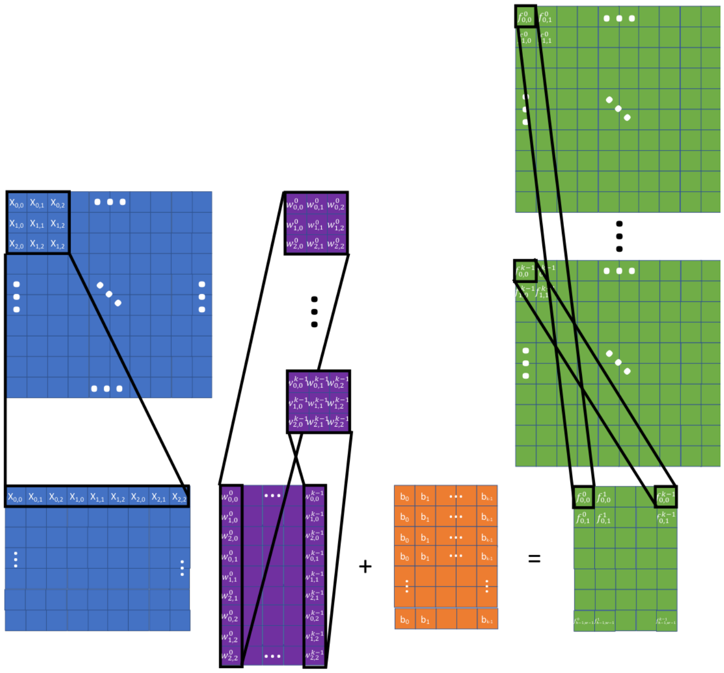

$x_{0,0}$ $x_{0,1}$ $x_{0,2}$
$x_{1,0}$ $x_{1,1}$ $x_{1,2}$
$x_{2,0}$ $x_{1,2}$ $x_{1,2}$
$x_{0,0}$ $x_{0,1}$ $x_{0,2}$ $x_{1,0}$ $x_{1,1}$ $x_{1,2}$ $x_{2,0}$ $x_{2,1}$ $x_{2,2}$
$w^0_{0,0}$ $w^0_{0,1}$ $w^0_{0,2}$
$w^0_{1,0}$ $w^0_{1,1}$ $w^0_{1,2}$
$w^0_{2,0}$ $w^0_{2,1}$ $w^0_{2,2}$
$w^{k-1}_{0,0}$ $w^{k-1}_{0,1}$ $w^{k-1}_{0,2}$
$w^{k-1}_{1,0}$ $w^{k-1}_{1,1}$ $w^{k-1}_{1,2}$
$w^{k-1}_{2,0}$ $w^{k-1}_{2,1}$ $w^{k-1}_{2,2}$
$w^0_{0,0}$ $w^0_{1,0}$ $w^0_{2,0}$ $w^0_{0,1}$ $w^0_{1,1}$ $w^0_{2,1}$ $w^0_{0,2}$ $w^0_{1,2}$ $w^0_{2,2}$
$w^{k-1}_{0,0}$ $w^{k-1}_{1,0}$ $w^{k-1}_{2,0}$ $w^{k-1}_{0,1}$ $w^{k-1}_{1,1}$ $w^{k-1}_{2,1}$ $w^{k-1}_{0,2}$ $w^{k-1}_{1,2}$ $w^{k-1}_{2,2}$
+
$b_0$ $b_1$ $b_{k-1}$
=
$f^0_{0,0}$ $f^0_{0,1}$
$f^0_{1,0}$ $f^0_{1,1}$
$f^{k-1}_{0,0}$
$f^0_{0,0}$ $f^1_{0,0}$ $f^{k-1}_{0,0}$
$f^0_{0,1}$ $f^1_{0,1}$ $f^{k-1}_{0,1}$

Figure 2.5: Visual depiction of the decomposition of a convolution operation into a locally connected implementation.

calculated as follows:

$$\bar{x}_i = \frac{1}{T} \sum_{i=(0,T)} x_i \tag{2.6}$$

$$\sigma_i^2 = \frac{1}{T} \sum_{i=(0,T)} (x_i - \bar{x}_i)^2 \tag{2.7}$$

$$\hat{x}_i = \frac{(x_i - \bar{x}_i)}{\sqrt{\sigma_i^2}} \tag{2.8}$$

Where $T$ is the number of samples in the mini-batch, and $i$ is the is the dimension index of the input. Normalizing the activations across each mini-batch ensures that the gradients for each update set are better suited for training, especially in early epochs with randomly initialized weights. Batch Normalization allows for much faster training with higher learning rates. Traditionally the statistics and scale/shift values for the activations are learned at training time and frozen at test time, meaning that the activations are whitened based on the statistics of the training set, but not the test set.

Another technique that is commonly used to train deep neural networks is dropout [133]. The aim of dropout is to prevent deep networks from overfitting by forcing redundancy into the network. Dropout forces redundancy by randomly selecting a set of neurons in each layer with a given probability, generally between 25% to 50%, and setting their activations to zero. This forces the network to rely on the remaining features in the layer, and thus builds redundancy into the network. The introduction of dropout into the training process allows deeper networks to be trained with a significantly lower risk of overfitting. Convolutional neural network architectures have made significant advances in recent years. In this section we only highlight three commonly used architectures: VGG [129], Residual Networks (ResNet) [49] and Densely Connected Networks (DenseNet) [57]. There are many others that are used for various applications.

The VGG architecture is a traditional convolutional network architecture that was built with the same design in the original CNNs introduced in [80] and popularized in [73]. The VGG architecture generally has several convolutional layers utilizing 3x3 filters followed by a max pooling layer. These convolutional blocks are repeated until the last convolutional features are vectorized and fed into several fully connected layers. This type of architecture is the fundamental type of CNN, yet it is still commonly used throughout the field of deep learning. A common variation is that most researchers introduce batch normalization layers into the network as well.

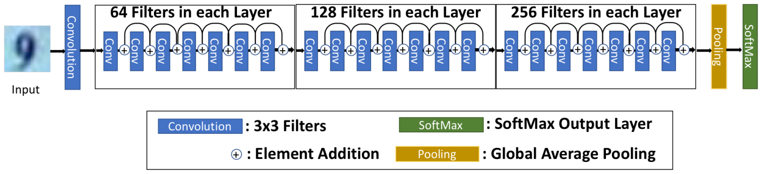

Input
Convolution
64 Filters in each Layer
128 Filters in each Layer
256 Filters in each Layer
Conv
Pooling
SoftMax
Convolution : 3x3 Filters
SoftMax : SoftMax Output Layer
⊕ : Element Addition
Pooling : Global Average Pooling

Figure 2.6: ResNet convolutional network architecture.

More recently, techniques such as ResNet [49] have introduced feed forward connections into networks that allow for training deeper networks. Residual connections provide a pathway around each convolution layer through which the gradients can flow. This is done by summing the input to each convolutional layer with its output. A visual example of a ResNet is presented in Figure 2.6

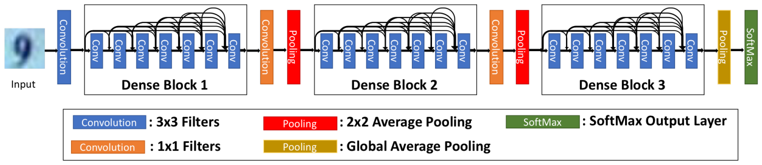

Input
Convolution
Conv
Dense Block 1
Convolution
Pooling
Dense Block 2
Convolution
Pooling
Dense Block 3
Pooling
SoftMax
Convolution : 3x3 Filters
Pooling : 2x2 Average Pooling
SoftMax : SoftMax Output Layer
Convolution : 1x1 Filters
Pooling : Global Average Pooling

Figure 2.7: Densely connected convolutional network architecture.

The concept of residual connections was further advanced in [57] which introduced densely connected networks (DenseNets). DenseNets are different than ResNets in that they are built using Dense Blocks where the feature maps from each layer are used as inputs for every subsequent layer in the block. This architecture allows for the combination of both high and low-level features throughout the network. An example DenseNet architecture with 25 layers split into three Dense Blocks is shown in Figure 2.7. In this work we primarily chose to use a DenseNet architecture because the state-of-the-art results it achieved on many of the standard image classification datasets [107] [74] [25]. DenseNets have also been shown to have a smoother error surface and thus are more likely to converge to a global minimum, as shown in [86].

# Chapter 3

# Manifold Optimization

Manifolds are used through the field of machine learning for various applications such as: dimensionality reduction [20], invariant subspace learning [144, 145], clustering [16, 148], regularization [59], and many more. Of particular interest to this work are matrix manifolds, the Stiefel manifold and the Grassmann manifold. We develop the Proxy Matrix Optimization (PMO) method which improves the convergence of iterative manifold optimization by reformulating the optimization problem to directly incorporate the manifold retraction into the gradient calculation. This reformulation alleviates the constraint on update step size required by previous methods due to their reliance on the tangent space.

In this chapter, we first provide the reader with the definitions of several important manifold structures. We then outline operations that can be performed on specific manifolds. After providing the theoretical background and a review of previous works in the field of manifold optimization, we propose our PMO method for optimizing over the Stiefel and Grassmann manifold. Manifold optimization methods form the core of three out of the four areas of application in this work: dimensionality reduction, improving learned feature embeddings, and network compression.

## 3.1 Mathematical Definitions

Prior to discussing the field of manifold optimization and our specific contributions, we provide a review of some of the core theory. The works of [2] and [9] provide a more detailed theoretical background on the field of optimization over matrix manifolds. We start with the question: what are manifolds and how are they used? A $d$-dimensional manifold is a set $\mathcal{M}$ that is fully covered by a set of charts that form one to one correspondences between points in the set $\mathcal{M}$ and elements in the $d$-dimensional real space $\mathbb{R}^d$. A more formal definition is supplied in [2]. A manifold $\mathcal{M}$ is a set in a couple $(\mathcal{M}, \mathcal{A}^+)$ where $\mathcal{A}^+$ is a maximal atlas of the set $\mathcal{M}$ into the $d$-dimensional real space $\mathbb{R}^d$. An atlas $\mathcal{A}$ is a set of charts, $(\mathcal{U}_a, \varphi_a)$, that cover the entire manifold, $\bigcup_a \mathcal{U}_a = \mathcal{M}$, and overlap smoothly for any point $x \in \mathcal{U}_a \cap \mathcal{U}_b$. A chart $(\mathcal{U}, \varphi)$ is a bijection, one-to-one mapping, of the each of the points in the set $\mathcal{U} \in \mathcal{M}$ to points in the $d$-dimensional real space $\mathbb{R}^d$. Given a chart $(\mathcal{U}, \varphi)$ and a point $x \in \mathcal{U}$, the bijection of $\varphi(x)$ maps the point to a coordinate in the real space, $\varphi(x) \in \mathbb{R}^d$. The manifold $\mathcal{M}$ is a smooth manifold if all charts in its atlas are infinitely differentiable over its own set. This is why the atlas is sometimes referred to as the differentiable structure of $\mathcal{M}$.

A $d$-dimensional vector space $\bar{\mathbf{v}}$ with basis $(v_1, v_2, ...v_d)$ for $i \in [1, ...d]$ is a manifold. The manifold structure of $\bar{\mathbf{v}}$ is given by the fact that a chart $\varphi$ exists for the set such that all points in $\bar{\mathbf{v}}$ are mapped to $\mathbb{R}^d$. This is most easily defined based on the inverse of mapping $\varphi^{-1} = x = \sum_{i=1}^{d} x^i v_i$. The manifolds formed by linear subspaces are a special type of manifold, called a linear manifold.

It can be shown that the set of real valued matrices $\mathbb{R}^{m \times p}$ form a $mp$-dimensional manifold, as the column vector of the matrix are the basis for the vector space $\varepsilon$. A chart for this manifold that maps from $\mathbb{R}^{m \times p}$ to $\mathbb{R}^{mp}$ is defined as the vectorization of the matrix, i.e. stacking each of the $m$ columns of length $p$ to form a single vector of length $mp$. This manifold of real-valued matrices is an important manifold for this work as it forms the set on which other submanifolds are defined. Another important property of the real matrix manifold is that it forms a Euclidean space with the traditional inner product given as $\langle \mathbf{X}, \mathbf{Y} \rangle = \varphi(\mathbf{X})^T \varphi(\mathbf{Y})$. This inner product can also be calculated using the trace of the transposed matrix multiplication as $\langle \mathbf{X}, \mathbf{Y} \rangle = \mathbf{tr}(\mathbf{X}^T \mathbf{Y})$. This inner product induces the traditional Frobenious norm $\|\mathbf{X}\|_F^2 = \mathbf{tr}(\mathbf{X}^T \mathbf{X})$. For the remainder of this work the manifold of real valued matrices is implied whenever we refer to the matrix manifold.

There are two categories of sub-manifolds of the real matrix manifold that are pertinent to this work: embedded manifolds and quotient manifolds. Embedded matrix manifolds are formulated as the subset of a set of real $(m \times p)$ matrices, $\mathbb{R}^{m \times p}$, where $1 \leq p \leq n$, and the matrices satisfy an explicit constraint. An example of an embedded manifold in the $\mathbb{R}^{m \times 1}$, is the unit sphere $S^{m-1}$, which contains all column vectors with unit norm i.e. if $\mathbf{X} \in S^{m-1}$ then $\mathbf{X}^T \mathbf{X} = 1$.

Quotient manifolds differ from embedding manifolds in that they are defined by the set of equivalence classes of $(m \times p)$ matrices. An example of a quotient manifold is the set of orthogonal $p$-dimensional subspaces in $\mathbb{R}^m$, this is known as the Grassmann manifold [44], denoted as $\mathcal{G}^{m\times p}$. The Grassmann manifold will be discussed in more detail later in this section. However, the important takeaway here is that if an $(m \times p)$ matrix $\mathbf{X}$ exists on the Grassmann manifold $\mathcal{G}^{m\times p}$, then any matrix resulting from the right multiplication by an orthonormal $(p \times p)$ matrix $\mathbf{Q} \in \mathcal{O}^{p\times p}$, exists on the same point in the Grassmann manifold. These are equivalent representations of the point on the Grassmann manifold. It is important to highlight the fact that because quotient manifolds represent equivalence classes, there is not an inherent singular matrix representation of each point on the manifold. Instead, any arbitrary matrix belonging to the given subspace on the manifold can be used to represent the point on the manifold. The results of matrix algebra operations performed on the representative matrix are equivalent regardless of which matrix was selected to represent the subspace.

The first important submanifold of the matrix manifold is the Orthogonal matrix manifold. The Orthogonal manifold $\mathcal{O}^{m\times m}$ is formed by the subset of real $m \times m$ matrices that satisfy the $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_m$. The matrices that make up $\mathcal{O}^{m\times m}$ have independent columns with unit norms.

Another important manifold is the Stiefel manifold, $\mathcal{S}^{m\times p}$. The Stiefel manifold is an embedded submanifold in the matrix manifold $\mathbb{R}^{m\times p}$ where all points on the manifold, $\mathbf{X} \in \mathcal{S}^{m\times p}$ satisfy $\mathbf{X}^T\mathbf{X} = \mathbf{I}_p$. Using this definition the Stiefel manifold is defined as an embedded submanifold in the matrix manifold which is itself an embedded manifold in $mp$-dimensional Euclidean space $\mathbb{R}^{mp}$. However, some embedded manifolds can also be defined in terms of quotient spaces. The benefit to defining a manifold as quotient spaces of well defined manifolds leads to simpler formulae for operations on the manifold. Both Stiefel $\mathcal{S}^{m\times p}$ and Grassmann manifolds $\mathcal{G}^{m\times p}$ can be defined as a quotient manifold of the Orthogonal manifold $\mathcal{O}^{m\times m}$. This can be done by representing the set of equivalency classes for $\mathcal{S}^{m\times p}$ in $\mathcal{O}^{m\times m}$ as the set $[\mathbf{X}]$ where $\mathbf{X} \in \mathcal{O}^{m\times m}$ and $\mathbf{Q_{n-p}} \in \mathcal{O}^{n-p\times n-p}$ then:

$$[X] = \begin{pmatrix} \mathbf{I_p} & \mathbf{0_{n-p}} \\ \mathbf{0_p} & \mathbf{Q_{n-p}} \end{pmatrix} \tag{3.1}$$

An equivalent expression to this equation is to say that the set of points on the orthogonal manifold $\mathcal{O}^{m\times m}$ form an equivalency class for $\mathcal{S}^{m\times p}$ if their first $p$ columns are the same. It should be noted that we provide the quotient space definition of Stiefel manifold for convenience to the reader, however, in the remainder of this work we will generally refer to the Stiefel manifold as an embedded manifold in Euclidean space.

A full derivation of the operations based on the quotient space definition of the Stiefel manifold is provided in [2,9,31].

The Grassmann manifold $\mathcal{G}^{m \times p}$ is a quotient manifold because it defines a set of equivalence classes, the $p$-dimensional subspaces in $\mathbb{R}^m$. Unlike the Stiefel manifold, a point on the Grassmann manifold can be be represented as any matrix whose columns span the subspace. Points on the Grassmann manifold are invariant to the ordering of the basis vectors, as long as they span the same subspace. More concretely two $(m \times p)$ matrices $\mathbf{X}$ and $\mathbf{Y}$ represent the same point on the Grassmann manifold $\mathcal{G}^{m \times p}$ if there exists an orthogonal square matrix $\mathbf{Q} \in \mathcal{O}^p$ that satisfies $\mathbf{X} = \mathbf{YQ}$.

In the next section we provide important definitions of geometric structures and operations on the Stiefel and Grassmann manifolds.

## 3.2 Manifold Operations

In this section we discuss geometries of manifolds and operations on manifolds that are pertinent to this work. The aim of this section is to simply introduce the concepts, but we do not provide their formal derivations.

The first important geometry of manifolds is the concept of the tangent space of a manifold $\mathcal{M}$ at point $\mathbf{Y} \in \mathcal{M}$. In order to derive the tangent space for an embedded submanifold of the Euclidean space, one must only take the derivative of the constraint of the manifold. In the case of the Stiefel manifold, the equation of its constraint is $\mathbf{Y}^T\mathbf{Y} = \mathbf{I}_p$, and its derivative can be calculated using the chain-rule as $\mathbf{Y}^T\dot{\mathbf{Y}} + \dot{\mathbf{Y}}^T\mathbf{Y} = \mathbf{0_p}$. The tangent space for the Stiefel manifold at point $\mathbf{Y}$ can be defined as the set of points $\mathbf{X}$ that satisfy $\mathbf{Y}^T\mathbf{X} + \mathbf{X}^T\mathbf{Y} = \mathbf{0_p}$.

We next define the normal space for a manifold as the space of points orthogonal to the tangent space, that is the set of points whose inner product with all points in the tangent space is 0. The inner product of two points in Euclidean space, $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{m \times p}$ is the Frobenius norm given as $g(\mathbf{X}, \mathbf{Y}) = tr(\mathbf{X}^T\mathbf{Y})$. The normal space of the manifold at point $\mathbf{Y}$ consists of all points $\mathbf{N}$ that have inner product of 0 with any element of the tangent space of the manifold at, $\mathbf{T}$, $tr(\mathbf{T}^T\mathbf{N}) = 0$.

One of the most important geometrical structures on manifolds is the concept of the geodesic. The geodesic of a manifold generalizes the concept of lines in Euclidean space, in that they represent the shortest length between two points on the manifold. The geodesic can also be described based on its direction in the tangent space at a given point on a manifold.

The tangent directional description of the geodesic on a manifold gives rise to the important concept of parallel transport of vectors in tangent space of points along a manifold. This concept follows from the idea of parallel translation of tangent space in Euclidean space, i.e. a tangent vector is transported along a line by moving the tail of the vector to the location on the line. However, this transport of tangent vectors does not hold for motion along the geodesic of a manifold. A vector that exists in the tangent space of one point on a manifold is not guaranteed to exist in the tangent space at a different location on the geodesic. It is important to define such parallel transport of vectors in the tangent space of a point along a geodesic, even if only for transporting the vector defining a geodesic to the tangent spaces of the points on a geodesic. Assuming the distance between points on the geodesic is not too large, the parallel transport of vectors in the tangent space can be performed by orthogonally projecting the point to the tangent space by removing the normal component of the point using:

$$\pi_{T,Y}(Z) = \mathbf{Y}\frac{1}{2}(\mathbf{Y}^T\mathbf{Z} - \mathbf{Z}^T\mathbf{Y}) + (\mathbf{I}_m - \mathbf{Y}\mathbf{Y}^T)\mathbf{Z} \tag{3.2}$$

Another important operation is the retraction of an arbitrary point in Euclidean space to a location on the manifold. There are many different methods for retracting a point on matrix manifolds [3]. In this case we use the one that minimizes the Frobenius norm between the point $\mathbf{Z} \in \mathbb{R}^{m\times p}$ in Euclidean space and its retraction $r_{\mathcal{S}}(\mathbf{Z}) \in \mathcal{S}^{m\times p}$. This retraction can be performed through the use of the singular value decomposition of $\mathbf{Z} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. The retraction then takes the form:

$$r_{\mathcal{S}}(\mathbf{Z}) = \mathbf{U}\mathbf{V}^T \tag{3.3}$$

This retraction method on the Stiefel, and inherently the Grassmann, manifold forms the basis for much of the literature for optimization on orthogonal matrix manifolds, where matrices are updated in unconstrained Euclidean space and then retracted back to the manifold.

Many manifold optimization techniques rely on the computation of the Gradient of function $F(\cdot)$ defined over the manifold, as $\nabla F$. The most common method to calculate the gradient direction of a loss function defined on a manifold is to calculate the partial derivatives of the loss function with respect to the components of the point $\mathbf{Y}$ as $\frac{\partial F}{\partial \mathbf{Y}_{i,j}}$ where $\mathbf{Y}_{i,j}$ is the $(i, j)$ elements of the matrix representation of the point $\mathbf{Y}$. The gradient is then orthogonally projected onto the tangent space of the manifold at $\mathbf{Y}$ to provide a direction in the tangent space that minimizes the function $F(\cdot)$ at the point $\mathbf{Y} \in \mathcal{S}^{m\times p}$ as:

$$\nabla F = \frac{\partial F}{\partial \mathbf{Y}} - \mathbf{Y}(\frac{\partial F}{\partial \mathbf{Y}})^T \mathbf{Y}^T \tag{3.4}$$

This is an important formula as it defines the direction in the tangent space at point $\mathbf{Y}$ that minimizes the loss of the function $F(\cdot)$. Thus, assuming the loss function is smooth, Equation (3.4) defines the loss minimizing geodesic along the manifold emanating from point $\mathbf{Y}$.

## 3.3 Proposed Proxy Matrix Manifold Optimization

In this section, we outline methods for iterative optimization of a function which is defined over either a Stiefel or Grassmann manifold. Here we discuss the design decisions made during the development of our method. We start by describing the simplest formulation and grow its complexity in order to improve its robustness and convergence. We first describe One-Step retraction, which is simple to implement but does not have a theoretical proof of convergence. We then describe the Two-Step retraction method, which does have a theoretical proof of convergence. Finally we outline our proposed proxy matrix manifold optimization method. All of these methods are implemented using Stochastic Gradient Descent due to its demonstrated history of robust optimization.

The aim of both the One-Step and Two-Step iterative optimization methods is to find a solution, $\mathbf{X}^*$ for the following problem:

$$\mathbf{X}^* = \underset{\mathbf{X}\in\mathcal{G}^{m\times p}}{\arg\min}\, F(\mathbf{X}) \tag{3.5}$$

Where $F(\mathbf{X})$ is some scalar loss function for the point $\mathbf{X} \in \mathcal{G}^{m\times p}$.

### 3.3.1 One-Step iterative manifold optimization

Let us begin with the simplest of the methods for iterative manifold optimization, One-Step retraction, a method based on a concept originally developed in [7]. A visual representation of the One-Step retraction method is shown in Figure 3.1. The core concept of the One-Step retraction method is the idea that any arbitrary point in $\mathbb{R}^{m\times p}$ can be retracted to the closest point on the manifold using Equation (3.3).

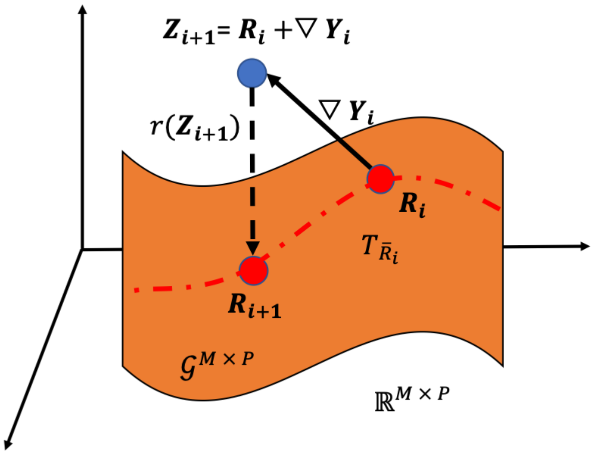

$Z_{i+1}= R_i +\nabla Y_i$
$\nabla Y_i$
$r(Z_{i+1})$
$R_i$
$T_{\bar{R}_i}$
$R_{i+1}$
$\mathcal{G}^{M \times P}$
$\mathbb{R}^{M \times P}$

Figure 3.1: Visual depiction of the One-Step retraction process. The point is first updated based on the gradients calculated through backpropagation. Then the point is retracted directly to the closest point on the manifold. This process is repeated until some stopping criteria is met.

The first step in the iterative process is to retract an initial point in ambient Euclidean space, $\mathbb{R}^{m \times p}$, to the manifold using Equation (3.3). Once the point is on the manifold, the gradient of $\mathbf{R_i}$ are calculated and used to update $\mathbf{R_i}$, shown as $\nabla Y_i$. This step will likely take the point out of the manifold, thus it must be retracted again to the manifold using Equation (3.3). Once the point is returned to the manifold, the process is repeated until a desired stopping criteria is met. In our implementation we ran for a set number of iterations with a decreasing learning rate to ensure relative convergence. We algorithm for the One-Step manifold optimization method is provided in Algorithm 1 . Though this method is easy to implement and in practice it does converge, there is no convergence proof, as the point's direct retraction is not guaranteed to fall on the geodesic defined by the direction of the projection of the gradients to the tangent space. We will highlight the reason for the lack of provable convergence for this method next, when we discuss the convergence proof of the Two-Step projection then retraction method.

```
Data: $\mathbf{X}$
Result: Locally Optimal $\mathbf{R}$ for $f_{\mathbf{X}}$
initialize $\mathbf{R} \in \mathcal{G}^{m \times p}$;
for $i > iter$ do
    $Y_i = f_{\mathbf{X}}(\mathbf{R_i})$;                    /* Calculate loss for $\mathbf{R_i}$ */
    $\mathbf{Z_{i+1}} = \mathbf{R_i} - \beta \nabla Y_i$;       /* Calculate update $\mathbf{Z_{i+1}}$ */
    $\mathbf{USV}^T = \mathbf{Z_{i+i}}$;                  /* Retract $\mathbf{Z_{i+1}}$ to $\mathcal{G}^{m \times p}$ */
    $\mathbf{R_{i+1}} = \mathbf{UV}^T$;            /* Set $\mathbf{R_{i+1}}$ to retracted $\mathbf{Z_{i+1}}$ */
end
```

**Algorithm 1:** One-Step iterative Grassmann manifold optimization algorithm

### 3.3.2 Two-step iterative manifold optimization

The Two-Step projection retraction method is a modification on the One-Step retraction method that guarantees convergence. A visual depiction of this method is shown in Figure 3.2. Like the One-Step retraction method, the Two-Step method relies on retraction of the gradients in the ambient Euclidean space to the manifold. However, unlike the One-Step, the gradients are first orthogonally projected to the local tangent space of the manifold $\mathcal{T}_{R_i}$, shown as the green plane in Figure 3.2. Then retraction to the manifold is performed on the projection of the gradients to the tangent space. This Two-Step procedure is performed to ensure convergence to a minimum of the loss function. The algorithm for the Two-Step manifold optimization method is provided in Algorithm 2

The Two-Step method is guaranteed to converge based on the following. The first important fact is that the gradients calculated in Euclidean space can be decomposed into two components: the first component captures gradients corresponding to the normal space to the manifold, and the second corresponds the to tangential space to the manifold. Therefore, the gradients can be projected onto the tangent space of the manifold using Equation (3.2). It is important to note that because the tangential and normal components of the gradients are orthogonal, removing one will not impact the other. Because, by definition, the gradients are along the direction that minimizes the loss in ambient Euclidean space, if any tangential component of the gradients exist, they indicate a loss minimizing direction on their own. By removing just the normal components of the gradients, the projection of the gradient indicates a loss minimizing direction in the tangent space based on the loss minimizing direction in the ambient space. Once the gradients are projected to the tangent space, generating a loss reducing direction, the gradients can be retracted to the manifold, with the guarantee that the closest point on the manifold will lay on the loss-reducing geodesic described by the current location on the manifold and the projected gradients on the tangent space, presuming the step is not too large.

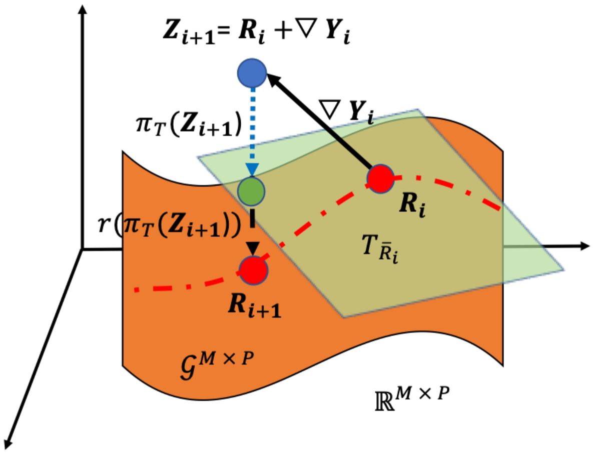

$Z_{i+1}= R_i +\nabla Y_i$
$\nabla Y_i$
$\pi_T(Z_{i+1})$
$r(\pi_T(Z_{i+1}))$
$R_i$
$T_{\bar{R}_i}$
$R_{i+1}$
$\mathcal{G}^{M \times P}$
$\mathbb{R}^{M \times P}$

Figure 3.2: Visual depiction of the Two-Step retraction process. The point is first updated based on the gradients calculated through backpropagation. Then the point is first projected to the tangent-space (green plane) at $\mathbf{R_i}$ using Equation (3.2), the projection is shown as the dotted blue line. Then the point is retracted from the tangent space to the closest point on the manifold using Equation (3.3). This process is repeated until some stopping criteria is met.

Unlike the One-Step retraction approach, the Two-Step method is guaranteed to converge because the retraction of the projected gradients is guaranteed to lay on the loss minimizing geodesic. Though it is likely that the updated locations on the manifold resulting from each method are quite similar, there is no guarantee of convergence if the normal components of the gradients are not first removed prior to retraction.

```
Data: $\mathbf{X}$
Result: Locally Optimal $\mathbf{R}$ for $f_{\mathbf{X}}$
initialize $\mathbf{R} \in \mathcal{G}^{m\times p}$;
for $i > iter$ do
    $Y_i = f_{\mathbf{X}}(\mathbf{R_i})$;                    /* Calculate loss for $\mathbf{R_i}$ */
    ;          /* Project gradients to tangent space */
    $\pi_T\big(\beta\nabla Y_i\big) = R_i\frac{1}{2}\big(\mathbf{R_i}^T(\beta\nabla Y_i) - (\beta\nabla Y_i)^T\mathbf{R_i}\big) + \big(\mathbf{I}_m - \mathbf{R_i}\mathbf{R_i}^T\big)(\beta\nabla Y_i)$;
    ;          /* Retract from tangent space to $\mathcal{G}^{m\times p}$ */
    $\mathbf{USV}^T = \pi_T\big(\beta\nabla Y_i\big)$;
    $r\Big(\pi_T\big(\beta\nabla Y_i\big)\Big) = \mathbf{UV}^T$;
    $\mathbf{R_{i+1}} = \mathbf{R_i} - r\Big(\pi_T\big(\beta\nabla Y_i\big)\Big)$;                  /* Update R */
end
```

**Algorithm 2:** Two-step iterative Grassmann manifold optimization algorithm

### 3.3.3 Proxy Matrix Optimization

We now present one of the key contributions of this work, the Proxy Matrix Optimization (PMO) for optimization over orthogonal matrix manifolds. We first provide the formulation for PMO, and then prove that PMO converges to a minimum for the loss function. After the proof of convergence, we highlight the differences between the Two-Step method and PMO, and derive the computational complexity. Following our complexity derivation, we provide some experimental results and discussion which highlight the difference between the iterative manifold optimization method discussed in this chapter.

In PMO the approach to optimization is flipped from the intuitive two-step approach. Instead of restricting the search to the local region of the manifold, PMO optimizes in ambient space such that each update moves the point in a direction that its mapping to the manifold reduces the loss. More formally the optimization problem in PMO is to find the optimal $\mathbf{X}^*$ by solving:

$$\mathbf{X}^* \operatorname*{arg\,min}_{X\in\mathbb{R}^{m\times p}} F(\mathbf{UV}^T) \tag{3.6}$$

Subject to:

$$\mathbf{X} = \mathbf{U\Sigma V}^T \tag{3.7}$$

Where $\mathbf{X} = \mathbf{U\Sigma V}^T$ is the singular value decomposition of $\mathbf{X}$.

This formulation of the manifold optimization alters the problem so that the point being optimized is not restricted to the manifold. In the Two-Step optimization method this restriction is enforced by projecting the ambient gradients to the tangent space, then retracting to the manifold. The PMO method embeds the manifold retraction inside the optimization function, instead of performing retraction after optimization. Therefore the optimizer is no longer limited in the degree that it can move at each step. The Two-Step method requires that the point remains in the region of the manifold where the tangent-space is still applicable. This restriction is to ensure the retraction to the manifold will be along the geodesic described by the projection of the ambient gradients at the current location. This unconstrained optimization allows for greater steps in ambient space, and therefore faster convergence and a lower likelihood of being stuck at local minima.

It is important to point out the difference between the gradients resulting from optimizing the Two-Step formulation, Equation (3.5), and the gradients resulting from optimizing the PMO formulation, Equation (3.6). The Two-Step gradients, once projected to the tangent space of the manifold, represent the direction along the manifold that minimizes the loss. The PMO gradients represent the unrestricted direction in $\mathbb{R}^{m \times p}$ that will minimize the loss of direct retraction of the point $\mathbf{X}$ to the manifold. A visual representation of the PMO method is given in Figure 3.3.

Unlike other iterative optimization and retraction methods, the Proxy Matrix Optimization method does not aim to directly optimize a matrix on the manifold. Instead PMO uses an auxiliary, or Proxy Matrix, that exists in the ambient Euclidean space and is retracted the closest location on the manifold using Equation (3.3). We depict the PMO process in Figure 3.3, and present the algorithm for PMO in Algorithm 3. The first step in the PMO process is to retract the proxy matrix, $\mathbf{P}_i$ to its closest location on the manifold, $\mathbf{Y}_i$. Once the proxy matrix is retracted to the manifold, the loss is calculated based on the loss function at $\mathbf{Y}_i$. This loss is then backpropagated through the singular value decomposition of proxy matrix using a method developed by [66]. Unlike backpropagation through standard layers of neural networks, such as convolution and fully connected layers, backpropagation through structured layers is significantly more complex.

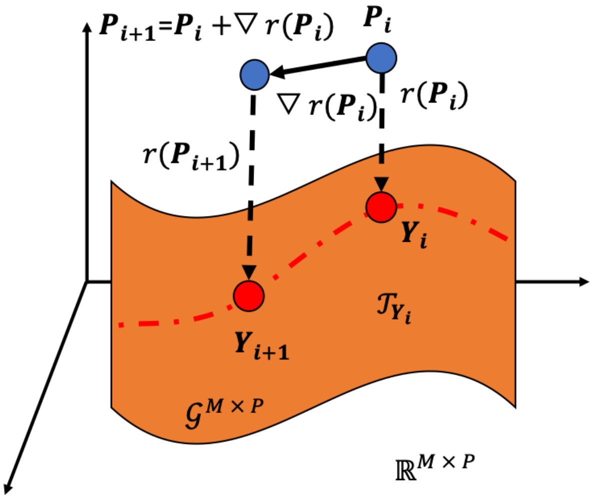


Figure 3.3: Visual depiction of the Proxy Matrix Optimization process. The proxy matrix, $\mathbf{P}_i$, is first retracted to the manifold using Equation (3.3). Then the gradients $\nabla R(\mathbf{P}_i)$ in ambient space, to move the proxy matrix in a direction that minimize the loss of its retraction. These gradients are used to move the proxy matrix in Euclidean space in the direction that minimizes the loss.

```
Data: $\mathbf{X}$
Result: Locally Optimal $\mathbf{P}$ such that $r\big(\mathbf{P_i}\big)$ minimizes the loss $f_\mathbf{X}$
initialize $\mathbf{P} \in \mathbb{R}^{m\times p}$ and $\mathbf{P} \notin \mathcal{G}^{m\times p}$;
for $i > iter$ do
    $\mathbf{USV}^T = \mathbf{P_i}$;                                    /* Retract $\mathbf{P_i}$ to $\mathcal{G}^{m\times p}$ */
    $r\big(\mathbf{P_i}\big) = \mathbf{UV}^T$;
    $\nabla\mathbf{r}\big(\mathbf{P_i}\big) = \frac{\delta}{\delta\mathbf{P_i}} f_\mathbf{X}\Big(r\big(\mathbf{P_i}\big)\Big)$;   /* Calculate gradients for $\mathbf{P_i}$
     */
    $\mathbf{P_{i+1}} = \mathbf{P_i} - \beta\nabla\mathbf{r}\big(\mathbf{P_i}\big)$;                          /* Update $\mathbf{P}$ */
end
```

**Algorithm 3:** Proxy-Matrix iterative Grassmann manifold optimization algorithm.

The formulation for calculating the partial derivatives through SVD was only recently proposed by Ionescu et al. in [66]. We will provide a brief description of the formulation for backpropagation through SVD, but a full derivation is provided in [66]. That work combines the concept of a variation, from the field of calculus of variations, with partial derivatives using Taylor expansion, where the variation of the function $f(\cdot)$, which takes as input the matrix $\mathbf{X}$ and outputs a matrix $\mathbf{Y}$ as $f(\mathbf{X}) = \mathbf{Y}$, is given as:

$$d\mathbf{Y} = df(\mathbf{X}; d\mathbf{X}) = f(\mathbf{X} + d\mathbf{X}) - f(\mathbf{X}) \tag{3.8}$$

The chain rule can then be derived by forcing the first order terms from the Taylor expansion of both sides to match:

$$\frac{\partial L \circ f}{\partial \mathbf{X}} : d\mathbf{X} = \frac{\partial L}{\partial \mathbf{Y}} : d\mathbf{Y} \tag{3.9}$$

Where $L$ is a function that maps to real numbers and $\mathbf{A} : \mathbf{B} = \mathbf{tr}\big(\mathbf{A}^T\mathbf{B}\big)$. Ionescu et al. use this identity to derive the expression of the partial derivative that we are interested in $\frac{\partial L \circ f}{\partial \mathbf{X}}$ as a function of the right hand side. To do this they use a two-step procedure: They first derive the variations $d\mathbf{Y}$, with respect to the variations of the inputs, using the identity in Equation (3.8). Then, given $d\mathbf{Y}$, the chain rule can be used, based on the identity in Equation (3.9), to obtain the partial derivatives of the loss with respect to $\mathbf{X}$.

For completeness we provide the derived equation for backpropagation of the loss to calculate the partial derivatives of the loss with respect to the proxy matrix $\mathbf{P}$ as derived by Ionescu et al.

$$\frac{\partial F}{\partial \mathbf{P}} = \mathbf{D}\mathbf{V}^T + \mathbf{U}\Big(-\mathbf{U}^T\mathbf{D}\Big)_{diag}\mathbf{V}^T + 2\mathbf{U}\mathbf{\Sigma}\left(\mathbf{K}^T \circ \left(\mathbf{V}^T\Big(\frac{\partial F}{\partial \mathbf{V}} - \mathbf{V}\mathbf{D}^T\mathbf{U}\mathbf{\Sigma}\Big)\right)\right)_{sym}\mathbf{V}^T \tag{3.10}$$

where $\mathbf{A}_{sym}$ is the symmetric part of matrix $\mathbf{A}$ given as $\mathbf{A}_{sym} = \frac{1}{2}(\mathbf{A}^T + \mathbf{A})$, and $\mathbf{A}_{diag}$ the diagonalization operator on A, where all of elements on the main diagonal are the same as $\mathbf{A}$ and all elements off the main diagonal are set to 0. More formally the elements of $\mathbf{A}_{diag}$ are given as:

$$\mathbf{A}_{i,j} = \begin{cases} \mathbf{A}_{i,j}, & i = j \\ 0, & i \neq j \end{cases} \tag{3.11}$$

The formulation for the matrix $\mathbf{D}$ is given as:

$$D = \Big(\frac{\partial F}{\partial \mathbf{U}}\Big)_1 \mathbf{\Sigma}_n^{-1} - \mathbf{U}_2\Big(\frac{\partial F}{\partial \mathbf{U}}\Big)_2^T \mathbf{U}_1\mathbf{\Sigma}_n^{-1} \tag{3.12}$$

The elements of the matrix $\mathbf{K}$ are given as:

$$\mathbf{K}_{i,j} = \begin{cases} \frac{1}{\Sigma_{ii}^2 - \Sigma_{jj}^2}, & i \neq j \\ 0, & i = j \end{cases} \tag{3.13}$$

where $\Sigma_{ii}$ is the $(i, i)$ element of the matrix $\mathbf{\Sigma}$.

It is important to note that because the retracted point on the manifold does not depend on the matrix $\mathbf{\Sigma}$, the partial of the loss with respect to the proxy matrix does not have any component associated with the partial of the loss with respect to $\mathbf{\Sigma}$.

We now provide a theoretical proof that PMO converges to a minimum. Unlike the proof of convergence for the Two-Step solution, the proof of convergence of PMO is comparably straight forward. Because the invariant of $\mathbf{U}$ and $\mathbf{V}$ is preserved in the variations $d\mathbf{U}$ and $d\mathbf{V}$, namely that they satisfy the orthogonality constraints:

$$\mathbf{U}^T d\mathbf{U} + d\mathbf{U}^T \mathbf{U} = 0 \tag{3.14}$$

and:

$$\mathbf{V}^T d\mathbf{V} + d\mathbf{V}^T \mathbf{V} = 0 \tag{3.15}$$

Then it follows that the update of the gradient of $\mathbf{X}$ is the direction that minimizes the loss function based on maintaining the invariants of $\mathbf{U}$ and $\mathbf{V}$. Because of the retraction to the manifold, Equation (3.3), is embedded in the loss function, Equation (3.6), the gradient of the loss function will move $\mathbf{X}$ in the direction that minimizes the loss, assuming the step size is not too large.

## 3.4 Experimental Results

In this section we present our experimental results comparing the iterative manifold optimization techniques based on Stochastic Gradient Descent. The proposed PMO method is used heavily throughout the remainder of this work, so the aim of this section is to highlight its improvement over current manifold optimization methods based on SGD. We focus on the Two-Step and PMO methods as they have theoretical guarantees of convergence, where as the One-Step method does not.

### 3.4.1 Learning Rate Convergence Analysis

The first step in our examination of the behavior of the Two-Step and PMO manifold optimization methods is explore the impact different learning rates have on convergence. This step is important for ensuring the optimal convergence behavior with the fewest number of iterations. This is particularly true for the Two-Step method, as it relies on the on the assumption that the retraction of the projected point on the tangent plane falling on the geodesic of the gradient. If the update step is too large for the

Two-Step method the retraction from the tangent space may not fall on the geodesic, leading to divergent behavior.

In order to explore the behaviors of the Two-Step and PMO methods we ran multiple sets of experiments with different loss functions and varied the learning rate for each optimizer. Though the differences between the Two-Step and PMO methods were very pronounced, we found that each method behaved consistently across the different loss functions. This suggests that a single learning rate can be used irrespective of changes in the function that is optimized. Though we did find it is important to normalize the loss function so that the magnitude of the loss is of the same scale. In our work we found that normalizing the magnitudes of the loss so that it was in the range of $\pm 1.0$ worked well, and alleviated the need for finding new learning rates for different loss functions.

To highlight the behavior the two methods with different loss functions we selected the $L_1$-PCA problem as an example optimization problem. The details of $L_1$-PCA and the dataset are highlighted in Chapter 4 but all of the various losses we used demonstrated similar behavior for the different learning rates. For these experiments we extracted the top 10 principal components from a dataset consisting of 400 25-dimensional vectors. In all of these experiments we use a learning rate schedule that reduced the learning rate by an order of magnitude at 50% and again at 75% through training. We provide example plots of the loss over the course of training with the Two-Step and PMO methods in Figures 3.4 and 3.5, respectively.

The plot in Figure 3.4 shows the significant impact the learning rate has on the Two-Step method. If the learning rate is too-high the optimizer takes excessively large steps and fails to converge, depicted by the Red and Blue plots in Figure 3.4. This verifies a key requirement in the proof of convergence for the Two-Step method: the step size must be small enough that the update is in the local region of the manifold where the tangent space is still representative. However, if the loss function is too small the optimizer will fail to converge to the global minimum, depicted by the Black plot in Figure 3.4. We found that a learning rate of 0.1 generally assured convergence to the global minimum, shown as the green plot in Figure 3.4.

There is a drawback to using lower learning rates for the Two-Step method: very slow convergence. The Two-Step method with a learning rate of 0.1 took over 3000 epochs to plateau. An interesting behavior that can be seen from the Blue plot (learning rate of 1.0) in Figure 3.4 is that even though the learning rate is too large for convergence, the optimizer is pushed in the direction of the minima and therefore converges faster once the learning rate is reduced. We decided to leverage this behavior

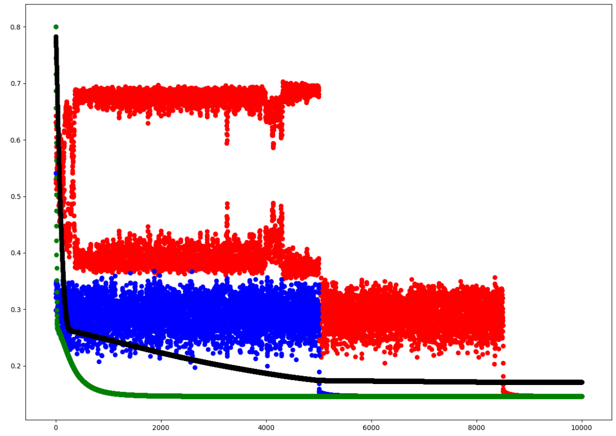


Figure 3.4: Example behavior of the loss when varying the learning rate using the Two-Step retraction method. The learning rates used were: Red 10.0, Blue 1.0, Green 0.1, Black 0.01.

to accelerate the convergence of the Two-Step method using a customized learning rate schedule. We initially train with a high learning rate of 1.0 for the first 5% of training, then the learning rate is reduced by an order of magnitude at 60% and 85% through training. We found that this configuration allowed for the optimal convergence of the Two-Step method. Even with the optimized learning rate schedule, the Two-step method often takes over 2000 epochs to converge.

The behavior of the Proxy-Matrix Optimizer with various learning rates is plotted in Figure 3.5. It should be noted that the scale of the horizontal axis in Figure 3.5 is an order of magnitude smaller than that in Figure 3.4. The first observation that is striking from the plot is the quick convergence of all methods, where all are reaching a plateau before 400 epochs. It is hard to visually see the differences between the different learning rates but we found slight numerical differences in the final results for each learning rate. Our experiments show that a learning rate above 100.0, Blue in figure, did not allow the network the converge to a constant solution. A learning rate of 1.0, Black in figure, had slower convergence. We therefore selected a learning rate of 10.0 for the Proxy-Matrix Optimization method. As was done with the Two-Step

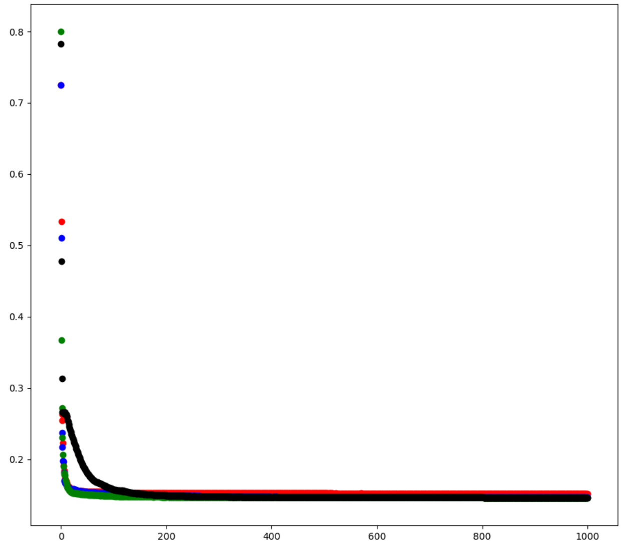


Figure 3.5: Example behavior of the loss when varying the learning rate using the Proxy-Matrix method. The learning rate was reduced by an order of magnitude at 50% and again at 75% through training.The learning rates used were: Red 1000.0, Blue 100.0, Green 10.0, Black 1.0.

method, we designed a learning rate schedule for PMO to accelerate convergence based on the behavior of demonstrated in the Figure 3.5. We found that PMO tends to make the majority of the reduction to the optimization loss with the largest learning rate, and only needs the learning rate to be reduced to minimize oscillation. Therefore we settled on a learning rate schedule for PMO where the learning rate was reduced by an order of magnitude at 80% and 90% through training.

### 3.4.2 Number of Iterations for Convergence

We next investigate how many epochs are required to train both the Two-Step and PMO methods until they converge to the global minima. We performed these experiments with various loss functions, however we will provide only a few informative results for the reprojection loss formulation of PCA with various norms. The utility of such optimization problems is discussed in Chapter 4.

In order to demonstrate the consistent improved convergences of the Proxy-Matrix method, we plot the convergence plots for optimization of three different variants of PCA in Figures 3.8, 3.7, and 3.6, where the $L_{0.5}$ $L_1$ and $L_2$ norms are optimized, respectively. To do this we trained each optimizer for a wide range of iterations, from 1 to 10000, and report the final reprojection accuracy they achieved relative to the optimal solution. For all experiments in this section the customized learning rate schedules were used for both methods.

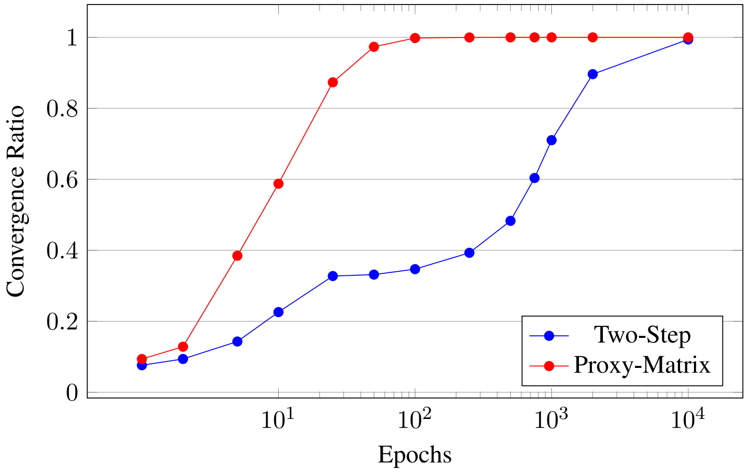


Figure 3.6: Convergence plot of Two-Step and Proxy Matrix methods, Blue and Red plots respectively, for $L_2$-PCA.

The PMO method demonstrates consistent convergence behavior and generally converges orders of magnitude faster than the Two-Step method. The faster convergence is important because it means PMO requires significantly fewer iterations without any loss in accuracy. The smooth degradation is also a favorable behavior, because it means that the accuracy of the result will be affected predictably if the number of iterations are reduced in order to accelerate the algorithm.

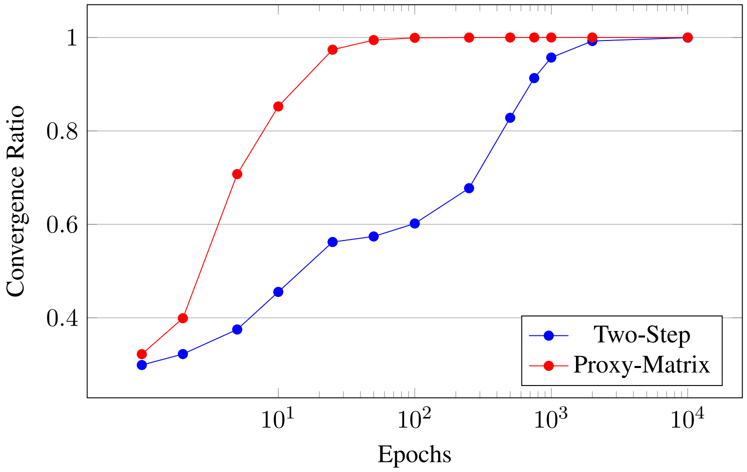


Figure 3.7: Convergence plot of Two-Step and Proxy Matrix methods, Blue and Red plots respectively, for $L_1$-PCA. (Note the horizontal axis is in log-scale.)

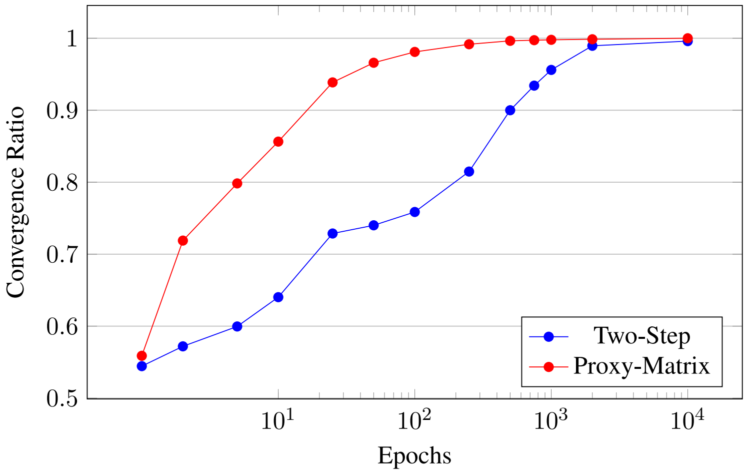


Figure 3.8: Convergence plot of Two-Step and Proxy Matrix methods, Blue and Red plots respectively, for $L_{0.5}$-PCA. (Note the horizontal axis is in log-scale.)

## 3.5 Implementation Details

We present a neural network-based implementation of Grassmann manifold optimization for dimensionality reduction based on our initial work in [102]. Many methods exist for optimization over the Grassmann manifold, but many have drawbacks including lack of flexibility or slow convergence. Many other methods for Grassmann manifold optimization require the partial derivative of the loss function either be explicitly provided or solved for using suboptimal numerical differentiation. These requirements limit the flexibility of the optimizer to be used for various loss functions. In contrast, our method leverages the auto-differentiation methods supplied in the PyTorch Deep Learning framework [108] to automatically calculate the gradients of the loss functions, allowing for much greater flexibility. The proposed approach leverages the PyTorch framework by reformulating the Grassmann manifold optimization problem into a neural network architecture. Our initial work in the field of manifold optimization was an implementation of the One-Step retraction method using TensorFlow Deep Learning framework [1] proposed GM-PCA method [102]. The GM-PCA method was based on the One-Step optimization method, so it did not have a theoretical proof of convergence, however we found experimental that it almost always did converge.

Since then, we reformulated our method into the Proxy-Matrix Optimization technique that proved to be greatly superior to our initial One-Step method. For PMO we have changed our deep learning framework to PyTorch [108], because of its increased flexibility. Another important factor in the framework change was that PyTorch provided a built in implementation for backpropgation through SVD. Backpropagation with Gradient Descent, and the mini-batch variant of Stochastic Gradient Descent, are the most widely used optimization methods in the field of deep learning. The robustness of SGD is primarily due to its ability to optimize over a large dataset and thus escape the local minima that can trap other optimization techniques [10].

# Chapter 4

# Dimensonality Reduction

Linear Dimensionality Reduction (LDR) is a broad range of methods which are widely used throughout the fields of machine learning and signal processing. Broadly speaking all LDR methods share a common goal, learning a low dimensional subspace that better captures the structure of the data. Different LDR methods focus on capturing different structures or features in the low dimensional representation of the data. In this section we discuss various LDR methods and how they can be reformulated to fit in a unified LDR framework which uses our proposed Proxy Matrix Optimization method.

## 4.1 Background

Linear Dimensionality Reduction methods are commonly used by many machine learning and signal processing methods to both reduce the processing time and extract the nominal data from noisy signals in high-dimensional data. Of particular interest in this dissertation is the application of different LDR methods to computer vision problems. Each LDR method has a unique optimization objective, $f_{\mathbf{X}}(\mathbf{R})$, based on the desired structure they aim to preserve or enhance in the low dimensional space. This optimization objective is used to search for a projection matrix $\mathbf{R} \in \mathbb{R}^{n \times p}$ which projects $n$-dimensional input data to a lower $p$-dimensional space such that it minimizes the objective loss. The input data is stored in an $(m \times n)$-dimensional matrix, $\mathbf{X} = [x_1, ...x_1]^T$, whose $m$ rows hold the $n$-dimensional data vectors. Thus projection of high-dimensional data, $\mathbf{X}$, to the low-dimensional space performed using to the projection matrix, $\mathbf{R}$, is given as:

$$\mathbf{Y} = \mathbf{X}\mathbf{R} \tag{4.1}$$

Where $\mathbf{Y}$ is the low-dimensional representation of the data, with dimensions $(m \times p)$. Some dimensionality reduction methods focus on optimizing projection matrix based on the low dimensionality representation, $\mathbf{Y}$. Others reproject the low-dimensional data back to the high-dimensional space using the following:

$$\dot{\mathbf{X}} = \mathbf{Y}\mathbf{R}^T = \mathbf{X}\mathbf{R}\mathbf{R}^T \tag{4.2}$$

This reprojection is often performed when the objective focuses on learning a representation of the data that is similar to the original input data.

For most LDR methods the projection matrix is often constrained to be an orthonormal matrix, i.e. it satisfies $\mathbf{R}^T\mathbf{R} = \mathbf{I_p}$ where $\mathbf{I_p}$ is the $(p \times p)$ identity matrix. If the matrix $\mathbf{R}$ conforms to this constraint, its columns are orthogonal basis vectors. The set of all orthogonal matrices belonging to $\mathbb{R}^{n \times p}$ is the well-studied Grassmann Manifold $\mathcal{G}^{n \times p}$ discussed in Chapter 3. It therefore follows that the general dimensionality reduction optimization problem can be given as:

$$\mathbf{R}^* = \underset{\mathbf{R} \in \mathcal{G}^{n \times p}}{\arg\min} f_{\mathbf{X}}(\mathbf{R}) \tag{4.3}$$

The field of linear dimensionality reduction is broad, and includes a huge variety of methods, each aimed at solving a particular problem with a unique optimization formulation. In this section, we present Linear Dimensionality Reduction methods that are pertinent to this work. We aim to provide both the theoretical basis for each method, as well as the formulation for their optimization objective based on Equation (4.3).

### 4.1.1 Grassmann Optimization for Linear Dimensionality Reduction

In [20], Cunningham and Ghahrammani reformulate Linear Dimensionality Reduction problems as variants of Grassmann manifold optimization with different loss functions. The authors demonstrate that, though particular heuristic optimal approaches may exist, identical results can be achieved by their Grassmann optimization approach. The heuristic approaches are sometimes more efficient, but they may not always be optimal. The benefit to a manifold optimization approach is two-fold: Firstly, it means that a single optimization framework can be used for a large number of LDR problems.

Secondly the Grassmann manifold optimization approach finds all components of the projection simultaneously.

Instead of the greedy approach that is typically taken for problems such as PCA, the authors propose an iterative two-step approach to optimization on the Grassmann manifold, outlined in Section 3.3.2. The first step minimizes the loss function, and the second step retracts the updated projection matrix $\mathbf{R}$ back to the Grassmann manifold. This work demonstrated that though it may not be the most efficient method for $L_2$-PCA, identical results to the traditional decomposition methods can be achieved with manifold optimization techniques. They went on to show that indeed greedy decomposition methods which are commonly used on similar dimensionality reduction methods are sub-optimal for the Linear Discriminant Analysis (LDA) problem.

This approach to Linear Dimensionality Reduction based on optimization on the Grassmann manifold serves as the inspiration for much of our work in the field of LDR using our proposed PMO manifold optimization technique.

### 4.1.2 $L_2$-norm Based PCA

A great deal of work has been done on $L_2$-norm based PCA methods [30, 110]. The $L_2$ variant of PCA is widely used because there is an efficient solution that is proven to be optimal. There are three equivalent formulations for $L_2$-PCA optimization $f_{\mathbf{X}}(\mathbf{R})$, given in Equations (4.4), (4.5), and (4.6).

The goal of the first optimization formulation of $L_2$-PCA, Equation (4.4), is to learn an orthogonal projection, $\mathbf{R}$, of a low-dimensional surrogate of the data, $\mathbf{S}$, such that the maximum amount of information in $\mathbf{X}$ is retained in $\mathbf{S}$. Where $\mathbf{X}$ is the $(m \times n)$ matrix composed $m$ data $n$-dimensional samples, $\mathbf{R}$ is a $(n \times p)$ orthogonal projection matrix, and $\mathbf{S}$ is a $(m \times p)$ matrix composed of the $p$-dimensional surrogates of the $m$ data samples. This can be represented similarly as a by replacing $\mathbf{S}$ using Equation (4.1), $\mathbf{S} = \mathbf{XR}$, resulting in an equivalent optimization problem, Equation (4.5). These two formulations are often referred to as reprojection error minimization problems.

There is a third formulation of the $L_2$-PCA optimization that results from properties of the $L_2$-norm and aims to maximize the energy of the data in the projection space, given in Equation (4.6). The third formulation is the most commonly used today because it has an efficient solution that can be obtained in quadratic time by decomposing the covariance matrix of the input data, $\mathbf{X}^T\mathbf{X}$, and taking the eigenvectors corresponding to the K highest eigenvalues.

$$\{\mathbf{R}^*, \mathbf{S}^*\} = \underset{\mathbf{R}\in\mathcal{G}^{n\times p}, \mathbf{S}}{\arg\min}\, (\|\mathbf{X} - \mathbf{S}\mathbf{R}^T\|_2^2) \tag{4.4}$$

$$\mathbf{R}^* = \underset{\mathbf{R}\in\mathcal{G}^{n\times p}}{\arg\min}(\|\mathbf{X} - \mathbf{X}\mathbf{R}\mathbf{R}^T\|_2^2) \tag{4.5}$$

$$\mathbf{R}^* = \underset{\mathbf{R}\in\mathcal{G}^{n\times p}}{\arg\max}(\|\mathbf{X}\mathbf{R}\|_2^2) \tag{4.6}$$

### 4.1.3 $L_1$-norm Based PCA Background

$L_2$-PCA is commonly used in many applications, but it is sensitive to outliers in the dataset [77]. If the data contain outliers from a significantly different distribution, the noise has a destructive impact on the principal components. This effect is more apparent in the case of the $L_2$-PCA because of the squared magnitude in the $L_2$-Norm, as explained in [14].

One approach to minimizing the impact of outlying data points is to replace the $L_2$-Norm with the $L_1$-Norm when calculating the principal components (PCs). The drawback to using the $L_1$-Norm is that the reprojection theorem does not hold under the $L_1$-Norm, thus the Singular Value Decomposition (SVD) method used for the $L_2$ norm can no longer be applied. Additionally, the three corollary $L_1$ optimization problems, Equations (4.7), (4.8), and (4.9), are no longer equivalent. Focus has primarily been on two of the formulations: reprojection error minimization, Equation (4.8), and projection energy maximization, Equation (4.9).

$$\{\mathbf{R}^*, \mathbf{S}^*\} = \underset{\mathbf{R}\in\mathcal{G}^{n\times p}, \mathbf{S}}{\arg\min}\, (\|\mathbf{X} - \mathbf{S}\mathbf{R}^T\|_1) \tag{4.7}$$

$$\mathbf{R}^* = \underset{\mathbf{R}\in\mathcal{G}^{n\times p}}{\arg\min}(\|\mathbf{X} - \mathbf{X}\mathbf{R}\mathbf{R}^T\|_1) \tag{4.8}$$

$$\mathbf{R}^* = \underset{\mathbf{R}\in\mathcal{G}^{n\times p}}{\arg\max}(\|\mathbf{X}\mathbf{R}\|_1) \tag{4.9}$$

It is not known which optimization formulation is best suited for $L_1$-PCA. One of the advantages of our approach is that it can be used with either of the above loss functions. Some works have focused on minimizing the reprojection error in Equation (4.8), for example [12, 13, 70, 71]. However, due to the non-smooth error surface, an optimal solution has not yet been developed. The $L_1$ energy maximization formulation, Equation (4.9), has been demonstrated to be more tractable. Two optimal solutions have been developed in [96,97] with computational complexity $\mathcal{O}(2^{NK})$ and $\mathcal{O}(N^{rank(\mathbf{X})K-K+1})$, respectively, where $N$ is the number of data points. More recently a sub-optimal but more efficient method for $L_1$-PCA was proposed in [98]. Markopoulos et al. develop a method which greedily searches for each $L_1$ principal component using bit flipping to find the components that maximize the projection energy.

### 4.1.4 $L_p$-Norm Based PCA

The concept of energy maximization in the low-dimensional space was generalized from the $L_1$ and $L_2$ norms to the more general $L_p$-norm. The equation for the $L_p$-norm of the $n$-dimensional vector $\mathbf{x}$ is given as:

$$\|\mathbf{x}\|_P^P = \sum_{i=0}^{n} |x_i|^P \tag{4.10}$$

Where $|\cdot|$ is the absolute value function, and $p$ is an arbitrary real value greater than zero. The optimization problem for energy maximization based $L_p$-PCA is:

$$\mathbf{R}^* = \underset{\mathbf{R}\in\mathcal{G}^{n\times p}}{\arg\max}(\|\mathbf{XR}\|_P^P) \tag{4.11}$$

In [78] Kwak suggests that the use of fractional $L_p$-norms allows PCA to better fit datasets that are non-Gaussian distribution. Kwak proposed a solution for $L_p$-norm PCA based on a greedy component by component search using either Gradient ascent or Lagrangian multiplier methods. Their results demonstrate that for some datasets the components extracted using $P < 1.0$ work best, however they also demonstrate how the error surface is highly non-convex when $p < 1.0$. Therefore, they suggest that optimizing for lower values of $p$ may not always converge.

### 4.1.5 Linear Discriminant Analysis

Fisher's Linear Discriminant Analysis (LDA) [8,33,36,113] is a popular LDR method that uses class labels to find the projection which minimizes the covariance between samples from the same class, $\Sigma_W$, while maximizes the covariance of samples from different classes, $\Sigma_B$. The intra-class, within-class, variance is given as:

$$\Sigma_W = \sum_{i=0}^{n} (x_i - \mu_{c_i})(x_i - \mu_{c_i})^T \tag{4.12}$$

The inter-class, between-class, covariance is given as:

$$\Sigma_B = \sum_{i=0}^{n} (\mu_{c_i - \mu})(\mu_{c_i} - \mu)^T \tag{4.13}$$

Where $\mu$ is the mean for all of the data, and $\mu_{c_i}$ is the mean of the class that the $i^{\text{th}}$ sample belongs to. If the data is already zeros-mean prior to preforming LDA, Equation (4.13) simplifies to:

$$\Sigma_B = \sum_{i=0}^{n} \mu_{c_i} \mu_{c_i}^T \tag{4.14}$$

The loss function formulation of the LDA optimization objective is given as:

$$\mathbf{R}^* = \underset{\mathbf{R} \in \mathcal{G}^{n \times p}}{\arg\max} \left( \frac{\mathbf{tr}\big(\mathbf{R}^T \Sigma_B \mathbf{R}\big)}{\mathbf{tr}\big(\mathbf{R}^T \Sigma_W \mathbf{R}\big)} \right) \tag{4.15}$$

Where is $\mathbf{tr}(\cdot)$ is the matrix trace operator, which sums the elements on the main diagonal of the matrix.

Cunningham and Ghahramani [20] highlight an important error in a commonly used formulation of LDA, which uses the a greedy eigenvector decomposition of $\mathbf{\Sigma_W}^T \mathbf{\Sigma_B}$. The authors point out that this solution minimizes the following equation, which is related to but not the actual optimization objective of LDA.

$$\mathbf{R}^* = \underset{\mathbf{R} \in \mathcal{G}^{n \times p}}{\arg\max} \left( \mathbf{tr}\Big( \Big(\mathbf{R}^T \Sigma_W \mathbf{R}\Big)^{-1} \Big(\mathbf{R}^T \Sigma_B \mathbf{R}\Big) \Big) \right) \tag{4.16}$$

The difference between the optimization objective in Equation (4.15) and Equation 4.16, is the difference between minimizing the quotient of the traces, the correct objective, and minimizing the trace of the quotients, the incorrect objective. This difference is further discussed in [127, 156].

## 4.2 Dimensionality Reduction Using Proxy-Matrix Optimization

In this work we apply the Proxy Matrix Optimization methods to the problem of Linear Dimensional Reduction. At its core, PMO is a neural network-based implementation of Grassmann manifold optimization for dimensionality reduction. This method was inspired by our work in [102], however, the method was reformulated to accelerate convergence to the optimal solution. This approach is based on the iterative Grassmann manifold optimization approach to dimensionality reduction originally presented in [20]. Cunningham and Ghahramani present a method to optimize the entire projection matrix $\mathbb{R}^{k \times n}$ simultaneously using an iterative approach of a line-search optimization technique and Grassmann manifold retraction step. This approach was highly influential for the development of the proposed work, but had several drawbacks including lack of flexibility due to it requiring the partial derivative of the loss function either be explicitly provided or solved for using suboptimal numerical differentiation. In contrast, the PMO method leverages the auto differentiation methods supplied in the PyTorch Deep Learning framework [108] to automatically calculate the gradients of the loss functions, allowing for much greater flexibility.

One of the key benefits of the neural network based approach of the PMO over other LDR methods is its flexibility. Proxy Matrix Optimization can be used for any LDR method, as long as the projection matrix is required to be orthogonal. The only other requirement is that the LDR method's optimization objective can be formulated as a loss function that can be implemented in the PyTorch framework, which is often relatively straight forward. In this section we highlight some specific details for a handful of example LDR methods implemented using PMO. However, these method just serve as some key examples, and do not exhaust the full variety of LDR methods that can be implemented using PMO.

### 4.2.1 $L_p$-PCA

In this section we formulate our PCA method for the general $L_p$-norm, where if $p = 2$ the problem is the special case of $L_2$-PCA and if $p = 1$ the problem is the $L_1$-PCA. This is the first time the behavior of the reprojection minimization and the projection maximization formulations of PCA could be directly compared for cases where $p \neq 2$. Previously most works on PCA that were based on non-$L_2$ norms have focused on the projections maximization, due to its improved tractability. We hope that by providing a direct comparison we gain insight into the behaviors of each formulation and encourage future work on the reprojection formulation.

We also note that we scale the loss for each problem by the norm of the input data $\|\mathbf{X}\|_P^P$ so that the same learning rate can be used regardless of the magnitudes of the values in the input or number of samples. Because the scaling factor is constant, it can be precomputed and does not impact the geometry of the error surface. It only affects the magnitudes of the gradients.

We implement the reprojection error minimization formulation for general $L_p$-PCA as:

$$\mathbf{R}^* = \arg\min_{\mathbf{R}} \left( \frac{1}{\|\mathbf{X}\|_P^P} \|\mathbf{X} - \mathbf{X}\mathbf{R}\mathbf{R}^T\|_P^P \right) \tag{4.17}$$

The projection maximization is implemented in a similar fashion, however the norm is negated so that it can be treated as a a minimization problem. This was done because all optimizers in PyTorch are implemented for minimization. Thus, the loss function we use for the projection maximization problem is given as:

$$\mathbf{R}^* = \arg\min_{\mathbf{R}} \left( - \frac{1}{\|\mathbf{X}\|_P^P} \|\mathbf{X}\mathbf{R}\|_P^P \right) \tag{4.18}$$

By formulating our two loss functions for PCA in the more general $L_p$-norm manner we have the flexibly to explore the impact of different $L_p$-norms for PCA and instigate their impact on robustness to outliers. Previously the area of $L_p$-norm PCA has been under-explored because of the lack of flexibility, computational complexity, and instability of many solutions to the general $L_p$-PCA problem. In Section 4.3 we perform these experiments on multiple datasets.

### 4.2.2 Weighted Contribution PCA

The main motivation for using $L_p$-norms is that it has been shown that when $p < 2$ the components extracted using $L_p$-PCA are more robust to outliers [78]. As previously discussed this improved robustness is due to the lower impact on the loss when the norm is raised to a power less than 2.0. In this section we propose a new method for outlier robustness based on the concept of explicitly weighting the contribution of the loss of each point based on how well it fits our model for the data. We call this method Weighted contribution PCA (W-PCA).

There has been limited work in this area in the past. Some related works [5,23,143] aimed to learn a combination of a weighting matrix $\mathbf{W}$ in combination with the projection $\mathbf{R}$ with the goal of performing PCA on data with lost or corrupted samples. These works used an Expectation-Maximization approach to finding a solution for a $\mathbf{W}$ and $\mathbf{R}$ and were only focused on $L_2$-PCA. A key difference between these previous works is not only that they optimize $\mathbf{W}$ along with $\mathbf{R}$, but also that the weights for each entry of $\mathbf{W}_{i,j}$ were specific to each dimension of each input. In this work we fix $\mathbf{W}$ based some precomputed metric, and equally scale the contribution of all dimensions for each data point.

Our formulation for the Reprojection minimization variant of W-PCA is then given as:

$$\arg\min_{\mathbf{R}} \left( \frac{1}{\|\mathbf{X}\|_P^P} \|\mathbf{X} - \mathbf{X}\mathbf{R}\mathbf{R}^T\|_P^P \right) \tag{4.19}$$

The projection maximization optimization objective is given as:

$$\arg\min_{\mathbf{R}} \left( - \frac{1}{\|\mathbf{X}\|_P^P} \|\mathbf{X}\mathbf{R}\|_P^P \right) \tag{4.20}$$

Where $W(\mathbf{X}_i)$ is the weighted calculated by some weighting function for the input data point $\mathbf{X}_i$.

The exact weighting function used in W-PCA can be any arbitrary function that produces a real valued scalar based on a $n$-dimensional vector input. In this work we implement weighting function $W(\dot{})$ based on the distance of each the datapoint to the median of the dataset. Our weighting metric is given as:

$$W(\mathbf{X}_i) = \frac{n}{\sum_{j=0}^{n} \|\mathbf{X}_j - \tilde{\mathbf{X}}\|_P^P} \|\mathbf{X}_i - \tilde{\mathbf{X}}\|_P^P \tag{4.21}$$

Where row vector corresponding to the $i^{\text{th}}$ data point, and $\tilde{\mathbf{X}}$ $n$-dimensional vector containing the median value for each dimension of all the data points.

We choose this simple weighting function because it does not impose to constraint of prior knowledge about the data. However, if more information about the data is known, a data specific metric could be used. One potential example is that if the data is know to come from a Gaussian distribution, then a Gaussian distribution could be fit to the data. With the weight for each data point would be the probability density sampled at the location corresponding to the data point.

## 4.3 Experiments

In this section we outline our experimental results using our proposed PMO Grassmann manifold Optimization framework for dimensionality reduction. We first present the results from a series of experiments performed on an artificially generated dataset, we call the Synthetic-Dataset. The Synthetic-Dataset is a collection of multiple sample sets, each consisting of a training set with 500 samples and a test set with 100 samples. The data in both sets are samples from a randomly generated 25-dimensional Gaussian distribution with a unique random covariance matrix. These experiments are useful because the provide a controlled environment to test the methods with the ability to verify the repeatably of the results.

For our initial experiments with $L_2$-PCA we compare the proposed PMO based method against the proven optimal SVD-based PCA method. We perform these experiments on an uncorrupted version of the Synthetic-Dataset. After our $L_2$-PCA experiments, we perform a set of experiments with $L_1$-PCA. We compare our PM-$L_1$-PCA to a sate-of-the-art $L_1$ PCA method, on the version Synthetic-Dataset that has been corrupted with outliers. After comparing the special cases of $L_2$ and $L_1$ PCA to the state of the art methods in the field, we explore the behavior of the PMO method in the general $L_p$-PCA problem. We pay specific attention to the impact of both the value of $p$ and difference between the reprojection and projection objectives.

Following our experiments on the impact of different $L_p$-norms, we demonstrate how adding our W-PCA method can improve the robustness of $L_p$-PCA to outliers in the training set. All experiments on the Synthetic-Dataset, except for the $L_2$-PCA experiments, were performed on a variation of the Synthetic-Dataset where 10% of the data points were replaced with samples for a significantly different distribution, i.e. outlier points.

Once we verify that the PMO methods perform as expected in our experiments on the Synthetic-Dataset, we then demonstrate that PMO based PCA methods perform similarly on real-world computer vision tasks. The second dataset is a large collection of facial images that is commonly used for facial identification, known as Labeled Faces in the Wild (LFW) [58]. We test the PMO method on two types of dataset corruption on the LFW dataset, outlier datapoints and partial missing/corrupted data. The outlier experiments were performed in a manner similar to the experiments on the Synthetic-Dataset. In the missing/corrupted data experiments a randomly selected block covering 10% of every image is replaced with random noise.

The full set of experiments performed in this section demonstrates the potential improvement that is provided by our PMO based LDR methods. However, more importantly we hope that the results provide deeper insight into the general field of LDR using Grassmann manifold optimization and inspire future work in the area.

### 4.3.1 Initialization Experiments

Prior to preforming our experiments we acknowledge one drawback of using an optimization method based on SGD on a nonconvex loss surface. That drawback is that the optimizer may converge to different local minimum depending on its initialization. There are two popular approaches that are often used to mitigate the problem of converging to a local minima. The first approach is to randomly initialize multiple times and select the best solution. This method has the drawback of requiring the optimization to be run multiple times, thus significantly increasing the processing time. The second approach is to use prior knowledge to initialize in a region that is close to the global minimum. However, if the loss surface is non-convex, there is no guarantee that the optimizer will converge to the global minimum, even if initialized relatively close to it.

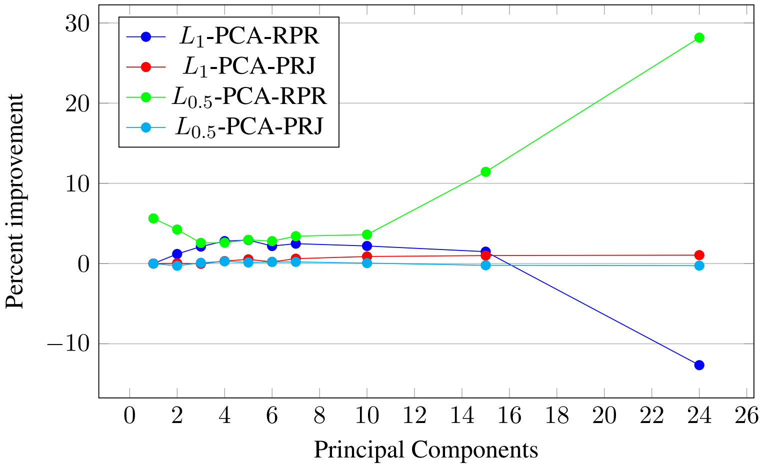


Figure 4.1: Relative percent reduction in error for 5 random initializations vs. initializing with the $L_2$ solution. Experiments were run on 10 sets of data with four different dimensionality reduction objectives.

There is no consensus on which approach produces better results, thus, we decided to run multiple experiments with both initialization methods. We ran our method once with an initialization based on the $L_2$ solution. Then we ran for 5 runs with a random initial locations and selected the solution with the minimum loss on the training set as the final solution. In Figure 4.1 we then plot the percent improvement of random initialization method relative to the $L_2$ based initialization. It is clear in the plot that most of the results were relatively similar for both initialization methods. However, random initialization did generally perform better. In the end we selected a compromise between the two methods. For the remainder of this work we ran the optimizer three times for each problem, once with the $L_2$ based initialization and twice with a random initialization. We should note that for any experiment where we were using a p value of 2, we ran with three random initializations.

### 4.3.2 $L_2$-PCA Experiments

As previously mentioned, there already exists an optimal solution for $L_2$-PCA. Thus in this work we do not aim to further improve on the reprojection results of current $L_2$-PCA methods. Instead we aim to demonstrate that our proposed neural network based Grassmann manifold optimization approach converges to the same minima as the optimal solution. For these validation experiments we used our Synthetic-Dataset. The Synthetic-Dataset consists of 10 separate sets of data-points each generated from a 25-dimensional multi-dimensional Gaussian distribution, with randomly selected covariances. All data is centered (zero-mean) before processing by any LDR methods so all of the data was inherently zero mean.

We extracted the top principal components using the standard method for $L_2$-PCA, greedily selecting the top principal components using the Singular Value Decomposition of the covariance matrix. We compare the baseline $L_2$-PCA method with our proposed method which simultaneously optimizes all of the principal components using our proposed Grassmann manifold based method, PM-$L_2$-PCA. For these experiments we randomly seed the initial state of the principal components for the PM-$L_2$-PCA method.

The plots in Figure 4.2 demonstrate that the reprojection error for the two loss functions optimized using PMO matched the result of the proven optimal solution. This is an important result as it demonstrates that the proposed PMO technique does converge to the optimal Principal Components. By demonstrating that the proposed method achieves identical results these experiments provided evidence that the proposed method is in fact an optimal solution for $L_2$-PCA.

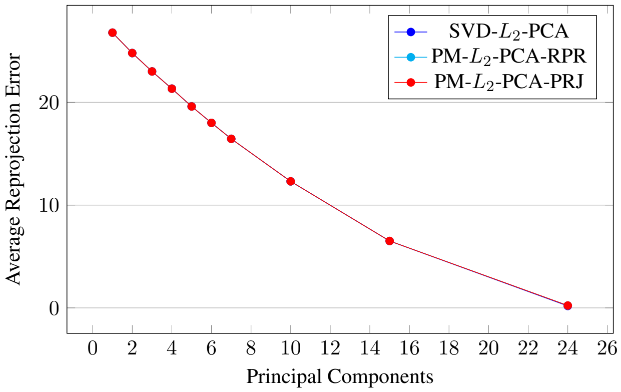


Figure 4.2: Average reprojection error for standard SVD based $L_2$-PCA and proposed PM-$L_2$-PCA methods for varying number of principal components.

### 4.3.3 $L_1$-PCA Experiments

In this section we first provide a direct comparison of our proposed method with the current state of the art $L_1$-PCA method, Bit-Flipping $L_1$-PCA [98]. In these experiments we compare the behavior of both the reprojection minimization and projection maximization formulations, Equations (4.8) and (4.9) respectively.

The experiments in this section are performed on the Synthetic-Dataset. We use a similar Synthetic-Dataset as in Section 4.3.2, however, in these experiments the training set for each of these datasets was corrupted by replacing 10% of the data with data from a significantly different distribution. A separate test partition was kept uncorrupted to test the reconstruction ability of each PCA method.

In order to better demonstrate the benefit of using $L_1$-PCA versus the $L_2$-PCA, we first present results with the current state of the art Bit-Flipping $L_1$-PCA [98] method, on our Synthetic-Dataset. We choose this method over a method for the exact optimal solution [97], because it has a better balance of computational time and accuracy.

Our first performance comparison is focused on the ability of each method to generate accurate reprojections of a clean dataset using principal components that were exacted from a training set that is corrupted with outlying data points. This metric is a good indication of how well each method captured the structure of the signal, clean data, and disregarded the noise, outlying data points, in the training set. It is commonly assumed that the lower the reconstruction on the clean data, the better the signal is captured by the Principal Components.

The plot in Figure 4.3 presents the results for the reprojection error of both the $L_2$-PCA and $L_1$-PCA methods relative to the number of principal components extracted from the 25-dimensional data, ranging from 1 to 24 principal components. Here it is clear that in almost all instances the $L_1$-PCA outperforms the $L_2$-PCA until 24 principal components are extracted. At that point the reprojection error for both methods is almost zero and any difference is insignificant.

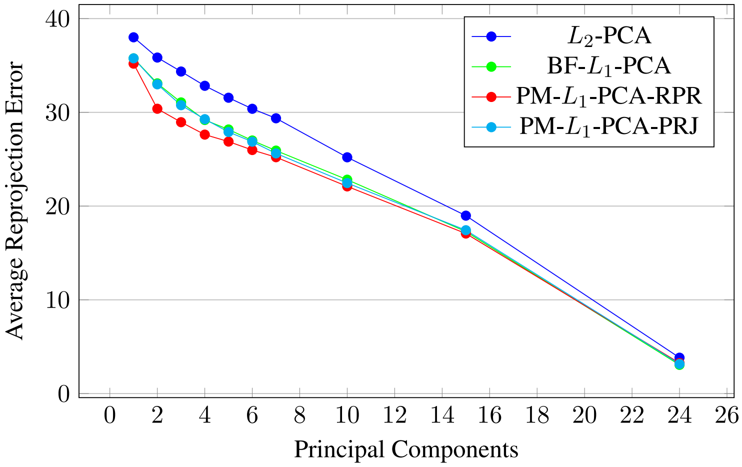


Figure 4.3: Average reprojection error for four PCA methods for varying number of principal components. The mean from runs on 10 separate dataset are plot. The results from $L_2$-PCA are shown in blue, BF-$L_1$-PC are shown in green, PM-$L_1$-PCA-RPR are shown in red, and PM-$L_1$-PCA-PRJ are shown in cyan.

We next plot the percent improvement of the $L_1$-PCA method relative to $L_2$-PCA, in terms of reprojection error. This is formulated as, $P_{im} = \frac{E_{L_2} - E_{L_1}}{E_{L_2}} \times 100.0$. We plot the percent improvement result for the principal component sweep experiments in Figure 4.4. We don't show the difference for the experiments with 24 components because the loss is so small that even a small variance is over-amplified. It is clear in 4.3 that all methods produces similar errors in the case of the number of components being one less than the original dimension of the data. This plot shows that in general the $L_1$-PCA method reduces the reprojection error relative to $L_2$-PCA, at the very least $L_1$-PCA does not under-perform $L_2$-PCA.

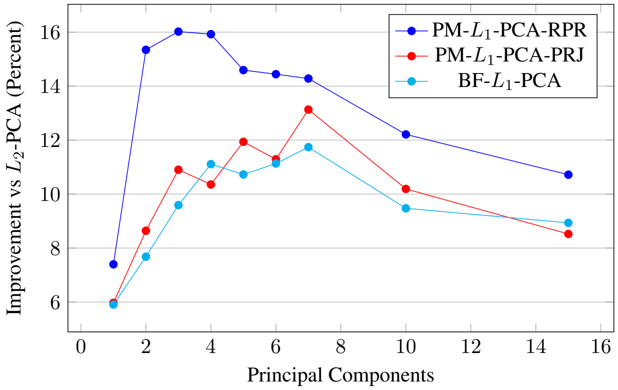


Figure 4.4: Percent improvement of the PMO and Bit-Flipping $L_1$-PCA methods relative to $L_2$-PCA for varying number of principal components. Mean calculated based on 10 runs on randomly initialized datasets.

The plots in Figure 4.4 show that, unlike our experiments with $L_2$-PCA in Figure 4.2, there is a clear difference between the reprojection and projection objectives for $L_1$-PCA. The results produced by using the projection objective with the PMO optimizer are very similar to the results produced by the bit-flipping method. This is consistent with what is expected because they share the same objective and the bit-flipping method was shown to produce results close to the optimal $L_1$-PCA method.

The more interesting result from this experiment is the result produced using the reprojection objective. The reprojection objective is generally less-well studied due to its highly nonconvex loss surface. In these experiments we demonstrate that the reprojection objective always outperforms the projection objective in terms of a lower reconstruction error of the clean data. These results show that in general the reprojection objective produces a solution that more accurately captures the structure of the data rather than the noise. The other interesting difference between the two is that the projection objective produces solutions that provide the maximal improvement over $L_2$-PCA when the number of components extracted is around half of the data's original dimension. The reprojection method, on the other hand, produces a maximal improvement when fewer components are extracted and its advantage steadily decreases as more components are extracted. This is consistent with the fact that there is a higher probability of capturing the structure of the noise in the components when more components are extracted.

After the principal component sweep experiments, we tested both $L_1$-PCA and $L_2$-PCA on varying dataset sizes, from 50-500 data points. We plot these results in the right plot of Figure 4.5. In these experiments the $L_1$-PCA method does not always outperform $L_2$-PCA. For very small datasets, with only 50 samples, the reprojection error of $L_1$-PCA is significantly worse for the projection objective and only marginally worse for the reprojection objective. However, for larger datasets the $L_1$-PCA methods produce better results than $L_2$-PCA. This under-performance is likely due to the fact the outliers have a larger impact in a small dataset.

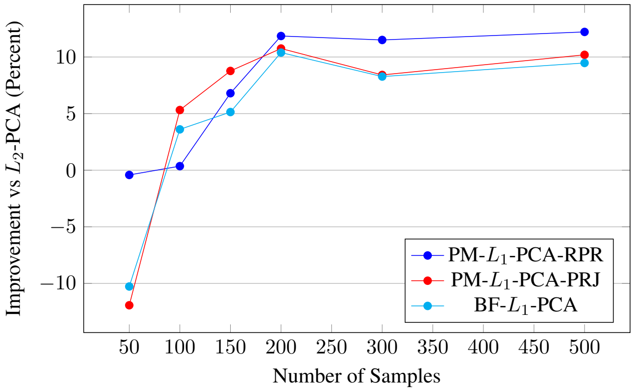


Figure 4.5: Percent improvement of the PMO and Bit-Flipping $L_1$-PCA methods relative to $L_2$-PCA for datasets with various number of samples. Mean calculated based on 10 runs on randomly initialized datasets.

A benefit of the proposed PM-$L_1$-PCA method is the reduced computational time required to find the principal components so that $L_1$-PCA can be more appealing for widespread use. In the plots in Figures 4.6 and 4.7 we demonstrate that the proposed method provides a significant speedup relative to the current state of the art BF-$L_1$-PCA method [98].

These figures show that the proposed PM-$L_1$-PCA method has a relatively low execution time irrespective of the number of principal components extracted or the dataset size. This is because the major contributing source of computations is the singular value decomposition of the proxy matrix, and the backpropgation through the SVD layer. The primary factors that affect the computation time of the PM-based PCA method are the number of input dimensions and number of output dimensions. The number of computations required by the PMO algorithm is only linear with respect

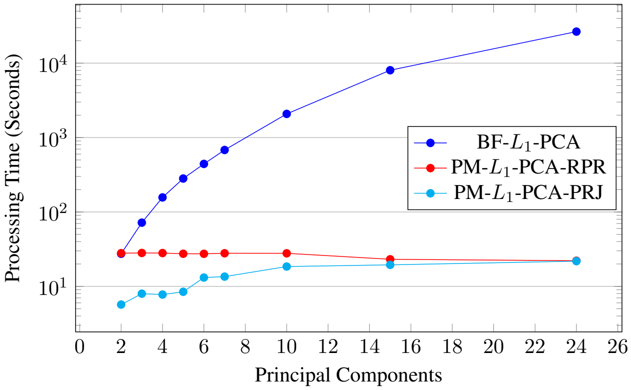


Figure 4.6: Processing time for $L_1$-PCA algorithms run on 500 samples from the toy datasets with varying number of principal components.

the number of input points. However, PM-based PCA can also be performed in a mini-batch manner, making the computation time constant with respect to the number input data points. This is in contrast to the exponential relationship between each of these parameters and the computation time of the BF-$L_1$-PCA method.

The reprojection variant of PM-PCA seems to maintain constant run times while the projection variant changes as the number of extracted components increases. We suggest that this is an artifact of our early termination criteria being too restrictive for the reprojection loss which tends to have more variation. None of the reprojection runs met the criteria for early termination while the projection methods did when fewer components were extracted.

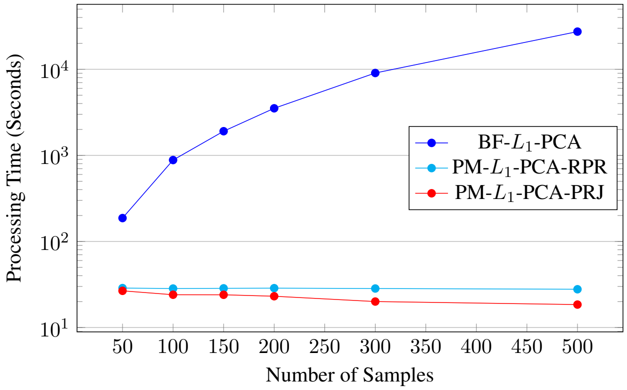


Figure 4.7: Processing time for $L_1$-PCA algorithms run on varying number of samples 50-500 from the toy calculating 10 principal components.

### 4.3.4 $L_p$-PCA Experiments

In this section we perform experiments to determine what is the impact to the reprojection error when the $p$-norm is varied from 0.1 to 2.0. Previous work in the area of $L_p$-PCA based on the projection objective [78] suggests that PCA can be made more robust when $0 < p < 1$. In this section we perform additional experiments on the Synthetic-Dataset corrupted with outliers, using both the reprojection and projection objectives.

In Table 4.1 we present the reprojection errors on a clean test set when PM-$L_p$-PCA is trained on the corrupted dataset. In the table we present the results for both the reprojection and projection objectives. We also plot the result for the two objectives with their varying $L_p$-norms in Figures 4.8 and 4.9. We plot the results for the two objectives separately to better highlight the impact of $p$ on the reprojection accuracy on the test set. We do not plot the results for the reprojeciton objective for $L_{0.1}$ because the error is so large it distorts the plots of the rest of the data.

The plots in Figure 4.8 indicate that for the reprojection objective, the optimal value for $p$ is always 0.5, providing an up to 30% improvement over $L_2$-PCA. The plots in Figure 4.9 show a similar story for the projection objective, a $p$-norm of 0.5 tends to be best. However, the reprojection error on the test set tends to be higher for the projection objective than the reprojection objective. Another trend that we noticed was the results with a $p$-norm of 0.1 were significantly worse than the rest.

| | P-norm | 1 | 2 | 5 | 10 | 15 | 24 |
|---|---|---|---|---|---|---|---|
| Reproject Min | $L_{0.1}$ | -64.6 | -61.9 | -61.1 | -43.6 | -73.4 | -100.6 |
| | $L_{0.25}$ | -38.9 | 5.4 | 4.9 | 14.1 | 18.5 | -6.9 |
| | $L_{0.5}$ | **13.4** | **25.3** | **25.9** | **28.5** | **20.9** | **20.4** |
| | $L_{0.75}$ | 11.0 | 22.6 | 20.9 | 18.6 | 17.5 | 4.8 |
| | $L_1$ | 7.4 | 15.3 | 14.6 | 12.2 | 10.7 | 11.8 |
| | $L_{1.5}$ | 1.9 | 3.2 | 3.5 | 3.6 | 2.7 | 7.9 |
| Project Max | $L_{0.1}$ | 0.4 | 7.9 | 14.7 | 13.9 | 13.5 | -2.5 |
| | $L_{0.25}$ | 7.6 | 9.6 | 19.2 | 15.6 | 11.4 | -2.5 |
| | $L_{.05}$ | **11.0** | **13.2** | **18.3** | **16.8** | **14.8** | 1.6 |
| | $L_{0.75}$ | 9.1 | 13.1 | 16.1 | 14.3 | 14.1 | **11.9** |
| | $L_1$ | 6.0 | 8.6 | 11.9 | 10.2 | 8.5 | 8.0 |
| | $L_{1.5}$ | 1.5 | 1.2 | 3.7 | 3.6 | 2.3 | -1.2 |

Table 4.1: Improvement reprojection errors, relative to $L_2$-PCA, for clean test set when PM-$L_p$-PCA with varying values of $p$ is trained on corrupted Synthetic-Dataset.

We believe that this is because the loss surface is nonconvex to an extent that makes it non-suitable for traditional gradient-based optimization techniques.

We also ran experiments with varying datasets sizes to determine how PCA with the different $L_p$-norms responded to smaller datasets. We plot the relative improvement, to $L_2$-PCA, for the reprojection and projection objectives in Figures 4.10 and 4.11, respectively. We observe similar behavior to our experiments perform in previous section, where for very small datasets $L_p$-PCA under-performs $L_2$-PCA. However, for larger datasets $L_p$-PCA significantly outperforms $L_2$-PC, with $L_{0.5}$-PCA providing the most improvement.

The plots in Figures 4.10 and 4.11 are consistent with the results from our experiments extracting different number of components, Figures 4.8 and 4.9. Generally using a norm with a lower $p$-norm provides the most robustness to outliers. However, using $p < 0.5$ generally does not result in any additional improvement. In fact, in the case of a $p$-norm of 0.1 the method performs worse. Our results demonstrate that $L_{0.5}$-PCA achieves a 30% reduction in reprojection error, relative to the standard $L_2$-PCA. This is a promising result and consistent with the results in [78]. Based on these results, for the remainder of our experiments in this chapter we use both $L_1$ and $L_{0.5}$ norms due to their robustness.

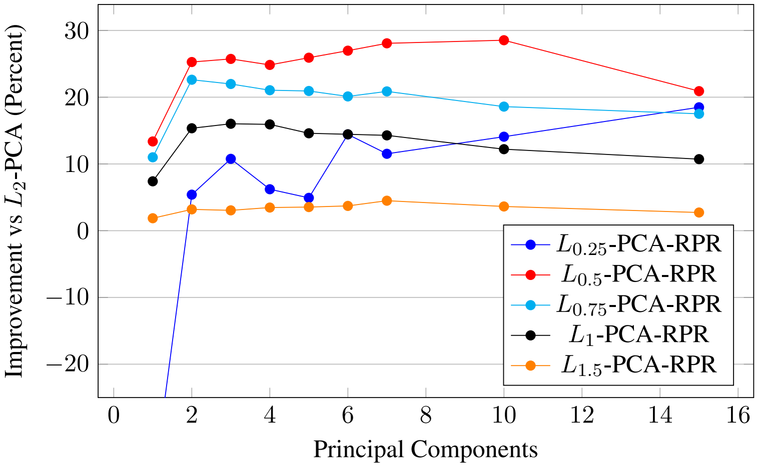


Figure 4.8: Improvement in reprojection error using the reprojection objective with the PM-$L_p$-PCA framework for different number of components, all relative to $L_2$-PCA.

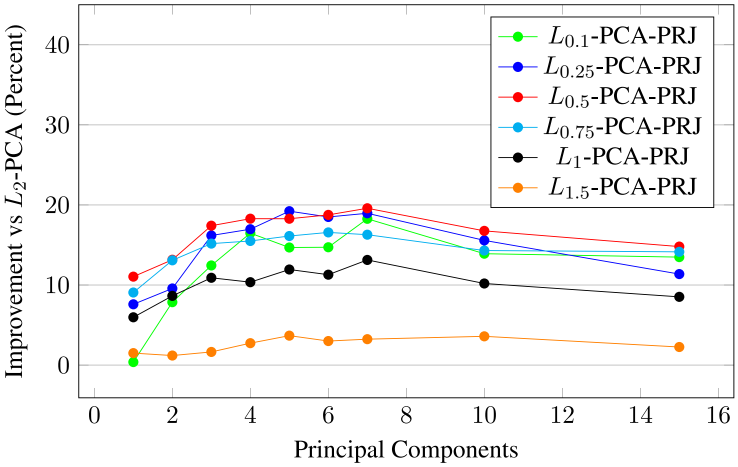


Figure 4.9: Improvement in reprojection error using the projection objective with the PM-$L_p$-PCA framework for different number of components, all relative to $L_2$-PCA.

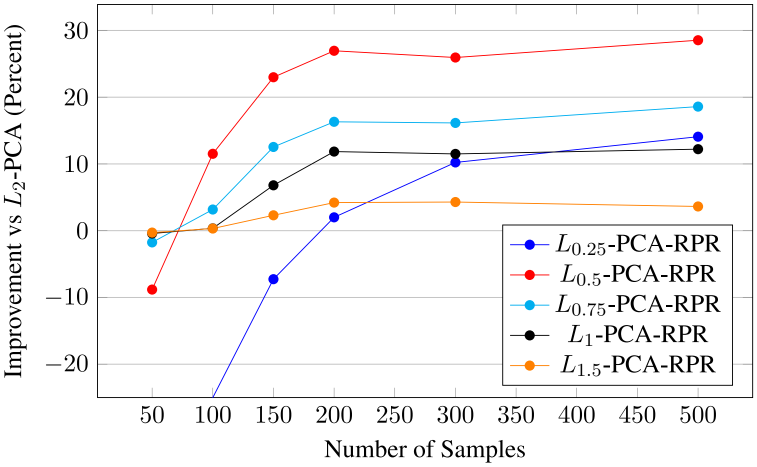


Figure 4.10: Improvement in reprojection error using the reprojection objective with the PM-$L_p$-PCA framework for different dataset sizes, all relative to $L_2$-PCA.

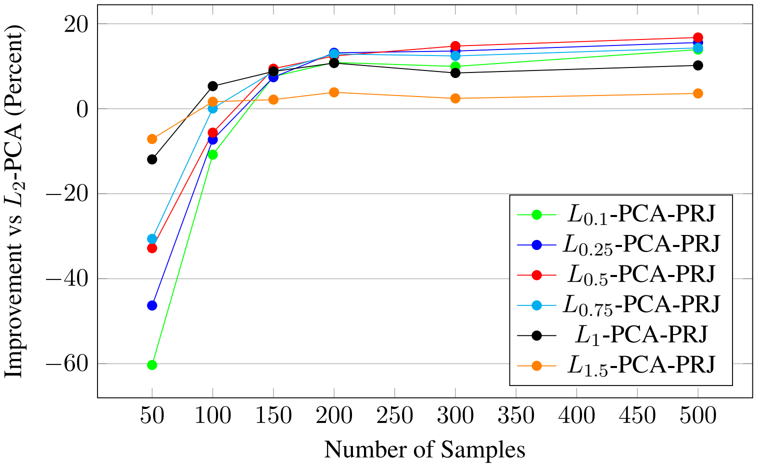


Figure 4.11: Improvement in reprojection error using the projection objective with the PM-$L_p$-PCA framework for different dataset sizes, all relative to $L_2$-PCA.

### 4.3.5 Weighted Contribution PCA Experiments

We next investigate the impact of introducing our weighted-contribution component to the PM-$LP$-PCA process, on the synthetic-dataset. For these experiments we focus on the impact of our W-PCA method for norms with $p$ of 0.5, 1.0, and 2.0. Again we compare the reconstruction error for each method against the baseline $L_2$-PCA method. We first present the results for the reprojection objective in Figure 4.12. Here we see that the W-PCA method, dashed lines, has a significant impact on PM-$L_p$-PCA with the reprojective objective for all three norms. Surprisingly even $L_2$ experiences a close to 20$\%$ improvement over the $L_2$-PCA baseline when the weighting component is used, suggesting it provides a significant improvement to robustness even for non-robust $L_p$-norms. The W-PCA has an even more significant impact on the $L_{0.5}$ and $L_1$ based PCA methods. By adding the weighted contribution component to PM-$L_1$-PCA it outperforms PM-$L_{0.5}$-PCA with the weighted contribution method.

In Figure 4.13 we plot the impact of W-PCA on the PM-$L_p$-PCA with the projection objective. Here we find that W-PCA has an even more pronounced impact on the PM-$L_2$-PCA, improving the results by up to 60$\%$ from the $L_2$-PCA baseline. The W-PCA method also significantly improves the results produced by PM-$L_1$-PCA. However, the PM-$L_{0.5}$-PCA is only marginally improved by the W-PCA method.

We next performed experiments varying the number of samples in the dataset to examine the impact on the W-PCA methods. In Figures 4.14 and 4.15 we present the results achieved with and without the W-PCA method for the reprojection and projection objectives, respectively.

In this section our experiments were performed on a dataset that was synthetically generated in order to better examine the behavior of our PM-$L_p$-PCA method. In the next sets of experiments we use real imagery to demonstrate that the improvements demonstrated in this section were not artifacts specific to our dataset.

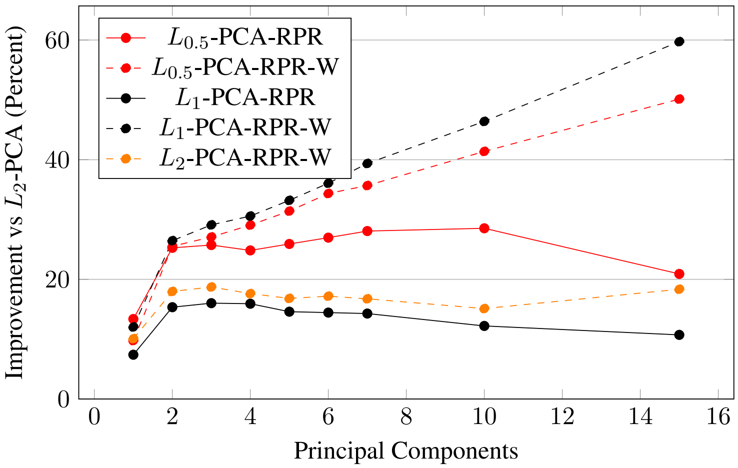


Figure 4.12: Improvement in reprojection error using the reprojection objective with the PM-$L_p$-PCA framework using $p$ of 0.5, 1.0, 2.0. both without and with weighted-contributions, solid and dashed lines respectively. All results are relative improvement to $L_2$-PCA.

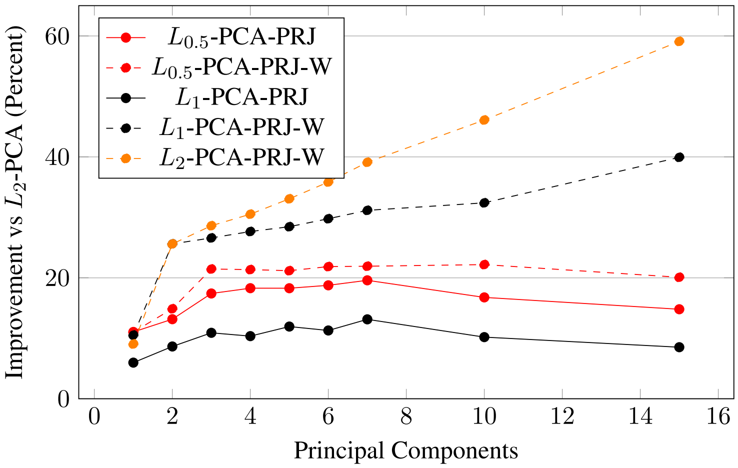


Figure 4.13: Improvement in reprojection error using the projection objective with the PM-$L_p$-PCA framework using $p$ of 0.5, 1.0, 2.0. both without and with weighted-contributions, solid and dashed lines respectively. All results are relative improvement to $L_2$-PCA.

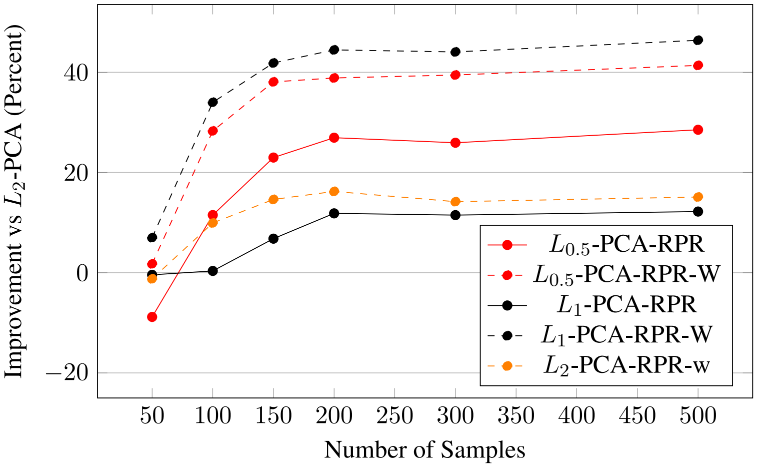


Figure 4.14: Improvement in reprojection error using the reprojection objective with the PM-$L_p$-PCA framework for different dataset sizes, all relative to $L_2$-PCA.

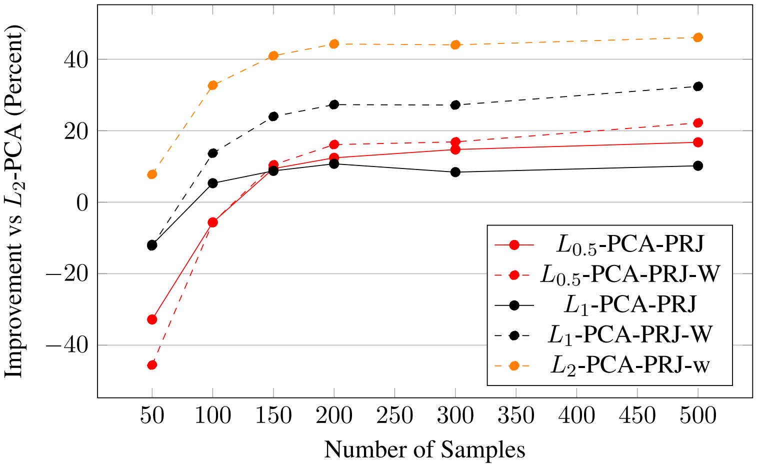


Figure 4.15: Improvement in reprojection error using the projection objective with the PM-$L_p$-PCA framework for different dataset sizes, all relative to $L_2$-PCA.

### 4.3.6 Labeled Faces in the Wild Outlier Experiments

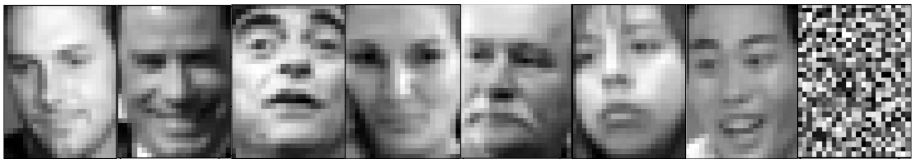


Figure 4.16: Example images of faces in the Labeled Faces in the Wild Dataset. The right-most image is an example of outlier noise image added to the training dataset in experiments.

In order to demonstrate the benefits of our proposed PMO based $L_p$-PCA method in a computer vision application we ran experiments extracting subspaces from a popular dataset consisting of images of faces. For these experiments we used the Labeled Faces in the Wild (LFW) Dataset [58], a dataset commonly used for facial identification. Examples from the LFW dataset and outlier noise samples are shown in Figure 4.16. Prior to the recent development of deep learning based methods, a common first step in many facial identification methods is to extract the principal components from the set of images, known as Eigenfaces [131]. In these experiments we demonstrate how such a technique would benefit from using the proposed $L_p$-PCA method if the dataset of faces is corrupted by outliers. We first present results where we simulate a corrupted dataset by outlying data samples. Then we present results on a dataset that simulates partially corrupted/lost/occluded training data.

For these experiments we downsampled each image to 25% of its original dimension to reduce computation time. We then split the LFW dataset into separate train and test partitions using an 80:20 split. For the outlier experiments the training set was then corrupted by replacing 10% of the samples with random noise. We then extracted the principal components from the corrupted training set using multiple methods.

We plot the improvement it terms of reduced reprojection error for our proposed PM-$L_1$-PCA method relative to the results from $L_2$-PCA in Figure 4.17. We found that when more than 50 components were extracted, both formulations of our proposed PM-$L_p$-PCA method outperform $L_2$-PCA. The increase in performance is most pronounced with the reprojection minimization formulation which achieves over a 40% reduction in reprojection error over $L_2$-PCA. Unlike in the synthetic-dataset there is not as pronounced a difference between the PM-$L_{0.5}$-PCA and PM-$L_1$-PCA results. One trend seems to be that PM-$L_1$-PCA tends to produce a lower reprojection error on the clean test set when fewer components are extracted. When more than 200 components are extracted PM-$L_{0.5}$-PCA seems to produces better results.

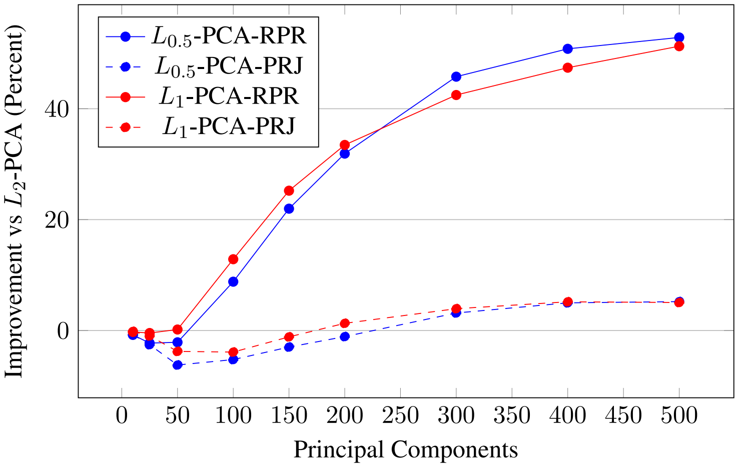


Figure 4.17: Improvement in reprojection error for the LFW dataset with varying values of $p$ using the proposed PM-$L_p$-PCA framework, all relative to $L_2$-PCA.

We then applied our weighted contribution method to the PM-$L_p$-PCA methods. The results for the reprojection and projection objectives are plotted in Figure 4.18 and 4.19, respectively. Again in these experiments the W-PCA versions of each method (dashed lines) significantly improve on their baselines (solid-lines) for the projection formulation. The reprojection formulation does not exhibit significant improvement, outside of PM-$L_2$-PCA, though it always improves. Again PM-$L_1$-PCA becomes the top performing method when the W-PCA method is used.

In addition to our quantitative results in this section we also provide qualitative results in the form of example reprojections of uncorrupted test images in Figure 4.20. These results were produced by extracting the top 100 components from a dataset corrupted with 10%. Then the components trained on the corrupted dataset were used to reproject clean images, to determine how much of the noise was was captured in the components.

It is clear from the examples in Figure 4.20 that the projection formulation of PM-$L_1$-PCA does not perform any better than the $L_2$-PCA. Unfortunately, the projection formulations of PM-$L_1$-PCA captures a fair amount of noise in the reprojection. This confirms the results in Figures 4.17 and 4.19, where the projection formulation of PM-

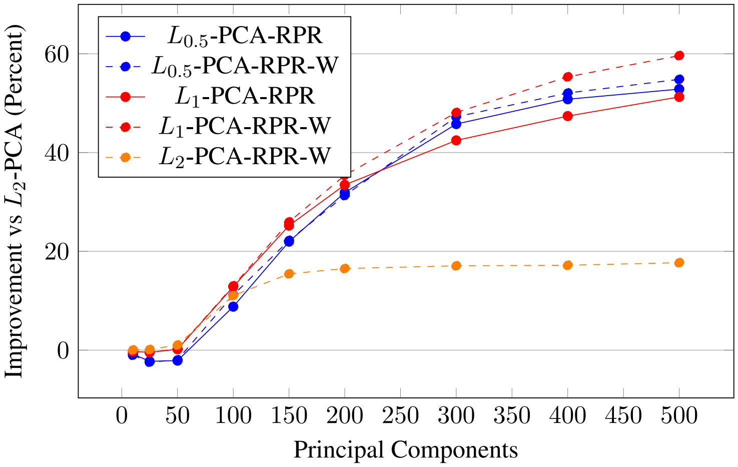


Figure 4.18: Improvement in reprojection error using the reprojection objective with the PM-$L_p$-PCA framework, both with and without the weighted loss functions, all relative to $L_2$-PCA.

$L_1$-PCA does not outperform $L_2$-PCA until more than 200 components are extracted. The reprojection formulation of PM-$L_1$-PCA, on the other hand, shows a significant reduction in noise in the reprojections. The differences are particularly evident in the areas with less variation, such as the forehead and cheeks. The improvement between the contribution variant is not as visually apparent however the quantitative results confirm there is small improvement when the weighted contribution method is uses.

The images that make up the columns in Figure 4.21 show the principal components (1,11,21,31,41,51,61,71,81,91) for: $L_2$ PCA (first row), PM-$L_1$-PCA with projection (second row), PM-$L_1$-PCA with reprojection (third row), and PM-$L_1$-PCA with reprojection and weighted contribution (fourth row), for these experiments. Notice how principal components in the first 5 columns of $L_2$-PCA do not contain much noise, while the last 5 columns are significantly influenced by noise. The projection formulation of PM-$L_1$-PCA appears to incorporate a small portion of noise into each component, though every component also appears to have some structure corresponding to the characteristics of a face. The components learned using PM-$L_1$-PCA with the reprojection formulations on the other hand have very little noise in their structure. Instead, each component seems to capture some facial structure such as edges around the nose, mouth and eyes.

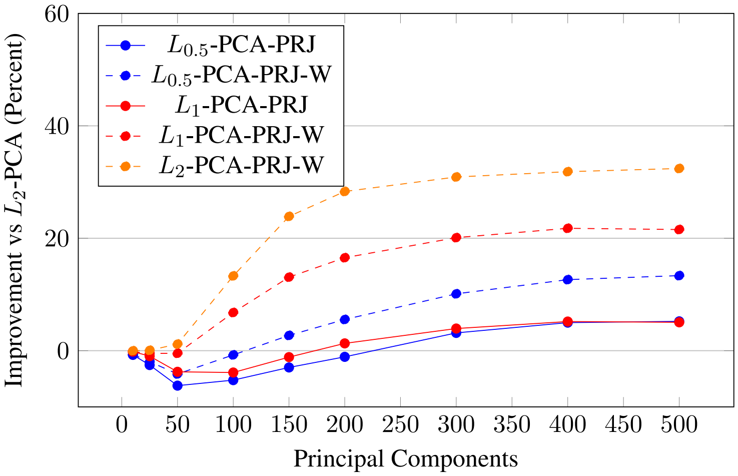


Figure 4.19: Improvement in reprojection error using the projection objective with the PM-$L_p$-PCA framework, both with and without the weighted loss functions, all relative to $L_2$-PCA.

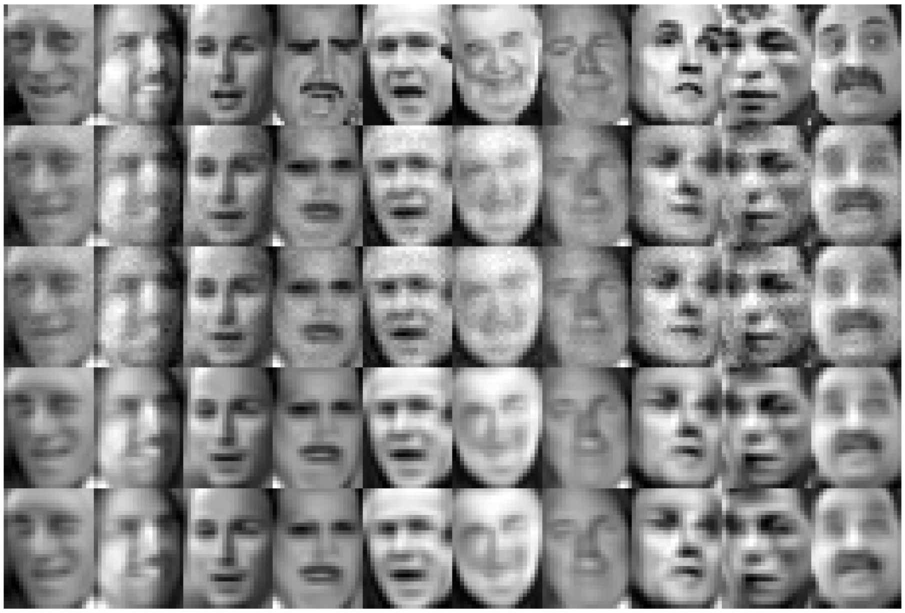


Figure 4.20: Reprojection of face images using the proposed POM-$L_1$-PCA and standard $L_2$-PCA principal components extracted from corrupted face training set. First row: Input Image. Second row: Reprojection using $L_2$-PCA. Third row: Reprojection using PM-$L_1$-PCA with projection formulation. Third row: Reprojection using PM-$L_1$-PCA with reprojection formulation. Fourth row: Reprojection using PM-$L_1$-PCA with reprojection formulation and weighted contribution. Fifth row: Reprojection using PM-$L_1$-PCA-W with Projection Maximization Formulation.

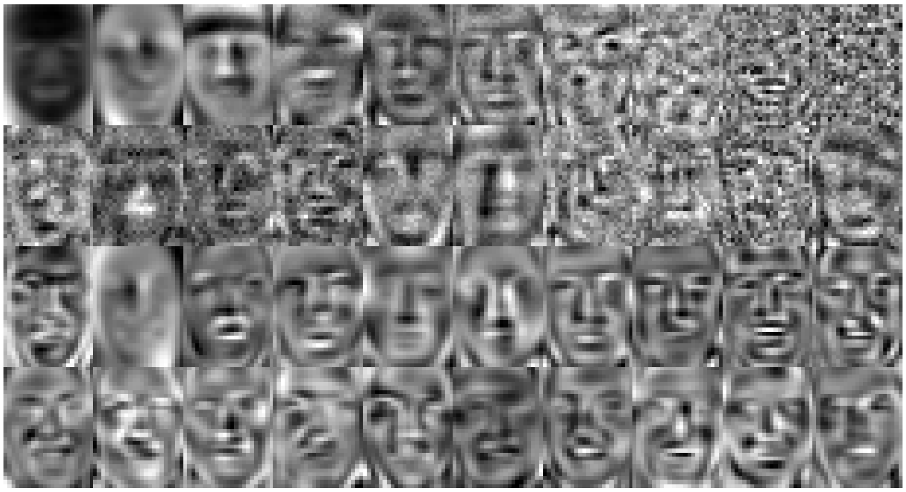


Figure 4.21: Example principal components resulting from training different PCA methods on a dataset of facial images with 10% of the image corrupted with uniform noise. The rows are organized as follows. First row: $L_2$-PCA. Second row: projection formulation of PM-$L_1$-PCA. Third row: reprojection formulation of PM-$L_1$-PCA. Forth row: reprojection formulation of PM-$L_1$-PCA with weighted contribution.

### 4.3.7 Labeled Faces in the Wild Occlusion Experiments

The definition of what constitutes an outlier is very broad and there are multiple ways to model the corruption of a dataset. In the previous sections we modeled outliers as points that were entirely replaced by samples from a different dataset. However, a more common type of corruption is when only a portion of the data for each sample is replaced with values from a different distribution. The partial corruption could result from lost data during transmission or partial occlusion of an object. In this section we test the robustness of our proposed method to such corruptions. To do this we replace a randomly placed patch which covers roughly 10% to to 30% of each image with noise sampled from a uniform distribution. This type of data corruption models a situation where all of the faces in a dataset are partially occluded by an object, such as a scarf/hat/glasses.

In this section we perform the same analysis as in the previous section. In Figures 4.22 and 4.23 we plot the the improvement to reprojection error of the the reprojection and projection formulations of PM-$L_p$-PCA, respectively. In Figure 4.24 we plot a selection of the components learned by the various PCA methods. We present the reconstructions of clean data samples using the features extracted from the corrupted dataset in Figure 4.25. In Figure 4.26 we show the reprojection of corrupted data points using the same components.

The plots in Figures 4.22 and 4.23 show an interesting difference in reconstruction behavior based on the type of corruption. In the previous section the weighted contribution method provided a significant benefit to all PM-$L_p$-PCA methods, whereas in the occlusion experiments it provided little to no benefit. This makes sense as it is primarily designed to discard datapoints that are far away from the average of the dataset. However, in the occlusion experiments all of the samples are corrupted to a similar extent. Thus it is not beneficial to minimize the contribution of any particular point, as they are all equally corrupted. The more interesting behavior is that in almost every case the projection maximization formulation of PM-$L_p$-PCA produces worse results than $L_2$-PCA. However, the reprojection formulations generally improve on $L_2$-PCA providing further evidence that the reprojection formulation is generally superior to the projection formulation.

The principal components, shown in Figure 4.24, show a similar behavior as in the previous section. The initial components extracted using $L_2$-PCA are generally unaffected by the noise, where as the higher order components tend to capture the noise, as seen at the center of the images. Again, the components extracted using the projection formulation of PM-$L_1$-PCA capture some face structure, but also capture

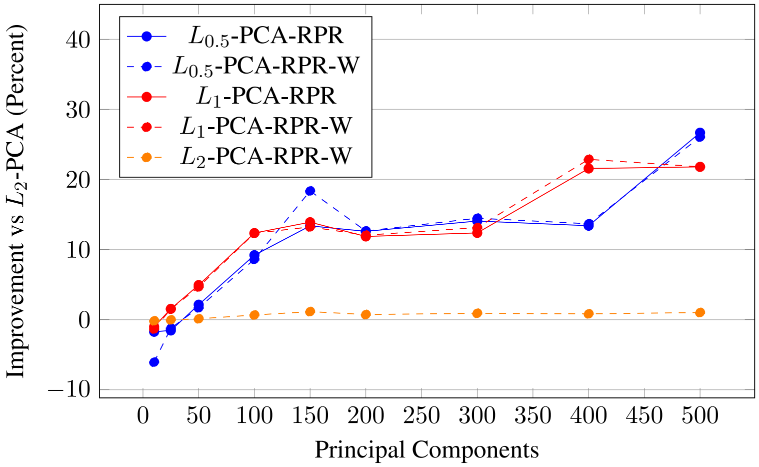


Figure 4.22: Improvement in reprojection error using the reprojection objective with the PM-$L_p$-PCA framework, both with and without the weighted loss functions, all relative to $L_2$-PCA.

the noise. Finally, the reprojection versions of PM-$L_1$-PCA both capture face structure and generally ignore the noise.

The images in Figure 4.25 show that the reprojections using $L_2$ capture some portion of the noise from the training set, while the reprojection formulation of PM-$L_1$-PCA focuses on the general structure of the faces. A more interesting set of results are those shown in Figure 4.26, where we plot the reconstruction ability of the different methods when the input images are partially corrupted. For these experiments we used input images that had the random corruption patches, and we show the reconstruction using different PCA methods. These experiments are interesting because they show how well each method captured the underlying structure of the faces. Here we see that there is a fair amount of noise in the $L_2$-PCA reprojections. While the reprojection PM-$Lp$-PCA methods generally fill in the regions with believable face structures, even if the brightness of the patch is distorted due to the nature of the corruption patch.

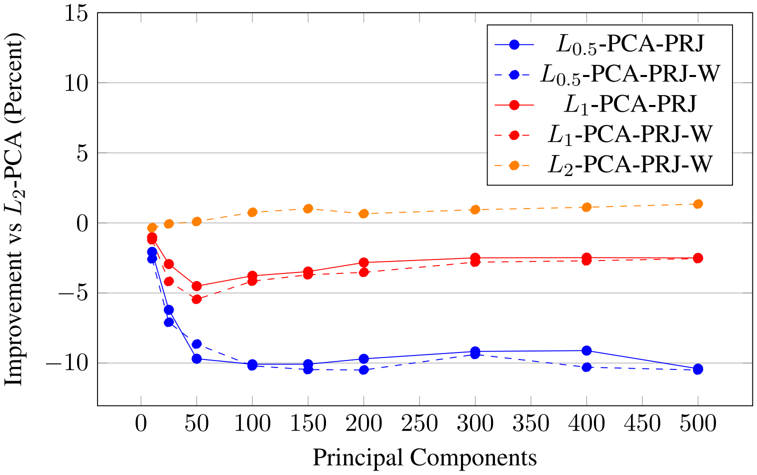


Figure 4.23: Improvement in reprojection error using the projection objective with the PM-$L_p$-PCA framework, both with and without the weighted loss functions, all relative to $L_2$-PCA.

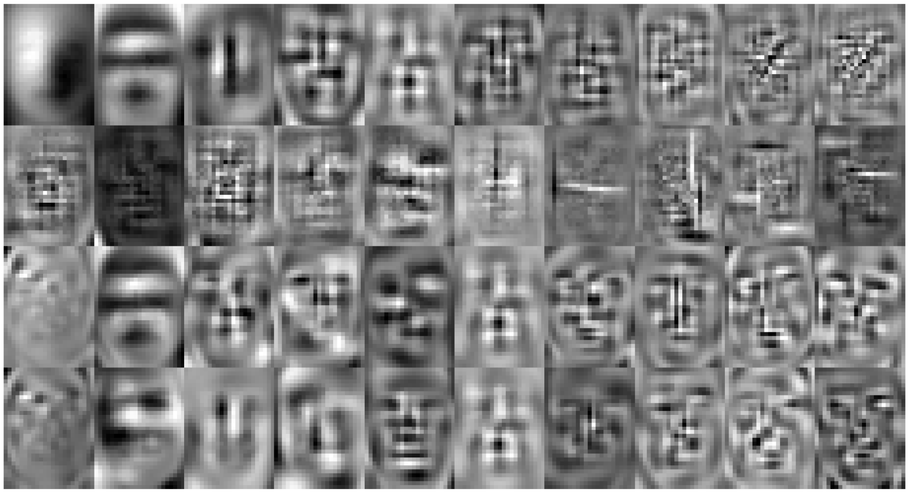


Figure 4.24: Example principal components resulting from training different PCA methods on a dataset of facial images with all images corrupted by a patch covering 10% to 30% of the image made of uniform noise. The rows are organized as follows. First row: $L_2$-PCA. Second row: projection formulation of PM-$L_1$-PCA. Third row: reprojection formulation of PM-$L_1$-PCA. Fourth row: reprojection formulation of PM-$L_1$-PCA with weighted contribution.

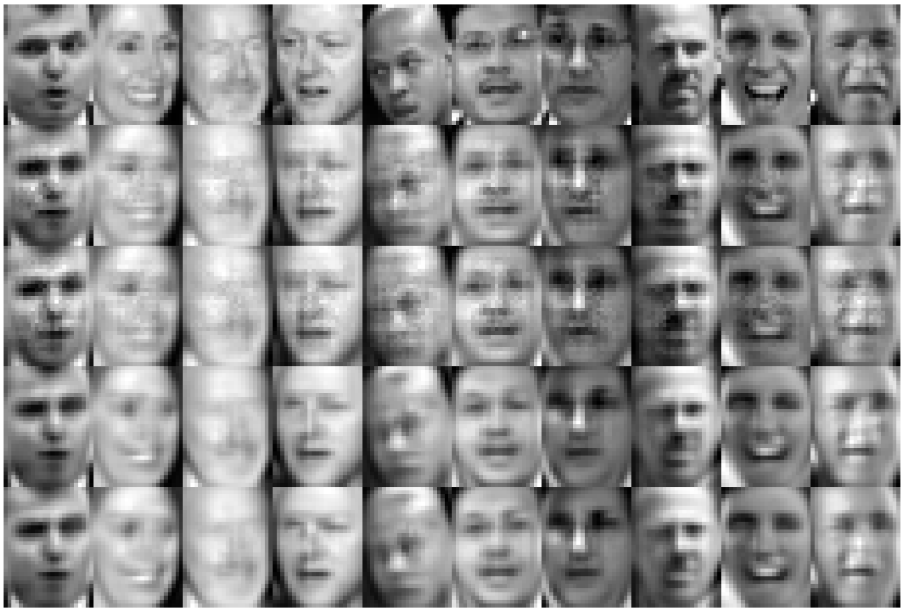


Figure 4.25: Reprojection of face images using the proposed PM-$L_1$-PCA and standard $L_2$-PCA principal components extracted from the corrupted face training set. First row: Input Image. Second row: Reprojection using $L_2$-PCA. Third row: Reprojection using PM-$L_1$-PCA with Reprojection Minimization Formulation. Fourth: Reprojection using PM-$L_1$-PCA with Projection Maximization Formulation. Fifth row: Reprojection using PM-$L_1$-PCA-W with Projection Maximization Formulation.

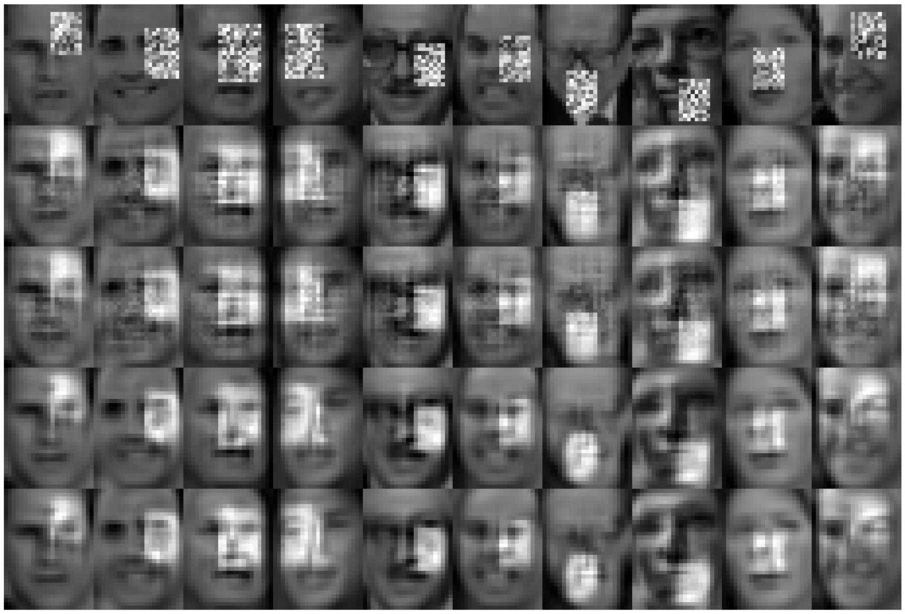


Figure 4.26: Reprojection of face images using the proposed PM-$L_1$-PCA and standard $L_2$-PCA principal components extracted from the corrupted face training set. First row: Input Image. Second row: Reprojection using $L_2$-PCA. Third row: Reprojection using PM-$L_1$-PCA with Reprojection Minimization Formulation. Fourth row: Reprojection using PM-$L_1$-PCA with Projection Maximization Formulation. Fifth row: Reprojection using PM-$L_1$-PCA-W with Projection Maximization Formulation.

## 4.4 Remarks

In this section we outlined how linear dimensionality reduction problems can be reformulated as Grassmann Manifold optimization problems. We then demonstrated how the Proxy-Matrix optimization methods developed in Chapter 3 can be used to find a solution the LDR problem, given a loss function. We then perform an in-depth analysis on multiple version of the most popular LDR method, Principal Component Analysis. We present multiple results for different versions of PCA on both synthetic and real-world datasets corrupted by outliers. In Chapter 5 we demonstrate how the methods developed in this chapter can be used to compress deep neural networks.

# Chapter 5

# Cascaded Projection Network Compression

We propose a data-driven approach for deep convolutional neural network compression that achieves high accuracy with high throughput and low memory requirements. Current network compression methods either find a low-rank factorization of the features that requires more memory, or select only a subset of features by pruning entire filter channels. We propose the Cascaded Projection (CaP) compression method that projects the output and input filter channels of successive layers to a unified low dimensional space based on a low-rank projection. We optimize the projection to minimize classification loss and the difference between the next layer's features in the compressed and uncompressed networks. To solve this non-convex optimization problem we propose a new optimization method of a proxy matrix using backpropagation and Stochastic Gradient Descent (SGD) with geometric constraints, as described in Chapters 3. Our cascaded projection approach leads to improvements in all critical areas of network compression: high accuracy, low memory consumption, low parameter count and high processing speed. The proposed CaP method demonstrates state-of-the-art results compressing VGG16 and ResNet networks with over 4× reduction in the number of computations and excellent performance in top-5 accuracy on the ImageNet dataset before and after fine-tuning.

## 5.1 Introduction

The compression of deep neural networks is gaining attention due to the effectiveness of deep networks and their potential applications on mobile and embedded devices. The powerful deep networks developed today are often overparameterized [26] and require large amounts of memory and computational resources [15]. Thus, efficient network compression, that reduces the number of computations and memory required to process images, enables the broader application of deep neural networks.

Methods for network compression can be categorized into four types, based on quantization, sparsification, factorization and pruning. In this work we leverage the advantages of factorization and pruning methods, as they are the most popular. Quantization methods accelerate deep networks and reduce storage by using mixed precision arithmetic and hashing codes [17, 19, 46]. However most of them require mixed precision arithmetic, which is not always available on standard hardware. Sparsification methods eliminate individual connections between nodes that have minimal impact on the network, however, they are not well suited for current applications because most neural network libraries are not optimized for sparse convolution operations and fail to achieve significant speedup.

Factorization methods [27, 68, 79, 161] reduce computations by factorizing the network kernels, often by splitting large kernels into a series of convolutions with smaller filters. These methods have the drawback of increasing memory consumption due to the intermediate convolution operations. Such memory requirements pose a problem for mobile applications, where network acceleration is needed most. Pruning methods [46, 53, 85, 93, 104, 132, 160, 164] compress layers of a network by removing entire convolutional filters and the corresponding channels in the filters of the next layer. They do not require feature map reprojection, however they discard a large amount of information when eliminating entire filter channels.

In this paper, we propose the Cascaded Projection (CaP) compression method which combines the superior reconstruction ability of factorization methods with the multi-layer cascaded compression of pruning methods. Instead of selecting a subset of features, as is done in pruning methods, CaP forms linear combinations of the original features that retain more information. However, unlike factorization methods, CaP brings the kernels in the next layer to low dimensional feature space and therefore does not require additional memory for reprojection.

Figure 5.1 provides a visual representation of the differences between the three methods: factorization (top row) reprojects to higher dimensional space and increases

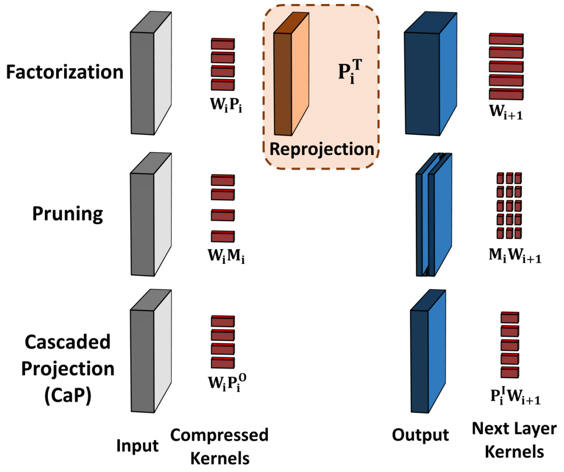


Figure 5.1: Visual representation of network compression methods on a single CNN layer. Top row: Factorization compression with a reprojection step that increases memory. Middle row: Pruning compression where individual filters are removed. Bottom row: Proposed CaP method which forms linear combinations of the filters without requiring reprojection.

memory, pruning (middle row) masks filters and eliminates their channels, and our proposed CaP methods (bottom row) combines filters to a smaller number without reprojecting. Our results demonstrate that by forming filters based on linear combinations instead of pruning with a mask, more information is kept in the filtering operations and better network classification accuracy is achieved.

The primary contributions of this chapter are the following:

1. We propose the CaP compression method that finds a low dimensional projection of the feature kernels and cascades the projection to compress the input channels of the kernels in the next layers.
2. We introduce proxy matrix projection backpropagation, the first method to optimize the compression projection for each layer using end-to-end training with standard backpropagation and stochastic gradient descent.
3. Our optimization method allows us to use a new loss function that combines the reconstruction loss with classification loss to find a better solution.
4. The CaP method is the first to simultaneously optimize the compression projection for all layers of residual networks.

5. Our results illustrate that CaP compressed networks achieve state-of-the-art accuracy while reducing the network's number of parameters, computational load and memory consumption.

## 5.2 Related Work

The goal of network compression and acceleration is to reduce the number of parameters and computations performed in deep networks without sacrificing accuracy. Early work in network pruning dates back to the 1990's [47]. However, the area did not gain much interest until deep convolutional networks became common [73, 74, 130] and the redundancy of network parameters became apparent [26]. Recent works aim to develop smaller network architectures that require fewer resources [55, 62, 114].

Quantization techniques [17, 19, 46, 67] use integer or mixed precision arithmetic only available on state-of-the-art GPUs [95]. These methods reduce the computation time and the amount of storage required for the network parameters. They can be applied in addition to other methods to further accelerate compressed networks, as was done in [72].

Network sparsification [89], sometimes referred to as unstructured pruning, reduces the number of connections in deep networks by imposing sparsity constraints. The work in [56] proposed recasting the sparsified network into separate groups of operations where the filters in each layer are only connected to a subset of the input channels. In [152] k-means clustering is used to encourage similarity between features to aid in compression. However, these methods require training the network from scratch which is not practical or efficient.

Filter factorization methods reduce computations at the cost of increased memory load for storing intermediate feature maps. Initial works focused on factorizing the three-dimensional convolutional kernels into three separable one-dimensional filters [27, 68]. In [79] CP-decomposition is used to decompose the convolutional layers into five layers with lower complexity. More recently [161] performed a channel decomposition that found a projection of the convolutional filters in each layer such that the asymmetric reprojection error was minimized.

Channel pruning methods [85, 93, 104, 132, 164] remove entire feature kernels for network compression. In [46] kernels are pruned based on their magnitudes, under the assumption that kernels with low magnitudes provide little information to the network. Li *et al.* [85] suggested a similar pruning technique based on kernel statistics. He *et al.* [53] proposed pruning filters based on minimizing the reconstruction error of each layer. Luo *et al.* [93] further extended the concepts in [53] to prune filters that have minimal impact on the reconstruction of the next layer. Yu *et al.* [160] proposed Neuron Importance Score Propagation (NISP) to calculate the importance of each neuron based on its contribution to the final feature representation and prune feature channels that provide minimal information to the final feature representation.

Other recent works have focused less on finding the optimal set of features to prune and more on finding the optimal amount of features to remove from each layer of the network. This is important to study because the amount of pruning performed in each layer is often set arbitrarily or through extensive experimentation. In [155, 160] the authors propose automatic pruning architecture methods based on statistical measures. In [52, 60] methods are proposed which use reinforcement learning to learn an optimal network compression architecture. Additional work has been done to reduce the number of parameters in the final layers of deep networks [18], however the fully connected layer only contributes a small fraction of the overall computations.

## 5.3 Cascaded Projection Methodology

In this section we provide an in depth discussion of the CaP compression and acceleration method. We first introduce projection compression when applied to a single layer, and explain the relationship between CaP and previous filter factorization methods [161]. One of the main goals of CaP compression is eliminating the feature reprojection step performed in factorization methods. To accomplish this, CaP extends the compression in the present layer to the inputs of the kernels in the next layer by projecting them to the same low dimensional space, as shown in Figure 5.2. Next we demonstrate that, with a few alterations, the CaP compression method can perform simultaneous optimization of the projections for all of the layers in residual networks [49]. Lastly we present the core component of the CaP method, which is our new end-to-end optimization method that optimizes the layer compression projections using standard back-propagation and stochastic gradient descent.

### 5.3.1 Problem Definition

In a convolutional network, as illustrated in the top row of Figure 5.2, the $i^{th}$ layer takes as input a 4-Tensor $\mathbf{I_i}$ of dimension $(n \times c_i \times h_i \times w_i)$, where $n$ is the number of images (mini-batch size) input into the network, $c_i$ is the number channels in the input and $w_i$ and $h_i$ are the height and width of the input. The input is convolved with a set of filters $\mathbf{W_i}$ represented as a 4-Tensor with dimensions $(c_{i+1} \times c_i \times k \times k)$, where $c_{i+1}$ is the number of kernels and $k$ is the spatial dimensions of the kernels, generally 3 pixels. In many networks, there is an additional bias, $\mathbf{b_i}$, of dimension $(c_{i+1} \times 1 \times 1 \times 1)$, that is added to each channel of the output. More formally, the convolution operation for layer $i$ of a CNN is given as:

$$\mathbf{O_i} = \mathbf{I_i} * \mathbf{W_i} + \mathbf{b_i} \tag{5.1}$$

where $(*)$ is the convolution operator. The input to the next layer is calculated by applying a nonlinearity to the output as $\mathbf{I_{i+1}} = G(\mathbf{O_i})$, where $G(\cdot)$ is often a ReLU [105].

Network compression aims to reduce the number of filters so that the classification accuracy of the network is minimally impacted. In this work we find a projection $\mathbf{P_i}$ that maps the features to a lower dimensional space by minimizing both the reconstruction error and the classification loss, as described in the rest of this section.

### 5.3.2 Single Layer Projection Compression

We first present how projection based compression is used to compress a single layer of a network. To compress layer $i$, the output features are projected to low dimensional representation of rank $r$ using an orthonormal projection matrix $\mathbf{P_i}$ represented as a 4-Tensor of dimension $(c_{i+1} \times r \times 1 \times 1)$. The optimal projection, $\mathbf{P_i^*}$ for layer $i$, based on minimizing the reconstruction loss is given as:

$$\mathbf{P_i^*} = \underset{\mathbf{P_i}}{\arg\min} \left\| \mathbf{O_i} - (\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} + \mathbf{b_i} * \mathbf{P_i}) * \mathbf{P_i^T} \right\|_F^2 \tag{5.2}$$

where $\|\cdot\|_F^2$ is the Frobenious norm.

Inspired by [161], we alter our optimization criteria to minimize the reconstruction loss of the input to the next layer. This results in the optimization:

$$\mathbf{P_i^*} = \underset{\mathbf{P_i}}{\arg\min} \left\| G(\mathbf{O_i}) - G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} + \mathbf{b_i} * \mathbf{P_i}) * \mathbf{P_i^T}) \right\|_F^2 \tag{5.3}$$

The inclusion of the nonlinearity makes this a more difficult optimization problem. In [161] the problem is relaxed and solved using Generalized SVD [43, 141, 142]. Our Cascaded Projection method is based on the end-to-end approach described next.

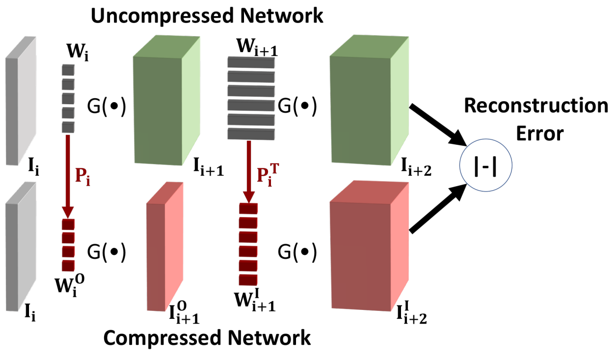


Figure 5.2: Visual representation of the compression of a CNN layer using the CaP method to compress the filters $\mathbf{W_i}$ and $\mathbf{W_{i+1}}$ in the current and next layers using projections $\mathbf{P_i}$ and $\mathbf{P_i^T}$ respectively. The reconstruction error in the next layer is computed after the nonlinearity $G(\cdot)$.

### 5.3.3 Cascaded Projection Compression

Factorization methods, including the single layer projection compression discussed above, are inefficient due to the additional convolution operations required to reproject the features to high dimensional space. Pruning methods avoid reprojection by removing all channels associated with the pruned filters. CaP takes a more powerful approach that forms linear combination of the kernels by projecting without the extra memory requirements of factorization methods. Following the diagram in Figure 5.2, we consider two successive convolutional layers, labeled $i$ and $i{+}1$, with kernels $\mathbf{W_i}$, $\mathbf{W_{i+1}}$ and biases $\mathbf{b_i}$, $\mathbf{b_{i+1}}$ respectively. The input to layer $i$ is $\mathbf{I_i}$, while the output of layer $i{+}1$ is the input to layer $i+2$, denoted by $\mathbf{I_{i+2}}$ and given below.

$$\mathbf{I_{i+2}} = G(G(\mathbf{I_i} * \mathbf{W_i} + \mathbf{b_i}) * \mathbf{W_{i+1}} + \mathbf{b_{i+1}}) \tag{5.4}$$

After substituting our compressed representation with reprojection for layer $i$ in the above we get:

$$\mathbf{I_{i+2}} = G(G((\mathbf{I_i} {*} \mathbf{W_i} {*} \mathbf{P_i} {+} \mathbf{b_i} {*} \mathbf{P_i}) {*} \mathbf{P_i^T}) {*} \mathbf{W_{i+1}} {+} \mathbf{b_{i+1}}) \tag{5.5}$$

To avoid reprojecting the low dimensional features back to higher dimensional space with $\mathbf{P_i^T}$, we seek two projections. The first $\mathbf{P_i^O}$ which captures the optimal lower dimensional representation of the features in the current layer, and the second $\mathbf{P_i^I}$ which pulls the kernels of the next layer down to lower dimensional space. This formulation leads to an optimization problem over the projection operators:

$$\begin{aligned}\{\mathbf{P_i^{I}}^*, \mathbf{P_i^{O}}^*\} = \operatorname*{arg\,min}_{\mathbf{P_i^I}, \mathbf{P_i^O}} \|\mathbf{I_{i+2}} - G(G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i^O} \\ + \mathbf{b_i} * \mathbf{P_i^O})) * \mathbf{P_i^I} * \mathbf{W_{i+1}} + \mathbf{b_{i+1}})\|_F^2\end{aligned} \tag{5.6}$$

To make the problem tractable, we enforce two strong constraints on the projections. We require that they are orthonormal and transposes of each other: $\mathbf{P_i^I} = (\mathbf{P_i^O})^T$. For the remainder of this work we replace $\mathbf{P_i^O}$ and $\mathbf{P_i^I}$ with $\mathbf{P_i}$ and $\mathbf{P_i^T}$, respectively. These constraints make the optimization problem more feasible by reducing the parameter search space to a single projection operator for each layer.

$$\begin{aligned}\mathbf{P_i^*} = \operatorname*{arg\,min}_{\mathbf{P_i}, \mathbf{P_i} \in \mathbb{O}^{\mathbf{n \times m}}} \|\mathbf{I_{i+2}} - G(G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} \\ + \mathbf{b_i} * \mathbf{P_i})) * \mathbf{P_i^T} * \mathbf{W_{i+1}} + \mathbf{b_{i+1}})\|_F^2\end{aligned} \tag{5.7}$$

We solve the optimization of a single projection operator for each layer using a novel data-driven optimization method for projection operators discussed in Section 5.3.6.

### Kernel Compression and Relaxation

Once the projection optimization is complete, we replace the kernels and biases in the current layer with their projected versions $\mathbf{W_i^O} = \mathbf{W_i} * \mathbf{P_i}$ and $\mathbf{b_i^O} = \mathbf{b_i} * \mathbf{P_i}$ respectively. We also replace the kernels in the next layer with their input compressed versions $\mathbf{W_{i+1}^I} = \mathbf{P_i^T} * \mathbf{W_{i+1}}$. Thus,

$$\mathbf{I_{i+2}} = G(G((\mathbf{I_i} * \mathbf{W_i^O} + \mathbf{b_i^O})) * \mathbf{W_{i+1}^I} + \mathbf{b_{i+1}}) \tag{5.8}$$

Figure 5.2 depicts how the filters $\mathbf{W_{i+1}^I}$ in the next layer are compressed using the projection $\mathbf{P_i^T}$ and are therefore smaller than the kernels in the original network. Utilizing the compressed kernels $\mathbf{W_i^O}$ and $\mathbf{W_i^I}$ results in twice the speedup over traditional factorization methods for all compressed intermediate layers (other than first and last layers).

Following kernel projection, we perform an additional round of training in which only the compressed kernels are optimized. We refer to this step as kernel relaxation because we are allowing the kernels to find a better optimal solution after our projection optimization step.

### 5.3.4 Mixture Loss

A benefit of gradient based optimization is that a loss function can be altered to minimize both reconstruction and classification error. Previous methods have focused on either reconstruction error minimization [53, 93] or classification [160] based metrics when pruning each layer. We propose using a combination of the standard cross entropy classification loss, $L_{Class}$, and the reconstruction loss $L_R$, shown in Figure 5.2. The reconstruction loss for layer $i$ is given as:

$$\begin{aligned} L_R(i) = \|&\mathbf{I_{i+2}} - G(G((\mathbf{I_i} * \mathbf{W_i} * \mathbf{P_i} \\ &+\mathbf{b_i} * \mathbf{P_i})) * \mathbf{P_i^T} * \mathbf{W_{i+1}} + \mathbf{b_{i+1}})\|_F^2 \end{aligned} \tag{5.9}$$

The mixture loss used to optimize the projections in layer $i$ is given as

$$L(i) = L_R(i) + \gamma L_{Class} \tag{5.10}$$

where $\gamma$ is a mixture parameter that allows adjusting the impact of each loss during training. By using a combination of the two losses we obtain a compressed network that maintains classification accuracy while having feature representations for each layer which contain the maximal amount of information from the original network.

### 5.3.5 Compressing Multi-Branch Networks

Multi-branch networks are popular due to their excellent performance and come in a variety of forms such as the Inception networks [137, 138, 139], Residual networks (ResNets) [49] and Dense Networks (DenseNets) [57] among others. We primarily focus on applying CaP network compression to ResNets, but our method can be integrated with other multi-branch networks. We select the ResNet architecture for two reasons. First, ResNets have a proven record of producing state-of-the art results [49, 50]. And second, the skip connections work well with network compression, as they allow propagating information through the network regardless of the compression process within the individual layers.

Our CaP modification for ResNet compression is illustrated in Figure 5.3. In our approach, we do not alter the structure of the residual block outputs, therefore we do not compress the outputs of the last convolution layers in each residual block, as was done by [93]. In [85, 160] pruning is performed on the residual connections, but we do not affect them, because pruning these layers has a large negative impact on the network's accuracy.

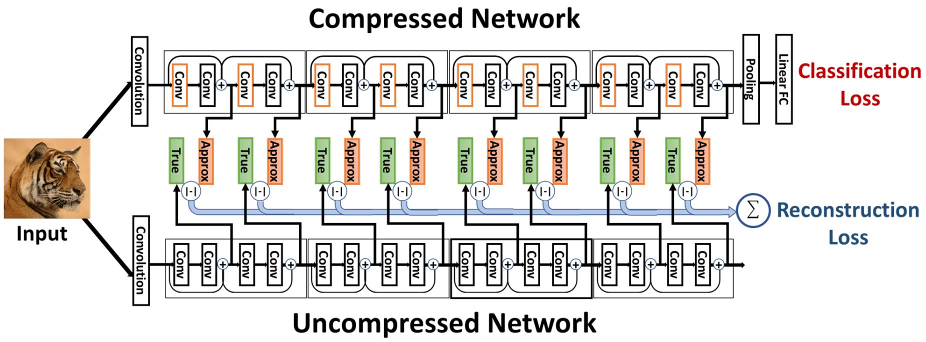


Figure 5.3: Illustration of simultaneous optimization of the projections for each layer of the ResNet18 network using a mixture loss that includes the classification loss and the reconstruction losses in each layer for intermediate supervision. We do not alter the structure of the residual block outputs, therefore we do not affect residual connections and we do not compress the outputs of the last convolution layers in each residual block.

We calculate the reconstruction error in ResNets at the outputs of each residual block, as shown in Figure 5.3, in contrast to single branch networks where we calculate the reconstruction error at the next layer as shown in Figure 5.2. By calculating the reconstruction error after the skip connections, we leverage the information in the skip connections in our projection optimization.

### Simultaneous Layer Compression

Most network compression methods apply a greedy layer-wise compression scheme, where one layer is compressed or pruned at a time. However, this layer-by-layer approach to network compression can lead to sub-optimal results [160]. We now present a version of CaP where all layers are simultaneously optimized. This approach allows the latter layers to help guide the projections of the earlier layers and minimize the total reconstruction error throughout the network.

In our experiments, we found that simultaneous optimization of the projection matrices has the risk of becoming unstable when we compress more than one layer in each residual block. To overcome this problem we split the training of the projections in residual blocks with more than one compressible layer into two rounds. In the first round, the projections for the odd layers are optimized, and in the second round the even layer projections are optimized.

Additionally, we found that using the reconstruction loss at the final layers did not provide enough supervision to the network. We therefore introduced deep supervision for each layer by minimizing the sum of normalized reconstruction losses for each layer, given by:

$$\underset{\mathbf{P_i}\in\mathbf{P}}{\arg\min}\sum_{i=0}^{N} L_R(i) + \gamma L_{Class} \tag{5.11}$$

where $\mathbf{P_i}$ is the projection for the $i^{th}$ layer, and $N$ is the total number of layers. We outline our approach to finding a solution for the above optimization using iterative backpropagation next.

### 5.3.6 Back-Propagated Projection Optimization

In this section we present an end-to-end Proxy Matrix Projection (PMaP) optimization method, which is an iterative optimization of the projection using backpropagation with Stochastic Gradient Descent (SGD). The proposed method efficiently optimizes the network compression by combining backpropagation with geometric constraints.

In our framework, we restrict the projection operators to be orthogonal and thus satisfy $\mathbf{P_i}^T\mathbf{P_i} = \mathbf{I}$. The set of $(n \times m)$ real-valued orthogonal matrices $\mathbb{O}^{\mathbf{n}\times\mathbf{m}}$, forms a smooth manifold known as a Grassmann manifold. There are several optimization methods on Grassmann manifolds, most of which include iterative optimization and retraction methods [2, 3, 20, 140, 151].

With CaP compression, the projection for each layer is dependent on the projections in all previous layers adding dependencies in the optimization across layers. Little work had been done in the field of optimization over multiple dependent Grassmann manifolds. Huang *et al.* [59] impose orthogonality constraints on the weights of a neural network during training using a method for backpropagation of gradients through structured linear algebra layers developed in [65, 66]. Inspired by these works, we utilize a similar approach where instead of optimizing each projection matrix directly, we use a proxy matrix $\mathbf{X_i}$ for each layer $i$ and a transformation $\Phi(\cdot)$ such that $\Phi(\mathbf{X_i}) = \mathbf{P_i}$.

We obtain the transformation $\Phi(\cdot)$ that projects each proxy matrix $\mathbf{X_i}$ to the closest location on the Grassmann manifold by performing Singular Value Decomposition (SVD) on $\mathbf{X_i}$, such that $\mathbf{X_i} = \mathbf{U_i}\mathbf{\Sigma_i}\mathbf{V_i^T}$, where $\mathbf{U_i}$ and $\mathbf{V_i^T}$ are orthogonal matrices and $\mathbf{\Sigma_i}$ is the matrix of singular values. The projection to the closest location on the Grassmann manifold is performed as $\Phi(\mathbf{X_i}) = \mathbf{U_i}\mathbf{V_i^T} = \mathbf{P_i}$.

During training, the projection matrix $\mathbf{P_i}$ is not updated directly; instead the proxy parameter $\mathbf{X_i}$ is updated based on the partial derivatives of the loss with respect to $\mathbf{U_i}$ and $\mathbf{V_i}$, $\frac{\partial L}{\partial \mathbf{U_i}}$ and $\frac{\partial L}{\partial \mathbf{V_i}}$ respectively. The partial derivative of the loss $L$ with respect to the proxy parameter $\mathbf{X_i}$ was derived in [65, 66] using the chain rule and is given by:

$$\frac{\partial L}{\partial \mathbf{X_i}} = \mathbf{U_i}\left\{2\mathbf{\Sigma_i}\left(\mathbf{K_i^T} \circ \left(\mathbf{V_i^T}\frac{\partial L}{\partial \mathbf{V_i}}\right)\right)_{sym} + \frac{\partial L}{\partial \mathbf{\Sigma_i}}\right\}\mathbf{V_i^T} \tag{5.12}$$

where $\circ$ is the Hadamard product, $\mathbf{A_{sym}}$ is the symmetric part of matrix A given as $\mathbf{A_{sym}} = \frac{1}{2}(\mathbf{A^T} + \mathbf{A})$. Since $\Phi(\mathbf{X_i}) = \mathbf{U_i}\mathbf{V_i^T}$, the loss does not depend on the matrix $\mathbf{\Sigma_i}$. Thus, $\frac{\partial L}{\partial \mathbf{\Sigma_i}} = 0$, and Equation (5.12) becomes:

$$\frac{\partial L}{\partial \mathbf{X_i}} = \mathbf{U_i}\left\{2\mathbf{\Sigma_i}\left(\mathbf{K_i^T} \circ \left(\mathbf{V_i^T}\frac{\partial L}{\partial \mathbf{V_i}}\right)\right)_{sym}\right\}\mathbf{V_i^T} \tag{5.13}$$

The above allows us to optimize our compression projection operators for each layer of the network using backpropagation and SGD. Our method allows for end-to-end network compression using standard deep learning frameworks for the first time.

## 5.4 Experiments

We first perform experiments on independent layer compression of the VGG16 network to investigate how each layer responds to various levels of compression. We then perform a set of ablation studies on the proposed CaP algorithm to determine the impact for each step of the algorithm on the final accuracy of the compressed network. We compare CaP to other state-of-the-art methods by compressing the VGG16 network to have over 4× fewer floating point operations. Finally we present our experiments with varying levels of compression of ResNet architectures, with 18 or 50 layers, trained on the CIFAR10 dataset.

All experiments were performed using PyTorch 0.4 [108] on a work station running Ubuntu 16.04. The workstation had an Intel i5-6500 3.20GHz CPU with 15 GB of RAM and a NVIDIA Titan V GPU.

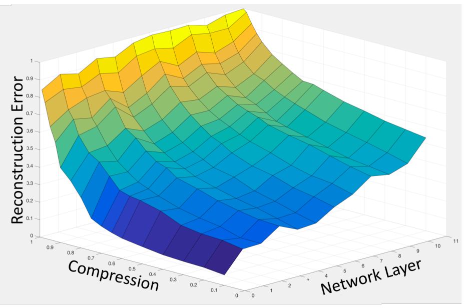


Figure 5.4: Plot of the reconstruction error (vertical axis) for the range of compression (left axis) for each layer of the network (right axis). The reconstruction error is lower when early layers are compressed.

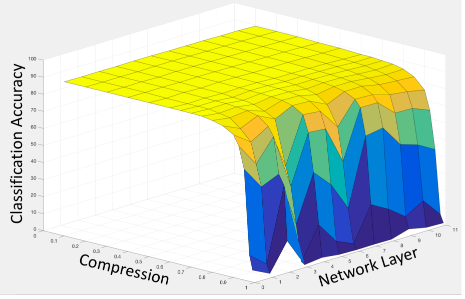


Figure 5.5: Plot of the classification accuracy (vertical axis) for the range of compression (left axis) for each layer of the network (right axis). The classification accuracy remains unaffected for large amounts of compression in a single layer anywhere in the network.

### 5.4.1 Layer-wise Experiments

In these experiment we investigate how each layer of the network is affected by increasing amounts of compression. We perform filter compression using CaP for each layer independently, while leaving all other layers uncompressed. We considered a range of compression for each layer, from 5% to 99%, and display the results in Figures and . This plot shows two trends. Firstly the reconstruction error does not increase much until 70% compression, indicating that a large portion of the parameters in each layer are redundant and could be reduced without much loss in accuracy. The second trend is the increase in reconstruction error for each level of compression for the deeper layers of the network (right axis).

In Figure 5.4.1 we plot the network accuracy resulting from each level of compression for each layer. The network is relatively unaffected for a large range of compression, despite the fact that there is a significant amount of reconstruction error introduced by the compression shown in Figure 5.4.1.

### 5.4.2 CaP Ablation Experiments

We ran additional experiments to determine the contribution of the projection optimization and kernel relaxation steps of our algorithm. We first trained the ResNet18 network on the CIFAR100 dataset and achieved a baseline accuracy of 78.23%. We then compressed the network to 50% of the original size using only parts of the CaP method to assess the effects of different components. We present these results in Table 5.1.

| **ResNet18 Network Variation** | **Accuracy** |
|---|---|
| ResNet18 Uncompressed (upper bound) | 78.23 |
| Compressed ResNet18 from Scratch | 77.22 |
| CaP Compression with Projection Only | 76.65 |
| CaP with Random Proj. & Kernel Relax | 76.27 |
| CaP with Projection & Kernel Relax | **77.47** |

Table 5.1: Network compression ablation study of the CaP method compressing the ResNet18 Network trained on the CIFAR100 dataset. (Bold numbers are best).

We also trained a compressed version ResNet18 from scratch for 350 epochs, to provide a baseline for the compressed ResNet18 network. When only projection compression is performed on the original ResNet18 network, there was a drop in accuracy of 1.58%. This loss in classification accuracy decreased to 0.76% after kernel relaxation. In contrast, when the optimized projections are replaced with random projections and only kernel relaxation training is performed, there is a 1.96% drop in accuracy, a 2.5 times increase in classification error. These results demonstrate that the projection optimization is an important aspect of our network compression algorithm, and the combination of both steps outperforms training the compressed network from scratch.

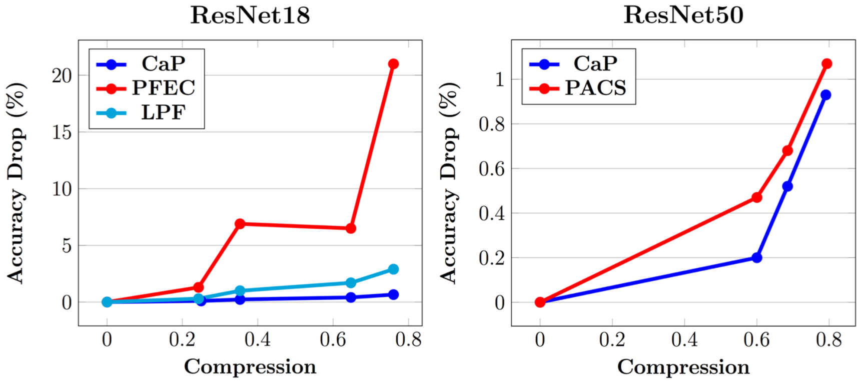


Figure 5.6: Classification accuracy drop on CIFAR10, relative to baseline, of compression methods (CaP, PCAS [155], PFEC [85] and LPF [60]) for a range of compression levels on ResNet18 (Right) and ResNet50 (Left).

| **ResNet** | **Method** | **FT** | **FLOPs %** | **Acc. / Base** |
|---|---|---|---|---|
| 56 | PFEC [85] | N | 72.4 | 91.31 / 93.04 |
| | CP [53] | N | 50.0 | 90.90 / 92.80 |
| | SFP [51] | N | 47.4 | 92.26 / 93.59 |
| | AMC [52] | N | 50.0 | 90.1 / 92.8 |
| | CaP | N | 50.2 | **92.92** / 93.51 |
| | PFEC [85] | Y | 72.4 | 93.06 / 93.04 |
| | NISP [160] | Y | 57.4 | (-0.03) * |
| | CP [53] | Y | 50.0 | 91.80 / 92.80 |
| | SFP [51] | Y | 47.4 | 93.35 / 93.59 |
| | AMC [52] | Y | 50.0 | 91.9 / 92.8 |
| | DCP [164] | Y | 35.0 | **93.7** / 93.6 |
| | CaP | Y | 50.2 | 93.22 / 93.51 |
| 110 | PFEC [85] | N | 61.4 | 92.94 / 93.53 |
| | MIL [29] | N | 65.8 | 93.44 / 93.63 |
| | SFP [51] | N | 59.2 | 93.38 / 93.68 |
| | CaP | N | 50.1 | **93.95/** 94.29 |
| | PFEC [85] | Y | 61.4 | 93.30 / 93.53 |
| | NISP [160] | Y | 56.3 | (-0.18) * |
| | SFP [51] | Y | 59.2 | 93.86 / 93.68 |
| | CaP | Y | 50.1 | **94.14/** 94.29 |

Table 5.2: Comparison of CaP with pruning and factorization based methods using ResNet56 and ResNet110 trained on CIFAR10. FT denotes fine-tuning. (Bold numbers are best). * Only the relative drop in accuracy was reported in [160] without baseline accuracy.

### 5.4.3 ResNet Compression on CIFAR 10

We perform two sets of experiments using ResNet18 and ResNet50 trained on the CIFAR10 dataset [74]. We compress 18 and 50 layer ResNets with varying levels of compression and compare the relative drop in accuracy of CaP with other state-of-the-art methods [60, 85, 155]. We plot the drop in classification accuracy for ResNet18 and ResNet50 in Figure 5.6. For both networks, the CaP method outperforms the other methods for the full range of compression.

In Table 5.2, we present classification accuracy of ResNet56 and ResNet110 with each residual block compressed to have 50% fewer FLOPs using CaPs. We compare the results obtained by CaP with those of [51, 52, 53, 85, 160] where the networks have been subjected to similar compression ratios. We report accuracy results with and without fine-tuning and include the baseline performance for comparison.

Results with fine-tuning are generally better, except in cases when there is overfitting. However, fine-tuning for a long period of time can hide the poor performance of a compression algorithm by retraining the network filters away from the compression results. The results of the CaP method without fine-tuning are based on projection optimization and kernel relaxation on the compressed filters with reconstruction loss, while the fine-tuning results are produced with an additional round of training based on mixture loss for all of the layers in the network.

### 5.4.4 VGG16 Compression with ImageNet

We compress the VGG16 network trained on ImageNet2012 [25] and compare the results of CaP with other state-of-the-art methods. We present two sets of results, without fine-tuning and with fine-tuning, in Tables 5.3 and 5.4 respectively. Fine-tuning on ImageNet is time intensive and requires significant computation power. This is a hindrance for many applications where users do not have enough resources to retrain a compressed network.

In Table 5.3 we compare CaP with factorization and pruning methods, all without fine-tuning. As expected, factorization methods suffer from increased memory load due to their additional intermediate feature maps. The channel pruning method in [53] has a significant reduction in memory consumption but under-performs the factorization method in [161] without fine-tuning. We present two sets of results for the CaP algorithm, each with different levels of compression for each layer. To match the architecture used in [53] we compressed layers 1-7 to 33% of their original size, and filters in layers 8-10 to 50% of their original size, while the remaining layers are

| **Method** | **Parameters** | **Memory (Mb)** | **FLOPs** | **GPU Speedup** | **Top-5 Acc/ Baseline** |
|---|---|---|---|---|---|
| VGG16 [130] (Baseline) | 14.71M | 3.39 | 30.9B | 1 | 89.9 |
| Low-Rank [68] | - | - | - | 1.01* | 80.02 / 89.9 |
| Asym. [161] | **5.11M** | 3.90 | 3.7B | 1.55* | 86.06 / 89.9 |
| Channel Pruning [53] | 7.48M | 1.35 | **6.8B** | 2.5* | 82.0 / 89.9 |
| CaP ( [53] arch) | 7.48M | 1.35 | **6.8B** | 3.05 | 86.57 / 90.38 |
| CaP Optimal | 7.93M | **1.11** | **6.8B** | **3.44** | **88.23** / 90.38 |

Table 5.3: Network compression results of pruning and factorization based methods without fine-tuning. The top-5 accuracy of the baseline VGG16 network varies slightly for each of the methods due to different models and frameworks. (Bold numbers are best). Results marked with * were obtained from [53].

| **Method** | **Mem. (Mb)** | **FLOPs** | **Top-5 Acc. / Baseline** |
|---|---|---|---|
| VGG16 [130] | 3.39 | 30.9B | 89.9 |
| Scratch [53] | 1.35 | 6.8B | 88.1 |
| COBLA [84] | 4.21 | 7.7B | 88.9 / 89.9 |
| Tucker [72] | 4.96 | **6.3B** | **89.4** / 89.9 |
| CP [53] | 1.35 | 6.8B | 88.9 / 89.9 |
| ThiNet-2 [93] | 1.44 | 6.7B | 88.86 / 90.01 |
| CaP | **1.11** | 6.8B | 89.39 / 90.38 |

Table 5.4: Network compression results of pruning and factorization based methods with fine-tuning. (Bold numbers are best).

left uncompressed . We also used the CaP method with a compression architecture that was selected based on our layer-wise training experiments. The results in Table 5.3 demonstrate that the proposed CaP compression achieves higher speedup and higher classification accuracy than the factorization or pruning methods.

In Table 5.4 we compare CaP with state-of-the-art network compression methods, all with fine-tuning. The uncompressed VGG16 results are from [130]. We include results from training a compressed version of VGG16 from scratch on the ImageNet dataset as reported in [53]. We compare CaP with the results of two factorization methods [72, 84] and two pruning methods [53], [93]. Both factorization methods achieve impressive classification accuracy, but this comes at the cost of increased memory consumption. The pruning methods reduce both the FLOPs and the memory consumption of the network, while maintaining high classification accuracy. However, they rely heavily on fine-tuning to achieve high accuracy. We lastly provide the results of the CaP compression optimized at each layer. Our results demonstrate that the CaP algorithm gives state-of-the-art results, has the largest reduction in memory consumption, and outperforms the pruning methods in terms of top-5 accuracy.

## 5.5 Observations

We propose cascaded projection, an end-to-end trainable framework for network compression that optimizes compression in each layer. Our CaP approach forms linear combinations of kernels in each layer of the network in a manner that both minimizes reconstruction error and maximizes classification accuracy. The CaP method is the first in the field of network compression to optimize the low dimensional projections of the layers of the network using backpropagation and SGD, using our proposed Proxy Matrix Projection optimization method.

We demonstrate state-of-the-art performance compared to pruning and factorization methods, when the CaP method is used to compress standard network architectures trained on standard datasets. A side benefit of the CaP formulation is that it can be performed using standard deep learning frameworks and hardware, and it does not require any specialized libraries for hardware for acceleration. In future work, the CaP method can be combined with other methods, such as quantization and hashing, to further accelerate deep networks.

# Chapter 6

# Feature Embedding

## 6.1 Feature Embedding Background

There are many variants of auxiliary loss functions used when training deep networks to encourage different behaviors. One of the first auxiliary losses proposed was feature regularization. The goal of regularizing the feature activations is to keep the values in the feature representation small or sparse by using $L_2$-norm or $L_1$-norm, respectively. The underlying assumption is that small-valued or sparse feature representations generally reduce overfitting. $L_2$ regularization encourages activations with small magnitudes

$$L_2 = \|G(\mathbf{x}\mathbf{W} + \mathbf{b})\|_2^2 \tag{6.1}$$

where $G(\cdot)$ is an activation function, $\mathbf{x}$ is the layer's input vector, $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ is the bias vector. $L_1$ regularization encourages sparsity in the activations

$$L_1 = \|G(\mathbf{x}\mathbf{W} + \mathbf{b})\|_1 \tag{6.2}$$

The use of these regularization techniques has waned due to reliance on robust training methods such as Batch Normalization [64] and Dropout [133]. Batch Normalization reduces the covariance shift and potential for vanishing or exploding gradients by normalizing the activations from each layer to be zero-mean and unit-variance. Dropout on the other hand reduces the potential for co-adaptation of neurons by randomly setting a fraction of neurons to zero on each training iteration, thus, forcing neurons to be more self-reliant. Though these techniques have improved training time and network accuracy, they do not address generalizable feature representations.

Recent work introduced an auxiliary function called the center loss [150], that increases the robustness of the feature representation by encouraging tightly grouped clusters. The center loss represents the average distance of each point $\mathbf{x_i}$, in feature space, to the mean $\mathbf{c_{y_i}}$ of the corresponding class $y_i$.

$$L_C = \frac{1}{2m}\sum_{i=1}^{m}\|\mathbf{x_i} - \mathbf{c_{y_i}}\|_2^2 \tag{6.3}$$

Where $m$ is the number of samples in the mini-batch, $\mathbf{x_i}$ is the feature space representation for the $i^{th}$ sample and $\mathbf{c_{y_i}}$ is the center for the class $y_i$ of the $i^{th}$ sample.

The tight Euclidean clustering in feature space encouraged by the center loss is useful in situations where the feature representations are compared to estimate similarity between samples, such as is done with k-Nearest Neighbors (k-NN). The work in [150] focused on person re-identification, a problem that requires robust feature representations that can be compared using Euclidean distance metrics.

More recently [111] introduced contrastive center loss, a technique that encourages tight clustering and increases class separation. The contrastive center loss is given as:

$$L_{CtC} = \frac{1}{m}\sum_{i=1}^{m}\frac{\|\mathbf{x_i} - \mathbf{c_{y_i}}\|_2^2}{\sum_{j=i,j\neq y_i}^{k}\|\mathbf{x_i} - \mathbf{c_j}\|_2^2 + \delta} \tag{6.4}$$

where $\delta$ is a small value that ensures the denominator is non-zero. Our work builds upon the center loss and contrastive center loss for better feature clustering and more robust performance.

Like many other methods in the field, both [150] and [111] use the Euclidean, $L_2$-norm as a similarity metric between deep feature vectors. This use of the $L_2$ norm can be problematic when applied to arbitrary feature spaces. The Euclidean norm is intended to operate in $\mathbb{R}^N$, represented by an orthogonal basis. However, the $L_2$-norm is often applied to vector representations without an orthogonal basis. We propose to produce feature representations with an $L_2$-norm similarity metric. This behavior is critical for many applications that use feature similarity, such as k-NN and other graphical methods. We ensure orthogonality by retracting the weight matrix of our feature representation layer to the Grassmann manifold, the set of orthogonal spaces in $\mathbb{R}^N$.

Learning feature embeddings using deep neural networks is a fast-growing area of research largely because of the number of potential applications that such a universal feature embedding network could have. To this point the feature embeddings learned using deep neural networks have largely been treated as naturally emergent behavior, i.e. the network will learn optimal feature embeddings based on the difficulty of the task on which it is trained. We suggest that this is not always the case and that often neural networks will learn sufficient features and not the most widely applicable, as demonstrated by their tendency for over-fitting. We also suggest that by using the responses from the last layer of a deep network, especially when the activations are used after being sent through a rectified linear function, is not the best approach for feature representations.

The research in the area of learning feature embeddings for images has largely focused on application to zero-shot or one-shot learning problems. Zero-Shot learning is a technique where a feature extraction method, often a deep network, is pretrained to learn a feature embedding that is semantically meaningful. i.e. images of similar objects are closer in feature space than images of different types of objects. Often with zero-shot learning, image embedding is mapped to a semantic space, such as Word2Vec [99]. Once the image embedding is aligned with the semantic space, zero-shot learning methods can make predictions for object categories outside of the training set. This is done by first using the deep neural network to generate the feature embedding for the image. Then the closest label in the aligned semantic feature space is chosen as the class label [35].

One-shot learning is a variant of the zero-shot learning problem, except that a single exemplar is provided for each class. This makes the problem significantly more tractable, as it relies on the semantic structure of the image embedding and does not also require alignment between the embedding spaces of the image and text modalities.

One of the most popular methods for zero-shot learning is the DeViSE [35] method. The DeViSE method learns an image embedding that is similar to the embedding of the image labels by fine-tuning a pretrained network to learn a dense feature embedding instead of the original classification task. That method achieved impressive results, however there is room for improvement. Enforcing the embedding space to have orthogonal basis could potentially use a simpler Euclidean loss function, instead of the rank loss formula.

## 6.2 Learning Deep Feature Embeddings on Euclidean manifolds

In its simplest form a CNN consists of a feature extraction convolutional network followed a linear classifier at the head of the network. One benefit of CNNs is that they are trained in an end to end manner, thus the maximum benefit can be extracted from each stage. However, the features learned by CNNs can be further improved for robustness. A robust feature representation is one that minimizes differences between samples of the same class and maximizes differences between samples from different classes.

We present a method referred to as DEFRAG [101], inspired by Linear Discriminate Analysis (LDA) [33]. Our approach shapes the feature representation through a novel auxiliary loss function, described in Section 6.2.1, and a constrained optimization problem that involves the Grassmann manifold, described in Section 6.2.2.

An auxiliary loss, $L_{aux}$, is secondary metric that is added to the loss from the main training objective, $L_{class}$, for the optimizer to minimize, as

$$L = L_{Class} + \lambda_{aux} L_{aux} \tag{6.5}$$

Where $\lambda_{aux}$ is a mixture parameter used to balance the impact of the auxiliary loss. In this work the main loss objective is the traditional categorical cross-entropy loss which learns to classify the samples, and a proposed new auxiliary loss function, the Silhouette Loss. Figure 6.2 illustrates the clustering characteristics of DEFRAG compared to other methods. The contributions of this approach are the following: We introduce a new auxiliary loss functions based on the Silhouette clustering metric which encourages tight intra-class clustering and inter-class separation. We propose an orthogonalization step which retracts the optimized feature projection matrix back on the Grassmann manifold.

The DEFRAG method consists of two components: an auxiliary loss component, and a retraction of the feature projection on the Grassmann manifold. The auxiliary loss is designed to encourage better feature clustering of samples based on their class labels, while the Grassmann manifold retraction ensures the features are in a space suitable for the $L_2$-norm similarity metric. A side benefit of our DEFFRAG training approach is that due to the robustness of the features generated, smaller networks can be used. The two components of DEFRAF are discussed next.

### 6.2.1 Clustering Auxiliary Loss

Robust feature representations are important for classification, as they increase the classifier's ability to generalize across different datasets and operating conditions. We formulate robust feature representations by seeking a feature space that encourages tight clusters for samples in the same class and large separations between clusters from different classes, for some similarity metric. Training deep networks with only the classification loss does not inherently encourage feature clustering.

Our auxiliary loss function is the Silhouette loss $L_{sil}$ shown below:

$$L_{Sil} = \frac{1}{m} \sum_{i=1}^{m} \frac{\|\mathbf{x_i} - \mathbf{c_{y_i}}\|_2^2}{\arg\min_{j \neq y_i} \|\mathbf{x_i} - \mathbf{c_j}\|_2^2 + \delta} \tag{6.6}$$

The Silhouette loss is inspired by the Silhouette score [116], which is used to assess clustering performance. The Silhouette loss is different from the center loss and contrastive center loss in that is focuses on separating classes that are close to each other, instead of maximizing the overall class separation. This criterion is important because it forces the network to focus on classes that are hard to separate and results in better classification performance. To reduce computation, we use a running average method to update the class centers as suggested in [150].

### 6.2.2 Grassmann manifold Retraction

The set of orthogonal spaces in $\mathbb{R}^N$ form a Grassmann manifold. We therefore formulate our feature learning process as an optimization problem on the Grassmann manifold. Our optimization process uses the Proxy-Matrix method described in Chapter 3. In this instance we optimize our classification and auxiliary losses instead of the loss functions developed for dimensionality reduction in Chapter 4. This is done by using a linear activation function for the last layer of the network, reducing it to a linear projection with projection matrix $\mathbf{M}$. The update step of the Grassmann optimization is done using the PMO method. This process enforces features that satisfy the Silhouette auxiliary loss criterion and reside in an orthonormal space.

## 6.3 Feature Learning Experiments

We performed a series of experiments by training a deep neural network using the proposed DEFRAG method and compared the results with the state of the art deep

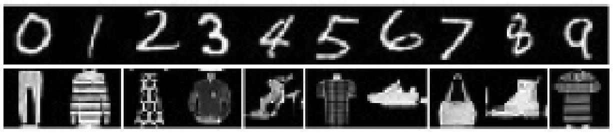


Figure 6.1: Example images from MNIST (top row) and Fashion MNIST (bottom row).

networks. The network architecture of our choice is small network with only two convolutional layers and three fully connected layers. The network parameters are given in Table 1. This network choice is intentionally simplistic so that the benefit of the DEFRAG method is made apparent.

Table 6.1: Network Architecture

| Stage | Layer Type | Size |
|---|---|---|
| Convolution Stage 1 | Conv | 32 (5x5) |
| | Pooling | 2x2 |
| Convolution Stage 2 | Conv | 256 (5x5) |
| | Pooling | 2x2 |
| Fully Connected | ReLU | 256 |
| Feature Representation | ReLU/Linear | 8 |
| Output | SoftMax | 10 |

We used standard classification datasets in these experiments including: MNIST [81] and Fashion MNIST [154]. Example images of the datasets used are shown in Figure 6.1. In Section 6.3.1 we show quantitative comparisons of the proposed DEFRAG method to standard deep networks on the Fashion MNIST dataset. Section 6.3.2 presents qualitative results of feature clustering with the MNIST [81] dataset.

### 6.3.1 Fashion MNIST Experiments

The Fashion MNIST dataset [154] was developed as a significantly more challenging alternative to the original MNIST dataset [81]. Like the original MNIST dataset, Fashion MNIST dataset consists of 60,000 training samples and 10,000 test samples from objects in 10 different classes. The objects in the Fashion MNIST set are ten different articles of clothing: T-Shirt/Top, Trouser, Pullover, Dress, Coat, Sandals, Shirt, Sneaker, Bag, Ankle boots. This dataset is significantly harder than the original MNIST dataset. In crowd-sourced experiments on a subset of 1000 examples, humans were only able to achieve 83.5% accuracy. The current state of the art performance on the dataset without augmentation is only 93.7%. Examples form the Fashion MNIST dataset are shown on the bottom row of Figure 6.1 and the experimental results are summarized in Table 6.2.

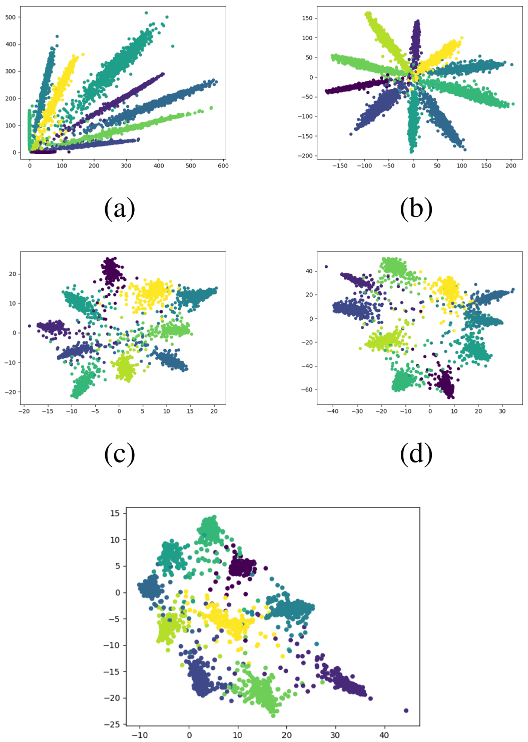
Our experiments with the Fashion MNIST dataset illustrate the impact on classification accuracy of both DEFRAG components, the Silhouette loss and orthogonalization step. We first trained the network using only the classification loss traditional ReLU activation function to get a baseline, Sparse (ReLU) in the table. We then considered a linear activation function for the feature representation layer, as well as Silhouette, Center and DEFRAG. The results in Table 6.2 demonstrate that the DEFRAG method outperforms the other methods with a relative reduction in error of 7% compared the original network. The results also demonstrate a significant improvement using just the Silhouette loss, however, DEFRAG shows improvement as a result of the orthonormal feature space.

Table 6.2: Fashion MNIST Results

| | Network | Accuracy | Parameters |
|---|---|---|---|
| Other Works | GoogleNet [17] | 0.9370 | 5M |
| | VGG16 [18] | 0.9350 | 26M |
| | HOG+SVM [19] | 0.9260 | N.A. |
| | Human [20] | 0.8350 | N.A. |
| Our Network | Sparse (ReLU) | 0.9347 | 1.4M |
| | Linear | 0.9369 | |
| | Silhouette | **0.9375** | |
| | Center | 0.9371 | |
| | Contr. Center | 0.9368 | |
| | DEFRAG | **0.9393** | |

Our experiments demonstrate that state of the art results can be achieved on the Fashion-MNIST dataset with a simple network that benefits from a feature space clustering regularization technique. The accuracy achieved by our network is better than the results achieved with the much larger GoogLeNet [136] and VGG16 [130] architectures. In comparison with the other two architectures, our network has 24 and

1.4 times fewer parameters, respectively. These comparisons are based on the updated results reported in [154].



(a) (b)

(c) (d)

(e)

Figure 6.2: Visualization of feature representations in 2D trained on the MNIST dataset using a combination of Classification Loss and Auxiliary Loss. Features are learned using (a) Softplus activation from classification loss only; (b) linear activation function; (c) center loss [150]; (d) contrastive center loss [111]; (e) our DEFRAG method.

### 6.3.2 Qualitative MNIST Experiments

The MNIST dataset serves as a useful tool in understanding the behavior of the network. In this chapter, we qualitatively investigate the clustering behavior of the proposed method. To visualize the clustering behavior the dimensions the feature representation was reduced to two, we replaced the fully connected layer with a layer that had only two neurons, and the resulting features were plotted in Figure 6.2. The X-axis corresponds to the response of the first unit and the Y-axis corresponds to the second unit.

A few observations can be made on these results. Firstly, it is clear that the SoftPlus activation function is not ideal for classification. This was confirmed by the results where the SoftPlus implementation achieved only 88.6% accuracy on the test set while the other methods achieved classification accuracy of over 99.0%.

Secondly, the plots of Figure 6.2 show that auxiliary cluster losses have a significant impact on the clustering behavior of the network. The linear implementation forms linearly separable clusters for each class, however, the intra-class variance is much higher than the inter-class separation. We measured the separability of each method by both the Silhouette score or the ratio of average inter- to intra-class distances. In both of these metrics the DEFRAG method outperformed all other methods. The DEFRAG method outperformed the center loss, the second-best method, with a 16% reduction in the Silhouette score and 13% in the distance ratio.

## 6.4 Remarks

We believe the area of deep feature embeddings offers a lot of potential for advancements through application of the Grassmann concepts proposed in this work. One avenue for additional research is finding new auxiliary loss function that encourage different types of behaviors of the feature embeddings learned with deep neural networks. These image embeddings can be useful in many applications where labeled data is limited such as visual object tracking and domain adaptation. Another application of interest is using the DEFRAG method in the area of out of distribution detection. The detection of input samples that are from a significantly different distribution than those those in the training set is important to provide a confidence of a networks prediction. If the feature representation extracted from the network is significantly different than the features from the training set the network should not make a prediction at all, because it will likely be wrong. We believe that the improved clustering and reduced feature manifold distortion of DEFRAG makes it well suited for this problem.

# Chapter 7

# Domain Adaptation

Domain Adaptation (DA) is process of adapting a classifier pre-trained on a specific dataset, the source dataset, to a separate dataset, the target dataset. The problem domain adaptation methods attempt to mitigate is known as the covariance shift, the difference between the data in different datasets. Covariance shifts can result from many different sources: difference between imaging sensors, image resolution, environmental conditions, and view point to name a few. The most common instance where DA is used is in situations where there is a large amount of labeled training data from a specific data source. However, the classifier is intended to be used on data from a different dataset that does not have as much labeled data. In these instances a classifier can be trained in the source dataset, with a large amount of labeled training data. Then the pretrained classifier can be adapted to the unlabeled target domain, the data the classifier is intended to be used on, using a DA method.

Domain adaptation has gained increased interest in recent year because of the importance of adaptability for methods that are deployed in real world applications. In the past 10 years computer vision and machine learning methods have been applied in many more commercial products including automatic copyright detection on YouTube [63], automatic face identification by Google [123], and traffic monitoring from Unmanned Aerial Vehicles (UAVs) [69], to name a few. A key factor that determines if methods are ready to commercial application is their robustness/adaptability to new data from potentially different domains. Many modern datasets such as IMAGENET [25] and MS-COCO [88], attempt to minimize the impact of covariance shift by having a large variety of samples in the training set that cover many of the environmental conditions faced by real world applications. However, it is difficult to fully capture all of the different conditions or imaging sensors encountered in the real world.

## 7.1 Domain Adaptation Background

Due to the importance of robustness and adaptability to new data for commercial applications there exists a large body of work in the area of Domain Adaptation (DA). This section provides background on the most important methods related to this work. The approaches to domain adaptation can be grouped into two primary categories: supervised and unsupervised methods. Supervised methods adapt to a new domain by using a small set of labeled data in the target domain. This is sometimes referred to as semi-supervised training depending on the size of the adaptation set. During unsupervised DA, the classifier must adapt to a new dataset without any knowledge of class labels for the target samples.

The choice of feature representation used the source and target data is crucial for robust domain adaptation. If features are not similar across domains it will be difficult to transfer knowledge from the source domain to the target domain. While traditional features, such as Scale-Invariant Feature Transform (SIFT) [92] and Histogram of Gradients (HoG) [22], have been used in multiple domain adaptation works. As with many other areas of computer vision and deep learning most recent domain adaptation works focus on adapting the features extracted using deep neural networks.

Fine-tuning a pre-trained network is the most commonly used method for supervised domain adaptation, [28]. However, fine-tuning based methods often face the problem of overfitting to the small adaptation set. Thus, most work in the field of supervised domain adaptation has focused on developing an adaptation methodology that allows the network to adapt to the new data without over-fitting. Various strategies have been considered to minimize overfitting including training only a subset of the network layers, reducing the learning rate, and introducing dropout to increase regularization. The conclusion is that there is no single optimal approach for fine-tuning a network. Each adaptation problem is unique and thus the optimal approach depends on the specific conditions.

This work primarily focuses on the unsupervised domain adaptation problem, because this problem is more challenging and most commonly faced when real world situations. Unsupervised domain adaptation is a particularly difficult task, as there are no labels for the target domain data that can be used to retrain the classifier. There have been a large variety of methods proposed for domain adaptation which take different approaches to solving the adaptation problem. In this section we categorize the methods based on which part of the feature extraction and classification pipeline they alter to adapt to the target domains. Each method can be placed into one of four categories: methods that adapt the feature extractor, methods that adapt the feature after extraction, methods that retrain the classifier, and lastly method that alter both the feature extractor and the classifier.

Domain adaptation methods that adapt the feature extraction method aim to increase the similarity of the features between the domains [37], [146], [38], [87]. This group can further be broken down into two sub-categories based on learning symmetric vs. asymmetric feature mappings for the source and target domains. Methods which learn symmetric feature mapping for source and target domains, such as [37], postulate that better features are learned by training on both domains simultaneously rather than on each domain separately. However, works by [75] and [135] have shown independently that a symmetric feature mapping/extraction is likely not optimal. Instead it is suggested that learning an asymmetric feature extractor for the source and target domains produces features that are better suited for domain adaptation. Thus, methods such as ADDA [146] and deep CORAL [135] learn asymmetric feature extraction methods that result in similar features from source and target domains.

While the first group of domain adaptation methods primarily focuses on deep learning base applications, the second group can be applied to both traditional hand-crafted features and deep features. This group alters the features after they have been extracted. These methods learn a transformation that maps both the source and target features into a unified feature space. The first method to generate a linear transformation from the target feature domain to the source feature domain was the domain invariant transform method [120]. Another method that maps source features space to the target feature space was Grassmann Subspace Sampling [42]. This method interpolates a series of feature subspaces between the source and target domains on a Grassmann manifold to transfer classification knowledge from the source to target domain. An asymmetric transformation of the target features to the source domain is learned in [40] by using landmarks in the source domain to transfer to the target domain. Gong et al. suggest that it is not always the best solution to use all the source data to train the feature transformation. Instead, an iterative training method is proposed where a subset of the source samples are selected for training which closely match the target distribution. Thus, during the course of training the subset of the source samples that is used for training grows as the two feature spaces become more aligned. These methods do not require any retraining of the classifier because the target features are adapted to match the source features.

The third group of domain adaptation methods leave the features unchanged, instead focusing on retraining the classifier based on the small set of label data in the training domain. The Adaptive-SVM [157] is an adaptation method which trains Support Vector Machines (SVMs) to maximize the classification accuracy in both domains, to minimize overfitting the limited labeled set in the source domain. The method of fine-tuning a pre-trained deep network [28] is a popular adaptation technique for deep neural networks. The premise behind fine-tuning a neural network is that if a training dataset is sufficiently large and diverse, the features learned of the course of training should be broadly applicable irrespective of target classification task or imagery. Thus, Decaf [28] proposes that by retraining only the last few fully connected layers of a neural network, the network can be adapted to a new task or imagery.

The fourth and final group of domain adaptation algorithms is a group that alters both the features extraction and classification components of the pipeline. The most common approach in this group is to learn a feature transformation from the two domains to a uniform subspace then learn a classifier based on the labeled examples in the unified subspace, such as [126], [32]. A popular DA method is Subspace-Alignment [32], which aligns the principal components for the source and target features. In SA, separate subspaces $\mathbf{R_s}$ and $\mathbf{R_t}$ are learned for the features extracted from the source and target domains, respectively. In [32] the subspaces were extracted using principal component analysis as:

$$\mathbf{R}^*_{\mathbf{S}} = \arg\max_{\mathbf{R_S} \in \mathcal{G}} (\|\mathbf{X_S}\mathbf{R_S}\|) \tag{7.1}$$

for the source domain and:

$$\mathbf{R}^*_{\mathbf{T}} = \arg\max_{\mathbf{R_T} \in \mathcal{G}} (\|\mathbf{X_T}\mathbf{R_T}\|) \tag{7.2}$$

for the target domain. The low-dimensional representation of the target features are then generated using:

$$\mathbf{S_T} = \mathbf{X_T}\mathbf{R}^*_{\mathbf{T}} \tag{7.3}$$

However, the subspace of the source features, $\mathbf{R_S}^*$, is not used directly to generate the aligned low-dimensional representations. Instead, the features are aligned with the target domain using:

$$\mathbf{S_S} = \mathbf{X_S}\mathbf{R}^*_{\mathbf{S}}{\mathbf{R}^*_{\mathbf{S}}}^T\mathbf{R}^*_{\mathbf{T}} \tag{7.4}$$

Once the source subspace is aligned to the target subspace a classifier is trained based on the labeled source domain samples. It was found that transforming the source feature to the target domain made the classifier more robust to novel examples from the target domain. More recently methods have been developed which adapt the feature extractor and classifier simultaneously, such as [21] and [28].

### 7.1.1 Deep domain adaptation

Due to the prevalence of deep learning techniques in the fields of machine learning and computer vision it is particularly important to focus on methods developed for deep learning based domain adaptation. One of the first deep learning based domain adaptation methods to be developed was fine-tuning [28], a supervised domain adaptation method which uses a small subset of labeled samples in the target domain to adapt a pretrained network. However, most recent works have focused on the problem of unsupervised domain adaptation for deep learning, with many methods adapting the feature extraction network. Some deep learning methods match the statistical distribution of source and target features explicitly [134], while other use different loss functions to adapt the feature extraction network in Deep Domain Confusion [147], Deep Adaptation Networks [91], and Deep Transfer Networks [162]. Each of these methods require the target data at training time, even though they do not have access to the target domain labels. This is different from the method proposed in this work which aims to adapt a network already trained on the source domain to a new target domain.

Recent work in the area of deep domain adaptation has focused on matching the statistical distributions of the feature from the source and target domains. The primary difference between these distribution matching methods are the different similarity metrics used to measure the similarity between the feature distributions. Sun et al. [134] for example propose using second order statistics to “whiten” and “recolor” the target features to match the source features by minimizing what their proposed CORAL loss. This method was then extended in [135] to retrain the entire target feature extraction network based on minimizing the CORAL loss. Adaptive Batch Normalization (ABN) [87] continues to update the statistics in batch normalization layers for the target data inorder to leverage the inherent whitening of batch normalization layers. By doing so ABN effectively normalizes the features in the source and target domains in a simple easy to implement manor. In [34] the authors propose the association loss function as a better similarity metric than the Maximum

Mean Discrepancy that is commonly use. In [45] the authors propose a student-teacher network configuration for training on the source and target domains. The student network (source domain) is trained to on both classification and embedding similarity and the teacher network (target domain) is trained to produce similar embeddings.

Adversarial Learning is a growing area of deep learning that recently received a lot of attention [41]. Adversarial methods iteratively training two networks with opposing objectives to learn optimal feature representations. These networks were originally developed to randomly generate synthetic imagery that was "believable" or looking similar to actual imagery. These networks have been used for domain adaptation, first in DANN [37], [38] and later in ADDA [146]. Adversarial learning is used for domain adaptation to learn feature representations for the source and target domains that are indistinguishable, and therefore theoretically robust across domains. The primary difference between these two methods is that whether or not the weights are shared for the source and target feature extraction network. The DANN method uses a single symmetric feature extraction network, where the same weights are used to extract the features form the source and target domains. Conversely, the ADDA method uses an asymmetric feature extraction configuration, where separate feature extraction networks are trained for the source and target domains.

Another group of domain adaptation techniques are known as Self-learning domain adaptation methods. Self-learning domain adaptation methods are closely related to the unsupervised learning algorithm proposed by Yarowsky [159]. These methods assumes that the source and target domains are close enough so that the classifier itself could be used to retrain on the target domain. These methods use "pseudo-labels" [83] to bootstrap the network and adapt based on its own predictions. Asymmetric Tri-training (ATT) [121] relies on a similar method for generating labels for the data except the labels are generated based on the results from three independently trained networks, to increase the accuracy of the "pseudo-labels". Both techniques use Inductive-Learning (IL) to adapt to the source domain.

A different subcategory of self-learning techniques uses Transductive-Labeling (TL) to adapt the network to the new domain. Transductive-labeling transfers the label from a sample in the source domain to a sample in the target domain to retrain the network. A common transductive-labeling approach is to use the dominant label of the closest K sample from the source domain, in feature space, to each sample from the target domain [125]. Information-Theoretical Learning of Discriminative Clusters (ITLDC) [128] proposes a transductive learning method that adapts both the feature space representation and the classifier, based on the clustering characteristics of the target data. ITLDC assumes that optimal features for both the source and target domain will generate tightly grouped clusters, based on class boundaries. Additionally, the Shi et al. assume that the clusters for the source and target domains will exist in close geometric proximity, thus this behavior can be used to train the network for the target domain. These techniques work well when the source and target domains are relatively similar, i.e. the geometric distance between the source and target features are close but fail when there is a stark difference between the source and target domain.

## 7.2 Domain Adaptation Proposed Work

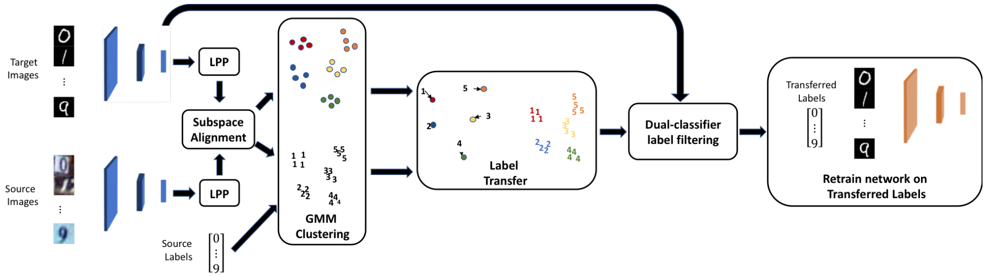


Figure 7.1: Overview unsupervised domain adaptation of deep network to the target domain. Adaptive batch normalization is used for training and adaptation in both source and target domains. Subspace alignment is performed for source and target features on the LPP manifold and the features are clustered to determine if label transfer is appropriate based on a clustering criterion. Label transfer is performed by assigning labels from the closest source cluster to each target cluster and using them to retrain the network.

We propose manifold-Aligned Label Transfer for Domain Adaptation (MALT-DA) [103] a new approach for unsupervised deep domain adaptation based on cluster association between source and target domains, as outlined in Figure 7.1. The first step in our approach is to use Adaptive Batch Normalization [87] to produce features that are similar between source and target domains. The features are then clustered in to groups, with each group ideally consisting of only elements of a specific category. The distributions of these groups are compared using the silhouette distance [117] as a clustering metric to determine if the target features are well suited for label transfer. The second step of our method transfers the labels from the source clusters to the target clusters to provided labels to train the classifier on target domain samples. Each step of the MALT-DA method is discussed in the rest of this section.

### 7.2.1 Center-loss feature training

The standard method for training a neural network is to use the cross-entropy loss of the output of the SoftMax layer and the one-hot encoded ground-truth label vectors. This loss encourages feature representations, layer before the SoftMax layer, that are linearly separable, however, this does not encourage tight clustering of the classes. Well-clustered feature representations are important to insuring generalization of a classifier across domains. These tight clusters are particularly important for domain adaptation methods, such as the one proposed in this work, that use the clusters behavior to adapt to the target domain.

In order to demonstrate how the SoftMax loss alone does not encourage well clustered features we trained a network with a two-dimensional feature representation and plotted the feature responses directly. We compare the SoftMax-trained features with features resulting from a network that was trained with an auxiliary loss that encourages tightly clustered features, the center-loss [150]. The center-loss was previously proposed for learning better spatially consistent feature representations for tasks that rely on distance comparisons between features, i.e. k-NN matching for human face identification. In this work we leverage the center-loss to learn better feature representations that are more well-suited for our domain adaptation method.

Quantitative results comparing the SoftMax and Center-loss networks trained on the MNIST [82] dataset are shown in Figure 7.1. Though both the SoftMax and Center-loss networks achieved the same classification accuracy on the MNIST test set, 99.65%. There were significant differences in the feature representations when inspecting the cluster separation metrics in Table 7.1. In this table the accuracies for each network are presented as well as the ratio of the distance between each point to its class center and all other class centers. A better metric of cluster separability, the silhouette score [117], is the ratio of the average distance of each point to its class center and the closest class center that the point does not belong to.

In Table 7.1 we present the Inter-Intra class ratio and silhouette score because they are good indicators of the difference in class separation between the networks. The center-loss network clearly outperforms the SoftMax network with a greater than 30% improvement in each metric. This translates to each point being on average 46 times closer to its class center than all other classes, and 37 times closer on average than

Table 7.1: Quantitative results for SoftMax and Center-loss networks with 2D feature representations.

| | MNIST-Network | |
|---|---|---|
| | softmax | center |
| Accuracy on MNIST Test-set | 99.65% | 99.65% |
| Inter-Intra class ratio. | 0.0312 | 0.0216 |
| Silhouette Score | 0.0382 | 0.0265 |
| Accuracy on USPS Test-set | 91.69% | 96.60% |

the closest point, whereas the separation for the SoftMax network is 32 and 26 times respectively.

We propose that adding one of our auxiliary loss functions proposed in Section 6.2, such as the silhouette loss, to the SoftMax loss while training our network on the source domain, we learn features that have a higher separability than the SoftMax loss alone. We suggest that the increased separability positively impacts the network's ability to generalize across domains. This hypothesis is confirmed by the data in the last row in Table 7.1 where we present the test accuracy of the networks, un-adapted, on the USPS test-set. The center-loss network has a relative reduction in error rate of 59%.

### 7.2.2 Adaptive Batch Normalization

The first and often most important step in domain adaptation is to adapt the features for the target domain to match those of the source domain. This could be done in two ways: adapt the feature extraction network, or align the features after extraction. The work in [146], [38] focuses on adapting the feature extraction method for the target domain in a way that the two sets of features are in the same subspace. In this work we opt to align the features from the two domains using two alignment techniques. Firstly, we normalized the features throughout the network using adaptive batch normalization, as done in [87]. Then, after extraction, we aligned the dimensionality reduced feature spaces using subspace alignement [32]. This method was selected because it allowed for quick alignment of the two domains and we preferred to alter the feature extraction network as part of the classifier update procedure instead of the feature alignment stage. We also found retraining the feature extraction network using adversarial training, as was done in [146], [38], proved to be a delicate process that could easily fall into a degenerate state.

In our framework, we do not explicitly retrain the feature extraction network, but instead we rely on adaptive batch normalization for a more effective method of ensuring that the source and target features exist in the same subspace. The difference between ABN and traditional batch normalization is that normally the whitening statistics are learned at training time and frozen at test time. However, as suggested in [87], by allowing the batch normalization layer to continue to adapt at test time one can achieve a domain adaptation without any additional training. The process of normalizing the activations in each layer of features for the mini-batches inherently aligns the distributions of the source and target domains, such that both have a zero-centered normal distribution. This approach has improved results on domain adaptation, without requiring any parameter tuning or re-training. Our results in Section 7.3.2 demonstrate how significant a role ABN plays in our domain adaption method, by reducing the classification error on the target data by an average of 30% without any retraining.

### 7.2.3 Subspace Alignment on the LPP manifold

In our work, we found that while ABN was useful for feature conditioning, the features of the source and target domains were not closely aligned. In our initial investigation [100] we used a variation of Subspace-Alignment [32] for aligning the subspaces of the two domains based on the basis vectors obtained using PCA [109], [54]. We found that the combination of dimensionality reduction and subspace alignment both positively impacted the clustering behavior for each domain and improved the cluster-correspondence between domains.

There is, however, one limitation to using PCA for dimensionality reduction. Since the objective of PCA is to maximize the variance of the overall data reconstruction based on the top principal components, the local neighborhood structure of the data is ignored, this can adversely affect local clustering. In this work, we utilize LPP which focuses on preserving the local structure of the data. The local structure of the data is encoded in a graph where the nodes represent each data point and the weighted edges between the nodes represent relative interactions based on some distance between these data points. There are several variations on methods that form this graph, e.g. one forms edges between nodes if their Euclidean distance is below a certain threshold, while another uses a k-Nearest Neighbor (k-NN) search to find the closest k (k=15) neighbors for each node and forms edges between them. Additionally, there are two methods for calculating the edge weights for the graph: the simplest method is a binary weighting scheme, where all edges between nodes that are connected based on the previous criteria are given a weight of 1 and all other edges are assigned a weight of 0. A more advanced method of determining the edge weights uses a heat kernel to calculate the edge weight $w_{i,j}$ based on the relative distance between two samples, $x_i$ and $x_j$, as follows:

$$w_{i,j} = e^{\frac{\|(x_i - x_j)\|}{\tau}} \tag{7.5}$$

Where $\tau$ is a constant parameter that is tuned to the given dataset. In this work, we selected a simple binary weighting scheme based on the k-NN for each node, because this required the fewest number of parameter to be tuned for each dataset.

Once the weight matrix is generated, a diagonal matrix, $D$, is formed as $D = vI$. where $I$ is the identity matrix and $v$ is a vector whose entries are the summation of the rows of the weight matrix:

$$v_i = \sum_j w_{i,j} \tag{7.6}$$

The calculation of the $D$ matrix was further simplified because of selecting a binary weighting scheme with the k-NN neighbor selection. Thus, all the rows will have k entries with weights of 1 and the remainder will be zeros. Therefore, $D$ is simply $D = kI$.

After $D$ is generated, it is used to calculate the Laplacian of the weight matrix as:

$$L = D - W \tag{7.7}$$

The Laplacian of the weight matrix is used to find the optimal linear transformation of the features, $a$, the preserves the data's local structure by solving:

$$\begin{aligned} \arg\min_{\mathbf{a}} \quad & \mathbf{a}^T X L X^T \mathbf{a} \\ \text{s.t.} \quad & \mathbf{a}^T X L X^T \mathbf{a} = 1 \end{aligned} \tag{7.8}$$

Where the columns of $X$ are made up of the features vectors of all the samples. This can be formulated as a generalized eigenvector problem:

$$X L X^T \mathbf{a} = \lambda X D X^T \mathbf{a} \tag{7.9}$$

Where $\lambda$ is the corresponding eigenvalue for each eigenvector that solves the equation. The linear transformation matrix can then be formed but selecting eigenvectors with the highest corresponding eigenvalues and forming the matrix A by ordering the eigenvectors as columns from highest eigenvalue to lowest.

The work in [163] demonstrates the improvement in clustering behavior of LPP relative to PCA or LDA for face recognition. These conclusions were later expanded to demonstrate the improvement of LPP relative to PCA for k-means clustering in [24]. Our experiments confirmed the previous results by demonstrating that clusters formed based on the LPP subspace had a higher purity than both the original features and the features in the PCA subspace. Once the transformation is calculated for the source and target domains independently, we align the subspaces based on LPP Subspace-Alignment [112]. This differs from a previous approach [100] where we slightly modified the subspace alignment method [32] to align the feature subspaces to the source domain in order to reduce the number of iterations of the clustering algorithm. In this work we found that, when using the LPP manifold, aligning the target subspace to the source subspace generally did not harm the clustering behavior and improved the cluster correspondence between the source and target clusters. We followed the procedure in [112] to calculate a transformation matrix, $M$, that optimally aligned the subspaces by solving:

$$\arg\min_{M} \|MA_S - A_T\|_F^2 - \beta\|M\|_F^2 \tag{7.10}$$

where the first term minimizes the disparity between the source transformation $A_S$ and the target transformation $A_T$. The second term is a regularization term regulated by a scaling constant $\beta$. A closed from of the solution is given as:

$$M = A_T A_S^T (A_S A_S^T + \beta I)^{-1} \tag{7.11}$$

The original, high dimensional, features from the source and target domains, $X_S$ and $X_T$ respectively, can be transformed to the low dimensional aligned subspace by:

$$Y_S{}^T = (MA_S)^T X_S \tag{7.12}$$

$$Y_T{}^T = A_T{}^T X_T \tag{7.13}$$

### 7.2.4 Feature Clustering using Gaussian Mixture Model

After the source and target subspaces are aligned, the next step in our process is to cluster the features from both domains in to C clusters, where C is the number of classes. Ideally, if the feature extraction network is robustly trained, the features in both domains will naturally form individual clusters for each of the classes. Our work leverages this assumption to improve label transfer. In [100], we used a k-Means algorithm to cluster our data. However, we found that the hard cluster boundaries produced by this method did not accurately represent all of the data. Because the network is not initially adapted to the target domain it is quite likely that the extracted features will not form tightly grouped clusters that are easily separable. Instead, clusters might be closer together with limited overlapping.

We consider a Gaussian Mixture Model (GMM) [24] to form clusters, such that each, in principle, corresponds to a unique class. We used two separate processes to calculate the Gaussian Mixtures for the source and target domains. For the source domain data, we leverage the sample labels and can thus calculate a single best fitting Gaussian for each subset based on its class labels. For the target domain, however, we don't have the labels to separate the data. Thus, we fit a GMM with C components, where C is the number of classes. The GMM uses an Expectation-Maximization (EM) [24] procedure to determine the components of the Gaussian Mixtures. During this process the optimal mean (cluster centers), covariance (shape), and weights of each of the C Gaussian mixtures must be calculated. We used K-Means clustering to provide initial parameters for the Gaussian mixtures. The first step of the EM algorithm, the expectation step, where the probability of each point, $x_i$, being a member of each cluster, $C_j$, as $P(x_iC_j|x_i)$. After the cluster probabilities for each of the points are calculated the mixture parameters are recalculated based on the new probabilities. This process is iterated until the algorithm converges to a stable solution.

In an effort to simplify the cluster correspondence process between the two domains, we impose an additional constraint on the GMM estimation. We enforce diagonal covariance matrices, i.e. we assume that the dimensions of the features in the LPP space are independent and the covariance between them is 0. We found that imposing this constraint had little impact on the cluster purity relative to an unconstrained covariance matrix. This does, however, significantly reduce the complexity of the cluster correspondence calculations, which makes it worthwhile to impose this constraint.

### 7.2.5 Dual-Classifier Pseudo-Label Filtering

In [100] a test was performed to determine the fitness for adaptation of a given set of target domain samples prior to doing any domain adaptation. The test was a binary decision based on the silhouette score [117] of the target dataset. This was done to determine if the features extracted from the target dataset formed well defined clusters that could be used to transfer labels from the source to target datasets. We found that if the target data did not form well defined clusters in the feature space, our method would degrade classification performance. Thus we developed a test prior to running adaptation to terminate the process if it would not be beneficial.

The next step in our work is to develop a domain adaptation method that would work in cases where the target domain forms ill-defined clusters. To do this we replaced the binary yes/no perform adaptation test with a data-filtering method to only adapt based on confident samples. This procedure is inspired by the approach in [121] where they required the agreement of two independent classifiers for the pseudo-labels to be used to train the adapted target network. In our work instead of training two separate classifiers, we propose using the original source classifier as one source for the pseudo-labels and the cluster assignment as the second source. If the pseudo-labels from each classifier match then the target sample is used for adapting to the target domain. If the labels do not match then the sample is not used for this adaptation process.

Our approach is novel because instead of relying on two classifiers of the same type, it uses two different types of classifiers, each focusing on different aspects of the data. The original classifier is trained to learn a linear separation of the high dimensional feature representations, whereas the second classifier is trained on the clustering behavior of the subspace aligned lower dimensional feature representations. When these two methods of classification agree, there is a high confidence that the target data is well-aligned with the source data, and thus the pseudo-label for the target data is correct.

### 7.2.6 Label Transfer via manifold clustering

The focus of our work is on retraining the classifier to adapt to the target domain. Unlike many previous works in unsupervised domain adaptation which focus solely on aligning the distributions of the high-dimensional feature representations of the source and target examples. Previous works assume that if the target and source feature distributions are perfectly aligned, the classifier should perform equally well on the target and source samples. Unfortunately, this is not always the case because there can be multiple transformations that align the features from the two domains. The work in [6] has shown that the optimal feature transformation belongs to the set of transformations that perfectly align the two distributions. However, there is no guarantee that the transformation used to align the two distributions is the optimal transformation. In our experience, we have found that the blindly aligning the two domains is just as likely to be destructive as it is to be beneficial to domain adaptation. The problem is that feature alignment is not enough for domain adaptation. The feature adaptation must be optimized for both aligning the two domains and the classification task. However, the difficulty is that there are no labels for the target data, thus traditional methods for retraining deep networks, such as [28], cannot be used.

Once the GMMs for the two domains are calculated, the next step is to form one-to-one correspondences between the clusters in the two domains. We propose using the Kullback–Leibler (KL) divergence [76] to determine the optimal correspondence between clusters in the target domain and clusters in the source domain. The KL divergence between two distributions is a measure of the information gain from using one distribution instead of another. If the two distributions are quite similar this information gain will be low, or 0 if the distributions are identical. If the distributions are dissimilar there will be a greater amount of information gain, thus the divergence will be higher, and completely unique distributions have a divergence of 1. It is important to note the direction of the KL divergence calculation is from the target to the source clusters, as the measurement does not satisfy the triangle inequality. Thus, the divergence between a given target cluster and a source cluster is different than the divergence between the same source cluster and same target cluster.

The KL divergence between two k-dimensional Normal Distributions, $\mathcal{N}_0$ and $\mathcal{N}_1$, with means, $\mu_0$ and $\mu_1$, and covariances, $\Sigma_0$ and $\Sigma_1$, respectively, is given as:

$$\begin{aligned} D_{KL}(\mathcal{N}_0||\mathcal{N}_0) =& \frac{1}{2}\operatorname{Tr}(\Sigma_1^{-1}\Sigma_0) \\ &+ \frac{1}{2}(\mu_1-\mu_0)^T\Sigma_1^{-1}(\mu_1-\mu_0) \\ &+ \frac{1}{2}\ln\frac{\det\Sigma_1}{\det\Sigma_0} - \frac{k}{2} \end{aligned} \tag{7.14}$$

This can be simplified because of the diagonal covariance matrix constraints imposed on the GMM process. The first component can be simplified as:

$$\operatorname{Tr}(\Sigma_1^{-1}\Sigma_0) = \sum_j \frac{\Sigma_{0,jj}}{\Sigma_{1,jj}} \tag{7.15}$$

Secondly the inverse of the covariance matrix $\Sigma_1$ can be calculated by inverting the elements along the main diagonal,

$$\Sigma_1^{-1} = \begin{cases} \frac{1}{\Sigma_{1,ij}} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{7.16}$$

Thirdly, the last component can be replaced, as follows:

$$\ln \frac{\det \Sigma_1}{\det \Sigma_0} = \ln \frac{\sum_i \Sigma_{1,i}}{\sum_i \Sigma_{1,i}} \tag{7.17}$$

The above results in the simplified KL Divergence equation used in this work:

$$\begin{aligned} D_{KL}(\mathcal{N}_0 || \mathcal{N}_0) =& \frac{1}{2} \sum_j \frac{\Sigma_{0,jj}}{\Sigma_{1,jj}} \\ &+ \frac{1}{2}(\mu_1 - \mu_0)^T \Sigma_1^{-1} (\mu_1 - \mu_0) \\ &+ \frac{1}{2} \ln \frac{\sum_i \Sigma_{1,i}}{\sum_i \Sigma_{1,i}} - \frac{k}{2} \end{aligned} \tag{7.18}$$

We use this KL divergence calculation to form the optimal on-to-one association between target and source clusters by greedily selecting the target and source pairs that have the lowest divergence. This method provides a significant improvement in the label transfer accuracy, which leads to improved accuracy of the final trained network using the transferred labels.

## 7.3 Domain Adaptation Experiments

### 7.3.1 Network Architecture

For our experiments, we used two separate DenseNet architectures, without the bottleneck configuration, as proposed in [57]. An example DenseNet architecture is shown in Figure 2.7. The DenseNet architecture lends itself well to ABN because it already has a batch normalization layer before each convolution operation, and the original architecture does not need to be changed to leverage ABN. Unlike the original DenseNet implementation which pre-whitened the input data, we normalized the input through a batch normalization layer.

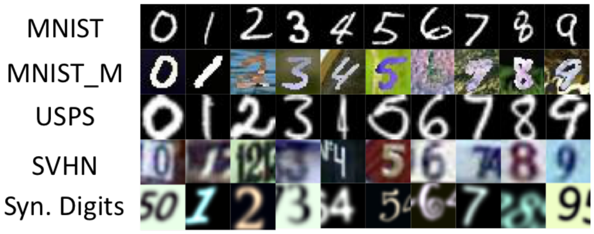


Figure 7.2: Sample images from different datasets showing variations in the same category across domains.

We further introduced a fully connected layer after the last global pooling layer in the DenseNet architecture to aid in the center-loss training by reducing the dimensionality of the feature representations. We found that this additional layer has no effect when training with the SoftMax loss but is important when training with the center loss.

For the digit classification experiments we used a smaller DenseNet that is 25 layers deep with three dense blocks and a growth rate of 12. We selected this shallower network for digit classification to reduce over-fitting, as the digits datasets are relatively small.

The network used for the aerial image classification was 40 layers deep and had three dense blocks and a growth rate of 12. The network for the aerial datasets is larger because the task is harder due to more classes, the input images have higher resolution (256x256 pixels), and the datasets are larger allowing the training of deeper networks.

The networks were trained on the source data for 300 epochs for the MNIST, USPS, MNIST_M datasets and 40 epochs on the SVHN, Syn. Digits, UCM, and AID dataset. The Stochastic Gradient Decent optimizer was used with a momentum of 0.9, a learning rate of 0.1 which was dropped to 0.01 50% through training and then again to 0.001 at 75% through training. We used a 20% dropout rate. For the digits experiments we used a mini-batch size of 256, and for the Aerial experiments we used a mini-batch size of 64.

After the label transfer procedure, the network was adapted using the filtered label for 15 epochs using the same learning rate schedule and optimizer as was done for the original training of the network.

### 7.3.2 Digit Classification

In our first set of experiments we tested the proposed MALT-DA method on digit classification across multiple datasets [82], [61], [107], [37], [38]. Examples from all 5 digit classification datasets can be seen in Figure 7.2. The MNIST [82] dataset is one of the first digit recognition datasets used in deep learning. It contains 60,000 training sample and 10,000 test samples of 28x28 black and white examples of handwritten digits 0-9. The USPS [61] dataset is quite similar to the MNIST dataset; however, the number of samples is much smaller, and the images are in gray scale.

The works in [37], [38], presented a more challenging digits dataset that was generated by combining the MNIST images with randomly cropped patches from the BSDS500 dataset [4]. Since the exact dataset used in [37] and [38] is not released publicly, we followed an identical procedure to that proposed in their works to generate a similar dataset that we refer to as MNIST_M.

The most recent digits dataset is the Street View House Numbers (SVHN) [107] dataset in which 600,000 labeled examples are extracted from RGB images, many include additional digits in the bounding box for each individual digit. The SVHN dataset is the most diverse and challenging of the digit's datasets. It is common for DA methods to work well when adapting from SVHN to a different domain. However, it is not common for methods to work well transferring to the SVHN domain because of its complexity. The SVHN dataset is a practically hard dataset because the image patches often include multiple digits, as illustrated in Figure 7.2.

Another synthetic digit classification dataset is the Synthetic Digits (Syn. Digits) dataset from [38]. The Synthetic Digits dataset was created using various computer fonts on different backgrounds with various alterations such as blur, rotations, color, etc. The dataset was deigned to be a synthetic set that more closely matches the SVHN dataset, and thus was of a similar size.

Since the MNIST and USPS datasets are single channel images, we duplicated the single channel in the MNIST and USPS data to simulate RGB imagery, so that they could be used in our experiments. Additionally, because the other three datasets have a mixture of light digits on dark backgrounds and dark digits on light backgrounds, 50% of the MNIST and USPS samples were inverted, as suggested in [34]. Note that this process was not data augmentation, because the number of samples in the dataset remained the same.

In Table 7.2 we compare the results produced by our method with the results reported by state-of-the-art DA methods on digit datasets. We consider methods that do not use data augmentation and methods without tuned parameters for each

Table 7.2: Accuracy of digit classification datasets.

| Source<br>Target | MNIST<br>USPS | USPS<br>MNIST | MNIST<br>SVHN | SVHN<br>MNIST | MNIST<br>MNIST_M | Syn. Digits<br>SVHN |
|---|---|---|---|---|---|---|
| Rev. Grad. [38] | 91.11% | 74.01% | 35.67% | 73.91% | 76.67 | 91.09% |
| DCRN [39] | 91.80% | 73.67% | 40.05% | 81.97% | - | - |
| G2A [122] | 92.50% | 90.80% | 36.40% | 84.70% | - | - |
| ADDA [146] | 89.40% | 90.10% | - | 76.00% | - | - |
| ATT [121] | - | - | 52.80% | 86.20% | - | 93.10% |
| SBADA-GAN [119] | **95.04%** | **97.60%** | **61.08%** | 76.14% | - | - |
| SE DA [34] | 88.14% | **92.35%** | 33.87% | 93.33% | - | **96.01%** |
| Assoc DA [45] | - | - | - | **95.68%** | **89.53%** | 91.30% |
| Transfer. Rep. [124] | - | - | - | 78.80% | **86.70%** | - |
| SA [32] | - | - | - | 59.32% | 56.90% | 86.44% |
| CORAL [134] | - | - | - | 63.10% | 57.70% | 85.20% |
| DSN w/ DANN [11] | - | - | - | 82.70% | 83.20% | 91.20% |
| DSN w/ MMD [11] | - | - | - | 72.20% | 80.50% | 88.50% |
| MMD [91] | - | - | - | 71.10% | 76.90% | 88.00% |
| Source | 36.05% | 10.28% | 32.27% | 93.68% | 58.53% | 70.70% |
| Source+ABN [87] | 96.60% | 19.59% | 60.01% | 90.60% | 68.86% | 81.20% |
| MALT-DA (ours) | **97.81%** | 10.28% | **78.36%** | **99.38%** | 66.25% | **95.63%** |

adaptation task, as this keeps the comparison more in the spirit of the unsupervised domain adaptation task. The results with the highest accuracy are in red and those with the second highest are shown in bold. These results illustrate that for several adaptation tasks the proposed MALT-DA method improves significantly on the state of the art results.

Table 7.2 compares the classification accuracies achieved by our adaptation method $A_a$ to the results from the network trained on just the source data $A_s$ as well as other state of the art DA methods.

The results in Tabel 7.3 show the coverage of the performance gap our adaptation method achieves relative to training on the target domain as:

$$Coverage = \frac{A_a - A_s}{A_t - A_s} \tag{7.19}$$

The performance gap coverage is a good metric to compare DA methods across different works because it is normalized with respect the networks base performance on the target domain, without any adaptation. In other words, if a more advanced network

Table 7.3: Coverage of performance gap by the proposed MALT-DA method relative to training on target domain.

| | | Target | | | | |
|---|---|---|---|---|---|---|
| | | MNIST | USPS | SVHN | Syn. Digits | MNIST_M |
| Source | MNIST | - | 100.1% | 70.1% | 65.8% | 18.9% |
| | USPS | 0.0% | - | 5.7% | -0.4% | -0.8% |
| | SVHN | 95.1% | 99.8% | - | 60.8% | 39.2% |
| | Syn. Digits | 81.2% | 83.8% | 91.3% | - | 58.1% |
| | MNIST_M | 95.7% | 59.2% | 29.1% | 92.0% | - |

is used for a domain adaptation task it will likely have a higher base performance and thus most-likely achieve a higher accuracy on the final adaptation task. However, when the results are normalized using the gap closure metric the impact of the improved base performance is accounted for. It is clear in this table that for most adaptation tasks the proposed MALT-DA method covers the majority of the performance gap between source only and target only training, averaging 57% coverage.

In addition to our quantitative results for the digit classification experiments in Tables 7.2 and 7.3, we present qualitative results of the features for several adaptation tasks in Figure 7.3. In this figure we used t-SNE visualization [94] to present a 2D representation of our multidimensional features through our adaptation process. Each column is a different step in the adaptation process. The first three rows show adaptation tasks where the proposed method performs well and the last one shows an instance where the proposed method does not work well.

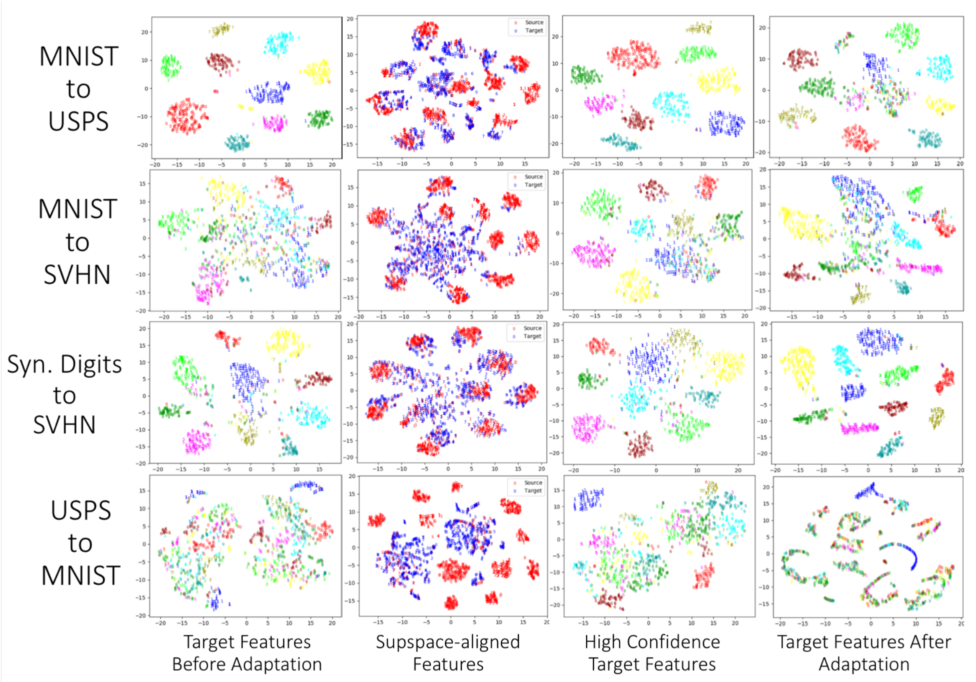


Figure 7.3: Feature visualization using t-SNE plots. The two leftmost columns show the Source (Red) and Target (Blue) features though each stage of the MALT-DA pipeline. The first column (Left) shows the source and target features, red and blue respectively, resulting from the un-adapted network with ABN. The second column depicts the same features after our Subspace Alignment process on the LPP manifold. The third column shows the only the features from the target domain that passed the dual-classifier pseudo label filtering, the different colors correspond to the class of each data-point. The last column displays the visualization of the, unfiltered, features resulting from the proposed network adaptation method.. Each row corresponds to a different adaptation problem. First: from MNIST to USPS dataset. Second: from MNIST to the SVHN dataset. Third: from Syn. Digits to the SVHN dataset. Last: from USPS the MNIST dataset.

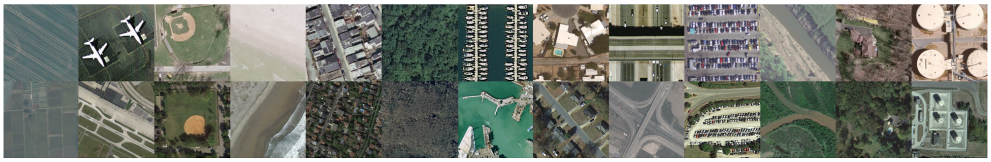


Figure 7.4: Sample images from the shared classes in the UCM (top row) and AID (bottom row) aerial datasets. From left to right, the classes are:Farmland/Agricultural, Airport/Runway & Airplane, Baseball Field, Beach, Dense Residential, Forest, Port/harbor, Medium Residential, Viaduct/Overpass, Parking lot, River, Sparse Residential, Storage Tanks.

### 7.3.3 Remote Sensing Datasets

We considered two remote sensing datasets that were independently developed and have different class labels that partially overlap. The UCM dataset [158] has 20 labeled classes whereas the AID dataset [153] has 30 different classes. In total we found 13 common classes between the two datasets. Examples of each of the shared classes for the UCM and AID datasets can be found in Figure 7.4.

As a means to treat these experiments as a better use case for domain adaptation we trained each network on the full set of classes on the source domain and only adapted to the shared classes in the target domain. This might have made the adaptation task harder because the network was trained to differentiate between more classes than were present in the target domain, but this is a good example of problems faced when using domain adaptation.

We present quantitative results for each aerial adaptation task in Table 7.4. The results in this table demonstrate how the proposed method improves the classification accuracy on the target domain significantly over the training on the source alone, but also indicate that there is room for improvement.

Table 7.4: Accuracy of MALT-DA on remote sensing datasets.

| Source | AID | UCM |
|---|---|---|
| Target | UCM | AID |
| Source only | 50.00% | 42.17% |
| LPP SA [112] | 58.18% | 45.20% |
| MALT-DA (ours) | 58.04% | 55.90% |

## 7.4 Remarks

In this chapter, we present a novel approach to unsupervised domain adaptation of a deep neural network to a new, unlabeled domain. The proposed approach relies on Adaptive Batch Normalization to condition the statistics of the source and target features for effective feature alignment and label transfer. Transductive label transfer, based on feature alignment on the LPP manifold and GMM cluster association with KL divergence, is used for improved labeling of target samples after aligning feature clusters in the source and target domains. We found that our approach works well when the source domain contains sufficient variation, relative to the target domain, to generate meaningful features that capture the variation in both domains. The results produced by the MALT-DA approach outperform state-of-the-art methods for many of the domain adaptation experiments with digits datasets. Furthermore, we present results on a new domain adaptation task between two visible spectrum remote sensing datasets.

# Chapter 8

# Conclusion

This dissertation presented novel methods in the areas of: manifold optimization, linear dimensionality reduction, network compression, deep feature embeddings, and visual domain adaptation. In the field of Manifold Optimization we propose the Proxy-Matrix Optimization method, a new iterative approach to constrained optimization. Proxy Matrix Optimization is a new approach on the manifold optimization that leverages deep learning to reformulate the problem for faster convergence. We then use our Proxy-Matrix method for four different applications: dimensionality reduction, network compression, improved deep feature embeddings, and domain adaptation.

Our work in the field of dimensionality reduction used the Proxy-Matrix method to reformulate common linear dimensionality reduction methods into a single unified framework. We demonstrate how a solution to many different linear dimensionality reduction methods can be obtained using the same manifold optimization framework. After this we propose multiple variations to the commonly used Principal Component Analysis method to improve its robustness to outlying datapoints.

We then extend our dimensionality reduction method to a more general tensor decomposition method and use it to compress deep neural networks. The cascaded projections method that we developed in this dissertation reduced the parameters in a state of the art deep network by 50%, and speed up the processing time of an image by 3.44×, while only experiencing a 0.51% drop in accuracy.

The Proxy Matrix method was then used in deep neural networks to improve the learned feature embeddings. The proposed Deep Euclidean Feature Representations through Adaptation on the Grassmann manifold (DEFRAG) method encourages better clustering behavior of the features, while maintaining orthogonal basis for feature embedding space. The importance of orthogonal basis is often overlooked when features are compared using the Euclidean distance metric.

Lastly we apply our manifold optimization to the field of domain adaptation, where we use subspace alignment and clustering methods to aid in the process of label transfer for adaptation of deep networks. Our proposed Manifold-Aligned Label Transfer for Domain Adaptation (MALT-DA) method achieves state of the art performance on many challenging adaptation tasks.

The methods proposed in this work have many potential applications to real-world problems such as: scene analysis from aerial platforms, object and pedestrian detection in autonomous vehicles, face identification on mobile devices, visual object tracking, online adaptation of deep neural networks, and many more.

# Bibliography

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] P. A. Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[3] P. A. Absil and Jérôme Malick. Projection-like retractions on matrix manifolds. *SIAM Journal on Optimization*, 22(1):135–158, 2012.

[4] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(5):898–916, 2011.

[5] Stephen Bailey. Principal component analysis with noisy and/or missing data. *Publications of the Astronomical Society of the Pacific*, 124(919):1015, 2012.

[6] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2010.

[7] Dimitri P. Bertsekas. On the goldstein-levitin-polyak gradient projection method. In *Conference on Decision and Control including the 13th Symposium on Adaptive Processes*, pages 47–52, 1974.

[8] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[9] William M. Boothby. *An introduction to differentiable manifolds and Riemannian geometry*, volume 120. Academic press, 1986.

[10] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):0, 1991.

[11] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 7, 2017.

[12] J. Paul Brooks and José H. Dulá. The L1-norm best-fit hyperplane problem. *Applied Mathematics Letters*, 26(1):51–55, 2013.

[13] J. Paul Brooks, José H. Dulá, and Edward L. Boone. A pure L1-norm principal component analysis. *Computational Statistics and Data Analysis*, 61:83–98, 2013.

[14] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM*, 58(3):11, 2011.

[15] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[16] Hasan Ertan Cetingul and René Vidal. Intrinsic mean shift for clustering on stiefel and grassmann manifolds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1896–1902, 2009.

[17] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2285–2294, 2015.

[18] Yu Cheng, Felix X. Yu, Rogerio S. Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2857–2865, 2015.

[19] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop*, 2014.

[20] John P. Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: survey, insights, and generalizations. *Journal of Machine Learning Research*, 16(1):2859–2900, 2015.

[21] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *Proceedings of AAAI conference on Artificial Intelligence*, volume 7, pages 540–545, 2007.

[22] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893, 2005.

[23] Ludovic Delchambre. Weighted principal component analysis: a weighted covariance eigendecomposition approach. *Monthly Notices of the Royal Astronomical Society*, 446(4):3545–3555, 2014.

[24] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38, 1977.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[26] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2148–2156, 2013.

[27] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1269–1277, 2014.

[28] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for

generic visual recognition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 647–655, 2014.

[29] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[30] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[31] Alan Edelman, Tomás A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[32] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2960–2967, 2013.

[33] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[34] Geoffrey French, Michal Mackiewicz, and Mark Fisher. Self-ensembling for domain adaptation. *arXiv preprint arXiv:1706.05208v3*, 2017.

[35] Andrea Frome, Greg S. Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2121–2129, 2013.

[36] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Elsevier, 2013.

[37] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014.

[38] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research (JMLR)*, 17(1):2096–2030, 2016.

[39] Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Deep reconstruction-classification networks for unsupervised domain adaptation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 597–613. Springer, 2016.

[40] Boqing Gong, Kristen Grauman, and Fei Sha. Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 222–230, 2013.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[42] Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 999–1006, 2011.

[43] John C. Gower, Garmt B. Dijksterhuis, et al. *Procrustes problems*, volume 30. Oxford University Press on Demand, 2004.

[44] Hermann Grassmann. Die ausdehnungslehre. 1862.

[45] Philip Haeusser, Thomas Frerix, Alexander Mordvintsev, and Daniel Cremers. Associative domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, page 6, 2017.

[46] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[47] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems (NIPS*, pages 164–171, 1993.

[48] K. He, X. Zhang, S. Ren, and J.: Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 630–645. Springer, 2016.

[51] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[52] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[53] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[54] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417, 1933.

[55] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[56] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[57] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 3, 2017.

[58] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[59] Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. *arXiv preprint arXiv:1709.06079*, 2017.

[60] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718, 2018.

[61] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 16(5):550–554, 1994.

[62] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[63] Vance E. Ikezoye and James B. Schrempp. Method and apparatus for identifying media content presented on a media playing device, March 29 2011. US Patent 7,917,645.

[64] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, page 448–456, 2015.

[65] C. Ionescu, O. Vantzos, and C. Sminchisescu. Matrix Backpropagation for Deep Networks with Structured Layers. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[66] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*, 2015.

[67] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *arXiv preprint arXiv:1712.05877*, 2017.

[68] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*, 2014.

[69] Konstantinos Kanistras, Goncalo Martins, Matthew J. Rutherford, and Kimon P Valavanis. Survey of unmanned aerial vehicles (uavs) for traffic monitoring. *Handbook of Unmanned Aerial Vehicles*, pages 2643–2666, 2015.

[70] Qifa Ke and Takeo Kanade. Robust subspace computation using L1 norm. Technical Report TR-CMU-CS-03-172-, Computer Science Dept. Carnegie Mellon Univ., 2003.

[71] Qifa Ke and Takeo Kanade. Robust l/sub 1/norm factorization in the presence of outliers and missing data by alternative convex programming. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 739–746, 2005.

[72] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[73] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 25:1097–1105, 2012.

[74] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

[75] Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1785–1792, 2011.

[76] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The annals of Mathematical Statistics*, 22(1):79–86, 1951.

[77] Nojun Kwak. Principal component analysis based on $l_1$-norm maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(9):1672–1680, 2008.

[78] Nojun Kwak. Principal component analysis by $l\_\{p\}$-norm maximization. *IEEE Transactions on Cybernetics*, 44(5):594–609, 2014.

[79] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[80] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[81] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *IEEE*, page 2278–2323, no. 11, 1998. vol. 86.

[82] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[83] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Proceedings of the Proceedings of the International Conference on Machine Learning (ICML) Workshop on Challenges in Representation Learning*, volume 3, page 2, 2013.

[84] Chong Li and C. J. Richard Shi. Constrained optimization based low-rank approximation of deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 732–747, 2018.

[85] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[86] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

[87] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. *arXiv preprint arXiv:1603.04779*, 2016.

[88] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[89] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 806–814, 2015.

[90] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[91] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.

[92] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[93] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[94] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[95] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. *arXiv preprint arXiv:1803.04014*, 2018.

[96] Panos P. Markopoulos, George N. Karystinos, and Dimitris A. Pados. Some options for L1-subspace signal processing. In *International Symposium on Wireless Communication Systems*, pages 622–626, 2013.

[97] Panos P. Markopoulos, George N. Karystinos, and Dimitris A. Pados. Optimal algorithms for $l\_\{1\}$-subspace signal processing. *IEEE Transactions on Signal Processing*, 62(19):5046–5058, 2014.

[98] Panos P. Markopoulos, Sandipan Kundu, Shubham Chamadia, and Dimitris A. Pados. Efficient $l_1$-norm principal-component analysis via bit flipping. *IEEE Transactions on Signal Processing*, 65(16):4252–4264, 2017.

[99] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[100] Breton Minnehan and Andreas Savakis. Manifold guided label transfer for deep domain adaptation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 65–73, 2017.

[101] Breton Minnehan and Andreas Savakis. Defrag: Deep euclidean feature representations through adaptation on the grassmann manifold. *arXiv preprint arXiv:1806.07688*, 2018.

[102] Breton Minnehan and Andreas Savakis. Grassmann manifold optimization for $l_1$-norm principal component analysis. *IEEE Signal Processing Letters*, 2018.

[103] Breton Minnehan and Andreas Savakis. Deep domain adaptation with manifold aligned label transfer. *Machine Vision and Applications*, pages 1–13, 2019.

[104] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[105] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[106] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4293–4302, 2016.

[107] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature

learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pages 1–9, 2011.

[108] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems (NIPS) workshops*, 2017.

[109] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. volume 2, pages 559–572. Taylor & Francis, 1901.

[110] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[111] C. Qi and F. Su. Contrastive-center loss for deep neural networks. preprint, 2017.

[112] Viresh Ranjan, Gaurav Harit, and C. V. Jawahar. Domain adaptation by aligning locality preserving subspaces. In *International Conference on Advances in Pattern Recognition (ICAPR)*, pages 1–6, 2015.

[113] C. Radhakrishna Rao. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2):159–203, 1948.

[114] Joseph Redmon and Ali Farhadi. Yolo v3: An incremental improvement. *arXiv*, 2018.

[115] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[116] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Computational and Applied Mathematics*, 20:53–65, 1987.

[117] Peter J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[118] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[119] Paolo Russo, Fabio Maria Carlucci, Tatiana Tommasi, and Barbara Caputo. From source to target and back: symmetric bi-directional adaptive gan. *arXiv preprint arXiv:1705.08824*, 2017.

[120] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 213–226. Springer, 2010.

[121] Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada. Asymmetric tri-training for unsupervised domain adaptation. *arXiv preprint arXiv:1702.08400*, 2017.

[122] Swami Sankaranarayanan, Yogesh Balaji, Carlos D. Castillo, and Rama Chellappa. Generate to adapt: Aligning domains using generative adversarial networks. *arXiv preprint arXiv:1704.01705*, 2017.

[123] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[124] Ozan Sener, Hyun Oh Song, Ashutosh Saxena, and Silvio Savarese. Learning transferrable representations for unsupervised domain adaptation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2110–2118, 2016.

[125] Ozan Sener, Hyun Oh Song, Ashutosh Saxena, and Silvio Savarese. Unsupervised transductive domain adaptation. *arXiv preprint arXiv:1602.03534*, 2016.

[126] Sumit Shekhar, Vishal M. Patel, Hien V. Nguyen, and Rama Chellappa. Generalized domain-adaptive dictionaries. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 361–368, 2013.

[127] Chunhua Shen, Hongdong Li, and Michael J Brooks. A convex programming approach to the trace quotient problem. In *Asian Conference on Computer Vision (ACCV)*, pages 227–235. Springer, 2007.

[128] Yuan Shi and Fei Sha. Information-theoretical learning of discriminative clusters for unsupervised domain adaptation. *arXiv preprint arXiv:1206.6438*, 2012.

[129] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[130] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[131] Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *Josa a*, 4(3):519–524, 1987.

[132] Suraj Srinivas and R. Venkatesh Babu. Data-free parameter pruning for deep neural networks. In *British Machine Vision Conference (BMVC)*, 2015.

[133] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal Machine Learning Research*, 15:1929–1958, 2014.

[134] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of AAAI conference on Artificial Intelligence*, volume 6, pages 2058–2065, 2016.

[135] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 443–450. Springer, 2016.

[136] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. preprint, 2014.

[137] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of AAAI conference on Artificial Intelligence*, volume 4, page 12, 2017.

[138] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[139] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[140] Hemant D. Tagare. Notes on optimization on stiefel manifolds. Technical report, Technical report, Yale University, 2011.

[141] Yoshio Takane and Heungsun Hwang. Regularized linear and kernel redundancy analysis. *Computational Statistics & Data Analysis*, 52(1):394–405, 2007.

[142] Yoshio Takane and Sunho Jung. Generalized constrained redundancy analysis. *Behaviormetrika*, 33(2):179–192, 2006.

[143] P. Tsalmantza and David W. Hogg. A data-driven model for spectra: Finding double redshifts in the sloan digital sky survey. *The Astrophysical Journal*, 753(2):122, 2012.

[144] Pavan Turaga, Ashok Veeraraghavan, and Rama Chellappa. Statistical analysis on stiefel and grassmann manifolds with applications in computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[145] Pavan Turaga, Ashok Veeraraghavan, Anuj Srivastava, and Rama Chellappa. Statistical computations on grassmann and stiefel manifolds for image and video-based recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(11):2273–2286, 2011.

[146] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 2962–2971, 2017.

[147] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[148] Qiong Wang, Junbin Gao, and Hong Li. Grassmannian manifold optimization assisted sparse spectral clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5258–5266, 2017.

[149] S. E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[150] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[151] Zaiwen Wen and Wotao Yin. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1-2):397–434, 2013.

[152] Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 5363–5372, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018.

[153] Gui-Song Xia, Jingwen Hu, Fan Hu, Baoguang Shi, Xiang Bai, Yanfei Zhong, Liangpei Zhang, and Xiaoqiang Lu. Aid: A benchmark data set for performance evaluation of aerial scene classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(7):3965–3981, 2017.

[154] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist a novel image dataset for benchmarking machine learning algorithms. 2017.

[155] Kohei Yamamoto and Kurato Maeno. Pcas: Pruning channels with attention statistics. *arXiv preprint arXiv:1806.05382*, 2018.

[156] Shuicheng Yan and Xiaoou Tang. Trace quotient problems revisited. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 232–244. Springer, 2006.

[157] Jun Yang, Rong Yan, and Alexander G. Hauptmann. Cross-domain video concept detection using adaptive svms. In *Proceedings of the ACM international conference on Multimedia*, pages 188–197. ACM, 2007.

[158] Yi Yang and Shawn Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL international*

*conference on advances in geographic information systems*, pages 270–279. ACM, 2010.

[159] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics, 1995.

[160] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[161] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1984–1992, 2015.

[162] Xu Zhang, Felix Xinnan Yu, Shih-Fu Chang, and Shengjin Wang. Deep transfer network: Unsupervised domain adaptation. *arXiv preprint arXiv:1503.00591*, 2015.

[163] Xin Zheng, Deng Cai, Xiaofei He, Wei-Ying Ma, and Xueyin Lin. Locality preserving clustering for image database. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 885–891. ACM, 2004.

[164] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 875–886. 2018.