FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

SPARSE DICTIONARY LEARNING & THE COMPACT SUPPORT NEURAL NETWORK

By

HONGYU MOU

A Dissertation submitted to the
Department of Statistics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2022

Hongyu Mou defended this dissertation on April 8,2022.
The members of the supervisory committee were:

Adrian Barbu

Professor Directing Dissertation

Xiuwen Liu

University Representative

Yiyuan She

Committee Member

Chong Wu

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

To my family and friends.

ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. Adrian Barbu, for his patient guidance on this dissertation. This dissertation would not be completed without his suggestions and support. I also would like to thank my committee members for their valuable advice and support.

Thank you all for helping me complete this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Chapter 2	
Symbols	**Meaning**
$\mathbf{x}$	Input
$\mathbf{D}$	Dictionary
$\boldsymbol{\alpha}$	Sparse Vector
λ	Penalty Parameter
$\mathbf{A}$	Sparse Matrix
k	Sparsity Level
η	Learning Rate
Chapter 3	
Symbols	**Meaning**
σ	Activation function (ReLU)
$\mathbf{w}$	Weights of Neuron
$\mathbf{x}$	Inputs of Neuron
b	Bias
α	Shape Parameter
R	Radius Parameter
$\mathbf{W}$	Weights Matrix
$\mathbf{b}$	Bias vector
$\mathbf{r}$	Radii Vector
d	Dimension of Input
$\boldsymbol{\mu}$	Mean
$\boldsymbol{\sigma}$	Standard Deviation
$\mathbf{u}$	Input Vector before Normalization
$\mathbf{v}$	Input Vector after Normalization
Chapter 4	
Symbols	**Meaning**
$\mathbf{w}$	Weights of Neuron
b	Bias (Spread Parameter for RBF neuron)
$\mathbf{x}$	Inputs of Neuron
R	Radius Parameter
α	Shape Parameter

LIST OF ABBREVIATIONS

Abbreviation	**Meaning**
ACET	Adversarial Confidence Enhanced Training
AUROC	Area under the ROC curve
CNN	Convolutional Neural Network
CSN	Compact Support Neuron
CSNN	Compact Support Neural Network
DUQ	Deterministic Uncertainty Quantification
FSA	Feature Selection with Annealing
GAN	Generative Adversarial Network
IRLS	Iterative Reweighed Least Squares
KNN	K-Nearest Neighbors
LARS	Least Angle Regression
MSE	Mean Square Error
NN	Neural Network
NZ	Non-Zero
OOD	Out of Distribution
RBF	Radial Basis Function
RF	Random Forest
ROC	Receiver Operating Characteristic
SCAD	smoothly clipped absolute deviation
SGD	Stochastic Gradient Descent
SNGP	Spectral-normalized Neural Gaussian Process
SVM	Support Vector Machine

ABSTRACT

Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. The tasks of computer vision include methods for acquiring, processing analyzing and understanding digital images. In this dissertation, we present approaches for obtaining meaningful representations in computer vision. The second chapter proposes a new method for dictionary learning that uses Feature Selection with Annealing(FSA) to control the representation sparsity directly. This method obtains a smaller reconstruction error than LARS or IRLS and the sparse representation can be used for image classification, where it obtains a smaller misclassification error than LARS and IRLS. In the third chapter, we introduce a novel neuron formulation that can be used for obtaining a tight representation near the training examples and thus preventing high confidence predictions on examples that are far away from the training data. The proposed a neuron uses ReLU as the activation function and has compact support, which means that its output is zero outside a bounded domain. We also show how to avoid difficulties in training a neural network with such neurons. Furthermore, the experimental findings on standard benchmark datasets show that the proposed approach has smaller test errors than the state-of-the-art competing methods and outperforms the competing methods in detecting out-of-distribution samples on two out of three datasets. In the fourth chapter, we prove that a neural network with such compact support neurons has the universal approximation property. This means that the network can approximate any continuous and integrable function with an arbitrary degree of accuracy. In the last chapter, we use the feature of CSNN for few shot learning by recognizing data from a new class as out-of-distribution data and it shows good performance.

CHAPTER 1

INTRODUCTION

Computer vision is a part of artificial intelligence that makes computers have the ability to gain high-level understanding from images or videos. It works like human vision that can tell us what it is, whether there is a certain object, how many objects are there in an image and humans can achieve this effortlessly due to millions of years of evolution. However, we need computers to learn these abilities in a short time with cameras, algorithms and data. Thus, how to train these functions and how to train them faster become very important parts of computer vision.

It is known that sparse representations can help computers save memory and computing time, and we can shorten the training time by using sparse representations. Dictionary learning is a popular method for obtaining sparse linear representations for high dimensional data which has been applied on many domains such as compressed sensing, image classification and signal recovery. There are plenty of methods to select important features for a training model, for example, Feature Selection with Annealing (FSA) (Barbu et al., 2016), Least Angle Regression (LARS) (Efron et al., 2004) and Iteratively Re-weighted Least Squares (IRLS)(Guo et al., 2013). So in the first part this dissertation, I'm going to present an accurate dictionary learning with direct sparsity control using FSA as the feature selection method, which uses an L_0 constraint and can obtain a more accurate sparse representation. I'll also compare the accuracy and efficiency in dictionary learning with classical methods based on the L_1 penalty.

Neural networks have been proven to be extremely useful in many applications, such as facial recognition, object detection, image restoration and etc. However, they also have limitations for requiring lots of training samples and giving high confidence predictions on data far away from the training data. The root cause might be the design of the neuron which is sensitive to far away signals and cannot give a constraint to the distribution of data by the neuron itself. Thus, in the second part, a novel neuron formulation will be presented which makes neurons have compact support. With these compact support neurons, the network can be more reliable, reflected through good performance on detecting out-of-distribution data. The proof of universal approximation

using CSNN is also shown in Chapter 4, in that it can approximate any function from $L^p(\mathbb{R}^d)$ with arbitrary accuracy for any $p \geq 1$.

Further, since the compact support neuron can reject the data far-away from the training data under the assumption that observations from one class are close to each other, I will present in Chapter 5 an approach for few shot learning that uses the prior knowledge of a pretrained CSNN. By recognizing data from a new class as out-of-distribution data, CSNN can be trained with one or few new CSN neurons to model the new class and give acceptable results.

CHAPTER 2

ACCURATE DICTIONARY LEARNING WITH DIRECT SPARSITY CONTROL

2.1 Introduction

Dictionary learning is a popular method for obtaining sparse linear representations for high dimensional data, with many applications in image classification, signal processing and machine learning. In this chapter, we introduce a novel dictionary learning method based on a recent variable selection algorithm called Feature Selection with Annealing (FSA) which uses an L_0 constraint instead of the L_1 penalty that does not introduce any bias in the coefficients and can obtains a more accurate sparse representation. The L_0 constraint makes it easy to directly specify the desired sparsity level instead of indirectly through a L_1 penalty. Furthermore, experimental validation on real gray-scale images shows that the proposed method obtains higher accuracy and efficiency in dictionary learning compared to classical methods based on the L_1 penalty.

Sparse dictionary learning is a feature learning method that aims to represent the input data as a linear combination of a small number of elements of a dictionary, called atoms. Unlike principal component analysis, the atoms in the dictionary are not required to be orthogonal. Furthermore, the dictionary usually contains more atoms that the dimensionality of the data, which means that the dictionary gives an over-complete representation.

In 1997, Olshausen (Olshausen and Field, 1997) introduced the idea of sparse coding with an overcomplete basis, as a possible explanation of the receptive cells in the V1 part of the brain. There has been a lot of work on sparse coding since then, which could be grouped into different categories based on the type of regularization that was used to obtain the sparse representation. A good survey of the different methods has been done in Zhang et al. (2015).

Given an observation $\mathbf{x} \in \mathbb{R}^d$ and a dictionary $\mathbf{D}$ with p atoms as a $d \times p$ matrix, sparse coding can be obtained by minimizing the l_0-norm with the constraint of exact reconstruction (Donoho and Elad, 2003):

$$\boldsymbol{\alpha} = \text{argmin}\, ||\boldsymbol{\alpha}||_0 \text{ s.t. } \mathbf{x} = \mathbf{D}\boldsymbol{\alpha}$$

where $||\boldsymbol{\alpha}||_0$ is the number of nonzero elements in the sparse vector $\boldsymbol{\alpha}$.

Because l_0-norm minimization is an NP-hard problem, a popular approximation used in machine learning and statistics (Patel and Chellappa, 2011; Yuan et al., 2014) is the l_1-norm minimization, especially the Lasso (Tibshirani, 1996):

$$\boldsymbol{\alpha} = \underset{\boldsymbol{\alpha}}{\text{argmin}}\, ||\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}||_2^2 + \lambda\|\boldsymbol{\alpha}\|_1, \tag{2.1}$$

where the penalty parameter λ can be tuned for a desired sparsity of the solution. One can also obtain the entire regularization path using least angle regression(LARS)(Efron et al., 2004). Recently, other more efficient methods for global optimization of the convex loss (2.1) have been proposed using gradient projection and homotopy (Yang et al., 2010).

Another way to introduce sparse coding is through l_p-norm minimization, where $0 < p < 1$. There are three typical algorithms for l_p minimization (Lyu et al., 2013): General Iteratively Reweighted Least Squares (GIRLS), Iteratively Thresholding Method (ITM) and Iteratively Reweighted Least Squares (IRLS). Experimental comparison of the three methods revealed that IRLS has the best performance and is the fastest as well. Furthermore, l_p minimization with IRLS has been used for robust face recognition (Guo et al., 2013).

Most recent methods for dictionary learning are based on regression with sparsity inducing penalties such as the convex l_1 penalty or non-convex penalties such as the SCAD penalty (Fan and Li, 2001). Examples include the online dictionary learning (Mairal et al., 2009) and Fisher discrimination dictionary learning (Yang et al., 2011). Dictionary learning has been applied to face recognition using discriminative K-SVD (Zhang and Li, 2010).

In this chapter we will investigate a novel approach to sparse dictionary learning based on a recent l_0-based optimization method named Feature Selection with Annealing (FSA)(Barbu et al., 2016). In our context of dictionary learning, FSA can be used for solving the l_0-constraint optimization problem:

$$\boldsymbol{\alpha} = \underset{||\boldsymbol{\alpha}||_0 \leq k}{\text{argmin}}\, ||\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}||_2^2$$

where k is the desired number of nonzero entries of $\boldsymbol{\alpha}$. Compared with Lasso, the FSA method can directly control the sparsity of the solution and has better performance than l_1-norm and l_p-norm minimization methods.

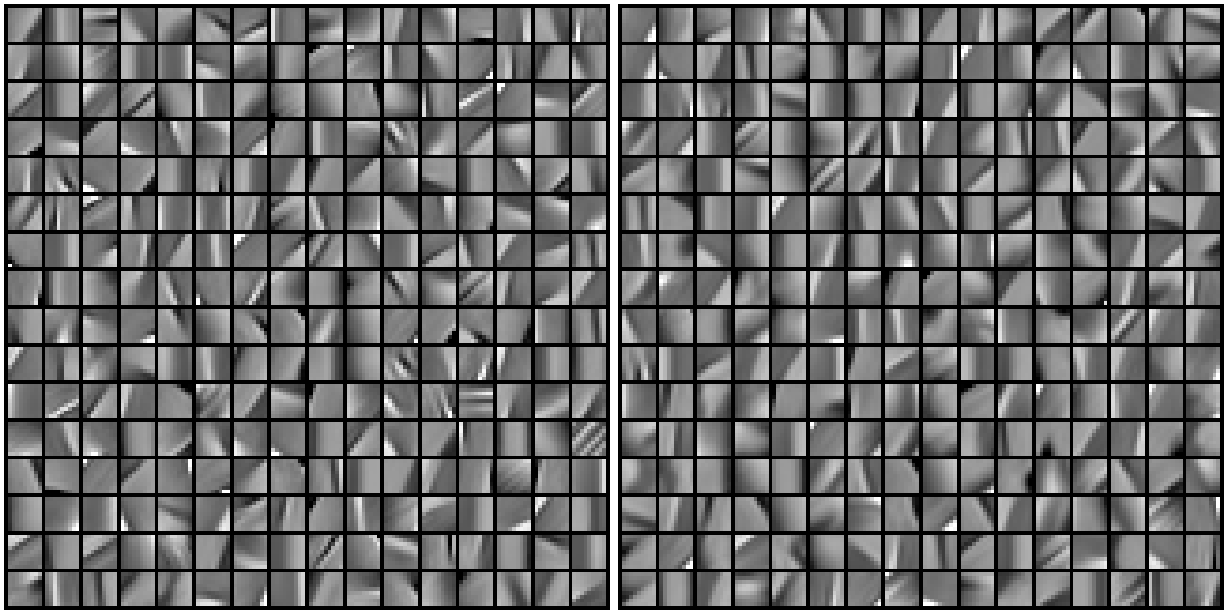


Figure 2.1: Learned dictionaries. Left: dictionary obtained using the l_1 penalty. Right: dictionary obtained using FSA

Through experiments we will see that FSA obtains a more accurate reconstruction of the data for the same sparsity level than LARS or ILRS, while being faster. We will also see that the obtained dictionary can be used for classification on the MNIST data using different learning methods, obtaining better results than LARS, IRLS and direct learning without sparse coding.

2.2 Dictionary Learning with FSA

According to the classical dictionary learning (Olshausen and Field, 1997), suppose we have an input data set $\mathbf{X} \in \mathbb{R}^{d \times n}$, each column representing an $m \times m$ image patch and $n >> d$ is the number of extracted image patches. The goal of dictionary learning is to find a dictionary $\mathbf{D} \in \mathbb{R}^{d \times p}$ where each column is an atom and:

$$\mathbf{X} \approx \mathbf{DA}$$

where $\mathbf{A} = (\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_n)$ contains sparse vectors $\boldsymbol{\alpha}_i \in \mathbb{R}^p$.

Learning the dictionary $\mathbf{D}$ and the sparse representation $\mathbf{A}$ is done by minimizing a cost function:

$$f_n(\mathbf{D}, \mathbf{A}) \triangleq \frac{1}{n} \sum_{i=1}^{n} l(\mathbf{x}_i, \mathbf{D}, \boldsymbol{\alpha}_i) \tag{2.2}$$

where the loss function $l(\mathbf{x}_i, \mathbf{D}, \boldsymbol{\alpha}_i)$ measures the difference between $\mathbf{x}_i$ and $\mathbf{D}\boldsymbol{\alpha}_i$ and encourages a sparse $\boldsymbol{\alpha}_i$.

One popular loss function is the l_1-penalized square loss also known as the Lasso (Tibshirani, 1996; Fu, 1998; Efron et al., 2004):

$$l(\mathbf{x}_i, \mathbf{D}, \boldsymbol{\alpha}_i) \triangleq \frac{1}{2}||\mathbf{x}_i - \mathbf{D}\boldsymbol{\alpha}_i||_2^2 + \lambda||\boldsymbol{\alpha}_i||_1 \tag{2.3}$$

where λ is a regularization parameter that imposes the desired level of sparsity for $\boldsymbol{\alpha}_i$. The Lasso has been used in sparse sparse dictionary learning before, for example in online dictionary learning (Mairal et al., 2009).

However, the l_1 approach has the disadvantage that it controls the sparsity of $\boldsymbol{\alpha}_i$ only indirectly through the penalty λ. We are interested in an approach where the sparsity of $\boldsymbol{\alpha}_i$ can be directly specified as a constraint $||\boldsymbol{\alpha}_i||_0 \leq k$.

$$l(\mathbf{x}_i, \mathbf{D}, \boldsymbol{\alpha}_i) \triangleq \begin{cases} ||\mathbf{x}_i - \mathbf{D}\boldsymbol{\alpha}_i||_2^2 & \text{if } ||\boldsymbol{\alpha}_i||_0 \leq k \\ \infty & \text{else} \end{cases} \tag{2.4}$$

The cost function (2.2) depends on two variables: $\mathbf{D}$ and $\mathbf{A}$. We will use a straightforward approach that minimizes the cost (2.2) by alternately minimizing over one variable while keeping the other one fixed. When $\mathbf{D}$ is fixed, minimizing over $\mathbf{A}$ can be obtained by independently minimizing (2.4) for each $\mathbf{x}_i$.

The whole alternating procedure for dictionary learning is described in Algorithm 1. It depends on two other algorithms, which will be described in the next two sections.

Algorithm 1 Dictionary learning with FSA

Input: Data matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ with n observations $\mathbf{x}_1, ..., \mathbf{x}_n$ as columns, sparsity level k.

Output: Trained dictionary $\mathbf{D}$.

1: Initialize $\mathbf{D}_0$ with p random observations from $\mathbf{X}$.
2: **for** t=1 to T **do**
3: **for** i=1 to n **do**
4: Use Algorithm 2 to compute

$$\boldsymbol{\alpha}_i = \underset{||\boldsymbol{\alpha}||_0 \leq k}{\operatorname{argmin}} ||\mathbf{x}_i - \mathbf{D}_{t-1}\boldsymbol{\alpha}||_2^2$$

5: **end for**
6: Set $\mathbf{A} = (\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_n) \in \mathbb{R}^{p \times n}$.
7: Compute $\mathbf{D}_t$ by Algorithm 3, with $(\mathbf{X}, \mathbf{A}, \mathbf{D}_{t-1})$ as input

$$\mathbf{D}_t = \underset{\mathbf{D}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{D}_{t-1}\boldsymbol{\alpha}_i||_2^2 \tag{2.5}$$

8: **end for**
9: Return $\mathbf{D} = \mathbf{D}_T$

2.2.1 Feature Selection with Annealing (FSA)

FSA is a novel variable selection method introduced in (Barbu et al., 2016) that minimizes a differentiable loss function $L(\boldsymbol{\alpha})$ with sparsity constraints

$$\boldsymbol{\alpha} = \underset{||\boldsymbol{\alpha}||_0 \leq k}{\operatorname{argmin}} L(\boldsymbol{\alpha}).$$

In our case, the loss function $L(\boldsymbol{\alpha})$ is:

$$L(\boldsymbol{\alpha}) = ||\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}||_2^2$$

where $\mathbf{x}$ is any of the columns of $\mathbf{X}$ and $\mathbf{D}$ is the current dictionary.

FSA achieves sparsity by gradually reducing the dimensionality of $\boldsymbol{\alpha}$ from p to k, according to an annealing schedule. It starts with a full $\boldsymbol{\alpha} \in \mathbb{R}^p$ and alternates the removal of some variables with the updating of the remaining parameters by gradient descent. The whole procedure is described in Algorithm 2.

Algorithm 2 Feature Selection with Annealing

Input: Observation $\mathbf{x} \in \mathbb{R}^d$, current dictionary $\mathbf{D} \in \mathbb{R}^{d \times p}$, sparsity level k

Output: Sparse $\boldsymbol{\alpha} \in \mathbb{R}^p$ with $\|\boldsymbol{\alpha}\|_0 \leq k$.

1: Initialize $\boldsymbol{\beta} = 0 \in \mathbb{R}^p, J = \{1, ..., p\}$

2: **for** e=1 to N^{iter} **do**

3: Update $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \mathbf{D}^T(\mathbf{D}\boldsymbol{\beta} - \mathbf{x})$

4: Find the indexes $I, |I| = p_e$ corresponding to the highest p_e elements of $|\boldsymbol{\beta}|$.

5: Keep only the entries with index I in $\boldsymbol{\beta}, \mathbf{x}, J$ and $\mathbf{D}$, i.e.

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_I, \mathbf{x} \leftarrow \mathbf{x}_I, J \leftarrow J_I, \mathbf{D} \leftarrow \mathbf{D}_{II}$$

6: **end for**

7: Set $\boldsymbol{\alpha} = 0 \in \mathbb{R}^p$, then $\boldsymbol{\alpha}_J = \boldsymbol{\beta}$.

The annealing schedule p_e represents the number of non-zero variables that are kept at the e^{th} iteration. A fast annealing schedule will save computation time but loose some accuracy in selecting the correct variables. So it is important to find a proper annealing schedule that balances speed and accuracy. In (Barbu et al., 2016), the authors provide an inverse schedule:

$$p_e = k + (p - k) \max(0, \frac{N^{iter} - 2e}{2e\mu + N^{iter}})$$

where p is the number of total variables and k is the sparsity level. The parameter μ controls the speed of removing the variables. Together with the learning rate η, they can be tuned to obtain a small value of the loss function at the completion of the algorithm.

2.2.2 Dictionary Update

The loss function (2.5) is quadratic in $\mathbf{D}$ and could be minimized analytically. To avoid large matrix operations, we use block-coordinate descent with warms starts, as described in (Mairal et al., 2009) and in Algorithm 3 below.

Algorithm 3 Dictionary Update

Input: Data matrix $\mathbf{X} \in \mathbb{R}^{d\times n}$, sparse matrix $\mathbf{A} = (\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_n) \in \mathbb{R}^{p\times n}$, input dictionary $\mathbf{D} = [\mathbf{d}_1, ..., \mathbf{d}_p] \in \mathbb{R}^{d\times p}$

Output: dictionary $\mathbf{D} = [\mathbf{d}_1, ..., \mathbf{d}_p] \in \mathbb{R}^{d\times p}$

Set $\mathbf{B} = \mathbf{A}\mathbf{A}^T \in \mathbb{R}^{p\times p}$,
Set $\mathbf{C} = \mathbf{X}\mathbf{A}^T = [\mathbf{c}_1, ..., \mathbf{c}_p] \in \mathbb{R}^{d\times p}$.
repeat
  for $j = 1$ to p **do**
    Set $\mathbf{u}_j \leftarrow \frac{1}{\mathbf{B}_{jj}}(\mathbf{c}_j - \mathbf{D}\mathbf{b}_j) + \mathbf{d}_j$
    Set $\mathbf{d}_j \leftarrow \frac{\mathbf{u}_j}{\max(||\mathbf{u}_j||_2, 1)}$
  end for
until convergence
Return updated dictionary $\mathbf{D}$.

2.3 Experiments

Data Description. In the experiments we use two standard gray images: Lena and Boat, resized to 128×128. We work with overlapping patches of size of 9×9 extracted from these images, and our input data is $\mathbf{X} \in \mathbb{R}^{81\times 14400}$.

For evaluating the quality of the learned dictionary $\mathbf{D}$ we will use the MSE of the data reconstruction:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{D}\boldsymbol{\alpha}_i||_2^2$$

FSA Parameter Experiments. First, we need to find proper hyper-parameter values for μ, η, and N^{iter} for using FSA in dictionary learning. We experimented with learning a 1024 atom dictionary on the Lena image. In Figure 2.2, left are shown the MSE vs iteration number for different values the learning rate η, for $N^{iter} = 100$ and $\mu = 80$. We see that the quality of the dictionary is almost the same for η between 0.01 and 0.03. In Figure 2.2, middle are shown the MSE vs iteration number for different values of N^{iter}, when $\mu = 80, \eta = 0.01$. We see that with more iterations, the quality of the dictionary is better, at an increased computation cost. We fixed $N^{iter} = 500$, which has a comparable computation cost with LARS. In Figure 2.2, right are shown the MSE vs iteration number for different values the annealing parameter μ, for $N^{iter} = 500$ and $\eta = 0.01$. We see that the quality of the dictionary is almost the same for μ between 100 and 500, the best being for $\mu = 200$.

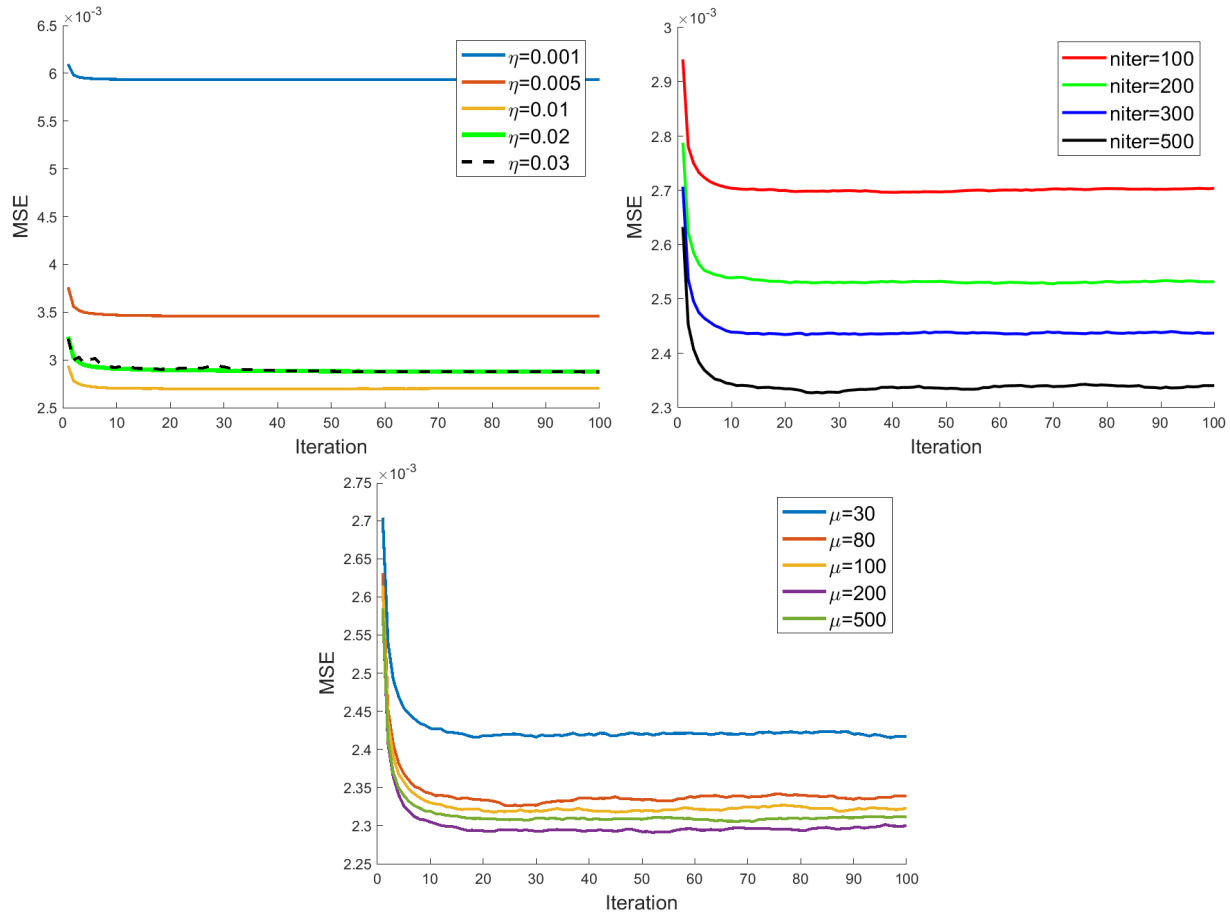


Figure 2.2: Data reconstruction error vs number of learning iterations for the three FSA hyper-parameters: η, N^{iter} and μ.

Comparison with Other Methods. In this experiment we compare the dictionary learned with FSA with approaches where Algorithm 2 was replaced by LARS or IRLS. We investigated three sparsity levels $k = 3, 4, 5$.

In Figure 2.3 are shown the reconstruction MSE for the three methods for 256 and 1024 atoms on Lena, and 1024 atoms on the Lena and Boats images simultaneously. We see that FSA obtains better dictionaries than LARS which is better than IRLS. Furthermore, FSA with $k = 4$ has smaller MSE than LARS and IRLS with $k = 5$ in all three cases. Moreover, IRLS is at least 10 times slower than FSA, which is why we didn't include the Lena+Boats results for IRLS.

Examples of reconstructed images with the learned dictionary are shown in Figure 2.4. We can see that FSA obtains a more accurate and more clear image than LARS.

Digit Recognition Application. In this section, we use dictionary learning to obtain a sparse

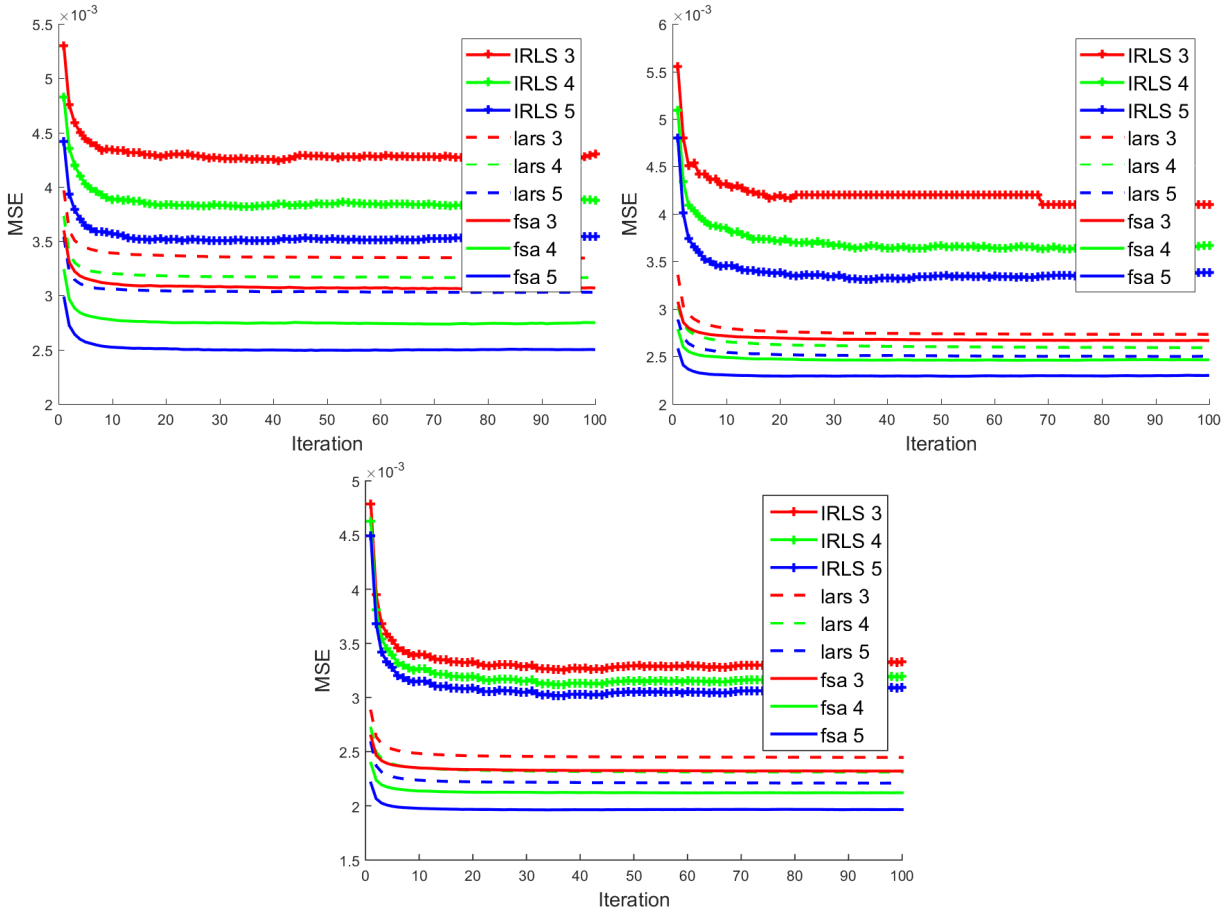


Figure 2.3: Comparison with other methods. Left: dictionary with 256 atoms on Lena. Middle: dictionary with 1024 atoms on Lena. Right: dictionary with 1024 atoms on Lena+Boats.

representation of the MNIST (LeCun, 1998) handwritten digit database and apply several multi-class classification methods on the sparse feature vectors to compare the classification accuracy.

MNIST is a dataset containing 60,000 training examples and 10,000 test examples as grayscale images of size 28×28 of handwritten digits.

In our experiment, we generate 256-atom dictionaries using different dictionary learning methods, one dictionary for each digit from 0 to 9. Then for each observation we generated one sparse feature vector from each dictionary, and concatenated them to obtain a 2560 dimensional sparse feature vector. This sparse feature vector was used as input for training and testing different classifiers.

As classifiers, we used SVM, Random Forest with 500 trees, and K-Nearest Neighbors with

Table 2.1: Test misclassification errors on MNIST data.

Method	Sparsity	SVM	KNN(K=3)	RF
FSA	5	0.0362	0.0915	0.0915
FSA	10	0.0257	0.0682	0.0394
FSA	20	**0.0223**	0.0561	0.0331
FSA	50	0.0239	**0.0364**	**0.0270**
LARS	5	0.0372	0.1062	0.0564
LARS	10	0.0323	0.1312	0.0463
LARS	20	0.0300	0.1012	0.0397
LARS	50	0.0330	0.0802	0.0357
IRLS	5	0.0848	0.1585	0.1030
IRLS	10	0.0759	0.1501	0.0904
IRLS	20	0.0527	0.1465	0.0582
IRLS	50	0.0503	0.0955	0.0539
Original data	-	0.0562	0.0619	0.0352

$K = 3$. We compared different sparsity levels in the sparse representation, and also tested the classifiers on the original data with no sparsity.

The results are shown in Table 2.1. We see that for each type of classifier the sparse representation obtained by FSA has smaller test misclassification error than LARS, IRLS and the original data.

2.4 Conclusion

We introduced a new method to solve the sparse coding problem in dictionary learning that replaces the L_1 penalty in the loss function with a sparsity constraint $||\boldsymbol{\alpha}||_0 \leq k$. The method relies on a recent feature selection methods called Feature Selection with Annealing (FSA). Using FSA we can directly specify the number of non-zero variables we want in the sparse representation, unlike the L_1 penalized methods where the sparsity is controlled indirectly through the regularization parameter λ.

The experimental results on image reconstruction showed that the proposed method obtains smaller reconstruction errors than LARS or IRLS for the same sparsity level and dictionary size. Furthermore, experiments on the MNIST dataset using SVM, Random Forest and K-Nearest Neighbors showed that the method can be used to obtain a sparse image representation that obtains a

smaller misclassification error on than directly using the image as input, Furthermore the sparse representation by FSA again outperforms LARS and IRLS in this classification task.

Figure 2.4: Reconstructed images. Top: original images. Middle: images reconstructed using LARS. $MSE_{Lena} = 0.0027, MSE_{Boat} = 0.0035$. Bottom: images reconstructed using FSA. $MSE_{Lena} = 0.0024, MSE_{Boat} = 0.0031$.

CHAPTER 3

THE COMPACT SUPPORT NEURAL NETWORK

3.1 Introduction

Neural networks are popular and useful in many fields, but they have the problem of giving high confidence responses for examples that are far away from the training data. This makes the neural networks very confident in their prediction while making gross mistakes, thus limiting their reliability for safety critical applications such as autonomous driving, space exploration, etc. In this chapter, we present a neuron generalization that has the standard dot-product based neuron and the RBF neuron as two extreme cases of a shape parameter. Using ReLU as the activation function we obtain a novel neuron that has compact support, which means its output is zero outside a bounded domain. We show how to avoid difficulties in training a neural network with such neurons, by starting with a trained standard neural network and gradually increasing the shape parameter to the desired value.

Through experiments on standard benchmark datasets, we show the promise of the proposed approach, in that it can have good prediction on in-distribution samples, while being able to consistently detect and have low confidence on out of distribution samples. Neural networks have been proven to be extremely useful in all sorts of applications, including object detection, speech and handwriting recognition, medical imaging, etc. They have become the state of the art in these applications, and in some cases they even surpass human performance. However, neural networks have been observed to have a major disadvantage: they don't know when they don't know, i.e. don't know when the input is far away from the type of data they have been trained on. Instead of saying "I don't know", they give some output with high confidence (Goodfellow et al., 2014; Nguyen et al., 2015). An explanation of why this is happening for ReLU based networks has been given in Hein et al. (2019). This issue is very important for safety-critical applications such as space exploration, autonomous driving, medical diagnosis, etc. In these cases it is important that the system know when the input data is outside its nominal range, to alert the human (e.g. driver for autonomous driving or radiologist for medical diagnostic) to take charge in such cases.

In this chapter we suspect that the root of this problem is actually the neuron design, and propose a different type of neuron to address what we think are its issues. The standard neuron can be written as $f(x) = \sigma(\mathbf{w}^T\mathbf{x}+b)$, which can be regarded as a projection (dot product) $\mathbf{x} \to \mathbf{w}^T\mathbf{x}+b$ onto a direction $\mathbf{w}$, followed by a nonlinearity $\sigma(\cdot)$. In this design, the neuron has a large response for vectors $\mathbf{x} \in \mathbb{R}^p$ that are in a half-space. This can be an advantage when training the NN since it creates high connectivity in the weight space and makes the neurons sensitive to far-away signals. However, it is a disadvantage when using the trained NN, since it can lead to the neurons unpredictably firing with high responses to far-away signals, which can result (with some probability) in high confidence responses of the whole network for examples that are far away from the training data.

To address these problems, we use a type of radial basis function neuron (Broomhead and Lowe, 1988), $f(\mathbf{x}) = g(\|\mathbf{x} - \boldsymbol{\mu}\|^2)$, which we modify to have a high response only for examples that are close to $\boldsymbol{\mu}$, and to have zero response at distance at least R from $\boldsymbol{\mu}$. Therefore the neuron has compact support, and the same applies to a layer formed entirely of such neurons. Using one such compact support layer before the output layer we can guarantee that the space where the NN has a non-zero response is bounded, obtaining a more reliable neural network.

In this formulation, the parameter vector $\boldsymbol{\mu}$ is directly comparable to the neuron inputs $\mathbf{x}$, thus $\boldsymbol{\mu}$ has a simple and direct interpretation as a "template". A layer consisting of such neurons forms can be interpreted as a sparse coordinate system on the manifold containing the inputs of that layer.

Because of the compact support, the loss function of such a compact support NN has many flat areas and it can be difficult to training it directly by backpropagation. However, we will show how to train such a NN, by starting with a trained regular NN and gradually bending the neuron decision boundaries to make them have smaller and smaller support.

The contributions of this chapter are the following:

- We introduce a type of neuron formulation that generalizes the standard neuron and the RBF neuron as two extreme cases of a shape parameter. Moreover one can smoothly transition from a regular neuron to a RBF neuron by gradually changing this parameter. We introduce the RBF correspondent to a ReLU neuron and observe that it has compact support, i.e. its output is zero outside a bounded domain.

- The above construction allows us to smoothly bend the decision boundary of a standard ReLU based neuron, obtaining a compact support neuron. We use this idea to train a compact support neural network (CSNN) starting from a pre-trained regular neural network.
- We show through experiments on standard datasets that the proposed CSNN can achieve comparable test errors with regular CNNs, and at the same time it can detect and have low confidence on out-of-distribution data.

3.1.1 Related Work

A common way to address the problem of high confidence predictions for out of distribution (OOD) examples is through ensembles (Lakshminarayanan et al., 2017), where multiple neural networks are trained with different random initializations and their outputs are averaged in some way. The reason why ensemble methods have low confidence on OOD samples is that the high-confidence domain of each NN is random outside the training data, and the common high-confidence domain is therefore shrunk by the averaging process. This reasoning works well when the representation space (the space of the NN before the output layer) is high dimensional, but it fails when this space is low dimensional (see (van Amersfoort et al., 2020) for example).

Another popular approach is adversarial training (Madry et al., 2018), where the training set is augmented with adversarial examples generated by maximizing the loss starting from slightly perturbed examples. This method is modified in adversarial confidence enhanced training (ACET) (Hein et al., 2019) where the adversarial samples are added through a hybrid loss function. However, we believe that training with out of distribution samples could be a computationally expensive if not hopeless endeavor, since the instance space is extremely vast when it is high dimensional. Consequently, a finite number of training examples can only cover an insignificant part of it and no matter how many out-of-distribution examples are used, there always will be other parts of the instance space that have not been explored.Other methods include the estimation of the uncertainty using dropout (Gal and Ghahramani, 2016), softmax calibration (Guo et al., 2017), and the detection of OOD inputs (Hendrycks and Gimpel, 2017). CutMix (Yun et al., 2019) is a method to generate training samples with larger variability, which help improve generalization and OOD detection. All these methods are complementary to the proposed approach and could be used together with the classifiers introduced in this paper to improve accuracy and OOD detection.

A number of works assume that the distance in the representation space is meaningful. A trust score was proposed in (Jiang et al., 2018) to measure the agreement between a given classifier and

a modified version of a k-nearest neighbor (k-NN) classifier. While this approach does consider the distance of the test samples to the training set, it only does so to a certain extent since the k-NN does not have a concept of "too far", and is also computationally expensive.

A simple method based on the Mahalanobis distance is presented in (Lee et al., 2018). It assumes that the observations are normally distributed in the representation space, with a shared covariance matrix for all classes. Our distribution assumption is much weaker, assuming that the observations are clustered into a number of clusters, not necessarily Gaussian. In our representation, each class is usually covered by one or more compact support neurons, and each neuron could be involved in multiple classes. Furthermore, (Lee et al., 2018) simply replaces the last layer of the NN with their Mahalanobis measure and makes no attempt to further train the new model, while the CSN layers can be trained together with the whole network.

The Generalized ODIN (Hsu et al., 2020) decomposes the output into a ratio of a class-specific function $h_i(\mathbf{x})$ and a common denominator $g(\mathbf{x})$, both defined over instances $\mathbf{x}$ of the representation space. Good results are obtained using h_i based on the Euclidean distance or cosine similarity. Again, this approach assumes that the observations are grouped in a single cluster for each class, which explains why it uses very deep models (with 34-100 layers) that are more capable to obtain representations that satisfy this assumption. Our method does not make the single cluster per class assumption, and can use deep or shallow models.

The Deterministic Uncertainty Quantification (DUQ) (van Amersfoort et al., 2020) method uses an RBF network and a special gradient penalty to decrease the prediction confidence away from the training examples. The authors also propose a centroid updating scheme to handle the difficulties in training an RBF network. They claim that regularization of the gradient is needed in deep networks to enforce a local Lipschitz condition on the prediction function that will limit how fast the output will change away from the training examples. While their smoothness and Lipschitz conditions might be necessary conditions, they are not sufficient conditions since a smoothly changing function could still have arbitrarily high confidence far away from the training examples. In contrast, our proposed Compact Support Neural Network (CSNN) is guaranteed to have zero outputs away from the training examples, which reflects in lowest possible confidence. Furthermore, the maximum gradient of the CSN layer can be computed explicitly and the Lipschitz condition can be directly enforced by decreasing the neuron support and weight decay. The authors of DUQ also encourage

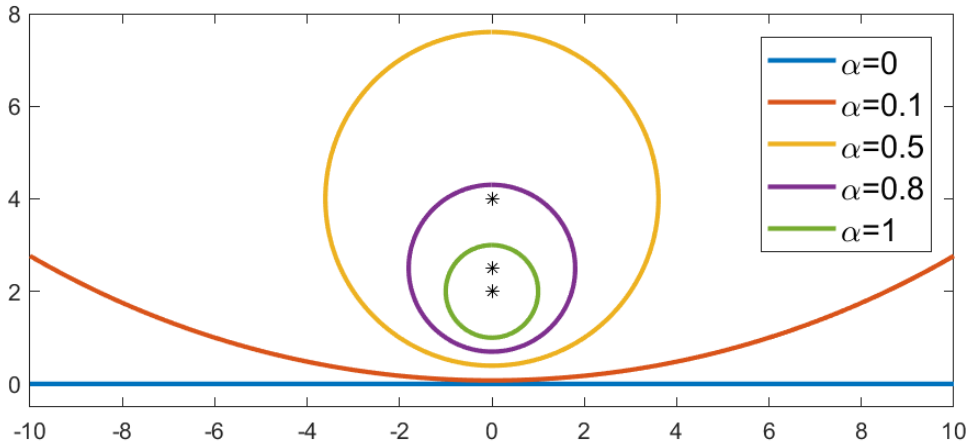


Figure 3.1: The construction (3.3) smoothly interpolates between a standard neuron ($\alpha = 0$) and an RBF-type of neuron ($\alpha = 1$). Shown are the neuron decision boundaries for various values of α.

their gradient to be bounded away from zero everywhere, which they recognize is based on a speculative argument. In contrast, the CSNN gradient is zero away from the training examples, while still obtaining better OOD detection and smaller test errors than DUQ.

3.2 The Compact Support Neural Network

The compact support neural network consists of a number of layers, where the last layer before the output layer contains only compact support neurons, which will be described next. The other layers could be regular neural network or convolutional neural network layers, or compact support layers. The final output layer is a regular linear layer without a bias term, so that it can output a vector of all zeros when appropriate.

3.2.1 The Compact Support Neuron

We start with the radial basis function (RBF) neuron (Broomhead and Lowe, 1988),

$$f(\mathbf{x}, \mathbf{w}) = g(\|\mathbf{x} - \mathbf{w}\|^2). \tag{3.1}$$

The RBF neuron has $g(u) = \exp(-\beta u)$ as the activation function, but in this chapter we will use $g(u) = \max(R - u, 0)$ because it is related to the ReLU.

A flexible representation. We can introduce an extra parameter $\alpha = 1$ and rewrite eq. (3.1) as

$$f_\alpha(\mathbf{x}, \mathbf{w}) = g(\mathbf{x}^T\mathbf{x} + \mathbf{w}^t\mathbf{w} - 2\mathbf{w}^T\mathbf{x}) = g(\alpha(\|\mathbf{x}\|^2 + \|\mathbf{w}\|^2) - 2\mathbf{w}^T\mathbf{x}). \tag{3.2}$$

Using the parameter α, we obtain a representation that smoothly changes between an RBF neuron when $\alpha = 1$ and a standard projection neuron when $\alpha = 0$. However, starting with an RBF neuron with $g(u) = \exp(-\beta u)$, we obtain the projection neuron for $\alpha = 0$ as $f_{\mathbf{w}}(\mathbf{x}) = \exp(2\mathbf{w}^T\mathbf{x})$, which has an exponential activation function.

The compact support neuron. We want to obtain a standard ReLU based neuron $f_\alpha(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x})$ with $\sigma(u) = \max(u, 0)$ for $\alpha = 0$. For this purpose we will use $g(u) = \sigma(R-u)$, and modify the above construction to obtain the compact support neuron:

$$f_\alpha(\mathbf{x}, \mathbf{w}, \mathbf{b}, R) = \sigma(R - \mathbf{x}^T\mathbf{x} - \mathbf{w}^T\mathbf{w} + 2\mathbf{w}^T\mathbf{x}) = \sigma[\alpha(R - \|\mathbf{x}\|^2 - \|\mathbf{w}\|^2 - b) + 2\mathbf{w}^T\mathbf{x} + b], \tag{3.3}$$

where we also introduced a bias term b for the standard neuron. We usually make $b = 0$ for simplicity.

The parameter R determines the radius of the support of the neuron when $\alpha > 0$. In fact, One can easily check that the support of $f_\alpha(\mathbf{x}, \mathbf{w}, \mathbf{b}, R)$ from eq. (3.3) (i.e. the domain where it takes nonzero values) is in a sphere of radius

$$R_\alpha^2 = R + b(1/\alpha - 1) + \|\mathbf{w}\|^2(1/\alpha^2 - 1) \tag{3.4}$$

centered at $\mathbf{w}_\alpha = \mathbf{w}/\alpha$. Therefore the neuron from eq. (3.3) has compact support for any $\alpha > 0$ and the larger the value of α, the smaller the support of the neuron will be. In Figure 3.1 is shown the support for several values of $\alpha \in [0, 1]$ of the neuron (3.3) with $\mathbf{w} = (0, 2)^T, b = 0$ and $R = 1$.

3.2.2 The Compact Support Neural Network

If we have a layer containing only compact support neurons (CSN), combining the weights into a matrix $\mathbf{W}^T = (\mathbf{w}_1, ..., \mathbf{w}_K)$, the biases into a vector $\mathbf{b} = (b_1, ..., b_K)$ and the radii into a vector $\mathbf{r} = (R_1, ..., R_K)$, we can write the CSN layer as:

$$\mathbf{f}_\alpha(\mathbf{x}, \mathbf{W}, \mathbf{b}, \mathbf{r}) = \sigma[\alpha(\mathbf{r} - \mathbf{b} - \mathbf{x}^T\mathbf{x} - Tr(\mathbf{W}\mathbf{W}^T)] + 2\mathbf{W}^T * \mathbf{x} + \mathbf{b}] \tag{3.5}$$

where $\mathbf{f}_\alpha(\mathbf{x}, \mathbf{W}, \mathbf{b}, \mathbf{r}) = (f_1(\mathbf{x}), ..., f_K(\mathbf{x}))T$ is a the vector of neuron outputs of that layer. This formulation enables the use of standard neural network machinery (e.g. Pytorch) to train a CSN. The radius $\mathbf{r}$ is trainable.

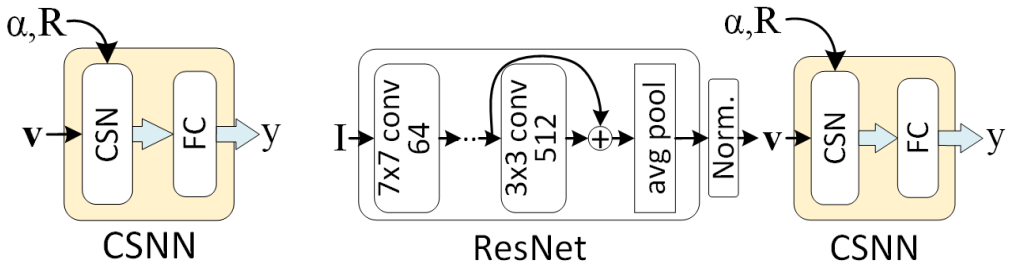


Figure 3.2: Left: Diagram of the compact support neural network (CSNN), with the CSN layer described in Eq. (3.5). Right: an example of the CSNN with normalized input from ResNet. Only the full arrows have backpropagation.

The simplest compact support neural network (CSNN) has two layers: a hidden layer containing compact support neurons (3.3) and an output layer which is a standard fully connected layer without bias. It is illustrated in Figure 3.2, left.

Normalization. It is desirable that $\|\mathbf{x}\|$ be approximately 1 on the training examples so that the value of the radius R does not depend on the dimension d of $\mathbf{x}$. These goals can be achieved by standardizing the variables to have zero mean and standard deviation $1/\sqrt{d}$ on the training examples (where d is the dimension of $\mathbf{x}$). This way $\|\mathbf{x}\|^2 \sim 1$ when the dimension d is large (under assumptions of normality and independence of the variables of $\mathbf{x}$). Our experiments on three datasets indicate that indeed $\|\mathbf{x}\| \sim 1$ on real data when the inputs $\mathbf{x}$ are normalized as described above, as exemplified by the histograms of $\|\mathbf{x}\|$ from Figure 3.3.To achieve this goal, we add a Batch Normalization layer without trainable parameters before the CSN layer.

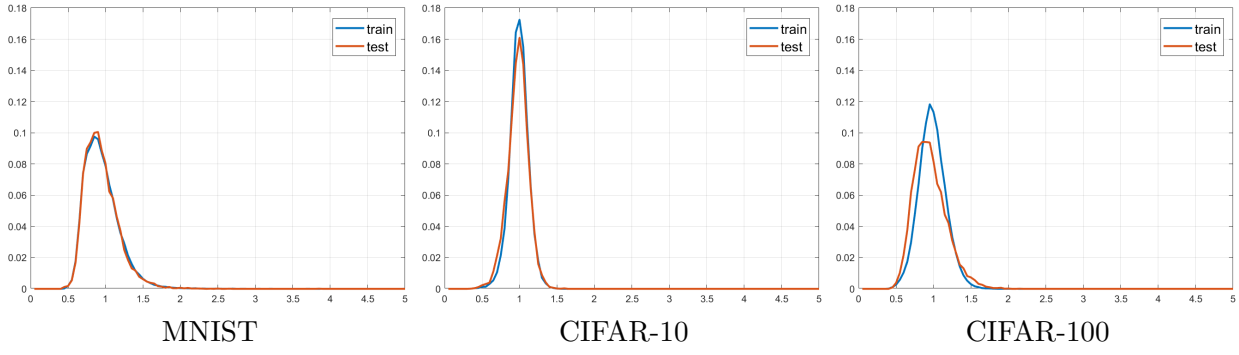


Figure 3.3: Histogram of the norms $\|\mathbf{v}_i\|$ of the normalized input features $\mathbf{v}_i$ to the CSN layer for the three datasets trained in our experiments.

Training. Like the RBF network, training a neural network with such neurons with $\alpha = 1$ is difficult because the loss function has many local optima. To make matters even worse, the compact support neurons have small support when α is close to 1, and consequently the loss function has flat regions between the local minima.

This is why we take another approach to training. Using equations (3.5) we can train a CSNN by first training a regular NN ($\alpha = 0$) and then gradually increasing the shape parameter α from 0 towards 1 while continuing to update the NN parameters. Observe that whenever $\alpha > 0$ the NN has compact support, but the support gets smaller as α gets closer to 1. The training procedure is described in detail in Algorithm 4.

Algorithm 4 Compact Support Neural Network (CSNN) Training

Input: Training set $T = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}\}_{i=1}^n$,

Output: Trained CSNN.

1: Train a regular CNN $\mathbf{f}(\mathbf{x}) = \mathbf{L}\sigma(2\mathbf{W}\mathbf{g}(\mathbf{x})+\mathbf{b})$ where $\mathbf{W}, \mathbf{L}$ are the last two layer weight matrices and $\mathbf{g}(\mathbf{x})$ is the rest of the CNN.

2: Freeze $\mathbf{g}(\mathbf{x})$, compute $\mathbf{u}_i = \mathbf{g}(\mathbf{x}_i), i = 1, ..., n$, their mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$.

3: Obtain normalized versions $\mathbf{v}_i$ of $\mathbf{u}_i$ as $\mathbf{v}_i = (\mathbf{u}_i - \boldsymbol{\mu})/\sqrt{d}\boldsymbol{\sigma}, i = 1, ..., n$.

4: **for** e= 1 to N^{epochs} **do**

5: Set $\alpha = e/N^{epochs}$

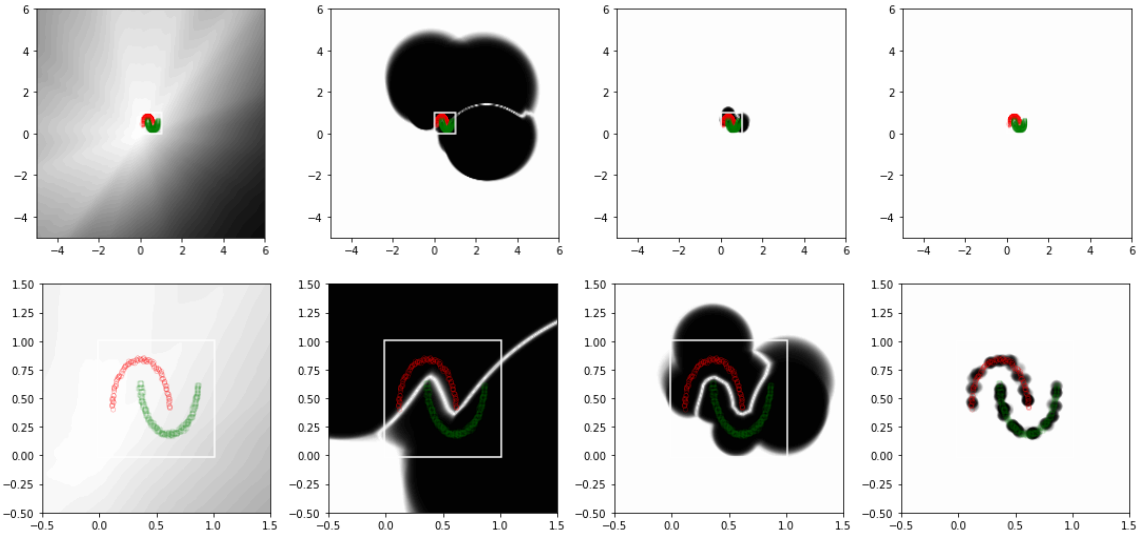
6: Use the examples $(\mathbf{v}_i, y_i)$ to update $(\mathbf{W}, \mathbf{L}, \mathbf{b}, \mathbf{r})$ based on one epoch of

$$\mathbf{f}(\mathbf{v}) = \mathbf{L}\sigma(\alpha[\mathbf{r} - \mathbf{v}^T\mathbf{v} - \mathrm{Tr}(\mathbf{W}\mathbf{W}^T) - \mathbf{b}] + 2\mathbf{W}\mathbf{v} + \mathbf{b})$$

7: **end for**

In the synthetic experiment in Figure 3.4 we succeeded to bring the train and test errors close to 0 for $\alpha = 1$ using a carefully crafted schedule for increasing the α. However, in the real data applications, the training, test and validation errors might first decrease a little bit but ultimately increase as α approaches 1. For example one could see the test errors vs α for the synthetic dataset in Figure 3.5 and for the real datasets in Figure 3.9. For this reason, in practice we stop the training at an $\alpha < 1$ where the training and validation errors still take acceptable values. However, we noticed that the larger the value of α, the tighter the support around the training data and the better the generalization.

It is worth noting that in contrast to the weights of a standard neuron, the weights of the compact support neuron exist in the same space as the neuron inputs and they can be regarded as



$\alpha = 0$ $\alpha = 0.6$ $\alpha = 0.94$ $\alpha = 1$

Figure 3.4: The confidence map (0.5 for white and 1 for black) of the trained CSNN on the moons dataset for different values of $\alpha \in [0, 1]$. Top: zoom out on the interval $[-5, 6]^2$. Bottom: zoom in view of the interval $[-0.5, 1.5]^2$.

templates. Thus they have more meaning, and one could easily visualize the type of responses that make them maximal, using standard neuron visualization techniques such as (Zeiler and Fergus, 2014). Furthermore, one can also obtain samples from the compact support neurons, e.g. for generative or GAN models.

3.3 Experiments

In this section we first present an experiment on 2D data to showcase what can be achieved with the proposed compact support neural network, and then experiments on real datasets to show the power of the CSNN to model real data and how it can detect out-of-distribution samples.

3.3.1 2D Example

We present a first experiment with the moons 2D dataset, where the data is organized on two intertwining half-circle like shapes, one containing the positives and one the negatives. The data is scaled so that all observations are in the interval $[0, 1]^2$ (shown as a white rectangle in Figure 3.4. As out of distribution data (OOD) we started with $100 \times 100 = 10000$ samples on a grid spanning

$[-0.5, 1.5]^2$ and we removed all samples at distance at most 0.1 from the moons data, obtaining 8763 samples.

We used a two layer CSNN, with the first layer having 128 CSN neurons, and the second layer being a standard NN layer without bias, as illustrated in Figure 3.2, left. The second layer is used to integrate the evidence from the CSNN neurons into the class prediction. We used 200 training examples and trained the CSNN using Algorithm 4. We trained 2000 epochs with R decreasing linearly from 0.04 to 0.01, and α increasing from 0 to 1 as $\alpha_i = \min(1, \max(0, (i^{0.1} - 1.5)/.6)), i = 1, ..., 2000$. This way α increases slower as it gets closer to 1. Using this special training we avoided the training and test errors blowing up when α gets close to 1. As specified in line 7 of Algorithm 4, the NN nodes that had zero response on all training examples were eliminated. These neurons cannot be trained anymore and only give uncontrolled responses on unseen data. This way from the 128 neurons, only 73 were left at the end of training.

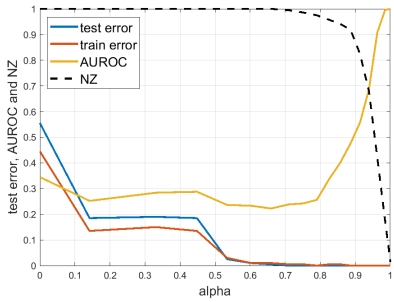
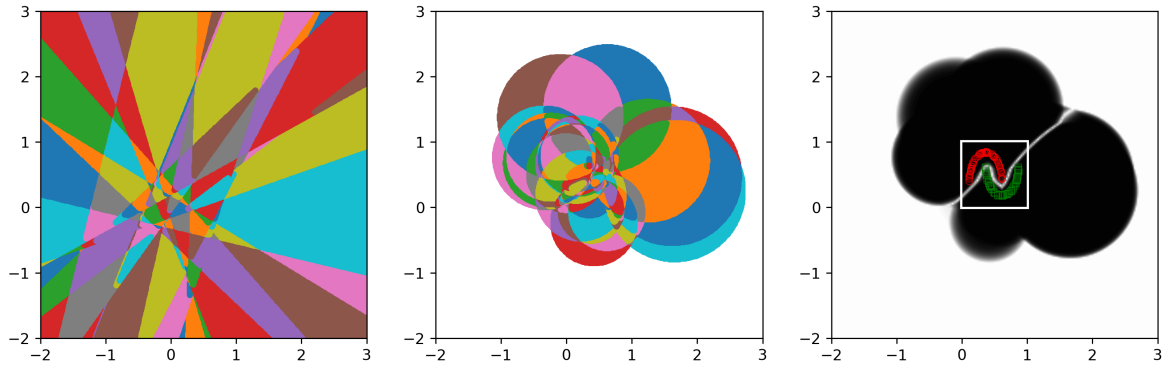


Figure 3.5: CSNN train and test errors, AUROC and percent nonzero outputs (NZ) vs. α for the moons data.

The training/test errors and the AUROC and NZ confidence measures for the OOD data described above vs. α are shown in Figure 3.5. Observe that the training and test errors for $\alpha = 0$ are quite large, because the standard NN with 128 neurons cannot fit the data well enough, and they decrease as the neuron support decreases and the model is better capable to fit the data.

The confidence map for the obtained classifier is shown in Figure 3.4. We can see that the confidence is 0.5 (white) almost everywhere except close to the training data, where it is close to 1 (black). This gives us an insight that the method works as expected, shrinking the support of the neurons to a small domain around the training data. We also see that the support is already reasonably small for $\alpha = 0.6$ and it gets tighter and tighter as α gets closer to 1.



activation patterns $\alpha = 0$ activation patterns $\alpha = 0.825$ confidence map $\alpha = 0.825$

Figure 3.6: Example of activation pattern domains for $\alpha = 0$ and $\alpha = 0.825$ and the resulting confidence map (0.5 for white and 1 for black) for $\alpha = 0.825$ for a 32 neuron 2-layer CSNN.

It is known (Croce and Hein, 2018; Hein et al., 2019) that the output of a ReLU-based neural network is piecewise linear and the domains of linearity are given by the activation pattern of the neurons. The activation pattern of the neurons contains the domains where the set of neurons that are active (i.e. their output is positive) does not change. These activation pattern domains are polytopes, as shown in in Figure 3.6, left, for a two-layer NN with 32 neurons. The activation domains for a CSNN are intersections of circles, as illustrated in Figure 3.6, middle, with the domain where all neurons are inactive shown in white. The corresponding confidence map is shown in Figure 3.6, right.

In real data applications we don't need to go all the way to $\alpha = 1$ since even for smaller α the support is still bounded and if the instance space is high dimensional (e.g. 512 to 1024 in the real data experiments below), the volume of the support of the CNN will be very small compared to the instance space, making it unlikely to have high confidence on out-of-distribution data.

3.3.2 Real Data Experiments

We conduct experiments by training on three different datasets: MNIST (LeCun, 1998), CIFAR-10 and CIFAR-100 (Krizhevsky, 2009). We evaluate the confidence on in-sample and out-of-sample data, by testing them on their respective test sets (in-sample) and on other datasets as shown in Table 3.1 and 3.2, including the test sets of EMNIST (Cohen et al., 2017), FashionMNIST (Xiao et al., 2017) and SVHN (Netzer et al., 2011), and the validation set of ImageNet (Deng et al., 2009).

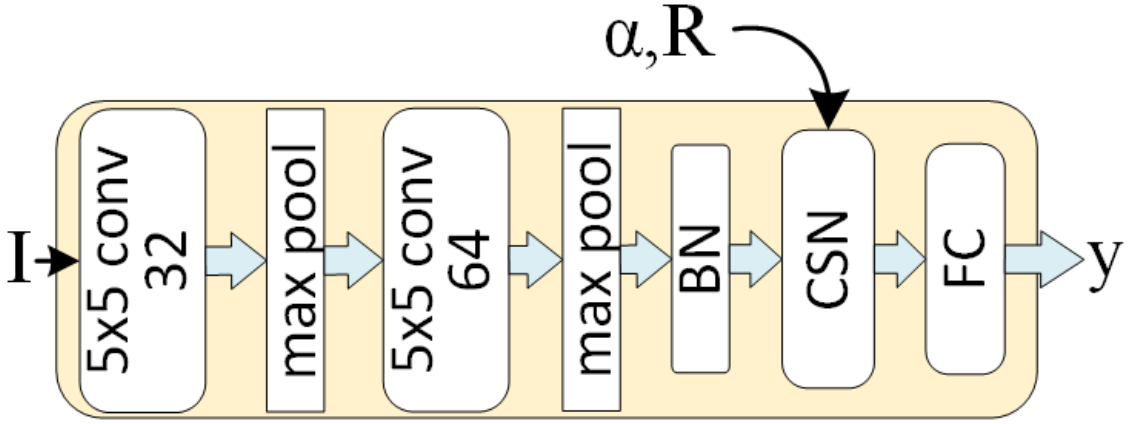


Figure 3.7: The CSNN-F with LeNet backbone, where all layers are trained by backpropagation.

For MNIST we also tested on a grayscale version of CIFAR-10, obtained by converting the 10,000 test images to gray-scale and resizing them to 28×28.

CNN architecture. For MNIST we use a 4-layer LeNet CNN as backbone, with two 5×5 convolution layers with 32 and 64 filters respectively, followed by ReLU and 2×2 max pooling, and two fully connected layers with 256 and 10 neurons. For the other two datasets, we used as backbone a ResNet-18 architecture (He et al., 2016) with 4 residual blocks with 64, 128, 256 and 512 filters respectively. After the backbone CNN has been trained, the FC layers were removed and only the convolutional layers were kept, as illustrated in Figure 3.2, right and Figure 3.7.

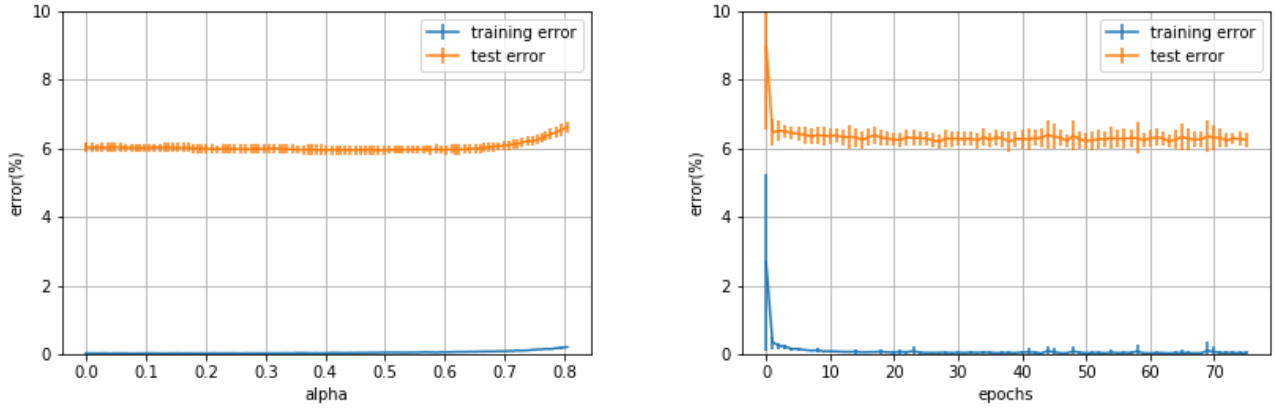
For the CSNN we will experiment with two architectures, illustrated in Figure 3.2 and Figure 3.7. The first is a small one (called CSNN) that takes as input the output of the last convolutional layer of the backbone, normalized as described in Section 3.2.2. The normalization of the CSNN input can also be achieved using a batch normalization layer without any learnable affine parameters. The second one is a full network (called CSNN-F), illustrated in Figure 3.7, where the backbone (LeNet or ResNet) is part of the backpropagation and a batch normalization layer (BN) without any learnable parameters has been introduced between the backbone and the CSN layer.

Training details. For all datasets we used data augmentation with padding (3 pixels for MNIST, 4 pixels for the rest) and random cropping to train the backbones. For CIFAR-100 we also used random rotation up to 15 degrees.We used no data augmentation when training the CSNN and CSNN-F.

The training/test data was passed through the backbone without the FC layers, and the output was normalized. A CSNN without bias term was trained for 510 epochs with $R = 0.1$, of which 10 epochs at $\alpha = 0$. For the CSNN training we used the Adam optimizer with learning rate 0.001 and weight decay 0.0001. We also tried SGD and obtained similar results. The CSNN-F was trained with SGD with a learning rate of 0.001 and weight decay 0.0005. Its layers were initialized with the trained backbone and the trained CSNN with an α smaller or equal than the desired α. The α was kept fixed for two epochs and then increased by 0.005 every epoch for 4 more epochs.

Training the CSNN from $\alpha = 0$ to $\alpha = 1$ for 510 epochs takes less than an hour on a MSI GS-60 Core I7 laptop with 16Gb RAM and Nvidia GTX 970M GPU. Each epoch of the CSNN-F took less than a minute with the LeNet backbone and about 3 minutes with the ResNet-18 backbone.

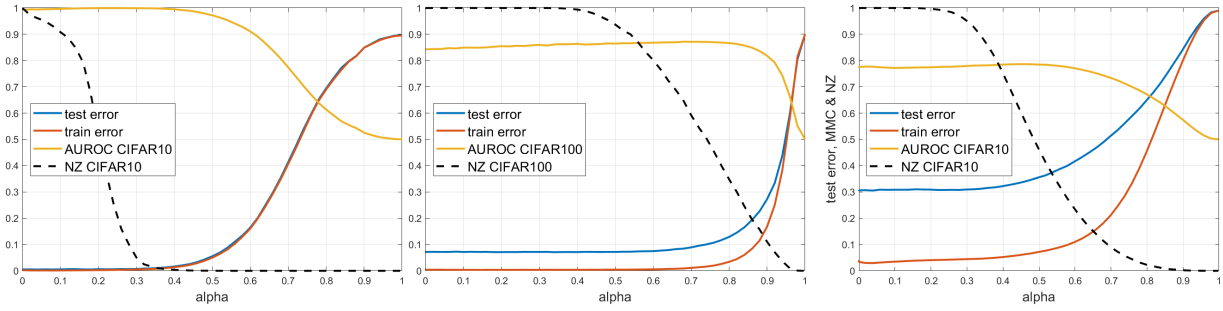
Figure 3.8 shows the mean and standard deviation of the train and test errors obtained by CSNN and RBF classifiers on the CIFAR-10 dataset. We trained 10 ResNet-18 networks as initial models and then trained CSNN and RBF starting from these pre-trained models. We used α as the x-axis for the CSNN plot to show the effect of the shape parameter. From the plot one can see that CSNN is more stable than RBF in this experiment, it has a smaller test error and standard deviation. Also, CSNN converges faster than the RBF network.



a) CSNN b) RBF

Figure 3.8: Mean and standard deviation of train and test errors for CSNN and RBF classifiers trained on CIFAR-10 over 10 independent training runs. a) errors vs. α for CSNN. b) errors vs. epochs for RBF.

OOD detection. The out of distribution (OOD) detection is performed similarly to the way it is done in a standard CNN. For any observation, the maximum value of CSNN raw outputs is used as the OOD score for predicting whether the observation is OOD or not. If the observation is in-distribution, its score will usually be large, and if it is OOD, it will be usually close to zero or even zero. The ROC curve based on these scores for the test set of the in-distribution data (as class 0) and one OOD dataset (as class 1) will give us the AUROC. If the two distributions are not separable (have concept overlap), some of the OOD scores will be large, but for the OOD observations that are away from the area of overlap they will be small or even zero.



MNIST CIFAR-10 CIFAR-100

Figure 3.9: Train and test errors, Area under ROC Curve (AUROC) and percent nonzero outputs (NZ) vs α for CSNN classifiers trained on three real datasets. These results are obtained from one training run.

In Figure 3.9 are shown the train/test errors vs α for the CSNN on the three datasets. Also shown are the Area under the ROC curve (AUROC) for OOD detection on CIFAR-10 or CIFAR-100 and the percentage of OOD samples with nonzero outputs (NZ). Observe that all curves on the real data are very smooth, even though they are obtained from one run, not averaged. We see that the training and test errors stay flat for a while then they start increasing from a certain α that depends on the dataset. At the same time, the AUROC stays flat and slightly increases, and there is a range of values of α where the test error is low and the AUROC is large. Looking in more detail at the CIFAR-10 dataset (middle plot in Figure 3.9), we see that for $\alpha = 0.7$ the NZ-CIFAR100 is about 0.6, which means that about 40% of the CIFAR-100 observations have all 0 CSNN outputs, therefore an OOD score of 0. Combined with the scores of the other observations and the fact that at most 10% of the CIFAR-10 test observations have an OOD score of 0 (because the test error is less than 0.1) it results in a slight increase in AUROC.

Table 3.1: OOD detection comparison between the proposed methods and standard classifiers (CNN and RBF) in terms of Area under the ROC curve (AUROC) for models trained and tested on several datasets. For each model the test error in % is shown in the "Train on" row. All the results are averaged over 10 runs and the standard deviation is shown in parentheses.

	CNN	RBF	CSNN	CSNN-F	10Ens
Trained on MNIST	TE: .53%(0.05)	TE: 0.65%(0.07)	TE: .52% (0.01)	TE:**0.50%** (0.02)	TE:0.51%(0.03)
EMNIST	0.983(0.001)	0.986(0.004)	**0.992(0.001)**	0.990(0.002)	0.985(0.001)
FashionMNIST	0.989(0.001)	**0.998**(0.002)	**0.998**(0.001)	0.997(0.001)	0.992(0.001)
grayCIFAR-10	0.995(0.001)	0.899(0.040)	**1.000**(0.0001)	**1.000**(0.0001)	0.992(0.002)
Average	0.989(0.005)	0.961(0.050)	**0.996**(0.003)	**0.996**(0.004)	0.990(0.004)
Trained on CIFAR-10	TE: **5.99%**(0.09)	TE:18.4%(5.0)	TE: 7.28%(0.06)	TE: 6.18%(0.09)	TE: 4.59(0.08)
CIFAR-100	0.860(0.001)	0.765(0.03)	0.865(0.001)	**0.882**(0.003)	0.897%(0.001)
SVHN	0.899(0.012)	0.810(0.04)	**0.908**(0.001)	0.900(0.013)	0.924(0.002)
ImageNet	0.834(0.002)	0.755(0.03)	0.848(0.001)	**0.854**(0.004)	0.869(0.001)
Average	0.865(0.029)	0.777(0.04)	0.874(0.026)	**0.879**(0.021)	0.897(0.023)
Trained on CIFAR-100	TE: 26.18%(0.28)	TE: 50.23%(1.8)	TE: 30.89%(0.1)	TE: **24.46%**(0.12)	TE:21.86%(0.11)
CIFAR-10	0.750(0.002)	0.652(0.018)	**0.783**(0.001)	0.762(0.002)	0.786(0.001)
SVHN	0.781(0.035)	0.684(0.071)	**0.872**(0.001)	0.860(0.006)	0.834(0.002)
ImageNet	0.766(0.002)	0.679(0.012)	0.755(0.001)	**0.793**(0.001)	0.803(0.001)
Average	0.766(0.023)	0.671(0.045)	0.804(0.050)	**0.805**(0.042)	0.808(0.021)

In practice, α should be chosen as large as possible where an acceptable validation error is still obtained, to have the smallest support possible. For example one could choose the largest α such that the validation error at α is less then or equal to the validation error at $\alpha = 0$. However, for better comparison with the other methods, for each dataset we chose the CSNN classifier corresponding to the largest α where the test error takes a value comparable to the other methods compared, and reported the AUROC values in Table 3.1 and 3.2 . The CSNN-F was obtained by merging the corresponding CSNN head with the ResNet or LeNet backbone and training them together for 6 epochs.

Methods compared. We compare our results with the Adversarial Confidence Enhanced Training (ACET) (Hein et al., 2019), Deterministic Uncertainty Quantification (DUQ) (van Amersfoort et al., 2020), Spectral-normalized Neural Gaussian Process (SNGP) (Liu et al., 2020), a standard CNN and RBF network. The ACET results are taken directly from (Hein et al., 2019), and the DUQ, CNN and ensemble results were obtained using the DUQ authors' code. For DUQ we trained multiple models with various combinations of the length scales $\sigma \in \{0.05, 0.1, 0.2, 0.3, 0.5, 1.0\}$ and gradient penalty $\lambda \in \{0, 0.05, 0.1, 0.2, 0.3, 0.5, 1.0\}$ and selected the combination with the best test error-AUROC trade-off.

Table 3.2: OOD detection comparison of the proposed methods with other non-ensemble OOD detection methods in terms of Area under the ROC curve (AUROC) for models trained and tested on several datasets. For each model the test error in % is shown in the "Train on" row. The ACET results are taken from (Hein et al., 2019). All other results are averaged over 10 runs and the standard deviation is shown in parentheses.

	SNGP	ACET	DUQ	CSNN	CSNN-F
Trained on MNIST	TE:0.54%(0.15)	TE: 0.66%	TE: 0.57%(0.04)	TE:0.52% (0.01)	TE:**0.50%** (0.02)
EMNIST	0.978(0.007)	0.912	0.988(0.001)	**0.992**(0.001)	0.990(0.002)
FashionMNIST	0.988(0.017)	**0.998**	**0.998**(0.001)	**0.998**(0.001)	0.997(0.001)
grayCIFAR-10	0.996(0.002)	**1.000**	0.978(0.005)	**1.000**(0.0001)	**1.000**(0.0001)
Average	0.987(0.013)	0.97	0.988(0.009)	**0.996**(0.003)	**0.996**(0.004)
Trained on CIFAR-10	TE:6.35%(0.08)	TE: 8.44%	TE:6.73%(0.32)	TE: 7.28%(0.06)	TE: **6.18%**(0.09)
CIFAR-100	**0.885**(0.001)	0.852	0.838(0.009)	0.865(0.001)	0.882(0.003)
SVHN	0.923(0.012)	**0.981**	0.914(0.022)	0.908(0.001)	0.900(0.013)
ImageNet	0.864(0.001)	**0.859**	0.824(0.009)	0.848(0.001)	0.854(0.004)
Average	0.891(0.026)	**0.897**	0.859(0.043)	0.874(0.026)	0.879(0.021)
Trained on CIFAR-100	TE: 26.30%(0.26)	TE: 32.24%	TE: 31.60%(0.44)	TE: 30.89%(0.1)	TE: **24.46%**(0.12)
CIFAR-10	0.739(0.002)	0.72	0.719(0.006)	**0.783**(0.001)	0.762(0.002)
SVHN	0.800(0.009)	**0.912**	0.752(0.043)	0.872(0.001)	0.860(0.006)
ImageNet	0.763(0.001)	0.752	0.741(0.004)	0.755(0.001)	**0.793**(0.001)
Average	0.768(0.026)	0.795	0.737(0.028)	0.804(0.050)	**0.805**(0.042)

Results. The results are shown in Table 3.1 and 3.2. All results except the ACET results are averaged over 10 runs and the standard deviation is shown in parentheses.

From Table 3.1 one could see that the trained RBF has difficulties in fitting the in distribution data well, with much larger test errors on CIFAR-10 and CIFAR-100 than the standard CNN. The most probable cause is that the training is stuck in a local optimum, because the training errors were also observed to be large (not shown in the table).

From Table 3.1 we observe that our methods obtain the best results on MNIST and the 10-ensemble obtains the best results on the other two datasets. The test errors of the CSNN-F approach are smaller than the CSNN, and the AUROCs are comparable. Further comparing the test errors, Table 3.1 reveals that the proposed CSNN-F method obtains the smallest test errors on two out of the three datasets. These findings strengthen the claim that the CSNN can fit the in-distribution data as well as a standard CNN. Comparing the detection AUROC on different OOD datasets, CSNN and CSNN-F have better performance than standard classifiers CNN and RBF.

From Table 3.2 one can see that compared to ACET, both CSNN and CSNN-F obtain smaller test errors on all three datasets and better average AUROC on two out of three datasets. Compared to DUQ, the CSNN and CSNN-F obtain comparable test errors and better average AUROC on all

three datasets. Compared to SNGP, the CSNN and CSNN-F obtain comparable test errors and better average AUROC on MNIST and CIFAR-100.

Furthermore, ACET has the highest test errors among the methods. Comparing the training time, both our methods are about 4 times faster than training a 5-ensemble, 8 times faster than a 10- ensemble and about 3 times faster than DUQ.

3.4 Conclusion

In this chapter, we presented a generic neuron formulation that encompasses the standard projection based neuron and the RBF neuron as two extreme cases of a shape parameter $\alpha \in [0, 1]$. By using ReLU as the activation function we obtained a novel type of neuron that has compact support. We showed how to avoid the difficulties in training the compact support NN by training a standard neural network first ($\alpha = 0$) and gradually shrinking the support by increasing α. We showed the advantages of the proposed compact support neural network in that it can still have good prediction on data coming from the same distribution, but it can detect out of distribution samples consistently well. This feature is important in safety critical applications such as autonomous driving, space exploration and medical imaging. Our results have been obtained without any adversarial training or ensembling, and adversarial training or ensembling could be used in our framework to obtain further improvements.

In the real data applications we used a compact support layer as the last layer before the output layer. This ensures that the compact support is involved in the most relevant representation space of the CNN. However, because the CNN still has many projection-based layers to obtain this representation space, it means that the corresponding representation in the original image space does not have compact support and high confidence erroneous predictions are still possible. In the future we plan to study architectures with multiple compact support layers that have even smaller support in the image space.

CHAPTER 4

UNIVERSAL APPROXIMATION USING COMPACT SUPPORT NEURAL NETWORKS

4.1 Introduction

The universal approximation property is the capability of some types of neural networks to approximate with arbitrary accuracy functions from a large class (e.g. continuous functions). In this chapter, we are going to prove that the compact support neural network has the universal approximation property.

There are many studies on universal approximation for feed-forward neural networks. Cybenko (1989) proved that a feed-forward network with one hidden-layer using sigmoid activation and arbitrary width can approximate any function. Then Hornik (1991) showed that the key point is not the activation function, but the architecture is what gives neural networks the ability to be a universal approximators. And in the same year, Park and Sandberg (1991) proved that under certain conditions, Radial-Basis-Function networks are also capable of universal approximation. Finally, Liao et al. (2003) showed that the integrability was not the necessary condition for RBF networks to be universal approximators.

As known, there are two popular kinds of neurons for neural networks. The first one is the standard projection-based neuron, which can be written as:

$$f(\mathbf{x}, \mathbf{w}, \mathbf{b}) = g(\mathbf{w}^T\mathbf{x} + b). \tag{4.1}$$

where $\mathbf{w}$ represents the weight vector and b is the bias for the neuron. The function g is the activation function and the sigmoid function or ReLU are commonly used.

The other type of neuron is the Radial-Basis Function (RBF) neuron, from which the CSNN is derived. It has the following form:

$$f(\mathbf{x}, \mathbf{w}, \mathbf{b}) = g(\frac{||\mathbf{x} - \mathbf{w}||}{b}). \tag{4.2}$$

where $\mathbf{w} \in \mathbb{R}^n$ is called a 'center vector', $b \in \mathbb{R}$ is the 'spread parameter'. For RBF networks, the Gaussian function is always used as activation function $g(x) = \exp(-x^2)$.

4.2 Main Results

In this section, we will introduce conditions for certain networks to be universal approximators and present the proof of universal approximation for the compact support neural network.

Define $L^\infty(\mathbb{R}^d)$ and $L^p(\mathbb{R}^d)$ as the space of functions $f : \mathbb{R}^d \to \mathbb{R}$ such that they are essentially bounded and respectively their p-th power f^p is integrable. Denote the L^p and L^∞ norms by $\| - \|_p$ and $\| - \|_\infty$ respectively.

In Cybenko (1989); Hornik (1991) it was proved that when the activation function g used in the hidden layer is continuous almost everywhere, locally essentially bounded, and not a polynomial, then a two-layered neural network can approximate any continuous function with respect to the $\| - \|_\infty$ norm.

In Park and Sandberg (1991) it was proved that if the RBF neuron used in the hidden layer is continuous almost everywhere, bounded and integrable on $\mathbb{R}^n$, the RBF network can approximate any function in $L^p(\mathbb{R}^n)$ with respect to the L^p norm with $1 \leq p < \infty$. More exactly:

Theorem 1. *(Park and Sandberg, 1991) Let $K : \mathbb{R}^d \to \mathbb{R}$ be a bounded integrable function such that K is continuous almost everywhere and $\int_{\mathbb{R}^d} K(\mathbf{x})d\mathbf{x} \neq 0$. Then the family of functions*

$$q(\mathbf{x}) = \sum_{i=1}^{H} \boldsymbol{\beta}_i K(\frac{\mathbf{x} - \mathbf{z}_i}{c})$$

is dense in $L^p(\mathbb{R}^r)$ for every $p \in [1, \infty)$.

In the above statement $q(\mathbf{x})$ is the two-layer RBF network and H is the number of neurons in the hidden layer.

We are going to use the above result to prove a similar statement for CSNN:

Theorem 2. *Let $R_0 \geq 0$ and $g : \mathbb{R} \to \mathbb{R}$ be a non-negative, continuous and increasing function such that $\int_{\mathbb{R}^p} g(R_0 - \|\mathbf{x}^2\|)d\mathbf{x} < \infty$. Then for any given $\alpha \in (0, 1]$, the family of two-layer compact support*

neural networks: $q(\mathbf{x}) = \sum_{i=1}^{H} \boldsymbol{\beta}_i f_\alpha(\mathbf{x}, \mathbf{w}_i, R_{\mathbf{w}_i})$*, with* $f_\alpha(\mathbf{x}, \mathbf{w}, R) = g(\alpha(R - \|\mathbf{x}\|^2 - \|\mathbf{w}\|^2) + 2\mathbf{w}^T\mathbf{x})$*, and*

$$R_{\mathbf{w}} = \frac{1}{\alpha} R_0 - \|\mathbf{w}\|^2 (\frac{1}{\alpha^2} - 1),$$

is dense in $L^p(\mathbb{R}^d)$ *for every* $p \in [1, \infty)$

Proof. When $0 < \alpha \leq 1$, we have that:

$$f_\alpha(\mathbf{x}, \mathbf{w}, R) = g(\alpha(R - \|\mathbf{x}\|^2 - \|\mathbf{w}\|^2) + 2\mathbf{w}^T\mathbf{x}) = g[\alpha(R + \|\mathbf{w}\|^2(\frac{1}{\alpha^2} - 1)) - \alpha\|\mathbf{x} - \frac{\mathbf{w}}{\alpha}\|^2]$$

To be in the conditions of Theorem 1, we can rewrite the above equation as:

$$f_\alpha(\mathbf{x}, \mathbf{w}, R_{\mathbf{w}}) = K(\frac{\mathbf{x} - \mathbf{w}/\alpha}{c})$$

where $c = 1/\sqrt{\alpha}$, $K(\mathbf{x}) = g(R_0 - \|\mathbf{x}\|^2)$ and

$$R_{\mathbf{w}} = \frac{1}{\alpha} R_0 - \|\mathbf{w}\|^2 (\frac{1}{\alpha^2} - 1)$$

Since $g \geq 0$ is increasing and $g(R_0 - \|x^2\|) \leq g(R_0)$, $K(\mathbf{x})$ is bounded. Since $K(\mathbf{x})$ is also integrable and continuous almost everywhere and $\int_{\mathbb{R}^d} K(\mathbf{x}) \neq 0$, we can apply Theorem 1 with $\mathbf{z}_i = \mathbf{w}_i/\alpha$ and $c = 1/\sqrt{\alpha}$, obtaining the desired result. □

Observe that when $\alpha = 0$, the CSNN becomes a standard neural network, which has been proved to be a universal approximator under certain conditions in Cybenko (1989); Hornik (1991).

CHAPTER 5

FEW SHOT LEARNING WITH CSNN

5.1 Introduction

Humans can learn a new object class from a small number of examples when they meet the new object, while most machine learning or deep learning models need hundreds or thousands of training samples. However, sometimes one cannot find so many observations when labeling the data is expensive or when it is hard to find examples. Therefore, algorithms are desired to learn a new object model from one or only a few training images, a task known as few-shot learning. In Chapter 2, we introduced a new type of neuron that has compact support and can reject the data far-away from the training data. We want to use this feature for few shot learning by recognizing data from a new class as out-of-distribution data.

In the few-shot learning setup, the CSNN will only have one or few OOD observations from the new class for training.

In the proposed approach, we have a CSNN pretrained on the training set. Then we add one or few observations from a new class to the training set, and we add one or few compact support neurons located on the new observations and with a radius given by the average radius of the existing neurons. Therefore, the new trained neurons will also have compact support which will give non-zero response for observations close to the new training data and zero response for data that is far away from the training data. Furthermore, we can control the confidence domains by adjusting the shape parameter α and radius parameter $\mathbf{r}$.

5.2 Related Work

There are many studies about learning with few examples, using prior knowledge that includes prior knowledge about data or about the model.

The model can be more reliable if trained on a augmented dataset using prior knowledge about data. Data augmentation conducted by using hand-crafted rules is wildly used in few shot learning with image data including flipping, cropping (Qi et al., 2018) etc. However, it requires much domain

knowledge and it is expensive to set up the rules. Another approach is to learn a transformation function on the observations and obtain more samples after transformation (Miller et al., 2000; Schwartz et al., 2018). Data augmentation can be used together with the proposed approach in this chapter to help improve performance.

Generative modeling methods use the prior knowledge of pretrained models and have good performance in many application domains, such as recognition, reconstruction (Gordon et al., 2018), etc.. They assume that the observations are drawn from some probability distribution $p(x;\theta)$, while our proposed approach assumes that the observations from the same class are close to each other in a certain instance space.

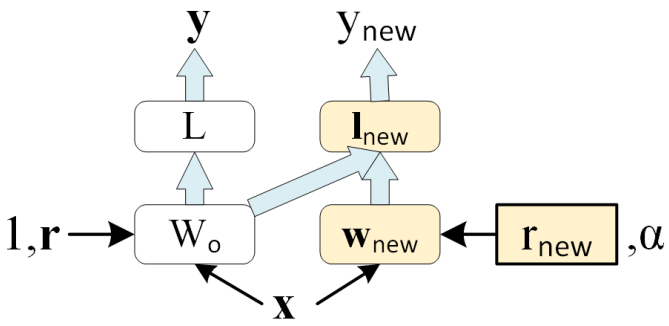


Figure 5.1: Diagram of adding a new class and a new CSN neuron for few shot learning.

5.3 Few Shot Learning with CSNN

The Compact Support Neuron Network was proposed to prevent high confidence predictions on data far away from the training data, and it can also be used to train new classes with few samples. When the outputs of OOD observations have many zeros, the network is telling us "I don't know what it is". In this situation, we can augment the model and train one or few neurons on the samples from the new class and add them to the pretrained CSNN. However, training only parts of the network encounters some implementation difficulties, so we set up a new network with only two layers. One is a CSN layer with only new compact support neurons and the other one is the output layer with only one neuron connected to all compact support neurons, as illustrated in Figure 5.1.

Let $T = \{T_1, T_2\}$ represent the training dataset, where $T_1 = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{Z}\}_{i=1}^{m}$ is the initial training data and $T_2 = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{Z}\}_{i=m+1}^{m+n}$ represents the samples from the new class. We obtain some features from the pretrained CSNN: $\mathbf{b}(\mathbf{x})$ from the batch normalization layer, $\mathbf{c}(\mathbf{x})$ from CSN layer and $\mathbf{s}$ from the output layer. Then we initialize the weights $\mathbf{w}_{new}$ of the new neurons with the mean value of $\mathbf{b}(\mathbf{x})\{\mathbf{x} \in T_2\}$, radius parameter $\mathbf{r}_{new}$ with r_μ(mean value of $\mathbf{r}$ from the pretrained CSNN), and $\mathbf{l}_{new}$ represents the weights of the output layer. The details about the training procedure are presented in Algorithm 5.

Algorithm 5 New class training with CSNN

Input: Training set $T = \{T_1, T_2\}, T_1 = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{Z}\}_{i=1}^{m}$, $T_2 = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{Z}\}_{i=m+1}^{m+n}$

Output: Trained new CSNN.

1: Obtain output features from the pre-trained CSNN. $\mathbf{b}(\mathbf{x})$ from Batch Normalization layer, $\mathbf{c}(\mathbf{x})$ from CSN layer and $\mathbf{s}$ from the output layer.

2: Initialize the weights for the new CSN as $\mathbf{w}_{new} = \frac{1}{|T_2|}\sum_{(\mathbf{x},y)\in T_2}\mathbf{b}(\mathbf{x})$, the output layer weights $\mathbf{l}_{new}$ and the radius parameter $r_{new} = r_\mu$.

3: **for** e= 1 to N^{epochs} **do**

4: $\alpha_{new} = e/N^{epochs}$

5: Use the examples $(\mathbf{x}_i, y_i)$ to update $(\mathbf{w}_{new}, \mathbf{l}_{new}, r_{new})$ for the new CSNN based on one epoch of the new classifier

$$\mathbf{f}(\mathbf{x}) = [\mathbf{L}f_1(\mathbf{x}, \mathbf{W}_o, 0, \mathbf{r}_o), (f_1(\mathbf{x}, \mathbf{W}_o, 0, \mathbf{r}_o), f_\alpha(\mathbf{x}, \mathbf{w}_{new}, 0, r_{new}))\mathbf{l}_{new}]$$

6: **end for**

5.4 Experiments

In this section, we conduct two experiments on the Iris dataset (Dua and Graff, 2017), which contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The observations in the Iris dataset have 4 features so we did principal component analysis(PCA) at first to reduce the dimension of the dataset in order to visualize the data, and the distribution of the 3 classes is shown in Figure 5.2(Left). After splitting the dataset, we obtained 120 training examples and 30 test examples. We use two CSNN architectures for few shot learning in the experiments. The first one is a simple CSNN with only two layers while the other one is deeper with 4 layers and we also set the radius parameter $\mathbf{r}$ trainable in the experiments. For both CSNNs, we only trained one more compact support neuron for the new class.

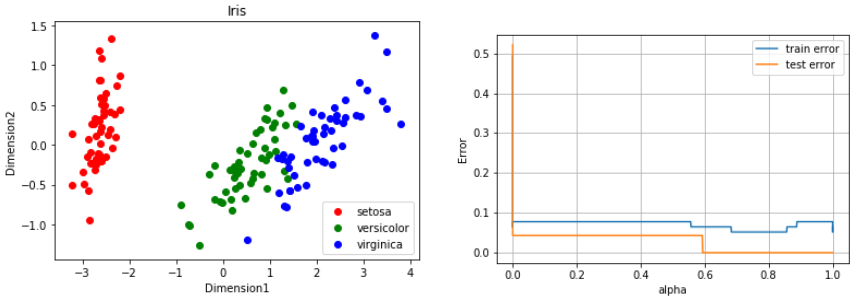
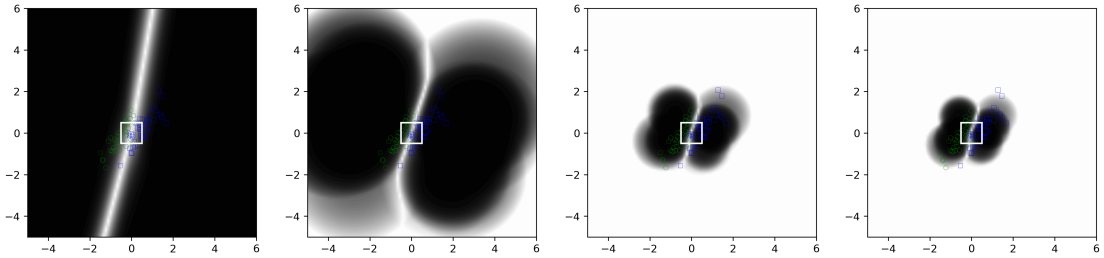


Figure 5.2: Left:Scatter plot of the dataset after principal component analysis. Right:Training/Test errors vs.α

5.4.1 Simple CSNN

In the first experiment, we use a very simple CSNN with only a CSN layer containing 20 neurons and an output layer. We take the versicolor(green) and virginica(blue) as the in-distribution data for training the CSNN while using the setosa(red) as the new class from which only a three samples will be added.

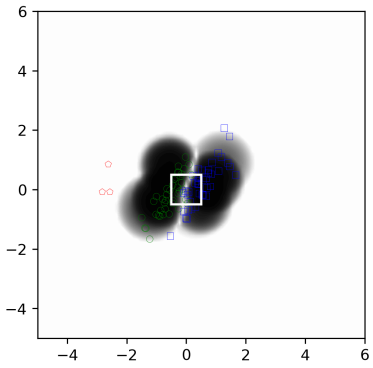
We present the train/test error vs. α in Figure5.2(Right) and the confidence maps of CSNN with various value of $\alpha \in [0, 1]$ and the mean value of $\mathbf{r}$ in Figure5.3.



$\alpha = 0$ $\mathbf{r}_{mu} = 1.000$ | $\alpha = 0.250$ $\mathbf{r}_{mu} = 1.009$ | $\alpha = 0.750$ $\mathbf{r}_{mu} = 1.043$ | $\alpha = 1$ $\mathbf{r}_{mu} = 1.080$

Figure 5.3: The confidence map (0.5 for white and 1 for black) of the trained CSNN on the iris dataset for different values of α and mean of $\mathbf{r}$. The interval is $[-5, 6]^2$.

Since the errors and confidence maps show that the model can fit the data well, we take 3 observations from the new class setosa(red) which is away from the in-distribution data to the model and present the confidence map in Figure 5.4. To train the neuron for the new class, we take the approach presented in Algothrim 5 and obtain results as expected.



$\alpha = 1$
$\mathbf{r}_{mu} = 1.080$

Figure 5.4: Left: Confidence map with three observations from the new class, which are far away from the training data.

Training Details. The training dataset we use contains 81 in-distribution data (initial training set) and 3 out-of-distribution data (new class) while the rest of the new class observations are added to the test dataset. We trained the new neuron with shape parameter α gradually increasing from 0 to 1 and then keep training the radius parameter $\mathbf{r}$ for some more epochs while $\alpha = 1$.

Results. In Figure 5.5, we present the confidence maps for various values of α and $\mathbf{r}_{new}$. We can observe that the compact support is decreasing when α is increasing and it begins to growing while α is fixed at 1. The confidence maps give us an insight that the model works as expected and the new neuron can fit the new class well which can also be obtained from Figure 5.6. The training/test errors are small for the three classes and the new neuron has high confidence for the new class. The histogram in Figure 5.6 shows the non-zero response for test dataset which contains 9 versicolor(green), 10 virginica(blue) and 47 setosa(red). This also proves that only the new neuron give response to the new class while other compact support neurons from pre-trained CSNN giving zero outputs for the new class.

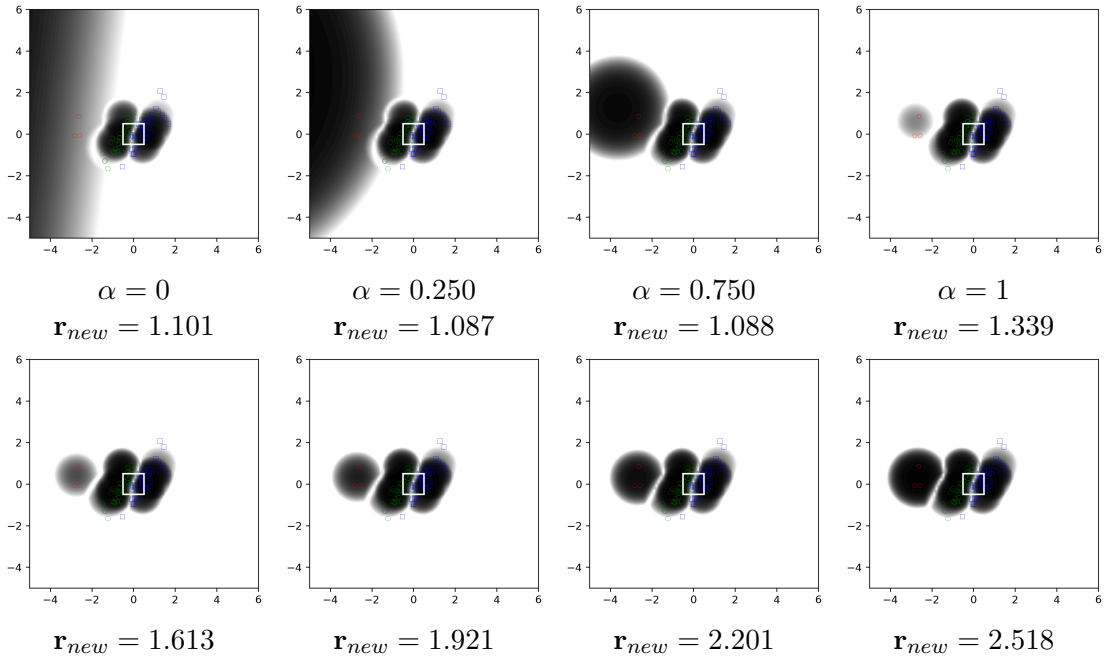


Figure 5.5: Top: The confidence map (0.5 for white and 1 for black) of the trained new CSNN on the iris dataset with different values of $\alpha \in [0, 1]$ and mean of $\mathbf{r}$. The interval is $[-5, 6]^2$. Bottom:The confidence map (0.5 for white and 1 for black) of the trained new CSNN on the iris dataset with $\alpha = 1$ and mean of $\mathbf{r}$.

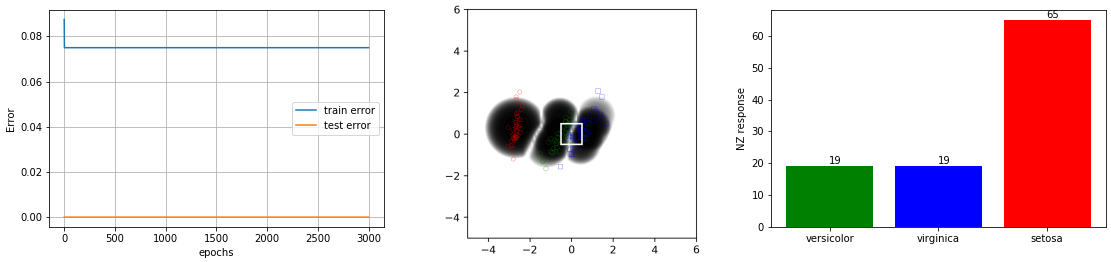


Figure 5.6: Left:Train/test errors for the network after adding a neuron vs. epochs. Middle: The confidence map with all out-of-distribution data. Right: Histogram of sum of non-zero outputs from the 3 output neurons on the test set.

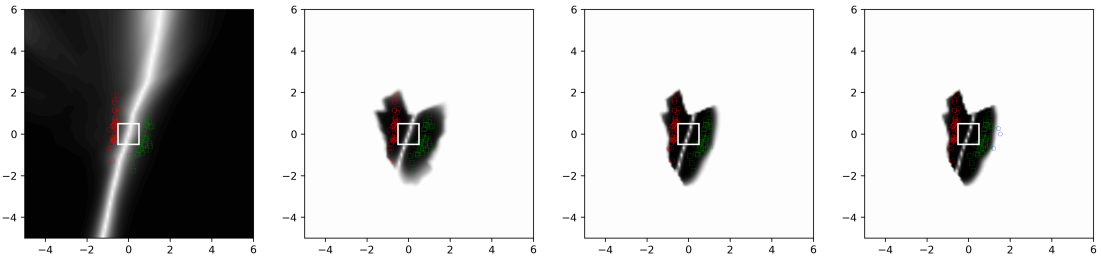
5.4.2 Deeper CSNN

In this experiment we take the setosa(red) and versicolor(green) as the in-distribution data to train a CSNN while using the virginica(blue) as the out of distribution dataset. A deeper CSNN with 4 layers is used in this part which contains 2 hidden layers with 64 and 128 standard projection neurons, a batch normalization layer without trainable parameters, a CSN layer with 256 compact

Table 5.1: Training/test error from new CSNN on different dataset

Methods(New CSNN)	Simple	Deeper
Training Error(Without new class samples)(%)	7.79	0.00
Training Error(With new class samples)(%)	7.50	1.18
Test Error(first two classes)(%)	0.00	0.00
Test Error(third class)(%)	0.00	14.89

support neurons and an output layer. We also take 3 out-of-distribution observations as training data for the new class.



$\alpha = 0$ $\mathbf{r}_{mu} = 1.000$ | $\alpha = 0.249$ $\mathbf{r}_{mu} = 1.005$ | $\alpha = 0.749$ $\mathbf{r}_{mu} = 1.015$ | $\alpha = 1$ $\mathbf{r}_{mu} = 1.015$

Figure 5.7: The confidence maps (0.5 for white and 1 for black) of the trained CSNN on the iris dataset for different values of α and mean of $\mathbf{r}$. The OOD data is also shown in the last figure. The interval is $[-5, 6]^2$.

Results. The confidence maps shown in Figure 5.7 provided an insight that how the CSNN worked on fitting the data in higher dimensional space while having low confidence for the OOD data. We also present the confidence maps for training the new class in Figure 5.8 with different values of α and $\mathbf{r}_{new}$. The confident area is decreasing while pushing the α from 0 to 1 and then increasing with the training radius parameter $\mathbf{r}_{new}$. At last, the shape of the confidence area of the new neuron has similar patterns as the OOD data. Further, we present in Table 5.1 the training/test errors on the first two classes and the third class for the two architectures. The most probable reason for the large training error of the simple CSNN is that the two training class are close to each other. This also happens to deeper CSNN where third class is close to one of the initial training classes.

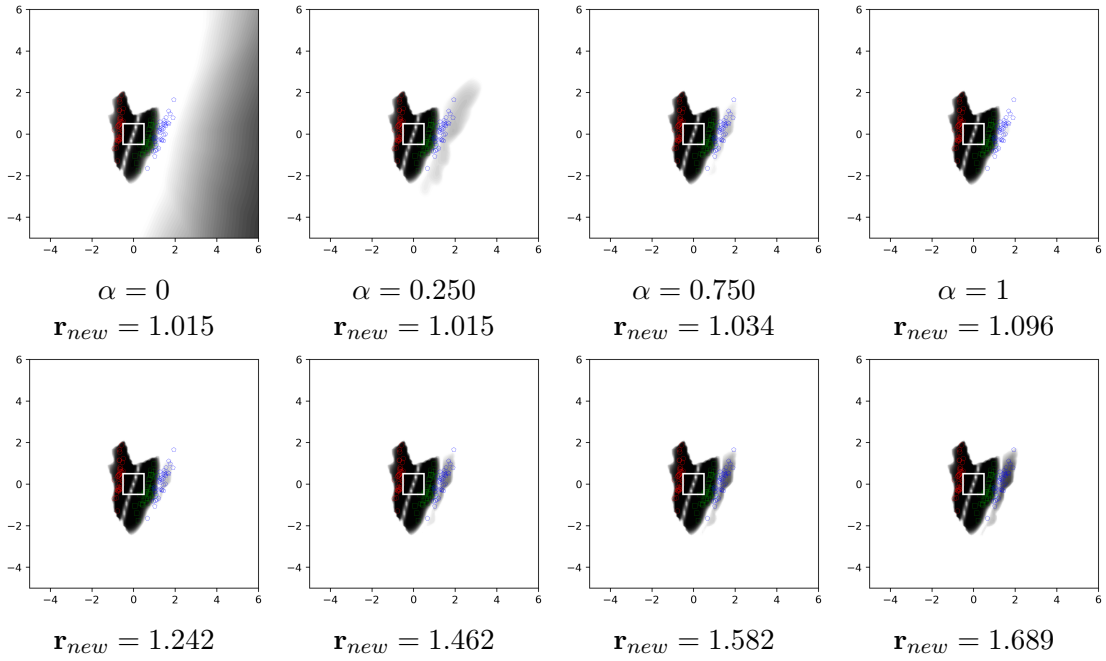


Figure 5.8: Top: The confidence map (0.5 for white and 1 for black) of the trained new CSNN on the iris dataset with different values of $\alpha \in [0, 1]$ and mean of $\mathbf{r}$. The interval is $[-5, 6]^2$. Bottom:The confidence map (0.5 for white and 1 for black) of the trained new CSNN on the iris dataset with $\alpha = 1$ and mean of $\mathbf{r}$.

5.5 Conclusion

In this chapter, we showed how to update a model with a few samples from a new class using prior knowledge from the pretrained CSNN, and obtained good performance on two different CSNNs. While training only parts of the CSNN using standard machinery (PyTorch) is difficult, we presented another way to train the new neurons by setting a new network with only two layers. Moreover, since the second experiment showed an insight of how compact support neurons work in the high dimensional space and can obtain good performance, we will try to apply this method on other more challenging datasets.

CHAPTER 6

CONCLUSION

In this dissertation, I bring four contributions to computer vision.

In Chapter 2, I apply a recent feature selection method FSA (Feature Selection with Annealing) to dictionary learning, which uses an L_0 constraint instead of the L_1 penalty and obtains a more accurate sparse representation. In experiments, I compare the proposed method with other feature selection methods such as LARS and IRLS and obtains smaller reconstruction errors for the same sparsity level and dictionary size. Furthermore, experiments on the MNIST dataset using SVM, Random Forest and K-Nearest Neighbors show that the method can produce a sparse image representation that obtains a smaller misclassification error than directly using the image as input and the sparse representation by FSA again outperforms LARS and IRLS on the classification task.

In Chapter 3, I introduce a new type of neural network with a novel neuron called compact support neuron, which has the standard dot-product based neuron and the RBF neuron as two extreme cases of a shape parameter α. Using ReLU as the activation function, I can obtain zero outputs when the input is far away from the training data. I also show how to avoid the difficulties in training the compact support NN by training a standard neural network first ($\alpha = 0$) and gradually shrinking the support by increasing α. Experiments indicate that the proposed compact support neural network can detect out of distribution samples consistently well while having good prediction on data from the same distribution. This feature is important in safety critical applications such as autonomous driving, space exploration and medical imaging.

In Chapter 4, I present the proof of the universal approximation property for compact support neural networks. By introducing the conditions for certain networks to be universal approximators and showing that the CSNN meets these conditions, I prove that the compact support neural network can approximate with arbitrary accuracy functions from a large class (e.g. L^p functions).

In the last chapter, I succeed to train OOD observations with few samples using pretrained CSNN and obtain good results on different dimensional space. I also propose an approach to train new neurons for a pretrained CSNN using standard neural network machinery (PyTorch). Since the

experiments show the insight of how compact support neurons worked in high dimensional space and can obtain good performance, I could try to apply this method on other more challenging datasets.

BIBLIOGRAPHY

Barbu, A., She, Y., Ding, L., and Gramajo, G. (2016). Feature selection with annealing for computer vision and big data learning. *IEEE transactions on pattern analysis and machine intelligence*, 39(2):272–286.

Broomhead, D. S. and Lowe, D. (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom).

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). Emnist: an extension of mnist to handwritten letters (2017). *arXiv preprint arXiv:1702.05373*.

Croce, F. and Hein, M. (2018). A randomized gradient-free attack on relu networks. In *German Conference on Pattern Recognition*, pages 215–227. Springer.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255.

Donoho, D. L. and Elad, M. (2003). Optimally sparse representation in general (nonorthogonal) dictionaries via ł1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al. (2004). Least angle regression. *The Annals of statistics*, 32(2):407–499.

Fan, J. and Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360.

Fu, W. J. (1998). Penalized regressions: the bridge versus the lasso. *Journal of computational and graphical statistics*, 7(3):397–416.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. E. (2018). Meta-learning probabilistic inference for prediction. *arXiv preprint arXiv:1805.09921*.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330.

Guo, S., Wang, Z., and Ruan, Q. (2013). Enhancing sparsity via łp (0¡ p¡ 1) minimization for robust face recognition. *Neurocomputing*, 99:592–602.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hein, M., Andriushchenko, M., and Bitterwolf, J. (2019). Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 41–50.

Hendrycks, D. and Gimpel, K. (2017). A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Hsu, Y.-C., Shen, Y., Jin, H., and Kira, Z. (2020). Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. In *CVPR*, pages 10951–10960.

Jiang, H., Kim, B., Guan, M., and Gupta, M. (2018). To trust or not to trust a classifier. In *Advances in neural information processing systems*, pages 5541–5552.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Master's thesis, University of Tronto*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Lee, K., Lee, K., Lee, H., and Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *NeurIPS*, pages 7167–7177.

Liao, Y., Fang, S.-C., and Nuttle, H. L. (2003). Relaxed conditions for radial-basis function networks to be universal approximators. *Neural Networks*, 16(7):1019–1028.

Liu, J., Lin, Z., Padhy, S., Tran, D., Bedrax Weiss, T., and Lakshminarayanan, B. (2020). Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems*, 33:7498–7512.

Lyu, Q., Lin, Z., She, Y., and Zhang, C. (2013). A comparison of typical łp minimization algorithms. *Neurocomputing*, 119:413–424.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *ICLR*.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696.

Miller, E. G., Matsakis, N. E., and Viola, P. A. (2000). Learning from one example through shared densities on transforms. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 1, pages 464–471. IEEE.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436.

Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325.

Park, J. and Sandberg, I. W. (1991). Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257.

Patel, V. M. and Chellappa, R. (2011). Sparse representations, compressive sensing and dictionaries for pattern recognition. In *The first Asian conference on pattern recognition*, pages 325–329. IEEE.

Qi, H., Brown, M., and Lowe, D. G. (2018). Low-shot learning with imprinted weights. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5822–5830.

Schwartz, E., Karlinsky, L., Shtok, J., Harary, S., Marder, M., Kumar, A., Feris, R., Giryes, R., and Bronstein, A. (2018). Delta-encoder: an effective sample synthesis method for few-shot object recognition. *Advances in Neural Information Processing Systems*, 31.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

van Amersfoort, J., Smith, L., Teh, Y. W., and Gal, Y. (2020). Uncertainty estimation using a single deep deterministic neural network. In *international conference on machine learning*.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Yang, A. Y., Sastry, S. S., Ganesh, A., and Ma, Y. (2010). Fast ł1-minimization algorithms and an application in robust face recognition: A review. In *2010 IEEE international conference on image processing*, pages 1849–1852. IEEE.

Yang, M., Zhang, L., Feng, X., and Zhang, D. (2011). Fisher discrimination dictionary learning for sparse representation. In *2011 International Conference on Computer Vision*, pages 543–550. IEEE.

Yuan, Y., Lu, X., and Li, X. (2014). Learning hash functions using sparse reconstruction. In *Proceedings of International Conference on Internet Multimedia Computing and Service*, pages 14–18.

Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, pages 6023–6032.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833.

Zhang, Q. and Li, B. (2010). Discriminative k-svd for dictionary learning in face recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2691–2698. IEEE.

Zhang, Z., Xu, Y., Yang, J., Li, X., and Zhang, D. (2015). A survey of sparse representation: algorithms and applications. *IEEE access*, 3:490–530.

BIOGRAPHICAL SKETCH

Hongyu Mou received his bachelor's degree from South Western University of Finance and Economics in 2015 in Statistics. Then he came to the Statistics Department at the Florida State University to pursue his master's degree. In 2017, he passed the qualifying exam and joined the Ph.D program. Since then, he was doing research in computer vision which included dictionary learning, deconvolutional networks, compact support neural networks and etc. He published a paper with his Ph.D advisor Dr,Barbu named 'Accurate Dictionary Learning with Direct Sparsity Control' in ICIP2018.