

# LDMNet: Low Dimensional Manifold Regularized Neural Networks

Wei Zhu

Duke University

[zhu@math.duke.edu](mailto:zhu@math.duke.edu)

Qiang Qiu

Duke University

[qiang.qiu@duke.edu](mailto:qiang.qiu@duke.edu)

Jiaji Huang

Baidu Silicon Valley AI Lab

[huangjiaji@baidu.com](mailto:huangjiaji@baidu.com)

Robert Calderbank

Duke University

[robert.calderbank@duke.edu](mailto:robert.calderbank@duke.edu)

Guillermo Sapiro

Duke University

[guillermo.sapiro@duke.edu](mailto:guillermo.sapiro@duke.edu)

Ingrid Daubechies

Duke University

[ingrid@math.duke.edu](mailto:ingrid@math.duke.edu)

## Abstract

Deep neural networks have proved very successful on archetypal tasks for which large training sets are available, but when the training data are scarce, their performance suffers from overfitting. Many existing methods of reducing overfitting are data-independent. Data-dependent regularizations are mostly motivated by the observation that data of interest lie close to a manifold, which is typically hard to parametrize explicitly. These methods usually only focus on the geometry of the input data, and do not necessarily encourage the networks to produce geometrically meaningful features. To resolve this, we propose the Low-Dimensional-Manifold-regularized neural Network (*LDMNet*), which incorporates a feature regularization method that focuses on the geometry of both the input data and the output features. In *LDMNet*, we regularize the network by encouraging the combination of the input data and the output features to sample a collection of low dimensional manifolds, which are searched efficiently without explicit parametrization. To achieve this, we directly use the manifold dimension as a regularization term in a variational functional. The resulting Euler-Lagrange equation is a Laplace-Beltrami equation over a point cloud, which is solved by the point integral method without increasing the computational complexity. In the experiments, we show that *LDMNet* significantly outperforms widely-used regularizers. Moreover, *LDMNet* can extract common features of an object imaged via different modalities, which is very useful in real-world applications such as cross-spectral face recognition.

## 1. Introduction

In this era of big data, deep neural networks (DNNs) have achieved great success in machine learning research and commercial applications. When large amounts of training data are available, the capacity of DNNs can easily be increased by adding more units or layers to extract more effective high-level features [9, 10, 26]. However, big net-

works with millions of parameters can easily overfit even the largest of datasets. It is thus crucial to regularize DNNs so that they can extract “meaningful” features not only from the training data, but also from the test data.

Many widely-used network regularizations are data-independent. Such techniques include weight decay, parameter sharing, *DropOut* [24], *DropConnect* [27], etc. Intuitively, weight decay alleviates overfitting by reducing the magnitude of the weights and the features, and *DropOut* and *DropConnect* can be viewed as computationally inexpensive ways to train an exponentially large ensemble of DNNs. Their effectiveness as network regularizers can be quantified by analyzing the Rademacher complexity, which provides an upper bound for the generalization error [1, 27].

Most of the data-dependent regularizations are motivated by the empirical observation that data of interest typically lie close to a manifold, an assumption that has previously assisted machine learning tasks such as nonlinear embedding [2], semi-supervised labeling [3], and multi-task classification [6]. In the context of DNN, data-dependent regularization techniques include the tangent distance algorithm [20, 22], tangent prop algorithm [21], and manifold tangent classifier [18]. Typically, these algorithms only focus on the geometry of the input data, and do not encourage the network to produce geometrically meaningful features. Moreover, it is typically hard to explicitly parametrize the underlying manifold, and some of the algorithms require human input of the tangent planes or tangent vectors [9].

Motivated by the same manifold observation, we propose a new network regularization technique that focuses on the geometry of both the input data and the output features, and build low-dimensional-manifold-regularized neural networks (*LDMNet*). This is inspired by a recent algorithm in image processing, the *low dimensional manifold model* [16, 19, 30]. The idea of *LDMNet* is that the concatenation  $(\mathbf{x}_i, \xi_i)$  of the input data  $\mathbf{x}_i$  and the output features  $\xi_i$  should sample a collection of low dimensional manifolds. This idea is loosely related to the Gaussian mixture model: instead of assuming that the generating distribution of the

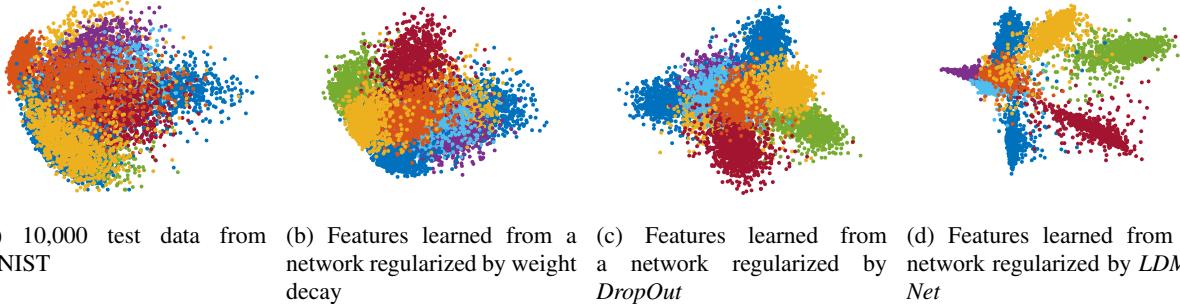


Figure 1: Test data of MNIST and their features learned by the same network with different regularizers. All networks are trained from the same set of 1,000 images. Data are visualized in two dimensions using PCA, and ten classes are distinguished by different colors. In (b) and (c), the features learned by weight decay and *DropOut* typically sample two dimensional regions. Whereas in (d), features learned by *LDMNet* tend to concentrate on one-dimensional and zero-dimensional manifolds (curves and points).

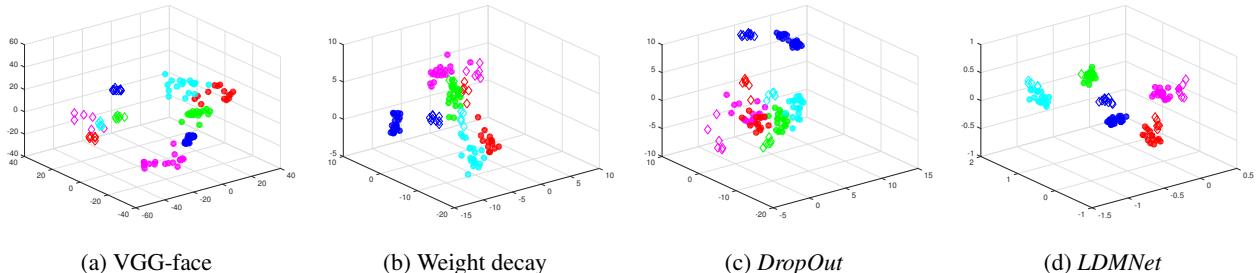


Figure 2: Features generated from VGG-face model [17] with or without embedding learned from the network specified in Table 8 with different regularizations. A total of five subjects are displayed, with one color per subject. VIS images are denoted as filled circles and NIR images are denoted as unfilled diamonds. All features are visualized in two dimensions using PCA. In (a), (b), and (c), features of the same subject typically form two clusters, one for NIR and the other for VIS. In (d), with *LDMNet*, features of the same subject from different modalities merge to form a single low dimensional manifold.

input data is a mixture of Gaussians, we assume that the input-feature tuples  $(\mathbf{x}_i, \xi_i)$  are generated by a mixture of low dimensional manifolds. To emphasize this, we explicitly penalize the loss function with a term using an elegant formula from differential geometry to compute the dimension of the underlying manifold. The resulting variational problem is then solved via alternating minimization with respect to the manifold and the network weights. The corresponding Euler-Lagrange equation is a Laplace-Beltrami equation on a point cloud, which is efficiently solved by the point integral method (PIM) [15] with  $O(N)$  computational complexity, where  $N$  is the size of the input data. In *LDM-Net*, we never have to explicitly parametrize the manifolds or derive the tangent planes, and the solution is obtained by solving only the variational problem.

In the experiments, we demonstrate two benefits of *LDMNet*: First, by extracting geometrically meaningful features, *LDMNet* significantly outperforms widely-used regularization techniques such as weight decay and *DropOut*. For example, Figure 1 shows the two-dimensional projections of the test data from MNIST and

their features learned from 1,000 training data by the same network with different regularizers. It can be observed that the features learned by weight decay and *DropOut* typically sample two-dimensional regions, whereas the features learned by *LDMNet* tend to lie close to one-dimensional and zero-dimensional manifolds (curves and points). Second, in some imaging problems, *LDMNet* is more likely to find the model that is subject to different illumination patterns. By regularizing the network outputs, *LDMNet* can extract common features of the same subject imaged via different modalities so that these features sample the same low dimensional manifold. This can be observed in Figure 2d, where the features of the same subject extracted by *LDMNet* from visible (VIS) spectrum images and near-infrared (NIR) spectrum images merge to form a single low dimensional manifold. This significantly increases the accuracy of cross-modality face recognition. The details of the experiments will be explained in Section 4.

## 2. Model Formulation

For simplicity of explanation, we consider a  $K$ -way classification using a DNN. Assume that  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N \subset \mathbb{R}^{d_1} \times \{1, \dots, K\}$  is the labeled training set, and  $\boldsymbol{\theta}$  is the collection of network weights. For every datum  $\mathbf{x}_i$  with class label  $y_i \in \{1, \dots, K\}$ , the network first learns a  $d_2$ -dimensional feature  $\xi_i = f_{\boldsymbol{\theta}}(\mathbf{x}_i) \in \mathbb{R}^{d_2}$ , and then applies a softmax classifier to obtain the probability distribution of  $\mathbf{x}_i$  over the  $K$  classes. The softmax loss  $\ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)$  is then calculated for  $\mathbf{x}_i$  as a negative log-probability of class  $y_i$ . The empirical loss function  $J(\boldsymbol{\theta})$  is defined as the average loss on the training set:

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i). \quad (1)$$

When the training samples are scarce, statistical learning theories predict that overfitting to the training data will occur [25]. What this means is that the average loss on the testing set can still be large even if the empirical loss  $J(\boldsymbol{\theta})$  is trained to be small.

*LDMNet* provides an explanation and a solution for network overfitting. Empirical observation suggests that many data of interest typically sample a collection of low dimensional manifolds, i.e.  $\{\mathbf{x}_i\}_{i=1}^N \subset \mathcal{N} = \cup_{l=1}^L \mathcal{N}_l \subset \mathbb{R}^{d_1}$ . One would also expect that the feature extractor,  $f_{\boldsymbol{\theta}}$ , of a good learning algorithm be a smooth function over  $\mathcal{N}$  so that small variation in  $\mathbf{x} \in \mathcal{N}$  would not lead to dramatic change in the learned feature  $\xi = f_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}^{d_2}$ . Therefore the concatenation of the input data and output features,  $\{(\mathbf{x}_i, \xi_i)\}_{i=1}^N$ , should sample a collection of low dimensional manifolds  $\mathcal{M} = \cup_{l=1}^L \mathcal{M}_l \subset \mathbb{R}^d$ , where  $d = d_1 + d_2$ , and  $\mathcal{M}_l = \{(\mathbf{x}, f_{\boldsymbol{\theta}}(\mathbf{x}))\}_{\mathbf{x} \in \mathcal{N}_l}$  is the graph of  $f_{\boldsymbol{\theta}}$  over  $\mathcal{N}_l$ . We suggest that network overfitting occurs when  $\dim(\mathcal{M}_l)$  is too large after training (see Figure 1). Therefore, to reduce overfitting, we explicitly use the dimensions of  $\mathcal{M}_l$  as a regularizer in the following variational form:

$$\begin{aligned} \min_{\boldsymbol{\theta}, \mathcal{M}} \quad & J(\boldsymbol{\theta}) + \frac{\lambda}{|\mathcal{M}|} \int_{\mathcal{M}} \dim(\mathcal{M}(\mathbf{p})) d\mathbf{p} \\ \text{s.t.} \quad & \{(\mathbf{x}_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))\}_{i=1}^N \subset \mathcal{M}, \end{aligned} \quad (2)$$

where for any  $\mathbf{p} \in \mathcal{M} = \cup_{l=1}^L \mathcal{M}_l$ ,  $\mathcal{M}(\mathbf{p})$  denotes the manifold  $\mathcal{M}_l$  to which  $\mathbf{p}$  belongs, and  $|\mathcal{M}| = \sum_{l=1}^L |\mathcal{M}_l|$  is the volume of  $\mathcal{M}$ . The following theorem from differential geometry provides an elegant way of calculating the manifold dimension in (2).

**Theorem 1.** [16] Let  $\mathcal{M}$  be a smooth submanifold isometrically embedded in  $\mathbb{R}^d$ . For any  $\mathbf{p} = (p_i)_{i=1}^d \in \mathcal{M}$ ,

$$\dim(\mathcal{M}) = \sum_{i=1}^d |\nabla_{\mathcal{M}} \alpha_i(\mathbf{p})|^2,$$

where  $\alpha_i(\mathbf{p}) = p_i$  is the coordinate function, and  $\nabla_{\mathcal{M}}$  is the gradient operator on the manifold  $\mathcal{M}$ . More specifically,  $\nabla_{\mathcal{M}} \alpha_i = \sum_{s,t=1}^k g^{st} \partial_t \alpha_i \partial_s$ , where  $k$  is the intrinsic dimension of  $\mathcal{M}$ , and  $g^{st}$  is the inverse of the metric tensor.

As a result of Theorem 1, (2) can be reformulated as:

$$\begin{aligned} \min_{\boldsymbol{\theta}, \mathcal{M}} \quad & J(\boldsymbol{\theta}) + \frac{\lambda}{|\mathcal{M}|} \sum_{j=1}^d \|\nabla_{\mathcal{M}} \alpha_j\|_{L^2(\mathcal{M})}^2 \\ \text{s.t.} \quad & \{(\mathbf{x}_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))\}_{i=1}^N \subset \mathcal{M} \end{aligned} \quad (3)$$

where  $\sum_{j=1}^d \|\nabla_{\mathcal{M}} \alpha_j\|_{L^2(\mathcal{M})}^2$  corresponds to the  $L^1$  norm of the local dimension. To solve (3), we alternate the direction of minimization with respect to  $\mathcal{M}$  and  $\boldsymbol{\theta}$ . More specifically, given  $(\boldsymbol{\theta}^{(k)}, \mathcal{M}^{(k)})$  at step  $k$  satisfying  $\{(\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i))\}_{i=1}^N \subset \mathcal{M}^{(k)}$ , step  $k+1$  consists of the following

- Update  $\boldsymbol{\theta}^{(k+1)}$  and the perturbed coordinate functions  $\boldsymbol{\alpha}^{(k+1)} = (\alpha_1^{(k+1)}, \dots, \alpha_d^{(k+1)})$  as the minimizers of (4) with the fixed manifold  $\mathcal{M}^{(k)}$ :

$$\begin{aligned} \min_{\boldsymbol{\theta}, \boldsymbol{\alpha}} \quad & J(\boldsymbol{\theta}) + \frac{\lambda}{|\mathcal{M}^{(k)}|} \sum_{j=1}^d \|\nabla_{\mathcal{M}^{(k)}} \alpha_j\|_{L^2(\mathcal{M}^{(k)})}^2 \\ \text{s.t.} \quad & \boldsymbol{\alpha}(\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i)) = (\mathbf{x}_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)), \quad \forall i = 1, \dots, N \end{aligned} \quad (4)$$

- Update  $\mathcal{M}^{(k+1)}$ :

$$\mathcal{M}^{(k+1)} = \boldsymbol{\alpha}^{(k+1)}(\mathcal{M}^{(k)}) \quad (5)$$

**Remark 1.** As mentioned in Theorem 1,  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_d)$  is supposed to be the coordinate functions. In (4), we solve a perturbed version  $\boldsymbol{\alpha}^{(k+1)}$  so that it maps the previous iterates of point cloud  $\{(\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i))\}_{i=1}^N$  and manifold  $\mathcal{M}^{(k)}$  to their corresponding updated versions. If the iteration converges to a fixed point, the consecutive iterates of the manifolds  $\mathcal{M}^{(k+1)}$  and  $\mathcal{M}^{(k)}$  will be very close to each other for sufficiently large  $k$ , and  $\boldsymbol{\alpha}^{(k+1)}$  will be very close to the coordinate functions.

Note that (5) is straightforward to implement, and (4) is an optimization problem with linear constraint, which can be solved via the alternating direction method of multipliers (ADMM). More specifically,

$$\begin{aligned} \boldsymbol{\alpha}_{\xi}^{(k+1)} = \arg \min_{\boldsymbol{\alpha}_{\xi}} \quad & \sum_{j=d_1+1}^d \|\nabla_{\mathcal{M}^{(k)}} \alpha_j\|_{L^2(\mathcal{M}^{(k)})} \\ & + \frac{\mu |\mathcal{M}^{(k)}|}{2\lambda N} \sum_{i=1}^N \|\boldsymbol{\alpha}_{\xi}(\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i)) - (f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i) - Z_i^{(k)})\|_2^2. \end{aligned} \quad (6)$$

$$\begin{aligned}\boldsymbol{\theta}^{(k+1)} = \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \frac{\mu}{2N} \sum_{i=1}^N & \| \boldsymbol{\alpha}_{\xi}^{(k+1)}(\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i)) \\ & - (f_{\boldsymbol{\theta}}(\mathbf{x}_i) - Z_i^{(k)}) \|_2^2.\end{aligned}\quad (7)$$

$$Z_i^{(k+1)} = Z_i^{(k)} + \boldsymbol{\alpha}_{\xi}^{(k+1)}(\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i)) - f_{\boldsymbol{\theta}^{(k+1)}}(\mathbf{x}_i), \quad (8)$$

where  $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_{\mathbf{x}}, \boldsymbol{\alpha}_{\xi}) = ((\alpha_1, \dots, \alpha_{d_1}), (\alpha_{d_1+1}, \dots, \alpha_d))$ , and  $Z_i$  is the dual variable. Note that we need to perturb only the coordinate functions  $\boldsymbol{\alpha}_{\xi}$  corresponding to the features in (6) because the inputs  $\mathbf{x}_i$  are given and fixed. Also note that because the gradient and the  $L^2$  norm in (6) are defined on  $\mathcal{M}$  instead of the projected manifold, we are *not* simply minimizing the dimension of the manifold  $\mathcal{M}$  projected onto the feature space. For computational efficiency, we update  $\boldsymbol{\alpha}$ ,  $\boldsymbol{\theta}$  and  $Z_i$  only once every manifold update (5).

Among (6), (7) and (8), (8) is the easiest to implement, (7) can be solved by stochastic gradient descent (SGD) with modified back propagation, and (6) can be solved by the point integral method (PIM) [15]. The detailed implementation of (6) and (7) will be explained in the next section.

### 3. Implementation Details and Complexity Analysis

In this section, we present the details of the algorithmic implementation, which includes back propagation for the  $\boldsymbol{\theta}$  update (7), point integral method for the  $\boldsymbol{\alpha}$  update (6), and the complexity analysis.

#### 3.1. Back Propagation for the $\boldsymbol{\theta}$ Update

We derive the gradient of the objective function in (7). Let

$$E_i(\boldsymbol{\theta}) = \frac{\mu}{2} \| \boldsymbol{\alpha}_{\xi}^{(k+1)}(\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i)) - (f_{\boldsymbol{\theta}}(\mathbf{x}_i) - Z_i^{(k)}) \|_2^2. \quad (9)$$

Then the objective function in (7) is

$$\tilde{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i) + \frac{1}{N} \sum_{i=1}^N E_i(\boldsymbol{\theta}). \quad (10)$$

Usually the back-propagation of the first term in (7) is known for a given network. As for the second term, let  $\mathbf{x}_i$  be a given datum in a mini-batch. The gradient of the second term with respect to the output layer  $f_{\boldsymbol{\theta}}(\mathbf{x}_i)$  is:

$$\frac{\partial E_i}{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)} = \mu \left( f_{\boldsymbol{\theta}}(\mathbf{x}_i) - Z_i^{(k)} - \boldsymbol{\alpha}_{\xi}^{(k+1)}(\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i)) \right) \quad (11)$$

This means that we need to only add the extra term (11) to the original gradient, and then use the already known procedure to back-propagate the gradient. This essentially leads to no extra computational cost in the SGD updates.

#### 3.2. Point Integral Method for the $\boldsymbol{\alpha}$ Update

Note that the objective function in (6) is decoupled with respect to  $j$ , and each  $\alpha_j$  update can be cast into:

$$\min_{u \in H^1(\mathcal{M})} \| \nabla_{\mathcal{M}} u \|_{L^2(\mathcal{M})}^2 + \gamma \sum_{\mathbf{q} \in P} |u(\mathbf{q}) - v(\mathbf{q})|^2, \quad (12)$$

where  $u = \alpha_j$ ,  $\mathcal{M} = \mathcal{M}^{(k)}$ ,  $\gamma = \mu |\mathcal{M}^{(k)}| / 2\lambda N$ , and  $P = \{\mathbf{p}_i = (\mathbf{x}_i, f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}_i))\}_{i=1}^N \subset \mathcal{M}$ . The Euler-Lagrange equation of (12) is:

$$\begin{aligned}-\Delta_{\mathcal{M}} u(\mathbf{p}) + \gamma \sum_{\mathbf{q} \in P} \delta(\mathbf{p} - \mathbf{q})(u(\mathbf{q}) - v(\mathbf{q})) &= 0, \quad \mathbf{p} \in \mathcal{M} \\ \frac{\partial u}{\partial n} &= 0, \quad \mathbf{p} \in \partial \mathcal{M}\end{aligned}\quad (13)$$

It is hard to discretize the Laplace-Beltrami operator  $\Delta_{\mathcal{M}}$  and the delta function  $\delta(x, y)$  on an unstructured point cloud  $P$ . We instead use the point integral method to solve (13). The key observation in PIM is the following theorem:

**Theorem 2.** [15] If  $u \in C^3(\mathcal{M})$  is a function on  $\mathcal{M}$ , then

$$\begin{aligned}\left\| \int_{\mathcal{M}} \Delta_{\mathcal{M}} u(\mathbf{q}) R_t(\mathbf{p}, \mathbf{q}) d\mathbf{q} - 2 \int_{\partial \mathcal{M}} \frac{\partial u(\mathbf{q})}{\partial n} R_t(\mathbf{p}, \mathbf{q}) d\tau_{\mathbf{q}} \right. \\ \left. + \frac{1}{t} \int_{\mathcal{M}} (u(\mathbf{p}) - u(\mathbf{q})) R_t(\mathbf{p}, \mathbf{q}) d\mathbf{q} \right\|_{L^2(\mathcal{M})} = O(t^{1/4}).\end{aligned}\quad (14)$$

where  $R_t$  is the normalized heat kernel:

$$R_t(\mathbf{p}, \mathbf{q}) = C_t \exp \left( -\frac{|\mathbf{p} - \mathbf{q}|^2}{4t} \right). \quad (15)$$

After convolving equation (13) with the heat kernel  $R_t$ , we know the solution  $u$  of (13) should satisfy

$$\begin{aligned}- \int_{\mathcal{M}} \Delta_{\mathcal{M}} u(\mathbf{q}) R_t(\mathbf{p}, \mathbf{q}) d\mathbf{q} \\ + \gamma \sum_{\mathbf{q} \in P} R_t(\mathbf{p}, \mathbf{q}) (u(\mathbf{q}) - v(\mathbf{q})) = 0.\end{aligned}\quad (16)$$

Combined with Theorem 2 and the Neumann boundary condition, this implies that  $u$  should approximately satisfy

$$\begin{aligned}\int_{\mathcal{M}} (u(\mathbf{p}) - u(\mathbf{q})) R_t(\mathbf{p}, \mathbf{q}) d\mathbf{q} \\ + \gamma t \sum_{\mathbf{q} \in P} R_t(\mathbf{p}, \mathbf{q}) (u(\mathbf{q}) - v(\mathbf{q})) = 0\end{aligned}\quad (17)$$

Note that (17) no longer involves the gradient  $\nabla_{\mathcal{M}}$  or the Laplace-Beltrami operator  $\Delta_{\mathcal{M}}$ . Assume that  $P =$

$\{\mathbf{p}_1, \dots, \mathbf{p}_N\}$  samples the manifold  $\mathcal{M}$  uniformly at random, then (17) can be discretized as

$$\frac{|\mathcal{M}|}{N} \sum_{j=1}^N R_{t,ij}(u_i - u_j) + \gamma t \sum_{j=1}^N R_{t,ij}(u_j - v_j) = 0, \quad (18)$$

where  $u_i = u(\mathbf{p}_i)$ , and  $R_{t,ij} = R_t(\mathbf{p}_i, \mathbf{p}_j)$ . Combining the definition of  $\gamma$  in (12), we can write (18) in the matrix form

$$\left( \mathbf{L} + \frac{\mu}{\tilde{\lambda}} \mathbf{W} \right) \mathbf{u} = \frac{\mu}{\tilde{\lambda}} \mathbf{W} \mathbf{v}, \quad \tilde{\lambda} = 2\lambda/t, \quad (19)$$

where  $\tilde{\lambda}$  can be chosen instead of  $\lambda$  as the hyperparameter to be tuned,  $\mathbf{u} = (u_1, \dots, u_N)^T$ ,  $\mathbf{W}$  is an  $N \times N$  matrix

$$W_{ij} = R_{t,ij} = \exp\left(-\frac{|\mathbf{p}_i - \mathbf{p}_j|^2}{4t}\right), \quad (20)$$

and  $\mathbf{L}$  is the graph Laplacian of  $\mathbf{W}$ :

$$\mathbf{L}_{ii} = \sum_{j \neq i} W_{ij}, \quad \text{and} \quad \mathbf{L}_{ij} = -W_{ij} \quad \text{if } i \neq j. \quad (21)$$

Therefore, the update of  $\alpha_\xi$ , which is cast into the canonical form (12), is achieved by solving a linear system (19).

### 3.3. Complexity Analysis

Based on the analysis above, we present a summary of the training for *LDMNet* in Algorithm 1.

---

#### Algorithm 1 *LDMNet* Training

---

**Require:** Training data  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N \subset \mathbb{R}^{d_1} \times \mathbb{R}$ , hyperparameters  $\tilde{\lambda}$  and  $\mu$ , and a neural network with the weights  $\theta$  and the output layer  $\xi_i = f_\theta(\mathbf{x}_i) \in \mathbb{R}^{d_2}$ .

**Ensure:** Trained network weights  $\theta^*$ .

Randomly initialize the network weights  $\theta^{(0)}$ . The dual variables  $Z_i^{(0)} \in \mathbb{R}^{d_2}$  are initialized to zero.

**while** not converge **do**

1. Compute the matrices  $\mathbf{W}$  and  $\mathbf{L}$  as in (20) and (21) with  $\mathbf{p}_i = (\mathbf{x}_i, f_{\theta^{(k)}}(\mathbf{x}_i))$ .

2. Update  $\alpha^{(k+1)}$  in (6): solve the linear systems (19), where

$$\mathbf{u}_i = \alpha_j(\mathbf{p}_i), \quad \mathbf{v}_i = f_{\theta^{(k)}}(\mathbf{x}_i)_j - Z_{i,j}^{(k)}. \quad (22)$$

3. Update  $\theta^{(k+1)}$  in (7): run SGD for  $M$  epochs with an extra gradient term (11).

4. Update  $Z^{(k+1)}$  in (8).

5.  $k \leftarrow k + 1$ .

**end while**

$$\theta^* \leftarrow \theta^{(k)}.$$


---

The additional computation in Algorithm 1 (in steps 1 and 2) comes from the update of weight matrices in (20)

and solving the linear system (19) from PIM once every  $M$  epochs of SGD. We now explain the computational complexity of these two steps.

When  $N$  is large, it is not computationally feasible to compute the pairwise distances in the entire training set. Therefore the weight matrix  $\mathbf{W}$  is truncated to only 20 nearest neighbors. To identify those nearest neighbors, we first organize the data points  $\{\mathbf{p}_1, \dots, \mathbf{p}_N\} \subset \mathbb{R}^d$  into a  $k$ -d tree [7], which is a binary tree that recursively partitions a  $k$ -dimensional space (in our case  $k = d$ ). Nearest neighbors can then be efficiently identified because branches can be eliminated from the search space quickly. Modern algorithms to build a balanced  $k$ -d tree generally at worst converge in  $O(N \log N)$  time [4, 5], and finding nearest neighbors for one query point in a balanced  $k$ -d tree takes  $O(\log N)$  time on average [8]. Therefore the complexity of the weight update is  $O(N \log N)$ .

Since  $\mathbf{W}$  and  $\mathbf{L}$  are sparse symmetric matrices with a fixed maximum number of non-zero entries in each row, the linear system (19) can be solved efficiently with the preconditioned conjugate gradients method. After restricting the number of matrix multiplications to a maximum of 50, the complexity of the  $\alpha$  update is  $O(N)$ .

## 4. Experiments

In this section, we compare the performance of *LDMNet* to widely-used network regularization techniques, weight decay and *DropOut*, using the same underlying network structure. We point out that our focus is to compare the effectiveness of the regularizers, and not to investigate the state-of-the-art performances on the benchmark datasets. Therefore we typically use simple network structures and relatively small training sets, and no data augmentation or early stopping is implemented.

Unless otherwise stated, all experiments use mini-batch SGD with momentum on batches of 100 images. The momentum parameter is fixed at 0.9. The networks are trained using a fixed learning rate  $r_0$  on the first 200 epochs, and then  $r_0/10$  for another 100 epochs.

As mentioned in Section 3.3, the weight matrices  $\mathbf{W}$  are truncated to 20 nearest neighbors. For classification tasks, nearest neighbors can be searched within each class in the labeled training set. We also normalize the weight matrices with local scaling factors  $\sigma(\mathbf{p})$  [29]:

$$w(\mathbf{p}, \mathbf{q}) = \exp\left(-\frac{\|\mathbf{p} - \mathbf{q}\|^2}{\sigma(\mathbf{p})\sigma(\mathbf{q})}\right), \quad (23)$$

where  $\sigma(\mathbf{p})$  is chosen as the distance between  $\mathbf{p}$  and its 10th nearest neighbor. This is based on the empirical analysis on choosing the parameter  $t$  in [15], and has been used in [30]. The weight matrices and  $\alpha$  are updated once every  $M = 2$  epochs of SGD.

All hyperparameters are optimized so that those reported are the best performance of each method. For *LDMNet*,  $\tilde{\lambda}$

Layer	Type	Parameters
1	conv	size: $5 \times 5 \times 1 \times 20$ stride: 1, pad: 0
2	max pool	size: $2 \times 2$ , stride: 2, pad: 0
3	conv	size: $5 \times 5 \times 20 \times 50$ stride: 1, pad: 0
4	max pool	size: $2 \times 2$ , stride: 2, pad: 0
5	conv	size: $4 \times 4 \times 50 \times 500$ stride: 1, pad: 0
6	ReLU ( <i>DropOut</i> )	N/A
7	fully connected	$500 \times 10$
8	softmaxloss	N/A

Table 1: Network structure in the MNIST experiments. The outputs of layer 6 are the extracted features, which will be fed into the softmax classifier (layer 7 and 8).

training per class	$\tilde{\lambda}$	$\mu$	$w$
50	0.05	0.01	0.1
100	0.05	0.01	0.05
400	0.01	0.01	0.01
700	0.01	0.01	0.005
1000	0.005	0.01	0.005
3000	0.001	0.01	0.001
6000	0.001	0.01	0.001

Table 2: Hyperparameters used in the MNIST experiments

defined in (19) typically decreases as the training set becomes larger, whereas the parameter  $\mu$  for augmented Lagrangian can be fixed to be a constant. For weight decay,

$$\min_{\theta} J(\theta) + w\|\theta\|_2^2, \quad (24)$$

the parameter  $w$  also usually decreases as the training size increases. For *DropOut*, the corresponding *DropOut* layer is always chosen to have a drop rate of 0.5.

#### 4.1. MNIST

The MNIST handwritten digit dataset contains approximately 60,000 training images ( $28 \times 28$ ) and 10,000 test images. Table 1 describes the network structure. The learned feature  $f_{\theta}(x_i)$  for the training data  $x_i$  is the output of layer six, and is regularized for *LDMNet* in (2).

While state-of-the-art methods often use the entire training set, we are interested in examining the performance of the regularization techniques with varying training sizes from 500 to 60,000. In this experiment, the initial learning rate is set to 0.001, and the hyperparameters are reported in Table 2. Table 3 displays the testing accuracy of the competing algorithms. The dependence of the classification error and generalization error (which is the difference between

training per class	weight decay	<i>DropOut</i>	<i>LDMNet</i>
50	91.32%	92.31%	<b>95.57%</b>
100	93.38%	94.05%	<b>96.73%</b>
400	97.23%	97.95%	<b>98.41%</b>
700	97.67%	98.07%	<b>98.61%</b>
1000	98.06%	98.71%	<b>98.89%</b>
3000	98.87%	99.21%	<b>99.24%</b>
6000	99.15%	<b>99.41%</b>	99.39%

Table 3: MNIST: testing accuracy for different regularizers

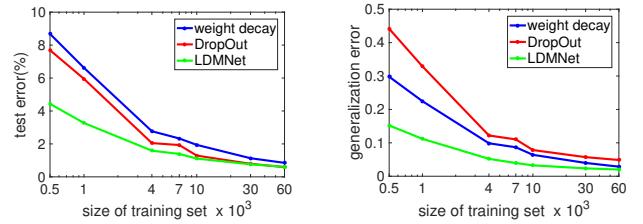


Figure 3: Comparison of the regularizers on the MNIST dataset. The first (second) figure shows the dependence of the classification (generalization) error on the size of the training set.

the softmax loss on the testing and training data) on the size of the training set is shown in Figure 3. Figure 1 provides a visual illustration of the features of the testing data learned from 1,000 training samples. It is clear to see that *LDMNet* significantly outperforms weight decay and *DropOut* when the training set is small, and the performance becomes broadly similar as the size of the training set reaches 60,000.

#### 4.2. SVHN and CIFAR-10

SVHN and CIFAR-10 are benchmark RGB image datasets, both of which contain 10 different classes. These two datasets are more challenging than the MNIST dataset because of their weaker intraclass correlation. All algorithms use the network structure (similar to [11]) specified in Table 4. The outputs of layer 13 are the learned features, and are regularized in *LDMNet*. All algorithms start with a learning rate of 0.005 for SVHN and 0.001 for CIFAR-10.  $\mu$  has been fixed as 0.5 for SVHN and 1 for CIFAR-10, and the remaining hyperparameters are reported in Table 5.

We report the testing accuracies of the competing regularizers in Table 6 and Table 7 when the number of training samples is varied from 50 to 700 per class. Again, it is clear to see that *LDMNet* outperforms weight decay and *DropOut* by a significant margin.

To demonstrate the generality of *LDMNet* to improve any pretrained network, we conduct another experiment on CIFAR-10 using the entire training data with a different network structure. We first train a DNN with VGG-16 ar-

Layer	Type	Parameters
1	conv	size: $5 \times 5 \times 3 \times 96$ stride: 1, pad: 2
2	ReLU	N/A
3	max pool	size: $3 \times 3$ , stride: 2, pad: 0
4	conv	size: $5 \times 5 \times 96 \times 128$ stride: 1, pad: 2
5	ReLU	N/A
6	max pool	size: $3 \times 3$ , stride: 2, pad: 0
7	conv	size: $4 \times 4 \times 128 \times 256$ stride: 1, pad: 0
8	ReLU	N/A
9	max pool	size: $3 \times 3$ , stride: 2, pad: 0
10	fully connected	output: 2048
11	ReLU ( <i>DropOut</i> )	N/A
12	fully connected	output: 2048
13	ReLU ( <i>DropOut</i> )	N/A
14	fully connected	$2048 \times 10$
15	softmaxloss	N/A

Table 4: Network structure in the SVHN and CIFAR-10 experiments. The outputs of layer 13 are the extracted features, which will be fed into the softmax classifier (layer 14 and 15).

training per class	SVHN		CIFAR-10	
	$\lambda$	$w$	$\lambda$	$w$
50	0.1	$10^{-6}$	0.01	$5 \times 10^{-4}$
100	0.05	$10^{-6}$	0.01	$5 \times 10^{-5}$
400	0.05	$10^{-7}$	0.01	$5 \times 10^{-5}$
700	0.01	$10^{-8}$	0.01	$5 \times 10^{-7}$

Table 5: Hyperparameters used in the SVHN and CIFAR-10 experiments.

training per class	weight decay	<i>DropOut</i>	<i>LDMNet</i>
50	71.46%	71.94%	<b>74.64%</b>
100	79.05%	79.94%	<b>81.36%</b>
400	87.38%	87.16%	<b>88.03%</b>
700	89.69%	89.83%	<b>90.07%</b>

Table 6: SVHN: testing accuracy for different regularizers

chitecture [23] on CIFAR-10 using both weight decay and *DropOut* without data augmentation. Then, we fine-tune the DNN by regularizing the output layer with *LDMNet*. The testing accuracies are reported in the last row of Table 7. Again, *LDMNet* outperforms weight decay and *DropOut*, demonstrating that *LDMNet* is a general framework that can be used to improve the performance of any network struc-

training per class	weight decay	<i>DropOut</i>	<i>LDMNet</i>
50	34.70%	35.94%	<b>41.55%</b>
100	42.45%	43.18%	<b>48.73%</b>
400	56.19%	56.79%	<b>60.08%</b>
700	61.84%	62.59%	<b>65.59%</b>
full data	87.72%		<b>88.21%</b>

Table 7: CIFAR-10: testing accuracy for different regularizers. The first four experiments use training sets varied in size from 500 to 7,000 and a simple network specified in Table 4. In the last experiment, we first train a DNN with VGG-16 architecture [23] from the full training data using both weight decay and *DropOut*, then the DNN is fine-tuned by regularizing the output layer with *LDMNet*. No data augmentation is implemented in any of the experiment.



Figure 4: Sample images of two subjects from the CASIA NIR-VIS 2.0 dataset after the pre-processing of alignment and cropping [12]. Top: NIR. Bottom: VIS.

ture.

### 4.3. NIR-VIS Heterogeneous Face Recognition

Finally, we demonstrate the effectiveness of *LDMNet* for NIR-VIS face recognition. The objective of the experiment is to match a probe image of a subject captured in the near-infrared spectrum (NIR) to the same subject from a gallery of visible spectrum (VIS) images. The CASIA NIR-VIS 2.0 benchmark dataset [14] is used to evaluate the performance. This dataset contains 17,580 NIR and VIS face images of 725 subjects. Figure 4 shows eight sample images of two subjects after facial landmark alignment and cropping [12]. Despite recent breakthroughs for VIS face recognition by training DNNs from millions of VIS images, such approach cannot be simply transferred to NIR-VIS face recognition. The reason is that, unlike VIS face images, we have only limited number of available NIR images. Moreover, the NIR-VIS face matching is a cross-modality comparison.

The authors in [13] introduced a way to transfer the breakthrough in VIS face recognition to the NIR spectrum. Their idea is to use a DNN pre-trained on VIS images as a feature extractor, while making two independent modifica-

Layer	Type	Parameters
1	fully connected	output:2000
2	ReLU ( <i>DropOut</i> )	N/A
3	fully connected	output:2000
4	ReLU ( <i>DropOut</i> )	N/A

Table 8: Fully connected network for the NIR-VIS nonlinear feature embedding. The outputs of layer 4 are the extracted features.

tions in the input and output of the DNN. They first modify the input by “hallucinating” a VIS image from the NIR sample, and then apply a low-rank embedding of the DNN features at the output. The combination of these two modifications achieves the state-of-the-art performance on cross-spectral face recognition.

We follow the second idea in [13], and learn a nonlinear low dimensional manifold embedding of the output features. The intuition is that faces of the same subject in two different modalities should sample the same low dimensional feature manifold in a transformed space. In our experiment, we use the VGG-face model (downloaded at [http://www.robots.ox.ac.uk/~vgg/software/vgg\\_face/](http://www.robots.ox.ac.uk/~vgg/software/vgg_face/)) [17] as a feature extractor. The learned 4,096 dimensional features can be reduced to a 2,000 dimensional space using PCA and used directly for face matching. Meanwhile, we also put the 4,096 dimensional features into a two-layer fully connected network described in Table 8 to learn a nonlinear embedding using different regularizations. The features extracted from layer 4 are regularized in *LDMNet*.

All nonlinear embeddings using the structure specified in Table 8 are trained with SGD on mini-batches of 100 images for 200 epochs. We use an exponentially decreasing learning rate that starts at 0.1 with a decaying factor of 0.99. The hyperparameters are chosen to achieve the optimal performance on the validation set. More specifically,  $\lambda$ ,  $\mu$ ,  $w$  are set to  $5 \times 10^{-5}$ , 5, and  $5 \times 10^{-4}$  respectively.

We report the rank-1 performance score for the standard CASIA NIR-VIS 2.0 evaluation protocol in Table 9. Because of the limited amount of training data (around 6,300 NIR and 2,500 VIS images), the fully-connected networks in Table 8 trained with weight decay and *DropOut* clearly overfit the training data: they actually yield testing accuracies that are worse than using a simple PCA embedding of the features learned from VGG-face. However, the same network regularized with *LDMNet* has achieved a significant 10.5% accuracy boost (from 74.51% to 85.02%) to using VGG-face directly. It is also better than the results reported in [13] using the popular triplet embedding [28] and low-rank embedding. Figure 2 provides a visual illustration of the learned features from different regularizations. The generated features of five subjects are visualized in two dimensions using PCA, with filled circle for VIS, and unfilled

	Accuracy (%)
VGG-face	$74.51 \pm 1.28$
VGG-face + triplet [13]	$75.96 \pm 2.90$
VGG-face + low-rank [13]	$80.69 \pm 1.02$
VGG-face weight Decay	$63.87 \pm 1.33$
VGG-face <i>DropOut</i>	$66.97 \pm 1.31$
VGG-face <i>LDMNet</i>	<b><math>85.02 \pm 0.86</math></b>

Table 9: NIR-VIS cross-spectral rank-1 identification rate on the 10-fold CASIA NIR-VIS 2.0 benchmark. The first result is obtained by reducing the features learned from VGG-face to 2,000 dimensional space using PCA. The next two results use triplet [28] and low-rank embedding of the learned features, and are reported in [13]. The last three results are achieved by training the nonlinear embedding network in Table 8 with the corresponding regularizations.

diamond for NIR, and one color for each subject. Note that in Figure 2a,2b,2c, features of one subject learned directly from VGG-face or from a nonlinear embedding regularized with weight decay or *DropOut* typically form two clusters, one for NIR and the other for VIS. This contrasts with the behavior in Figure 2d, where features of the same subject from two different modalities merge to form a single low dimensional manifold.

## 5. Conclusion

We proposed a general deep neural network regularization technique *LDMNet*. The intuition of *LDMNet* is that the concatenation of the input data and output features should sample a collection of low dimensional manifolds, an idea that is loosely related to the Gaussian mixture model. Unlike most data-dependent regularizations, *LDMNet* focuses on the geometry of both the input data and the output features, and does not require explicit parametrization of the underlying manifold. The dimensions of the manifolds are directly regularized in a variational form, which is solved by alternating direction of minimization with a slight increase in the computational complexity ( $O(N)$  in solving the linear system, and  $O(N \log N)$  in the weight update). Extensive experiments show that *LDMNet* has two benefits: First, it significantly outperforms widely used regularization techniques, such as weight decay and *DropOut*. Second, *LDMNet* can extract common features of the same subject imaged via different modalities so that these features sample the same low dimensional manifold, which significantly increases the accuracy of cross-spectral face recognition.

## Acknowledgements

Work partially supported by NSF (DMS 1516988), NIH, ONR, NGA, ARO, AFOSR, and Google.

## References

- [1] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536, 1998.
- [2] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.
- [3] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(Nov):2399–2434, 2006.
- [4] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448 – 461, 1973.
- [5] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.
- [6] T. Evgeniou, C. A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6(Apr):615–637, 2005.
- [7] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, Sept. 1977.
- [8] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, Sept. 1977.
- [9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [11] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
- [12] V. Kazemi and J. Sullivan. One millisecond face alignment with an ensemble of regression trees. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.
- [13] J. Lezama, Q. Qiu, and G. Sapiro. Not afraid of the dark: Nir-vis face recognition via cross-spectral hallucination and low-rank embedding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [14] S. Z. Li, D. Yi, Z. Lei, and S. Liao. The casia nir-vis 2.0 face database. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2013.
- [15] Z. Li, Z. Shi, and J. Sun. Point integral method for solving poisson-type equations on manifolds from point clouds with convergence guarantees. *Communications in Computational Physics*, 22(1):228258, 2017.
- [16] S. Osher, Z. Shi, and W. Zhu. Low dimensional manifold model for image processing. *SIAM Journal on Imaging Sciences*, 10(4):1669–1690, 2017.
- [17] O. M. Parkhi, A. Vedaldi, A. Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.
- [18] S. Rifai, Y. N. Dauphin, P. Vincent, Y. Bengio, and X. Muller. The manifold tangent classifier. In *Advances in Neural Information Processing Systems*, pages 2294–2302, 2011.
- [19] Z. Shi, S. Osher, and W. Zhu. Generalization of the weighted nonlocal laplacian in low dimensional manifold model. *Journal of Scientific Computing*, Sep 2017.
- [20] P. Simard, Y. LeCun, and J. S. Denker. Efficient pattern recognition using a new transformation distance. In *Advances in neural information processing systems*, pages 50–58, 1993.
- [21] P. Simard, B. Victorri, Y. LeCun, and J. Denker. Tangent prop-a formalism for specifying selected invariances in an adaptive network. In *Advances in neural information processing systems*, pages 895–903, 1992.
- [22] P. Y. Simard, Y. A. LeCun, J. S. Denker, and B. Victorri. *Transformation Invariance in Pattern Recognition – Tangent Distance and Tangent Propagation*, pages 235–269. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [24] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.
- [25] V. N. Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.
- [26] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In S. Dasgupta and D. Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058–1066. JMLR Workshop and Conference Proceedings, May 2013.
- [27] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In S. Dasgupta and D. Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058–1066. JMLR Workshop and Conference Proceedings, May 2013.
- [28] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2006.
- [29] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Advances in neural information processing systems*, pages 1601–1608, 2005.
- [30] W. Zhu, B. Wang, R. Barnard, C. D. Hauck, F. Jenko, and S. Osher. Scientific data interpolation with low dimensional manifold model. *Journal of Computational Physics*, 352:213–245, 2018.