

HYPERPARAMETER OPTIMIZATION OF DEEP
CONVOLUTIONAL NEURAL NETWORKS
ARCHITECTURES FOR OBJECT RECOGNITION

Saleh Albelwi

Under the Supervision of Dr. Ausif Mahmood

DISSERTATION
SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
AND ENGINEERING
THE SCHOOL OF ENGINEERING
UNIVERSITY OF BRIDGEPORT
CONNECTICUT

April 2018

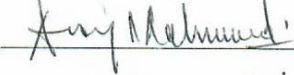
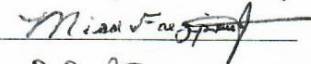
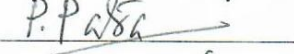
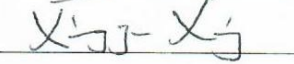

HYPERPARAMETER OPTIMIZATION OF DEEP CONVOLUTIONAL NEURAL NETWORKS ARCHITECTURES FOR OBJECT RECOGNITION

Saleh Albelwi

Under the Supervision of Dr. Ausif Mahmood

Approvals

Committee Members

Name	Signature	Date
Dr. Ausif Mahmood		2-13-2018
Dr. Miad Faezipour		Feb. 13, 2018
Dr. Prabir Patra		Feb 13, 18
Dr. Xingguo Xiong		02/13/18
Dr. Saeid Moslehpour		Feb 6, 18

Ph.D. Program Coordinator

Dr. Khaled M. Elleithy  Feb 15, 2018

Chairman, Computer Science and Engineering Department

Dr. Ausif Mahmood  2-13-2018

Dean, School of Engineering

Dr. Tarek M. Sobh  2/15/2018

HYPERPARAMETER OPTIMIZATION OF DEEP
CONVOLUTIONAL NEURAL NETWORKS
ARCHITECTURES FOR OBJECT RECOGNITION

© Copyright by Saleh Albelwi 2018

ABSTRACT

Recent advances in Convolutional Neural Networks (CNNs) have obtained promising results in difficult deep learning tasks. However, the success of a CNN depends on finding an architecture to fit a given problem. A hand-crafted architecture is a challenging, time-consuming process that requires expert knowledge and effort, due to a large number of architectural design choices. In this dissertation, we present an efficient framework that automatically designs a high-performing CNN architecture for a given problem. In this framework, we introduce a new optimization objective function that combines the error rate and the information learnt by a set of feature maps using deconvolutional networks (deconvnet). The new objective function allows the hyperparameters of the CNN architecture to be optimized in a way that enhances the performance by guiding the CNN through better visualization of learnt features via deconvnet. The actual optimization of the objective function is carried out via the Nelder-Mead Method (NMM). Further, our new objective function results in much faster convergence towards a better architecture. The proposed framework has the ability to explore a CNN architecture's numerous design choices in an efficient way and also allows effective, distributed execution and synchronization via web services. Empirically, we demonstrate that the CNN architecture designed with our approach outperforms several existing approaches in terms of its error rate. Our results are also competitive with state-of-the-art results on the MNIST dataset and perform reasonably against the state-of-the-art results on CIFAR-10 and CIFAR-100 datasets. Our approach has a significant role in

increasing the depth, reducing the size of strides, and constraining some convolutional layers not followed by pooling layers in order to find a CNN architecture that produces a high recognition performance.

Moreover, we evaluate the effectiveness of reducing the size of the training set on CNNs using a variety of instance selection methods to speed up the training time. We then study how these methods impact classification accuracy. Many instance selection methods require a long run-time to obtain a subset of the representative dataset, especially if the training set is large and has a high dimensionality. One example of these algorithms is Random Mutation Hill Climbing (RMHC). We improve RMHC so that it performs faster than the original algorithm with the same accuracy.

ACKNOWLEDGEMENTS

My thanks are wholly devoted to God, who has helped me complete this work successfully. I owe a debt of gratitude to my family for their understanding and encouragement. I am very grateful to my father for raising me and encouraging me to achieve my goal. Special thanks go to my wife and my kids; I could have never achieved this without their support.

I would like to express special thanks to my supervisor Dr. Ausif Mahmood for his constant guidance, comments, and valuable time. Without his support, I would not have been able to finish this work. My appreciation also goes to Dr. Khaled Elleithy for his feedback and support, and to my committee members Dr. Miad Faezipour, Dr. Prabir Patra, Dr. Xingguo Xiong, and Dr. Saeid Moslehpour for their valuable time and suggestions.

TABLE OF CONTENTS

ABSTRACT	iv
ACKNOWLEDGEMENTS	vi
TABLE OF CONTENTS	vii
LIST OF TABLES	x
LIST OF FIGURES	xii
CHAPTER 1: INTRODUCTION	1
1.1 Research Problem and Scope	3
1.2 Motivation behind the Research.....	4
1.3 Contributions.....	4
CHAPTER 2: BACKGROUND AND LITERATURE REVIEW.....	6
2.1 Deep Learning.....	6
2.2 Backpropagation and Gradient Descent.....	7
2.3 Convolutional Neural Networks.....	9
2.3.1 Convolutional Layers	10
2.3.2 Pooling Layers.....	12
2.3.3 Fully-connected Layers	13
2.4 CNN Architecture Design	14
2.4.1 Literature Review on CNN Architecture Design	16
2.5 Regularization	19
2.6 Weight initialization.....	19
2.7 Visualizing and Understanding CNNs	20
2.8 Similarity Measurements between Images.....	21
2.9 Sparse Autoencoder	22

2.10	Analysis of optimized instance selection algorithms on large datasets with CNNs.....	24
2.10.1	Literature Review for Instance Selection	26
2.11	A Deep Architecture for Face Recognition Based on Multiple Feature Extractors	28
2.11.1	Literature Review for Multiple Classifiers.....	29
CHAPTER 3: RESEARCH PLAN		32
3.1	A Framework for Designing the Architectures of Deep CNNs	32
3.1.1	Reducing the Training Set.....	33
3.1.2	CNN Feature Visualization Methods	34
3.1.3	Correlation Coefficient.....	38
3.1.4	Objective Function	39
3.1.5	Nelder Mead Method	40
3.1.1	Accelerating Processing Time with Parallelism.....	44
3.2	Analysis of optimized instance selection algorithms on large datasets with CNNs.....	47
3.3	A Deep Architecture for Face Recognition based on Multiple Feature Extractors.....	49
3.3.1	Principle Component Analysis (PCA)	49
3.3.2	Local Binary Pattern (LBP).....	50
3.3.3	Stacked Sparse Autoencoder (SSA).....	51
CHAPTER 4: IMPLEMENTATION AND RESULTS.....		54
4.1	A Framework for Designing the Architectures of CNNs.....	54
4.1.1	Datasets	54
4.1.2	Experimental Setup	55
4.1.3	Results and Discussion.....	58
4.1.4	Statistic Power Analysis.....	65
4.2	Analysis of optimized instance selection algorithms on large datasets with CNNs.....	67
4.2.1	Dataset.....	67

4.2.2 Training methodology	67
4.2.3 Experimental results and discussion.....	68
4.3 A Deep Architecture for Face Recognition Based on Multiple Feature Extractors	70
4.3.1 Datasets	70
4.3.2 Training setting	71
4.3.3 Experimental Results and Discussion	72
CHAPTER 5: CONCLUSIONS	75
REFERENCES	77

LIST OF TABLES

Table 4.1	Hyperparameter initialization ranges	58
Table 4.2	Error rate comparisons between the top CNN architectures obtained by our objective function and the error rate objective function via NMM	59
Table 4.3	Error rate comparison for different methods of designing CNN architectures on CIFAR-10 and CIFAR-100. These results are achieved without data augmentation	60
Table 4.4	Comparison of execution time by serial NMM and parallel NMM for Architecture Optimization	63
Table 4.5	Error rate comparisons with state-of-the-art methods and recent works on architecture design search. We report results for CIFAR-10 and CIFAR-100 after applying data augmentation and results for MNIST without any data augmentation	64
Table 4.6	Uniform CNN architecture summary	68
Table 4.7	Illustrates the accuracy of different instance	68
Table 4.8	Running time comparison between original RMHC and our proposed approach of RMHC for one iteration	69
Table 4.9	Performance of different classifiers on the ORL and AR	73

databases, including individual classifiers and MC systems

Table 4.10	Performance of classifiers with the replacement of classifiers with SA in the last stage	73
------------	--	----

LIST OF FIGURES

Figure 2.1	The standard structure of a CNN	10
Figure 2.2	Sparse Autoencoder structure: The number of units in input layer is equal to number of units in output layer	24
Figure 3.1	General components and a flowchart of our framework for discovering a high-performing CNN architecture	33
Figure 3.2	The top part illustrates the deconvnet layer on the left, attached to the convolutional layer on the right. The bottom part illustrates the pooling and unpooling operations [14]	36
Figure 3.3	Visualization from the last convolutional layer for three different CNN architectures. Grayscale input images are visualized after preprocessing	37
Figure 3.4	Nelder Mead method operations: reflection, expansion, contraction, and shrinkage	42
Figure 3.5	Our proposed system for face recognition	49
Figure 3.6	LBP operator computation for a 3×3 grid [103]	51
Figure 3.7	An architecture of stacked sparse autoencoder (SSA) that consists of two hidden layers	52
Figure 4.1	CIFAR-10 dataset, each rows shows different images of	55

	one class	
Figure 4.2	Objective functions progress during the iterations of NMM. (a) CIFAR-10; (b) CIFAR-100	61
Figure 4.3	The average of the best CNN architectures obtained by both objective functions. (a) The architecture averages for our framework; (b) The architecture averages for the error rate objective function	62
Figure 4.4	The running speed with different values of k .	70
Figure 4.5	Performance of all proposed classifiers in Tables 4.9 and Table 4.10.	74

CHAPTER 1: INTRODUCTION

Deep convolutional neural networks (CNNs) recently have shown remarkable success in a variety of areas such as computer vision [1-3] and natural language processing [4-6]. CNNs are biologically inspired by the structure of mammals' visual cortexes as presented in Hubel and Wiesel's model [7]. In 1998, LeCun et al. followed this idea and adapted it to computer vision. CNNs are typically comprised of different types of layers, including convolutional, pooling, and fully-connected layers. By stacking many of these layers, CNNs can automatically learn feature representation that is highly discriminative without requiring hand-crafted features [8, 9]. In 2012, Krizhevsky et al. [1] proposed AlexNet, a deep CNN architecture consisting of seven hidden layers with millions of parameters, which achieved state-of-the-art performance on the ImageNet dataset [10] with an error test of 15.3%, as compared to 26.2% obtained by second place. AlexNet's impressive result increased the popularity of CNNs within the computer vision community. Other motivators that renewed interest in CNNs include the number of large datasets, fast computation with Graphics Processing Units (GPUs), and powerful regularization techniques such as Dropout [11]. The success of CNNs has motivated many to apply them to solving other problems, such as extreme climate events detection [12] and skin cancer classification [13], etc.

Some works have tried to tune the AlexNet architecture design to achieve better accuracy. For example, in [14], state-of-the-art results are obtained in 2013 by making the filter size and stride in the first convolutional layer smaller. Then, [3] significantly improved accuracy by designing a very deep CNN architecture with 16 layers. The authors pointed out that increasing the depth of the CNN architecture is critical for achieving better accuracy. However, in [15, 16] showed that increasing the depth harmed the performance, as further proven by the experiments in [17]. Additionally, a deeper network makes the network more difficult to optimize and more prone to overfitting [18].

The performance of a deep CNN is critically sensitive to the settings of the architecture design. Determining the proper architecture for a CNN is challenging because the architecture will be different from one dataset to another. Therefore, the architecture design needs to be adjusted for each dataset. Setting the hyperparameters properly for a new dataset and/or application is critical [19]. Hyperparameters that specify a CNN's structure include: the number of layers, the filter sizes, the number of feature maps, stride, pooling regions, pooling sizes, the number of fully-connected layers, and the number of units in each fully-connected layer. The selection process often relies on trial and error, and hyperparameters are tuned empirically. Repeating this process many times is ineffective and can be very time-consuming for large datasets. Recently, researchers have formulated the selection of appropriate hyperparameters as an optimization problem. These automatic methods have produced results exceeding those accomplished by human experts [20, 21]. They utilize prior knowledge to select the next hyperparameter combination to reduce the misclassification rate [22, 23].

In this dissertation, we present an efficient optimization framework that aims to design a high-performing CNN architecture for a given dataset automatically. In this framework, we use deconvolutional networks (deconvnet) to visualize the information learnt by the feature maps. The deconvnet produces a reconstructed image that includes the activated parts of the input image. A good visualization shows that the CNN model has learnt properly, whereas a poor visualization shows ineffective learning. We use a correlation coefficient based on Fast Fourier Transform (FFT) to measure the similarity between the original images and their reconstructions. The quality of the reconstruction, using the correlation coefficient and the error rate, is combined into a new objective function to guide the search into promising CNN architecture designs. We use the Nelder-Mead Method (NMM) to automate the search for a high-performing CNN architecture through a large search space by minimizing the proposed objective function. We exploit web services to run three vertices of NMM simultaneously on distributed computers to accelerate the computation time [23].

1.1 Research Problem and Scope

Constructing a proper CNN architecture for a given problem domain is a challenge as there are numerous design choices that impact the performance [19]. Determining the proper architecture design is a challenge because it differs for each dataset and therefore each one will require adjustments. Many structural hyperparameters are involved in these decisions, such as depth (which includes the number of convolutional and fully-connected layers), the number of filters, stride (step-

size that the filter must be moved), pooling locations and sizes, and the number of units in fully-connected layers. It is difficult to find the appropriate hyperparameter combination for a given dataset because it is not well understood how these hyperparameters interact with each other to influence the accuracy of the resulting model [24]. Moreover, there is no mathematical formulation for calculating the appropriate hyperparameters for a given dataset, so the selection relies on trial and error. Hyperparameters must be tuned manually, which requires expert knowledge [25]; therefore, practitioners and non-expert users often employ a grid or random search to find the best combination of hyperparameters to yield a better design, which is very time-consuming given the numerous CNN design choices.

1.2 Motivation behind the Research

The success of a CNN depends on finding an architecture to fit a given problem. A hand-crafted architecture is a challenging, time-consuming process that requires expert knowledge and effort, due to the large number of architectural design choices. In this dissertation, we propose a framework that finds a good architecture automatically for a given dataset that will maximize the performance. This allows non-expert users and practitioners to find a good architecture for a given dataset in reasonable time without hand-crafting it.

1.3 Contributions

We propose an efficient framework for automatically discovering a high-

performing CNN architecture for a given problem through a very large search space without any human intervention. This framework also allows for an effective parallel and distributed execution.

We introduce a novel objective function that exploits the error rate on the validation set and the quality of the feature visualization via deconvnet. This objective function adjusts the CNN architecture design, which reduces the classification error and enhances the reconstruction via the use of visualization feature maps at the same time. Further, our new objective function results in much faster convergence towards a better architecture.

Instance selection is a subfield in machine learning that aims to reduce the size of the training set. One example of these algorithms is Random Mutation Hill Climbing (RMHC). We propose a new version of RMHC that works quickly and has the same accuracy as the original RMHC.

Some of the best current facial recognition approaches use feature extraction techniques based on Principle Component Analysis (PCA), Local Binary Patterns (LBP), or Autoencoder (non-linear PCA), etc. We employed the power of combining Multiple Classifiers (MC) and deep learning to build a system that uses different feature extraction algorithms PCA, LBP+PCA, LBP+NN. The features from the above three techniques are concatenated to form a joint feature vector. This feature vector is fed into a deep Stacked Sparse Autoencoder (SSA) as a classifier to generate the recognition results.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

2.1 Deep Learning

Deep learning is a subfield of machine learning that has achieved a great performance in a variety of applications in computer vision and natural language processing [3-6, 10]. Deep learning uses multiple hidden layers of non-linear transformations that attempt to learn a hierarchy of features and abstractions, where higher levels of the hierarchy are composed from lower-level features [6]. With enough such transformations, very complex functions can be learned. For object recognition, higher layers of representation amplify aspects of the inputs that are important for discrimination and suppress irrelative variation [26]. The pixels of the image are fed into the first layer, which can learn low-level features such as point, edges, and curves. In subsequent layers, these features are combined into a measure of the likely presence of higher level features; for example, lines are combined into shapes, which are then combined into more complex shapes. Once this is done, the network provides a probability that these high-level features comprise a particular object or scene. Deep learning is motivated by understanding how the human brain processes information. The brain is organized as a deep architecture with many layers that manipulates the information among many levels of non-linear transformation and representation [27].

The main aspect of deep learning is learning discriminative features from the raw data automatically without human-engineered features. The popular models for deep

learning include Deep Belief Network (DBN), Recurrent Neural Network (RNN), Stacked Autoencoder (SA), and Convolutional Neural Networks (CNN) [9, 28].

2.2 Backpropagation and Gradient Descent

The backpropagation algorithm [29] is a popular training method that uses gradient descent to update the parameters of deep learning algorithms to find the parameters (weights w and biases b) that minimize certain loss functions in order to map the arbitrary inputs to the targeted outputs as closely as possible.

During the forward phase, the algorithm forwards through the network layers to compute the outputs. As a result, the error of the loss function is compared to the expected outputs. During the backward phase, the model computes the gradient of the loss function with respect to the current parameters, after which the parameters are updated by taking a step in the direction that minimizes the loss function.

The forward phase starts by feeding the inputs through the first layer, so producing output activations for the successive layer. This procedure is repeated until the loss function at the last layer is computed. During the backward phase, the last layer calculates the derivative with respect to its own learnable parameters as well as its own input, which serves as the upstream derivatives for the previous layer. This procedure is repeated until the input layer is reached [30].

Gradient descent can be categorized into two main methods: Batch Gradient Descent (BGD) and Stochastic Gradient Descent (SGD). The main difference between both approaches is the size of the sample to consider for calculating the gradient. BGD

uses entire the training set to update the gradient at each iteration, while SGD performs the gradient for each training example $(x^{(i)}, y^{(i)})$. Since the gradient of BGD is calculated for the whole training set, it can be very slow and expensive, particularly when the size of the training set is very large. However, the convergence is smoother and the termination is more easily detectable. SGD is less expensive; however, it suffers from noisy steps and its frequent updates can make the loss function fluctuate heavily [31].

SGD with mini-batch takes the best of both BGD and SGD. It updates the gradient by taking the average gradient on a mini-batch of m' examples $\mathbb{Q} = ((x^{(i)}, y^{(i)}), \dots \dots \dots, (x^{(m')}, y^{(m')}))$ [32]. The advantage of mini-batch SGD is that it reduces the variance of the parameter updates, which can lead to more stable convergence. In addition, SGD with mini-batch allows the benefits from parallelism available in GPU, which are frequently used in deep learning frameworks such as Theano and Tensorflow. The size of m' mini-batch is defined by the user and can be up to few hundred examples. The estimate gradient of SGD with mini-batch is formed as:

$$\nabla_w = \frac{1}{m'} \nabla_w \sum_{i=1}^{m'} \ell(x^{(i)}, y^{(i)}, w) \quad (2.1)$$

$$\nabla_b = \frac{1}{m'} \nabla_b \sum_{i=1}^{m'} \ell(x^{(i)}, y^{(i)}, b) \quad (2.2)$$

where $\ell(x^{(i)}, y^{(i)}, w)$ is the loss function over the mini-batch samples \mathbb{Q} selected from the training set. Once the gradients of the loss are computed with backpropagation with respect to the parameters, they are used to perform a gradient descent parameter

update along the downhill direction of the gradient in order to decrease the loss function as follows:

$$w = w - \epsilon \cdot \nabla_w \tag{2.3}$$

$$b = b - \epsilon \cdot \nabla_b \tag{2.4}$$

where ϵ is the learning rate, which is a small positive value between $0 \leq \epsilon \leq 1$ that controls the step size of the update. Algorithm (2.1) highlights the essential steps of SGD with mini-batch in iteration k .

Algorithm 2.1. Stochastic Gradient Descent with mini-batch at iteration k

- 1: **Input:** Learning rate ϵ , initial parameters w, b , mini-batch size (m')
 - 2: **while** stopping criterion not met **do**
 - 3: Pick a random mini-batch with size m' from the training set
($x^{(1)}, \dots, x^{(m')}$) with corresponding outputs $y^{(i)}$
 - 4: Compute gradient for w : $\nabla_w = \frac{1}{m'} \nabla_w \sum_{i=1}^{m'} \ell(x^{(i)}, y^{(i)}, w)$
 - 5: Compute gradient for b : $\nabla_b = \frac{1}{m'} \nabla_b \sum_{i=1}^{m'} \ell(x^{(i)}, y^{(i)}, b)$
 - 6: Apply update for w : $w = w - \epsilon \cdot \nabla_w$
 - 7: Apply update for b : $b = b - \epsilon \cdot \nabla_b$
 - 8: **end while**
-

2.3 Convolutional Neural Networks

CNN is a subclass of neural networks that takes advantage of the spatial structure of the inputs. CNN models have a standard structure consisting of alternating convolutional layers and pooling layers (often each pooling layer is placed after a

convolutional layer). The last layers are a small number of fully-connected layers, and the final layer is a softmax classifier as shown in Figure 2.1.

The critical advantage of CNNs is that it is trained end-to-end from raw pixels to classifier outputs to learn feature representation automatically without depending totally on human-crafted features [10, 11]. Since 2012, many researches have improved the performance of CNNs in different directions, e.g. layer design, activation function, and regularization, or applying CNNs in other areas [12, 13]. CNNs have been implemented using large data sets such as MNIST [33], CIFAR-10/100 [34], and ImageNet [35] for image recognition.

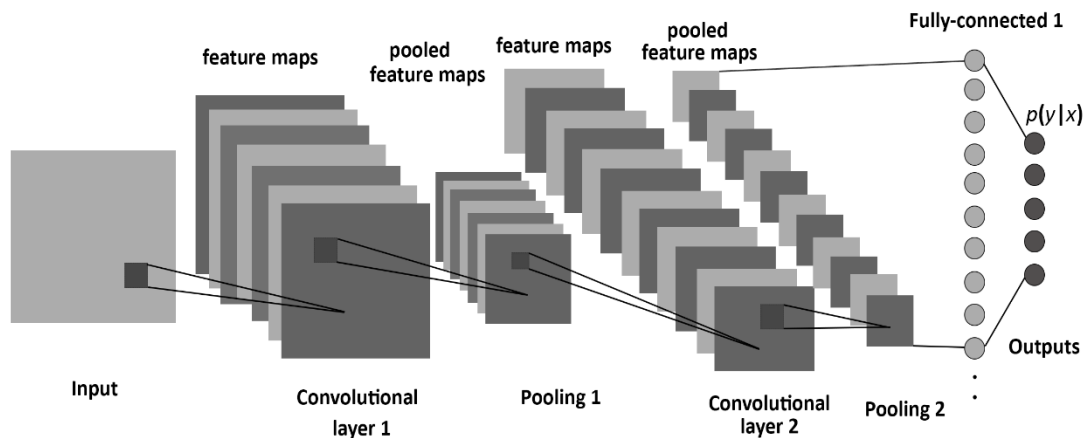


Figure 2.1. The standard structure of a CNN.

2.3.1 Convolutional Layers

The convolutional layer is comprised of a set of learnable kernels or filters which aim to extract local features from the input. Each kernel is used to calculate a feature map. The units of the feature maps can only connect to a small region of the input, called the

receptive field. A new feature map is typically generated by sliding a filter over the input and computing the dot product (which is similar to the convolution operation), followed by a non-linear activation function as shown in Equation 2.5 to introduce non-linearity into the model.

$$x_f^{(l)} = f \left(\sum_{F,F} x^{(l-1)} * w_f^{(l)} + b_f^{(l)} \right) \quad (2.5)$$

where $*$ is convolution operation, $w_f^{(l)}$ is convolution filter with size $F \times F$, $x^{(l-1)}$ is the output of previous layer, $b_f^{(l)}$ is shared bias of the feature map, and f is non-linear activation function.

During the backward phase, we compute the gradient of the loss function with respect to the weights (w) and biases (b) of the respective layer as follows:

$$\nabla_{w_f^{(l)}} \ell = \sum_{F,F} (\nabla_{x_f^{(l+1)}} \ell)_{F,F} (x_{F,F}^{(l)} * w_f^{(l)}) \quad (2.6)$$

$$\nabla_{b_f^{(l)}} \ell = \sum_{F,F} (\nabla_{x_f^{(l+1)}} \ell)_{F,F} (x_{F,F}^{(l)} * b_f^{(l)}) \quad (2.7)$$

All units share the same weights (filters) among each feature map. The advantage of sharing weights is the reduced number of parameters and the ability to detect the same feature, regardless of its location in the inputs [36].

Several nonlinear activation functions are available, such as sigmoid, tanh, and ReLU. However, ReLU [$f(x) = \max(0, x)$] is preferable because it makes training faster relative to the others [1, 37]. The size of the output feature map is based on the filter size

and stride, so when we convolve the input image with a size of $(H \times H)$ over a filter with a size of $(F \times F)$ and a stride of (S) , then the output size of $(W \times W)$ is given by:

$$W = \left\lfloor \frac{H - F}{S} \right\rfloor + 1 \quad (2.8)$$

The hyperparameters of each convolutional layer are filter size, the number of learnable filters, and stride. These hyperparameters must be chosen carefully in order to generate desired outputs.

2.3.2 Pooling Layers

The pooling, or down-sampling layer, reduces the resolution of the previous feature maps. Pooling produces invariance to a small transformation and/or distortion. Pooling splits the inputs into disjoint regions with a size of $(R \times R)$ to produce one output from each region [38]. Pooling can be max or average based. If a given input with a size of $(W \times W)$ is fed to the pooling layer, then the output size will be obtained by:

$$P = \left\lfloor \frac{W}{R} \right\rfloor \quad (2.9)$$

During the forward phase, the maximum value of non-overlapping blocks from the previous feature map $x^{(l-1)}$ is calculated as follows:

$$x^{(l)} = \max_{R,R} (x^{(l-1)})_{R,R} \quad (2.10)$$

Max pooling does not have any learnable parameters. During the backward phase, the gradient from the next layer is passed back only to the neuron that achieved the max

value; all of the other neurons receive zero gradient.

2.3.3 Fully-connected Layers

The top layers of CNNs are one or more fully-connected layers similar to a feed-forward neural network, which aims to extract the global features of the inputs. Units of these layers are connected to all of the hidden units in the preceding layer. The outputs of the fully-connected layer are computed as shown in Equation 2.11:

$$x^{(l)} = f((w^{(l)})^T \cdot x^{(l-1)} + b^{(l)}) \quad (2.11)$$

where \cdot is a dot product, $x^{(l-1)}$ is the output of the previous layer, $x^{(l)}$, $w^{(l)}$, and $b^{(l)}$ denotes the activations, weights, and biases of the current layer (l) respectively, and f is the non-linear activation function.

During the backward phase, the gradient is calculated with respect to the weights and biases as follows:

$$\nabla_{w^{(l)}} \ell = (x^{(l)})^T (\nabla_{x^{(l+1)}} \ell) \quad (2.12)$$

$$\nabla_{b^{(l)}} \ell = (x^{(l)})^T (\nabla_{x^{(l+1)}} \ell) \quad (2.13)$$

where $\nabla_{x^{(l+1)}}$ is the gradient of the next higher layer.

The fully-connected layer has only one hyperparameter, which is the number of neurons (the number of learnable parameters connecting the input to the output).

The last layer is a softmax classifier, which estimates the posterior probability of

each class label over K classes as shown in Equation (2.14) [27].

$$y_i = \frac{\exp(-z_i)}{\sum_{j=1}^K \exp(z_j)} \quad (2.14)$$

2.4 CNN Architecture Design

In this dissertation, our learning algorithm for the CNN (Λ) is specified by a structural hyperparameter λ which encapsulates the design of the CNN architecture as follows:

$$\lambda = \left((\lambda_1^i, \lambda_2^i, \lambda_3^i, \lambda_4^i)_{i=1, M_C}, (\lambda_1^j)_{j=1, N_f} \right) \quad (2.15)$$

where $\lambda \in \Psi$ defines the domain for each hyperparameter, (M_C) is the number of convolutional layers, and (N_f) is the number of fully-connected layers (i.e., the depth = $M_C + N_f$). Constructing any convolutional layer requires four hyperparameters that must be identified. For example, for convolutional layer i : λ_1^i is the number of filters, λ_2^i is the filter size (receptive field size), and λ_3^i defines the pooling locations and sizes. If λ_3^i is equal to one, this means there is no pooling layer placed after convolutional layer i ; otherwise, there is a pooling layer after convolutional layer i and the value λ_3^i defines the pooling region size. λ_4^i is stride step. λ_1^j is the number of units in fully-connected layer j . We also use $\ell(\Lambda, T_{TR}, T_V)$ to refer to the validation loss (e.g., classification error) obtained when we train model Λ with the training set (T_{TR}) and evaluate it on the validation set (T_V). The purpose of our framework is to optimize the combination of structural

hyperparameters λ^* that designs the architecture for a given dataset automatically, resulting in a minimization of the classification error as follows:

$$\lambda^* = \operatorname{argmin} \ell(\Lambda, T_{TR}, T_V) \quad (2.16)$$

We define the most important hyperparameters in designing a CNN architecture below:

Depth: defines the number of convolutional layers (M_C) and the number of fully-connected layers $N(f)$. So the depth= ($M_C + N_f$).

Filter size: The height and width of each filter. Generally, the sizes of the filters are quadratic, i.e. have the same width and height.

Number of filters: defines output volume and controls the number of learnable filters connected to the same region of the input volume. Each filter detects a different feature in the input.

Stride: step-size that the filter must be moved.

Pooling layer location: this defines whether the current convolutional layer is followed by a pooling layer.

Pooling region size: The amount of down-sampling to be performed. In current deep learning frameworks such as Keras and Tensorflow, the hyperparameters of the pooling layer are filter size and stride. In our work, the pooling region size is equivalent to the filter size, and we always assume that the stride is equal to the filter size, which means the pooling is always non-overlapped.

2.4.1 Literature Review on CNN Architecture Design

A simple technique for selecting a CNN architecture is cross-validation [39], which runs multiple architectures and selects the best one based on its performance on the validation set. However, cross-validation can only guarantee the selection of the best architecture amongst architectures that are composed manually through a large number of choices. The most popular strategy for hyperparameter optimization is an exhaustive grid search, which tries all possible combinations through a manually-defined range for each hyperparameter. The drawback of a grid search is its expensive computation, which increases exponentially with the number of hyperparameters and the depth of exploration desired [40]. Recently, random search [41], which selects hyperparameters randomly in a defined search space, has reported better results than grid search and requires less computation time. However, neither random nor grid search use previous evaluations to select the next set of hyperparameters for testing to improve upon the desired architecture.

Recently, Bayesian Optimization (BO) methods have been used for hyperparameter optimization [21, 42, 43]. BO constructs a probabilistic model \mathcal{M} based on the previous evolutions of the objective function f . Popular techniques that implement BO are Spearmint [21], which uses a Gaussian process model for \mathcal{M} , and Sequential Model-based Algorithm Configuration (SMAC) [42], based on a random forest of the Gaussian process. According to [44], BO methods are limited because they work poorly when high-dimensional hyperparameters are involved and are very computationally expensive. The work in [21] used BO with a Gaussian process to optimize nine hyperparameters of a CNN, including the learning rate, epoch, initial weights of the

convolutional and full-connected layers, and the response contrast normalization parameters. Many of these hyperparameters are continuous and related to regularization, but not to the CNN architecture. Similarly, Ref. [24, 45, 46] optimized continuous hyperparameters of deep neural networks. However, Ref. [46-51] proposed many adaptive techniques for automatically updating continuous hyperparameters, such as the learning rate momentum and weight decay for each iteration to improve the coverage speed of backpropagation. In addition, early stopping [52, 53] can be used when the error rate on a validation set or training set has not improved, or when the error rate increases for a number of epochs. In [54], an effective technique is proposed to initialize the weights of convolutional and fully-connected layers.

Evolutionary algorithms are widely used to automate the architecture design of learning algorithms. In [25], a genetic algorithm is used to optimize the filter sizes and the number of filters in the convolutional layers. Their architectures consisted of three convolutional layers and one fully-connected layer. Since several hyperparameters were not optimized, such as depth, pooling regions and sizes, the error rate was high, around 25%. Particle Swarm Optimization (PSO) is used to optimize the feed-forward neural network's architecture design [55]. Soft computing techniques are used to solve different real applications, such as rainfall and forecasting prediction [56, 57]. PSO is widely used for optimizing rainfall-runoff modeling. For example, Ref. [58] utilized PSO as well as extreme learning machines in the selection of data-driven input variables. Similarly, [59] used PSO for multiple ensemble pruning. However, the drawback of evolutionary algorithms is that the computation cost is very high, since each population member or

particle is an instance of a CNN architecture, and each one must be trained, adjusted and evaluated in each iteration. In [60], ℓ_1 Regularization is used to automate the selection of the number of units only for fully-connected layers for artificial neural networks.

Recently, interest in architecture design for deep learning has increased. The proposed work in [61] applied reinforcement learning and recurrent neural networks to explore architectures, which have shown impressive results. Ref. [62] proposed a CoDeepNEAT-based Neuron Evolution of Augmenting Topologies (NEAT) to determine the type of each layer and its hyperparameters. Ref. [63] used a genetic algorithm to design a complex CNN architecture through mutation operations and managing problems in filter sizes through zeroth order interpolation. Each experiment was distributed to over 250 parallel workers to find the best architecture. Reinforcement learning, based on Q -learning [64], was used to search the architecture design by discovering one layer at a time, where the depth is decided by the user. However, these promising results were achieved only with significant computational resources and a long execution time.

Visualization approach in [14, 65] is another technique used to visualize feature maps to monitor the evolution of features during training and thus discover problems in a trained CNN. As a result, the work presented in [14] visualized the second layer of the AlexNet model, which showed aliasing artifacts. They improved its performance by reducing the stride and kernel size in the first layer. However, potential problems in the CNN architecture are diagnosed manually, which requires expert knowledge. The selection of a new CNN architecture is then done manually as well.

2.5 Regularization

In CNNs, overfitting is a major problem, which regularization can effectively reduce. There are several techniques to combat this problem, including L1, L2 weight decay, KL-sparsity, early stopping, data augmentation, and dropout. Dropout has proven itself an effective method to reduce overfitting due to its ability to provide a better generalization on the testing set. Because dropout is such a powerful technique, it has encouraged recent success in CNNs

Dropout [11] is a powerful technique for regularizing full connected layers within neural networks or CNN. The idea of dropout is each neuron is selected randomly with probability p to be dropped (setting the activation to zero) for each training case. This helps to prevent hidden neurons from co-adapting with each other too much; forcing the model based on a subset of hidden neurons. The error back-propagated through only remaining neurons that are not dropped. On the other hand, we can look to the dropout as model averaging of large number of neural network models.

Early stopping [52, 53] is a kind of regularization that helps to avoid overfitting by monitoring the performance of the model on the validation set. Once the performance on the validation dataset decreases or saturates for a number of iterations, the model stops the training procedure.

2.6 Weight initialization

Weight initialization [18] is a critical step in CNNs that influences the training process. In order to initialize the model's parameters properly, the weights must be within

a reasonable range before the training process begins. As a result, this will make the convergence faster. Several weight initialization methods have been proposed, including random initialization, naive initialization, and Xavier initialization. The two most widely used are naive initialization and Xavier initialization.

- Naive initialization, the weights are initialized from a Gaussian distribution with a mean of zero and a small value of standard deviation.
- Xavier initialization [54] has become the default technique for weight initialization in CNNs. It tries to keep the variance between the layers approximately the same. The advantage of Xavier initialization is that it makes the network converge much faster than other approaches. The weight sets it produces are also more consistent than those produced by other techniques.

2.7 Visualizing and Understanding CNNs

There are three main methods for understanding and visualizing CNNs as follows:

- **Layer activations:** this simple technique shows the activations of a network during the forward pass. The drawback, however, is that some activation maps' outputs are zero for the input images, which indicates dead filters. Additionally, the size of the activation maps is not equal to the input image, especially in higher layers.
- **Retrieving images that maximally activate a neuron:** this strategy feeds a large set of images through the network and then keeps track of which images maximize the activations of the neurons. However, the limitation of this technique is that the

ReLU activation function does not always have semantic meaning by itself. This method can involve a high computational cost to find the images that maximize the neurons' values [66].

- **Deconvolutional networks:** aims to project the learned features in higher layers down to the input pixel space for a trained CNN. This results in a reconstructed image the same size as the input image. It contains the regions of the input image that were learned by a given feature map. A visualization similar to the input image indicates that the CNN architecture learned properly. Since the reconstructed image is the same size as the input, this allows us to measure the similarity between the inputs and their reconstruction effectively [67] (Details in 5.3)

2.8 Similarity Measurements between Images

Several methods are used to compare the similarity between two images or vectors. The most widely used are Euclidean distance, mutual information, and a correlation coefficient. Each one of these methods has advantages and disadvantages.

- **The Euclidean distance** between two images is the sum of the squared intensity differences of corresponding pixels in sequences as shown in Equation 2.17. The drawback of Euclidean distance is that it is very sensitive to normalization of the data, so any tiny errors will produce inaccurate results. Euclidean distance between two vectors q and p is computed by:

$$d(p, q) = \sqrt{\sum_{i=1}^y (p_i - q_i)^2} \quad (2.17)$$

- **Mutual information** is a concept derived from information theory. The mutual information measures the dependencies between two images, or the amount of information that one image contains about the other. The value of mutual information will be large when the similarity between a pair of images is high. Mutual information denoted as $MI(X, Y)$ between two grayscale images X and Y is given by:

$$MI(X, Y) = \sum_{x=0}^{255} \sum_{y=0}^{255} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \quad (2.18)$$

where $p(x)$ and $p(y)$ are marginal distributions of X, Y respectively, and $p(x, y)$ is a joint probability distribution. The key advantage of mutual information is that it detects the non-linear dependence between two images. The drawback of mutual information is that it is computationally very expensive [68].

- **Correlation** distance is the linear dependence between two images. Fast Fourier Transform (FFT) provides another approach to calculate the correlation coefficient with a high computational speed as compared to the original correlation coefficient formula and mutual information (further detail is provided in Chapter 3.1.3).

2.9 Sparse Autoencoder

Sparse autoencoder (SA) is a kind of unsupervised NN aiming to approximate an

identical function $f(x) = x$ by making the target outputs equal to the inputs during the training phase. SA applies backpropagation to learn meaningful features from unlabeled data. SA includes encoder and decoder steps. The encoder takes the input vector x to the hidden layer representation y with a non-linear activation function, such as sigmoid f , as shown in the following equation:

$$y = f(Wx + b) \quad (2.19)$$

The decoder maps the hidden representation y back into reconstruction z of the same input vector x as shown in Figure 2.2.

$$z = f(W'y + b) \simeq x \quad (2.20)$$

SA is trained by back-propagation usually via gradient descent to reduce the average reconstruction error $L(z, x) = ||z - x||^2$. SA imposes sparsity on many hidden neurons' outputs to make them zero or close to zero in order to discover interesting and feature representation and removing redundant and noisy information from the inputs [69].

Therefore, the cost function of sparse autoencoder is obtained by:

$$(W, b) = \frac{1}{2m} \sum_{i=1}^n \|x^{(i)} - z^{(i)}\|^2 + \frac{\lambda}{2} \|W\|^2 + \beta \sum_{j=1}^s KL(\rho \| \hat{\rho}) \quad (2.21)$$

The first term of cost function in Equation 2.11 is an average sum of squares error which describes the discrepancy error between the inputs $x^{(i)}$ and its reconstruction $z^{(i)}$ over the entire training samples. The second term is weight decay, which is a regularization technique for preventing overfitting. The last term is an extra penalty term

to provide sparsity constraint, where ρ is the sparsity parameter and typically takes a small value, n is number of neurons in the hidden layer, and the index j sums over the hidden neurons in our network. A $KL(\rho||\hat{\rho})$ is a Kullback-Leibler (KL) divergence between $\hat{\rho}$ which is an average activation (averaged over the training data) of hidden neuron j , and the target activations ρ_i which can be defined as:

$$KL(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j} \quad (2.22)$$

where ρ is a sparsity parameter, typically a small value close to zero (say $\rho = 0.05$). Therefore, we would like the average activation of each hidden neuron j to be close to 0.05. To satisfy this constraint, the hidden unit's activations must mostly be near zero.

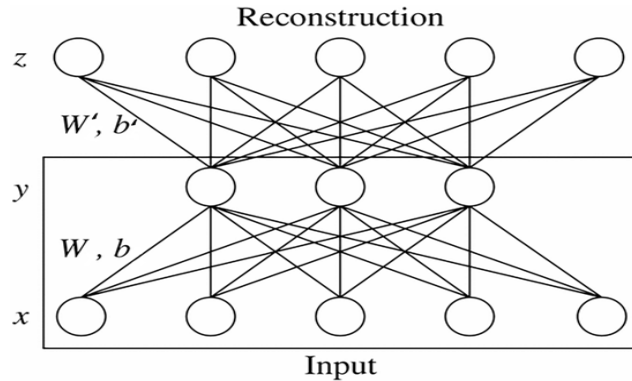


Figure 2.2. Sparse Autoencoder structure: The number of units in input layer is equal to number of units in output layer.

2.10 Analysis of optimized instance selection algorithms on large datasets with CNNs

It is common that training set consists of instances that are useless. Therefore, it is probable to get acceptable performance and enhance the training execution time

without non-useful instances; this process is called instance selection [70, 71].

Instance selection aims to choose subset (T_S) from training set (T_{TR}) where ($T_S \subset T_{TR}$) to accomplish the original task of classification application with little or no performance degradation as if the entire training set (T_{TR}) is used. This training set might contain superfluous instances which can be redundant or noisy. Removing these instances is required because they may cause performance deterioration [72]. Furthermore, reducing the training set will shrink the amount of computation and memory storage, especially if the amount of training set is large with high dimensionality. Each instance in a training set can be either border instance or interior instance. Border instance is its k nearest neighbors (k -NN) belonging to other classes that usually are closer to the decision boundary. Interior instance is its k nearest neighbors belonging to the same class [73].

Instance selection can be divided into three types of algorithms: condensation, edition, and hybrid [74]. Condensation techniques aim to retain border instances. This leads to make the accuracy over training set high but it might reduce the generalization accuracy over the testing set. The reduction rate is high in condensation methods [75]. Edition methods aim to discard the border instances and retain interior instances. Consequently, this leads to a smoother decision boundary between classes, which can improve the classifier accuracy over the testing set. Finally, hybrid methods seek to choose a subset of the training set containing border and interior instances to maintain or enhance the generalization accuracy.

An instance selection search can be incremental, decremental or mixed. Incremental methods start with empty subset $T_S = \emptyset$, and add each instance to T_S from T_{TR}

if it qualifies for some criteria. Decremental search starts with $T_S=T_{TR}$ and removes any instance I_{img} from T_S if it does not fulfill specific criteria. Mixed search starts with pre-selected subset T_S and iteratively can add or remove any instance meet the specific criteria [71, 74].

All instance selection methods work under the following assumption: T_{TR} is the training set; I_{img_i} is i -th instance in T_{TR} . Subset T_S is selected from T_{TR} ; I_{img_j} is j -th instance in T_S . T_V is the validation set. T_{TS} is the testing set. Typically, the accuracy of instance selection methods is determined by k Nearest Neighbors (k -NN) [76]. The Euclidean distance function is used to calculate the similarity between two instances q and p as shown in Equation 2.17.

2.10.1 Literature Review for Instance Selection

Condensed Nearest Neighbor (ConNN) [77] was first algorithm of instance selection. The algorithm is an incremental method that starts with adding one instance of each class to the subset T_S randomly from training set T_{TR} . Then, for each instance I_{img} in T_{TR} is classified using the instances in T_S . If the instance I_{img} is incorrectly classified, it will be added to T_S . This guarantees all instances in T_{TR} are classified correctly. Based on this criterion, noisy instance will be retained because they are commonly classified wrongly by their k -NN.

The Edited Nearest Neighbor (ENN) [78] algorithm starts with $T_S = T_{TR}$, and then each instance I_{img} in T_S is removed from T_S if it does not agree with the majority of k -NN

(e.g. $k=3$). The ENN discards noisy instances as well as border instances to yield smooth boundaries between classes by saving interior instances.

All k -NN [79] belongs to the family of ENN. The algorithm works as follows: for $i=1$ to k , flags as bad for each instance misclassified by its k -NN. Once the loop is ended, it discards any instance from T_S if it is flagged as bad.

Skalak [80] exploited Random Mutation Hill Climbing (RMHC) method [81] to select the subset T_S from T_{TR} . This algorithm has two parameters should be defined by the user early: (1) N_S is the size of the subset or the training sample T_S . (2) N_{iter} is number of iterations. The algorithm is based on coding instances in T_{TR} into binary string format. Each bit represents one instance, where N_S bits are equal to one randomly (they represent T_S). For N_{iter} iterations, the algorithm randomly mutates a single bit of zero to one. The accuracy will be computed, if the change increases the accuracy, the change will be kept, otherwise, roll-backed. This algorithm gives high chance to increase the size of T_S with increasing iteration

In [74] explained RMHC in a different way as follows: the algorithm begins to select N_S instances randomly from T_{TR} to represent T_S . For N_{iter} iterations: the algorithm replaces one instance selected randomly from T_S with instance selected randomly from $(T_{TR}-T_S)$. If the change improves the accuracy on the testing data using 1-NN, the change will be maintained, otherwise, the change will be roll-backed. The size of T_S in this way is fixed with length N_S .

2.11 A Deep Architecture for Face Recognition Based on Multiple Feature Extractors

In recent decades, face recognition has been widely explored in the areas of computer vision and image analysis due to its numerous application domains such as surveillance, smart cards, law enforcement, access control, and information security. With the number of face recognition algorithms that have been developed, face recognition is still a very challenging task with respect to the changes in facial expression, illumination, background and pose [82].

The performance of each individual classifier has shown sensitivity to some changes in facial appearance. Combining Multiple Classifiers (MC) in one system has become a new direction which integrates many information resources and is likely enhance the performance. The combination of MC can be applied at two levels: the decision level and the feature level. The decision level addresses how to combine the outputs of MC. In the feature level, each classifier produces a new representation that will be concatenated in a feature vector to be fed into the new classifier [83]. This is the approach we follow in our work. MC is only useful if combined classifiers are mutually complementary, and they do not make coincident errors [84]. MC is a very effective solution for classification problems involving a lot of classes and noisy input data [85].

In general, an MC system has three main topologies: parallel, serial and hybrid [86]. The design of an MC system consists of two main steps: the classifier ensemble and fuser. The classifier ensemble defines the selection of combined classifiers to be most effective. The fuser step combines the results that are obtained by each individual

classifier [87]. The final classification can be greatly improved using deep learning.

In this dissertation, we also employ the power of combining MC and deep learning to build a system that uses different feature extraction algorithms, namely PCA, LBP+PCA, and LBP+NN, to ensure that each classifier produces its own basis representation in a meaningful way. We then form a joint feature vector by concatenating the outputs of the above three MCs. This joint feature vector is then fed into a deep SA with two hidden layers to generate the classification results with probabilistic distribution to approximate the probability of each class label. We present the results of a series of experiments for different existing MC systems, replacing different classifiers with SSA, to exhibit the efficiency of deep SSA classification as compared to other classifiers implementation.

2.11.1 Literature Review for Multiple Classifiers

Lu et al [88] combined the basis vectors of PCA, ICA and LDA. The authors used sum rule as well as RBF network strategies to integrate the outputs of three classifiers, using matching scores. The outputs of the classifiers are concatenated into one vector to be used as input to the RBF network to get the final decision. All feature extractors used in this system are holistic techniques which utilize the information of the entire face to be projected into subspace. The drawback of this approach is that the classification results (not features) are combined from each classifier, whereas in this work, we combine the individual features from different classifiers in a balanced way into a deep learning based final classifier. Thus, preserving the entire feature information from an individual

classifier until the final result is produced.

Lawrence et al [89] proposed a hybrid neural network comprising of local image sampling, a self-organizing map (SOM) neural network and convolution neural networks (CNN). SOM is used for reducing dimensionality and invariance to small changes in the images. CNN provides partial distortion invariance. Replacing SOM by the Karhunen-Loeve transform produced a slightly worse result. While this approach produces good results and to some extent is invariant to small changes in the input image, the classification depends only on features obtained from dimensionality reduction. The best reported result on the ORL dataset was 96.2%, whereas the approach in this work which relies on diverse features from different techniques (including dimensionality reduction) achieves an accuracy of 98% on the ORL database.

Eleyan and Demirel [90] proposed two systems for face recognition; PCA followed by neural network (NN) and LDA followed by (NN). PCA and LDA are used for dimensionality reduction to be fed into NN for classification. LDA+ NN outperforms PCA+NN.

Lone et al [91] developed a single system for face recognition that combines four individual algorithms namely PCA, Discrete Cosine Transform (DCT), Template Matching using correlation (Corr) and Partitioned Iterative Function System (PIFS). In addition, they compared the results by combining two techniques of PCA-DCT, and three techniques based on PCA-DCT-Corr. The results show that combining four Algorithms outperforms combination of two as well as three Algorithms. They obtained 86.8% accuracy rate on ORL database.

Liong et al [92] proposed a new technique, called deep PCA, to obtain a deep representation of data, which will be discriminant and better for recognition. The approach comprises of two layers which are whitening and PCA. The outputs of first layer will be inserted into the second layer to perform whitening and PCA again.

CHAPTER 3: RESEARCH PLAN

3.1 A Framework for Designing the Architectures of Deep CNNs

In existing approaches to hyperparameter optimization and model selection, the dominant approach to evaluating multiple models is the minimization of the error rate on the validation set. In this work, we describe a better approach based on introducing a new objective function that exploits the error rate as well as the visualization results from feature activations via deconvnet. Another advantage of our objective function is that it does not get stuck easily in local minima during optimization using NMM. Our approach obtains a final architecture that outperforms others that use the error rate objective function alone.

In this section, we present information on the framework model consisting of deconvolutional networks, the correlation coefficient, and the objective function. NMM guides the CNN architecture by minimizing the proposed objective function, and web services help to obtain a high-performing CNN architecture for a given dataset. For large datasets, we use instance selection and statistics to determine the optimal, reduced training dataset as a preprocessing step. A general framework flowchart and components are shown in Figure 3.1. In order to accelerate the optimization process, we employ multiple techniques, including training on an optimized dataset, parallel and distributed execution, and correlation coefficient computation via FFT.

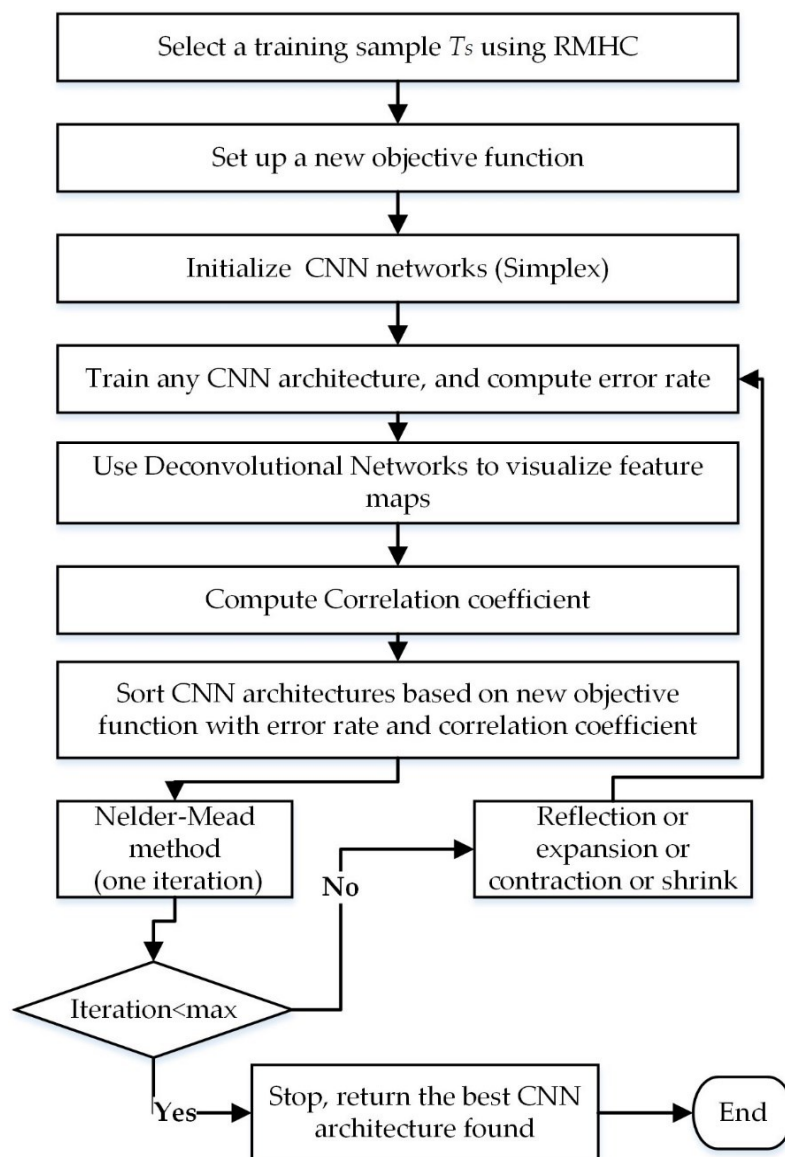


Figure 3.1. General components and a flowchart of our framework for discovering a high-performing CNN architecture

3.1.1 Reducing the Training Set

Training deep CNN architectures with a large training set involves a high computational time. The large dataset may contain redundant or useless images. In

machine learning, a common approach of dealing with a large dataset is instance selection, which aims to choose a subset or sample (T_S) of the training set (T_{TR}) to achieve acceptable accuracy as if the whole training set was being used. Many instance selection algorithms have been proposed and reviewed in [71]. Albelwi and Mahmood [76] evaluated and analyzed the performance of different instance selection algorithms on CNNs. In this framework, for very large datasets, we employ instance selection based on Random Mutual Hill Climbing (RMHC) [80] as a preprocessing step to select the training sample (T_S) which will be used during the exploration phase to find a high-performing architecture. The reason for selecting RMHC is that the user can predefine the size of the training sample, which is not possible with other algorithms. We employ statistics to determine the most representative sample size, which is critical to obtaining accurate results.

In statistics, calculating the optimal size of a sample depends on two main factors: the margin of error and confidence level. The margin of error defines the maximum range of error between the results of the whole population (training set) and the result of a sample (training sample). The confidence level measures the reliability of the results of the training sample, which reflects the training set. Typical confidence level values are 90%, 95%, or 99%. We use a 95% confidence interval in determining the optimal size of a training sample (based on RMHC) to represent the whole training set.

3.1.2 CNN Feature Visualization Methods

Recently, there has been a dramatic interest in the use of visualization methods to

explore the inner operations of a CNN, which enables us to understand what the neurons have learned. There are several visualization approaches. A simple technique called layer activation shows the activations of the feature maps [93] as a bitmap. However, to trace what has been detected in a CNN is very difficult. Another technique is activation maximization [94], which retrieves the images that maximally activate the neuron. The limitation of this method is that the ReLU activation function does not always have a semantic meaning by itself. Another technique is deconvolutional network [14], which shows the parts of the input image that are learned by a given feature map. The deconvolutional approach is selected in our work because it results in a more meaningful visualization and also allows us to diagnose potential problems with the architecture design.

3.1.2.1 Deconvolutional Networks

Deconvolutional networks (deconvnet) [14] are designed to map the activities of a given feature in higher layers to go back into the input space of a trained CNN. The output of deconvnet is a reconstructed image that displays the activated parts of the input image learned by a given feature map. Visualization is useful for evaluating the behavior of a trained architecture because a good visualization indicates that a CNN is learning properly, whereas a poor visualization shows ineffective learning. Thus, it can help tune the CNN architecture design accordingly in order to enhance its performance. We attach a deconvnet layer with each convolutional layer similar to [14], as illustrated at the top of Figure 3.2. Deconvnet applies the same operations of a CNN but in reverse, including unpooling, a non-linear activation function (in our framework, ReLU), and filtering.

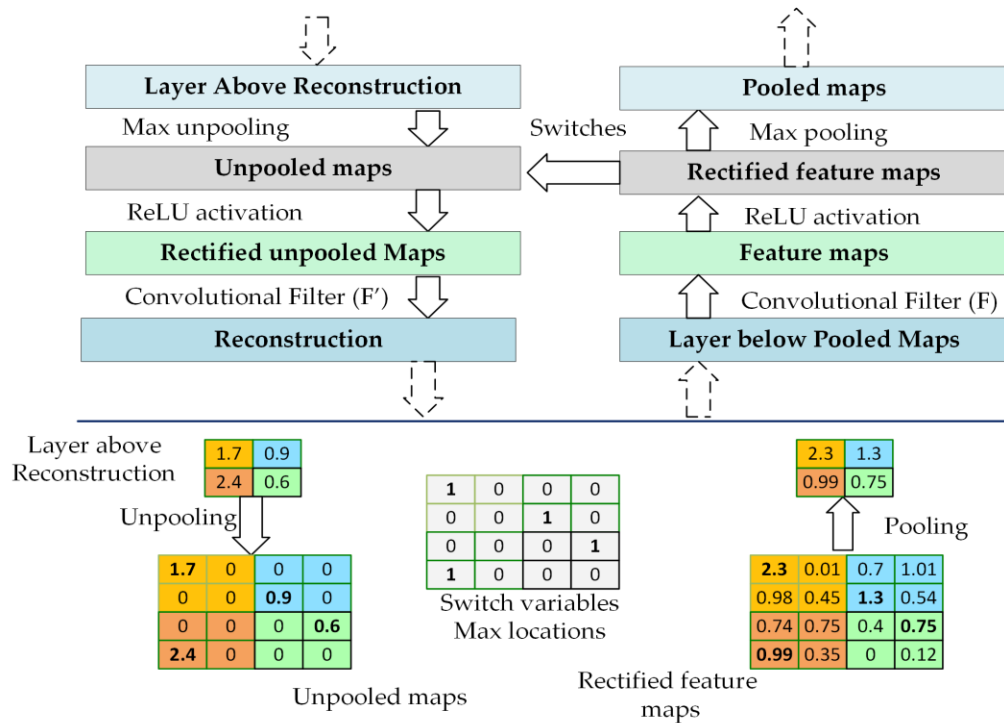


Figure 3.2. The top part illustrates the deconvnet layer on the left, attached to the convolutional layer on the right. The bottom part illustrates the pooling and unpooling operations [14].

The deconvnet process involves a standard forward pass through the CNN layers until it reaches the desired layer that contains the selected feature map to be visualized. In a max pooling operation, it is important to record the locations of the maxima of each pooling region in switch variables because max pooling is non-invertible. All feature maps in a desired layer will be set to zero except the one that is to be visualized. Now we can use deconvnet operations to go back to the input space for performing reconstruction. Unpooling aims to reconstruct the original size of the activations by using switch variables to return the activation from the layer above to its original position in the pooling layer, as shown at the bottom of Figure 3.2, thereby preserving the structure of the stimulus. Then, the output of the unpooling passes through the ReLU function. Finally, deconvnet applies a convolution operation on the rectified, unpooled maps with

transposed filters in the corresponding convolutional layer. Consequently, the result of deconvnet is a reconstructed image that contains the activated pieces of the input that were learnt. Figure 3.3 displays the visualization of different CNN architectures. As shown, the quality of the visualization varies from one architecture to another compared to the original images in grayscale. For example, CNN architecture 1 shows very good visualization; this gives a positive indication about the architecture design. On the other hand, CNN architecture 3 shows poor visualization, indicating this architecture has potential problems and did not learn properly.

The visualization of feature maps is thus useful in diagnosing potential problems in CNN architectures. This helps in modifying an architecture to enhance its performance, and also evaluating different architectures with criteria besides the classification error on the validation set. Once the reconstructed image is obtained, we use the correlation coefficient to measure the similarity between the input image and its reconstructed image in order to evaluate the reconstruction's representation quality.

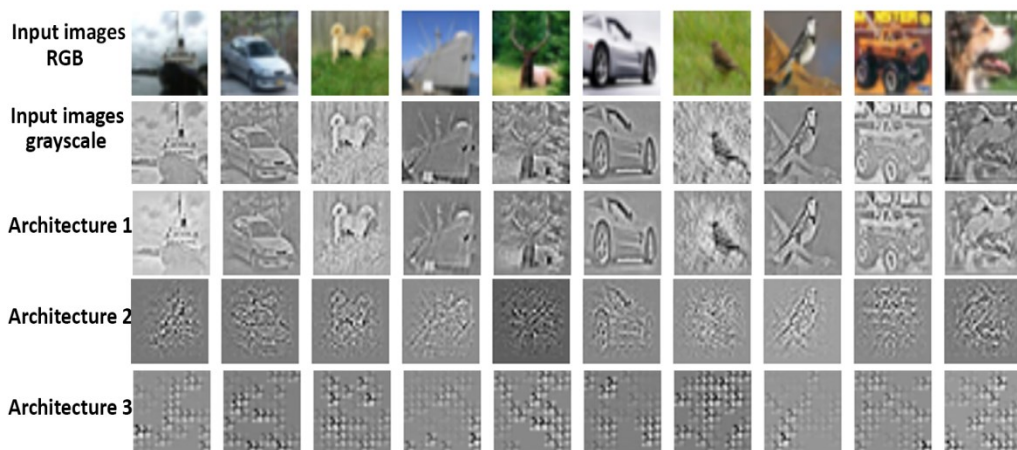


Figure 3.3. Visualization from the last convolutional layer for three different CNN architectures. Grayscale input images are visualized after preprocessing.

3.1.3 Correlation Coefficient

The correlation coefficient (*Corr*) [95] measures the level of similarity between two images or independent variables. The correlation coefficient is maximal when two images are highly similar. The correlation coefficient between two images A and B is given by:

$$Corr(A, B) = \frac{1}{n} \sum_{i=1}^n \left(\frac{a_i - \bar{a}}{\sigma_a} \right) \left(\frac{b_i - \bar{b}}{\sigma_b} \right) \quad (3.1)$$

where \bar{a} and \bar{b} are the averages of A and B respectively, σ_a denotes the standard deviation of A , and σ_b denotes the standard deviation of B . Fast Fourier Transform (FFT) provides an alternative approach to calculate the correlation coefficient with a high computational speed as compared to Equation (3.1) [96, 97]. The correlation coefficient between A and B is computed by locating the maximum value of the following equation:

$$Corr(A, B) = \mathcal{F}^{-1}[\mathcal{F}(A) \circ \mathcal{F}^*(B)] \quad (3.2)$$

where \mathcal{F} is an FFT for a two-dimensional image, \mathcal{F}^{-1} indicates inverse FFT, $*$ is the complex conjugate, and \circ implies element by element multiplication. This approach reduces the time complexity of the computing correlation from $O(N^2)$ to $O(N \log N)$. Once the training of the CNN is complete, we compute the error rate (*Err*) on the validation set, and choose N_{fm} feature maps at random from the last layer to visualize their learned parts using deconvnet. The motivation behind selecting the last convolutional layer is that it should show the highest level of visualization as compared to preceding layers. We choose N_{img} images from the training sample at random to test the deconvnet.

The correlation coefficient is used to calculate the similarity between the input images N_{img} and their reconstructions. Since each image of N_{img} has a correlation coefficient ($Corr$) value, the results of all $Corr$ values are accumulated in a scalar value called ($Corr_{Res}$). Algorithm 3.1 summarizes the processing procedure for training a CNN architecture:

Algorithm 3.1. Processing Steps for Training a Single CNN Architecture.

- 1: **Input:** training sample T_S , validation set T_V , N_{fm} feature maps, and N_{img} images
 - 2: **Output:** Err and $Corr_{Res}$
 - 3: Train CNN architecture design using SGD
 - 4: Compute error rate (Err) on validation set T_V
 - 5: $Corr_{Res} = 0$
 - 6: For $i = 1$ to N_{fm}
 - 7: Pick a feature map fm at random from the last convolutional layer
 - 8: For $j = 1$ to N_{img}
 - 9: Use deconvnet to visualize a selected feature map fm on image $N_{img}[j]$
 - 10: $Corr_{Res} = Corr_{Res} +$ correlation coefficient ($N_{img}[j]$, reconstructed image)
 - 11: Return Err and $Corr_{Res}$
-

3.1.4 Objective Function

Existing works on hyperparameter optimization for deep CNNs generally use the error rate on the validation set to decide whether one architecture design is better than another during the exploration phase. Since there is a variation in performance on the same architecture from one validation set to another, the model design cannot always be generalized. Therefore, we present a new objective function that exploits information from the error rate (Err) on the validation set as well as the correlation results ($Corr_{Res}$) obtained from deconvnet. The new objective function can be written as:

$$f(\lambda) = \eta(1 - Corr_{Res}) + (1 - \eta) Err \quad (3.3)$$

where η is a correlation coefficient parameter measuring the importance of Err and $Corr_{Res}$. The key reason to subtract $Corr_{Res}$ from one is to minimize both terms of the objective function. We can set up the objective function in Equation (3.3) as an optimization problem that needs to be minimized. Therefore, the objective function aims to find a CNN architecture that minimizes the classification error and provides a high level of visualization. We use the NMM to guide our search into a promising direction for discovering iteratively better-performing CNN architecture designs by minimizing the proposed objective function.

3.1.5 Nelder Mead Method

The Nelder-Mead algorithm (NMM), or simplex method [98], is a direct search technique widely used for solving optimization problems based on the values of the objective function when the derivative information is unknown. NMM uses a concept called a simplex, which is a geometric shape consisting of $n + 1$ vertices for optimizing n hyperparameters. First, NMM creates an initial simplex that is generated randomly. In this framework, let $[Z_1, Z_2, \dots, Z_{n+1}]$ refer to simplex vertices, where each vertex presents a CNN architecture. The vertices are sorted in ascending order based on the value of objective functions $f(Z_1) \leq f(Z_2) \leq \dots \leq f(Z_{n+1})$ so that Z_1 is the best vertex, which provides the best CNN architecture, and Z_{n+1} is the worst vertex. NMM seeks to find the best hyperparameters λ^* that designs a CNN architecture that minimizes the objective function in Equation 3.3 as follows:

$$\lambda^* = \arg \min_{\lambda \in \Psi} f(\lambda) \quad (3.4)$$

The search is performed based on four basic operations: reflection, expansion, contraction, and shrinkage, as shown in Figure 3.4. Each is associated with a scalar coefficient of α (reflection), β (expansion), γ (contraction), and δ (shrinkage). In each iteration, NMM tries to update a current simplex to generate a new simplex which decreases the value of the objective function. NMM replaces the worst vertex with the best that has been found from reflected, expanded or contracted vertices. Otherwise, all vertices of the simplex, except the best, will shrink around the best vertex. These processes are repeated until the stop criterion is accomplished. The vertex producing the lowest objective function value is the best solution that is returned. The main challenge in finding a high-performing CNN architecture is the execution time and, correspondingly, the number of computing resources required. We can apply our optimization objective function with any derivative-free algorithm such as genetic algorithms, particle swarm optimization, Bayesian optimization, and the Nelder-Mead method, etc. The reason for selecting NMM is that it is faster than other derivative-free optimization algorithms, because in each iteration, only a few vertices are evaluated. Further, NMM is easy to parallelize with a small number of workers to accelerate the execution time.

During the calculation of any vertex of NMM, we added some constraints to make the output values positive integers. The value of $Corr_{RES}$ is normalized between the minimum and maximum value of the error rate in each iteration of NMM. This is critical because it affects the value of η in Equation (3.3).

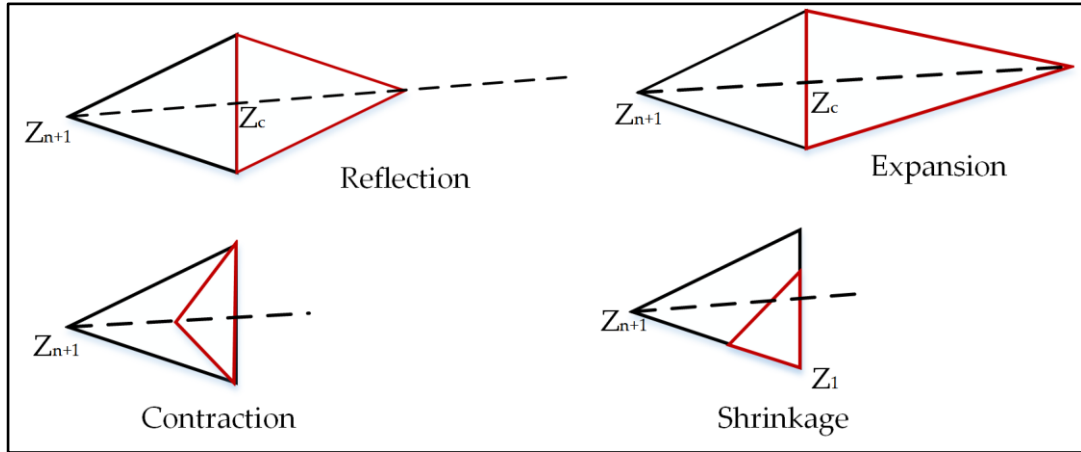


Figure 3.4. Nelder-Mead method operations: reflection, expansion, contraction, and shrinkage.

Below, we provide details of our proposed framework based on the serial NMM and the new objective function in Alg. 3.2 to obtain a good CNN architecture.

Algorithm 3.2. The Proposed Framework Pseudocode with Serial NMM

- 1: **Input:** n : Number of hyperparameters
- 2: **Input:** $\alpha, \rho, \gamma, \sigma$: reflection, expansion, contraction and shrink coefficients
- 3: **Output:** Best vertex ($Z [I]$) found
- 4: Create initial Simplex (Z) with $n+1$ vertices: $Z_{1:n+1}$
- 5: Determine training sample T_S using RMHC
- 6: Initialize the Simplex vertices ($Z_{1:n+1}$) randomly from Table 4.1
- 7: Train and evaluate each vertex of Z according **Alg. 3.1**
- 8: **While** stop criterion not met
- 9: Scale values of $Corr_{Res}$ between the *max* and *min* of Err of all vertices of (Z). Compute $f(Z_i)$ based on Equation. 33. for all vertices
- 10: $Z =$ sort vertices of current vertex so that $f(Z_1), f(Z_2), \dots, f(Z_{n+1})$ in the descending order.
- 11: Set $B = Z_1, A = Z_n, W = Z_{n+1}$
- 12: Compute centroid vertex without worst vertex $C = \sum_{i=1, n} (Z_c/n)$
- 13: Compute reflection: $R = C + \alpha(C - W)$. Train $f(R)$ according to **Alg. 3.1**

```

14:   Scale  $Corr_{Res}$  of  $f(R)$ 
15:   Compute new accuracy based on Equation. 3.3.
16:   Scale values of  $Corr_{Res}$  between the max and min of Err of all vertices of
      (Z.Compute  $f(Z_i)$  based on Equation. 33. for all vertices
17:   If  $f(B) < f(R) < f(Z)$ 
18:        $Z_{n+1} = R$ 
19:   Else
20       If  $f(R) \leq f(B)$ 
21           Expansion  $E = R + \gamma(R - C)$ 
22:           Train  $f(E)$  according to Alg. 3.1
23:           Scale  $Corr_{Res}$  of  $f(E)$ 
24:           If  $f(E) < f(R)$ 
25:                $Z_{n+1} = E$ 
26:           Else
27:                $Z_{n+1} = R$ 
28:       Else
29:            $b = \text{true}$ 
30:       If  $(f(R) \geq f(A))$ 
31:           Contraction  $Con = \rho R + (1 - \rho) C$ 
32:           Train  $f(Con)$  according to Alg. 3.1
33:           Scale  $Corr_{Res}$  of  $f(Con)$ 
34:           If  $f(Con) < f(R)$ 
35:                $Z_{n+1} = Con$ 
36:            $b = \text{false}$ 
37:       If  $b = \text{false}$ 
38:           shrink toward the best solution
39:           for  $i = 2$  to  $n+1$ :
40:                $Z_i = B + \sigma (Z_i - B)$ 
41:               Train  $Z_i$  according to Alg. 3.1
42:   end while
43:   Return  $Z [1]$ 

```

3.1.1 Accelerating Processing Time with Parallelism

Since serial NMM executes the vertices sequentially one vertex at a time, the optimization processing time is very expensive for deep CNN models. For this reason, it is necessary to utilize parallel computing to reduce the execution time; NMM provides a high degree of parallelism since there are no dependences between the vertices. In most iterations of NMM, the worst vertex is replaced with either the reflected, expanded, or contracted vertex. Therefore, these vertices can be evaluated simultaneously on distributed workers. There are two main types of parallelism models: asynchronous master-slave and synchronous master-slave.

Asynchronous master-slave model: The workers never stop to wait for any slower workers. However, it does not work exactly like a serial NMM. This technique is not suitable for our work because in some steps in the NMM, there is dependence in the calculation that requires all workers to stop. In addition, the implementation requires strict conditions and complex programming to ensure the program works properly. The final result is different than the serial NMM.

We implement a synchronous master-slave NMM model, which distributes the running of several vertices on workers while the master machine controls the whole optimization procedure. The master cannot move to the next step until all of the workers finish their tasks. A synchronous NMM has the same properties as a serial NMM, but it works faster. A serial NMM requires small changes in the implementation to work in a parallel way.

Recently, web services [68] provide a powerful technology for interacting

between distributed applications written in different programming languages and running on heterogeneous platforms, such as operating systems and hardware over the internet [69,70]. There are two popular methods for building a web service application to interact between distributed computers: Simple Object Access Protocol (SOAP) and Representational State Transfer (RESTful). We use RESTful [71] to create web services because it is simple to implement as well as lightweight, fast, and readable by humans; unlike RESTful, SOAP is difficult to develop, requires tools and is heavyweight. A RESTful service submits CNN hyperparameters into worker machines. Each worker builds and trains the architecture, computes the error rate and the correlation coefficient results, and returns both results to the master computer. Moreover, when shrinkage is selected, we run three vertices at the same time. This has a significant impact in reducing the computation time. Our framework is described in Algorithm 3.3, which details and integrates the techniques for finding the best CNN architecture.

Algorithm 3.3. The Proposed Framework Pseudocode with Parallel NMM

- 1: **Input:** n : Number of hyperparameters
 - 2: **Output:** best vertex ($Z[l]$) found that minimizes the objective function in
 - 3: Equation 3.3.
 - 4: Determine training sample T_S using RMHC
 - 5: Initialize the Simplex vertices ($Z_{1:n+1}$) randomly from Table 4.1
 - 6: $L = \lceil (n + 1)/3 \rceil$ # 3 is the number of workers
 - 7: For $j = 1$ to L
 - 8: Train each 3 vertices of $Z_{1:n+1}$ in parallel according to **Alg. 3.1**
 - 9: **For** $l= 1$ to Max_iterations :
 - 10: Normalize values of Corr_{Res} between the *max* and *min* of *Err* of vertices of
 - 11: (Z)
 - 12: Compute $f(Z_i)$ based on Equation 3.3. for all vertices $i = 1:n+1$
 - 13:
-

```

14:    $Z = \text{order the vertices so that } f(Z_1) \leq f(Z_2), \dots, < f(Z_{n+1}).$ 
15:   Set  $B = Z_1, A = Z_n, W = Z_{n+1}$ 
16:   Compute the centroid  $C$  of vertices without considering the worst vertex:  $C =$ 
17:    $\frac{1}{n} \sum_{i=1}^n Z_i$ 
18:   Compute reflected vertex:  $R = C + \alpha(C - W)$ 
19:   Compute Expanded vertex  $E = R + \gamma(R - C)$ 
20:   Compute Contracted vertex  $Con = \rho R + (1 - \rho)C$ 
21:   Train  $R, E,$  and  $Con$  simultaneously on workers 1,2, and 3 according to Alg.
22:   3.1
23:   Normalize  $Corr_{Res}$  of  $R, E,$  and  $Con$  between the max and min of Err of
24:   vertices of  $(Z)$ 
25:   Compute  $f(R), f(E),$  and  $f(con)$  based on Equation (3.3)
26:   If  $f(B) > R < W$ 
27:      $Z_{n+1} = W$ 
28:   Else If  $f(R) \leq f(B)$ 
29:     If  $f(E) < f(R)$ 
30:        $Z_{n+1} = E$ 
31:     Else
32:        $Z_{n+1} = R$ 
33:   Else
34:      $d = \text{true}$ 
35:     If  $f(R) \leq f(A)$ 
36:       If  $f(Con) \leq f(R)$ 
37:          $Z_{n+1} = Con$ 
38:        $d = \text{false}$ 
39:     If  $d = \text{true}$ 
40:       shrink toward the best vertex direction
41:        $L = \lceil n/3 \rceil$ 
42:       For  $k = 2$  to  $n+1$ : # do not include the best vertex
43:          $Z_k = B + \sigma(Z_k - B)$ 
44:       For  $j = 1$  to  $L$ :
45:         Train each 3 vertices of  $(Z_{2:n+1})$  in parallel on workers 1, 2, and 3
46:         according to Alg. 3.1
47:   Return  $Z[L]$ 

```

3.2 Analysis of optimized instance selection algorithms on large datasets with CNNs

When we implemented RMHL according to [74], the execution time of calculating the training sample T_S is very expensive. This is because each image in T_{TS} computes the accuracy using I -NN for all images in the T_S . However, adding or removing an image will affect the neighbors of the image that was added or removed in the testing set T_{TS} . Thus, there is no need to calculate the accuracy of all images in the testing set T_{TS} . We improved RMHC to make it works faster compared to the original RMHC with the same accuracy as follows: we first define k with a big value (e.g. $k \geq 50$). Once the training sample is selected randomly from the training set with size N_S . In each iteration of N_{iter} : choose one image I_{img1} randomly from T_S to be removed and replaced with another image I_{img2} that selected randomly from $(T_{TR} - T_S)$. Then, we find the k -NN images of I_{img1} in the testing set T_{TS} and store them in (A) . Also, we find the k -NN images of I_{img2} in the testing set T_{TS} and store them in (B) . Now, we compute the accuracy of A and B before and after the change. We have two cases as follows:

- **Case 1: before the change:**

Accuracy1= accuracy of A +accuracy of B using I -NN

- **Case 2: After the change:**

Accuracy2= accuracy of A +accuracy of U using I -NN

If accuracy2 > accuracy1 maintains the change, otherwise, roll-backed change. This means we compute the accuracy of $4k$ images of the testing set instead of all the testing

set $|T_{TS}|$ (see Algorithm 3.4). The experimental results show that our approach is much faster than the original RMHC with the same accuracy. We also investigated to study the impact of changing of the value of k on the running-speed and the accuracy.

Algorithm 3.4. Our proposed approach of RMHC

- 1: **Inputs:** Training set T_{TR} ,and Testing set (T_{TS})
 - 2: **Inputs:** N_S , N_{iter} , and k (where k is a large number $k \geq 50$)
 - 3: **Outputs:** T_S is a subset or training sample ($T_S \subset T_{TR}$)
 - 4: N_S instances or images will be selected randomly from T_{TR} to represent the training sample or subset T_S
 - 5: **For** $i=1$ to N_{iter} **do:**
 - 6: Select one image (I_{img1}) randomly from T_S to be removed
 - 7: Select one image (I_{img2}) randomly from $(T_{TR} - T_S)$ to be added to T_S
 - 8: Find the $k - NN$ images of I_{img1} in testing set T_{TS} and store them in (A)
 - 9: Find the $k - NN$ images of I_{img2} in testing set T_{TS} and store them in (B)
 - 10: *//Before the change:*
 - 11: Compute the accuracy of A and B ($Acc1 = Acc\ of\ A + Acc\ of\ B$)
 - 12: *//After the change*
 - 13: Compute the accuracy of A and B ($Acc2 = Acc\ of\ A + Acc\ of\ B$)
 - 14: **If** $Acc1 > Acc2$ **Then:**
 - 15: Keep the change
 - 16: **Else:**
 - 17: Roll-back the change
 - 18: **End For**
 - 19: **Return** T_S
-

3.3 A Deep Architecture for Face Recognition based on Multiple Feature Extractors

The architecture of our combined system is shown in Figure 3.5. We create different feature extractors such that each yields similar size feature vectors i.e., k features from each approach. This is done so that each feature type has equal importance until the final classification. The output of the three feature extractors i.e., PCA, LBP+PCA and LBP+NN is combined to produce a new feature vector of length $3k$. Note that the outputs from the LBP feature extractor are further fed into another PCA and NN to ensure that each classifier produces k features. The outputs of NN are taken from the neurons in the hidden layer after training it on the test dataset. The feature vector is fed into a Stacked Sparse Autoencoder (SSA) as followed by a softmax classification stage. We provide details of the different modules in our recognition system in the following subsections.

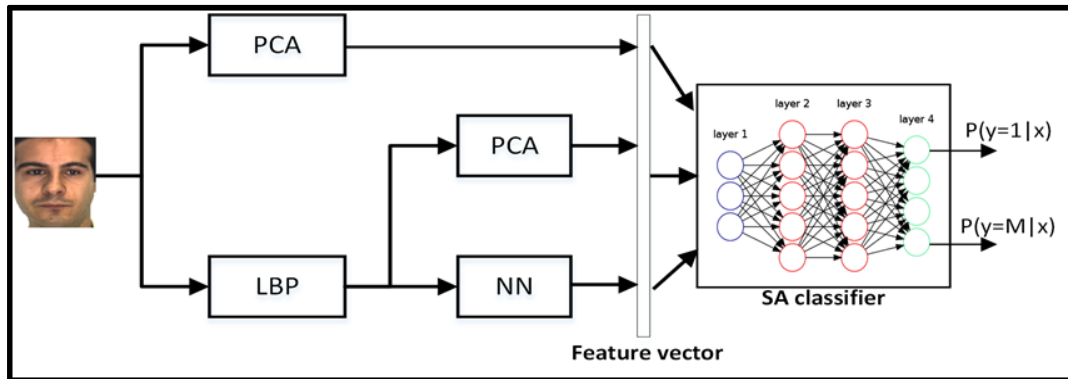


Figure 3.5. Our proposed system for face recognition

3.3.1 Principle Component Analysis (PCA)

PCA [99] is a dimensionality reduction algorithm, which uses a linear

transformation to convert high-dimensional data into low-dimensional data by finding the directions that maximize the variation between the data, by keeping the most information in high dimensions.

In PCA, a given M training set $[x_1, x_2, \dots, x_M]$ is transformed into $n \times 1$ vectors.

The covariance (C) of the combined data matrix is computed as:

$$C = \sum_{i=1}^M (x_i - \bar{x})(x_i - \bar{x})^T \quad (3.5)$$

where \bar{x} is the mean of M training samples. Singular value decomposition (SVD) of covariance (C) is used for computing Eigen values and Eigen vectors. To achieve dimensionality reduction, only k Eigen vectors that correspond to the largest Eigen values are selected. These Eigen vectors are combined into a matrix $U = [u_1, u_2, \dots, u_k]$. Each Eigen vector in the U matrix is referred to as an Eigen face. We then project each data x onto the k Eigen faces to reduce the input dimensionality of n to k dimensional subspace as shown in Equation 3.6.

$$P = U^T(x - \bar{x}) \quad (3.6)$$

3.3.2 Local Binary Pattern (LBP)

Originally, LBP [100] was developed as a texture descriptor. The main idea of LBP is that each center pixel will be compared with surrounding pixels. If the pixel value of the center is greater or equal to its neighbor, then it is given a value of 1, otherwise, a value of 0. The 1 or 0 assignment of each neighboring pixel of the center pixel is

combined into a binary number by selecting one neighbor pixel as a start point and then moving in a clockwise direction as shown Figure 3.6. The center pixel is then assigned this binary pattern.

In face recognition, LBP is used to produce a global description of the face by dividing the face into $(m \times n)$ blocks. Each block generates a local texture descriptor producing 59 histogram vectors for sampling in a 3×3 grid i.e., each center pixel has 8 neighbors; referred to as 8 points with radius 1 - $P(8, 1)$. The final vector for each image is equal to $m \times n \times 59$ [18]. To make the feature size of each feature extractor to k features, the outputs of LBP are further fed to a PCA, and separately to an NN to make each classifier's contribution equal in terms of number of features.

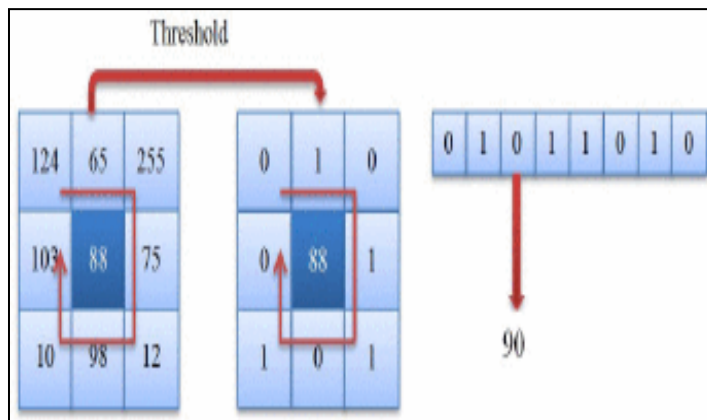


Figure 3.6. LBP operator computation for a 3×3 grid [101]

3.3.3 Stacked Sparse Autoencoder (SSA)

Stacked Sparse Autoencoder (SSA) is a neural network consisting of multiple layers of SA in which the outputs of each layer are connected to the inputs of the subsequent layer. The outputs of the last hidden layer are fed into the softmax classifier

for the classification task to estimate the probability of each class label in K classes as shown in equation 2.14

In this work, we construct an SSN with two hidden layers of SA ($h=2$) as illustrated in Figure 3.7. Each hidden layer of SSA is unsupervised, pre-training separately via SA in order to learn discriminative features on the input from the previous layer. So, after training the hidden layer using SA, we take the weights and biases of the encoder layer in the SA for an initialization of the hidden layer of the SSA. Once all weights and biases are initialized via SA, the second stage of SSA is a supervised fine-tune to train the entire network similar to the traditional neural network to minimize the prediction error on a supervised task to map the inputs into the desired outputs as possible. Algorithm 3.4 describes the steps for training SSA.

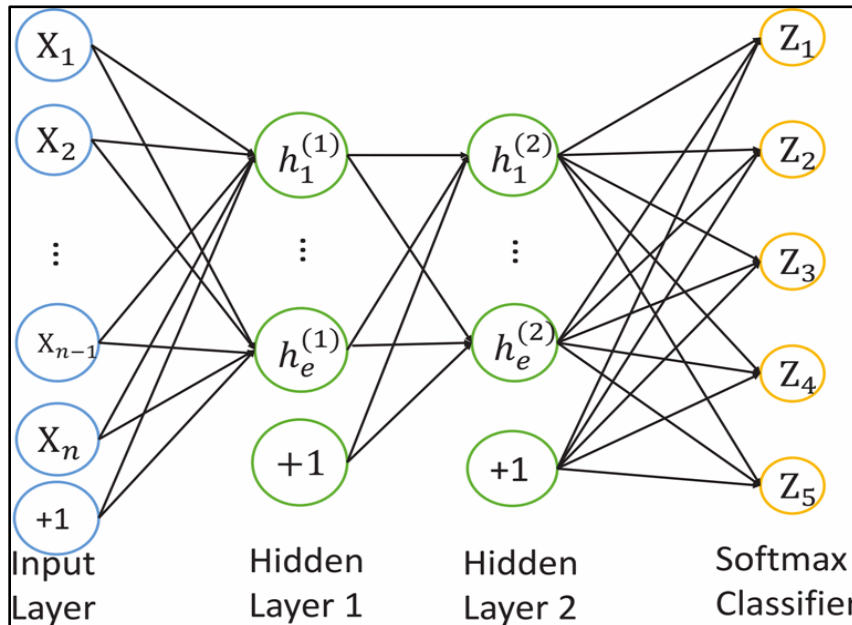


Figure 3.7. An architecture of stacked sparse autoencoder (SSA) that consists of two hidden layers.

Algorithm 3.4. Training SSA procedure

- 1: **Inputs:** joint feature vectors of the database, ρ, λ, h
 - 2: **Step 1: pre-training hidden layers:**
 - 3: **For** $i=0$ to h : # number of hidden layers
 - 4: Initialize (W^i, b^i) randomly for hidden layer $h=i$
 - 5: Find parameters of (W^i, b^i) for $h=i$ using SA by minimizing Equation 2.21
 - 6: **End For**
 - 7: **Step 2: Training softmax classifier**
 - 8: Train the softmax classifier (the inputs are the
 - 9: Outputs of last hidden layer)
 - 10: **Step 3: Fine-tune the whole network with supervised learning**
 - 11: Backpropagation with gradient descent
-

CHAPTER 4: IMPLEMENTATION AND RESULTS

4.1 A Framework for Designing the Architectures of CNNs

4.1.1 Datasets

The evaluation of our framework is done on the CIFAR-10, CIFAR-100, and MNIST datasets. The CIFAR-10 dataset [102] has 10 classes of 32×32 color images. There are 50K images for training, and 10K images for testing. CIFAR-100 [102] is the same size as CIFAR-10, except the number of classes is 100. Each class of CIFAR-100 contains 500 images for training and 100 for testing, which makes it more challenging. We apply the same preprocessing on both the CIFAR-10 and CIFAR-100 datasets, i.e., we normalize the pixels in each image by subtracting the mean pixel value and dividing it by the standard deviation. Then we apply ZCA whitening with an epsilon value of 0.01 for both datasets. Another dataset used is MNIST [33], which consists of the handwritten digits 0–9. Each digit image is 28×28 in size, and there are 10 classes. MNIST contains 60K images for training and 10K for testing. The normalization is done similarly to CIFAR-10 and CIFAR-100, without the ZCA whitening.

The previous works cited in our literature conduct experiments on CNN architecture design using four image classification datasets: MNIST, CIFAR-10, CIFAR-100, and SVHN. However, the previous works usually select and test the performance of their works on two or three of these datasets only. The advantage of selecting these

datasets is that we can compare our results with others instead of implementing their approaches.

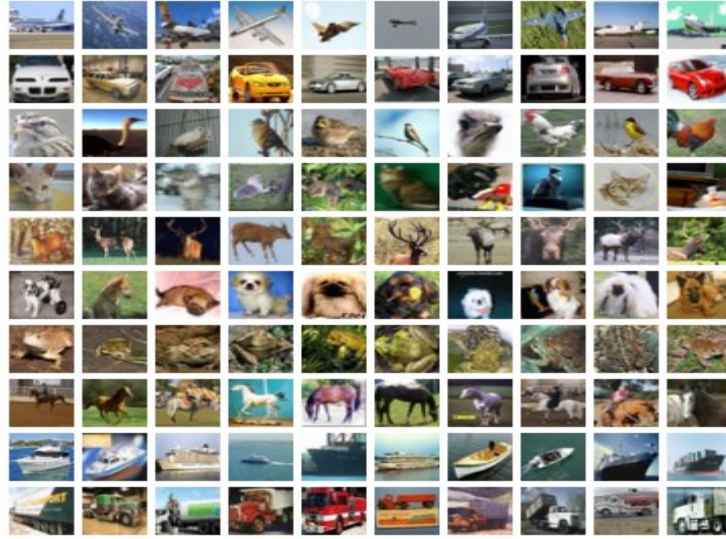


Figure 4.1. CIFAR-10 dataset, each rows shows different images of one class.

4.1.2 Experimental Setup

Our framework is implemented in Python using the Theano library [103] to train the CNN models. Theano provides automatic differentiation capabilities to compute gradients and allows the use of GPU to accelerate the computation. During the exploration phase via NMM, we select a training sample T_S using an RMHC algorithm with a sample size based on a margin error of 1 and confidence level of 95%. Then, we select 8000 images randomly from $(T_{TR}-T_S)$ for validation set T_V .

Training Settings: We use SGD to train CNN architectures. The final learning rate is set to 0.08 for 25 epochs and 0.008 for the last epochs; these values are selected after doing a small grid search among different values on the validation set. We set the

batch size to 32 images and the weight decay to 0.0005. The weights of all layers are initialized according to the Xavier initialization technique [54], and biases are set to zero. The advantage of Xavier initialization is that it makes the network converge much faster than other approaches. The weight sets it produces are also more consistent than those produced by other techniques. We apply ReLU with all layers and employ early stopping to prevent overfitting in the performance of the validation set. Once the error rate increases or saturates for a number of iterations, the model stops the training procedure. Since the training time of a CNN is expensive and some designs perform poorly, early stopping saves time by terminating poor architecture designs early. Dropout [11] is implemented with fully-connected layers with a rate of 0.5. It has proven to be an effective method in combating overfitting in CNNs, and a rate of 0.5 is a common practice. During the exploration phase of NMM, each of the experiments are run with 35 epochs. Once the best CNN architecture is obtained, we train it with the training set T_{TR} and evaluate it on a testing set with 200 epochs.

Nelder Mead Settings: The total number of hyperparameters n is required to construct the initial simplex with $n + 1$ vertices. However, this number is different for each dataset. In order to define n for a given dataset, we initialize 80 random CNN architectures as an additional step to return the maximum number of convolutional layers (C_{max}) in all architectures. Then, according to Equation 2.15, the number of hyperparameters n is given by:

$$n = C_{max} \times 4 + \text{the max number of fully-connected layers.} \quad (4.1)$$

NMM will then initialize a new initial simplex (Z_0) with $n + 1$ vertices. For all datasets, we set the value of the correlation coefficient parameter to $\eta = 0.20$. We select at random $N_{fm} = 10$ feature maps from the last convolutional layer to visualize their learned features and $N_{img} = 100$ images from the training sample to assess the visualization. The number of iterations for NMM is 25.

Table 4.1 summarizes the hyperparameter initialization ranges for the initial simplex vertices (CNN architectures) of NMM. The number of convolutional layers is calculated automatically by subtracting the depth from the number of fully-connected layers. However, the actual number of convolutional layers is controlled by the size of the input image, filter sizes, strides and pooling region sizes according to Equations 2.8 and 2.9. Some CNN architectures may not result in feasible configurations based on initial hyperparameter selections, because after a set of convolutional layers, the size of feature maps (W) or pooling (P) may become <1 , so the higher convolutional layers will be automatically eliminated. Therefore, the depth varies through CNN architectures; this will be helpful to optimize the depth. There is no restriction on fully-connected layers. For example, the hyperparameters of the following CNN architecture are initialized randomly from Table 4.1, which consists of six convolutional layers and two fully-connected layers as follows:

[[85, 3, 2, 1], [93, 5, 1, 1], [72, 3, 2, 2], [61, 7, 2, 1], [83, 7, 2, 3], [69, 3, 2, 3],
[715, 554]]

For an input image of size 32×32 , the framework designs a CNN architecture

with only three convolutional layers because the output size of a fourth layer would be negative. Thus, the remaining convolutional layers from the fourth layer will be deleted automatically by setting them to zeros as shown below:

[[85, 3, 2, 1], [93, 5, 1, 1], [72, 3, 2, 2], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [715, 554]]

Hyperparameter	Min.	Max.
Depth	5	10
Number of fully-connected layers	1	2
Number of filters	50	150
Kernel sizes	3	11
Number of pooling layers	4	7
Pooling region sizes	1	4
Number of neurons in fully-connected layers	250	800

Table 4.1. Hyperparameter initialization ranges.

4.1.3 Results and Discussion

First, we validate the effectiveness of the proposed objective function compared to the error rate objective function. After initializing a simplex (Z_0) of NMM, we optimize the architecture using NMM based on the proposed objective function (error rate as well as visualization). Then, from the same initialization (Z_0), we execute the NMM based on the error rate objective function alone by setting η to zero. Table 4.2 compares the error rate of five experiment runs obtained from the best CNN architectures found using the objective functions presented above on the CIFAR-10 and CIFAR-100 datasets respectively. The results illustrate that our new objective function outperforms the

optimization obtained from the error rate objective function alone. The error rate averages of 15.87% and 40.70% are obtained with our objective function, as compared to 17.69% and 42.72% when using the error rate objection function alone, on CIFAR-10 and CIAFAR-100 respectively. Our objective function searches the architecture that minimizes the error and improves the visualization of learned features, which impacts the search space direction, and thus produces a better CNN architecture.

Expt. Num.	Error Rate Based on the Error Objective Function	Error Rate Based on Our Objective Function
Results comparison on CIFAR-10		
1	18.10%	15.27%
2	18.15%	16.65%
3	17.81%	16.14%
4	17.12%	15.52%
5	17.27%	15.79%
Avg.	17.69%	15.87%
Results comparison on CIFAR-100		
1	42.10%	41.21%
2	43.84%	40.68%
3	42.44%	40.15%
4	42.98%	41.37%
5	42.26%	40.12%
Avg.	42.72%	40.70%

Table 4.2. Error rate comparisons between the top CNN architectures obtained by our objective function and the error rate objective function via NMM.

We also compare the performance of our approach with existing methods that did not apply data augmentation, namely hand-designed architectures by experts, random

search, genetic algorithms, and Bayesian optimization. For CIFAR-10, [102] reported that the best hand-crafted CNN architecture design tuned by human experts obtained an 18% error rate. In [25], genetic algorithms are used to optimize filter sizes and the number of filters for convolutional layers; it achieved a 25% error rate. In [104], SMAC is implemented to optimize the number of filters, filter sizes, pooling region sizes, and fully-connected layers with a fixed depth. They achieved an error rate of 17.47%. As shown in Table 4.3, our method outperforms others with an error rate of 15.27%. For CIFAR-100, we implement random search by picking CNN architectures from Table 4.1, and the lowest error rate that we obtained is 44.97%. In [104], which implemented SMAC, an error rate of 42.21% is reported. Our approach outperforms these methods with an error rate of 40.12%, as shown in Table 4.3.

Method	CIFAR-10	CIFAR-100
Human experts design [102]	18%	-
Random search (our implementation)	21.74%	44.97%
Genetic algorithms [25]	25%	-
SMAC [104]	17.47%	42.21%
Our approach	15.27%	40.12%

Table 4.3. Error rate comparison for different methods of designing CNN architectures on CIFAR-10 and CIFAR-100. These results are achieved without data augmentation.

In each iteration of NMM, a better-performing CNN architecture is potentially discovered that minimizes the value of the objective function; however, it can get stuck in a local minimum. Figure 4.2 shows the value of the best CNN architecture versus the

iteration using both objective functions, i.e., the objective function based on error rate alone and our new, combined objective function. In many iterations, NMM paired with the error rate objective function is unable to pick a new, better-performing CNN architecture and gets stuck in local minima early. The number of architectures that yield a higher performance during the optimization process is only 12 and 10 out of 25, on CIFAR-10 and CIFAR-100 respectively. In contrast, NMM with our new objective function is able to explore better-performing CNN architectures with a total number of 19 and 17 architectures on CIFAR-10 and CIFAR-100 respectively, and it does not get stuck early in a local minimum, as shown in Figure 4.2 (b). The main hallmark of our new objective function is that it is based on both the error rate and the correlation coefficient (obtained from the visualization of the CNN via deconvnet), which gives us flexibility to sample a new architecture, thereby improving the performance.

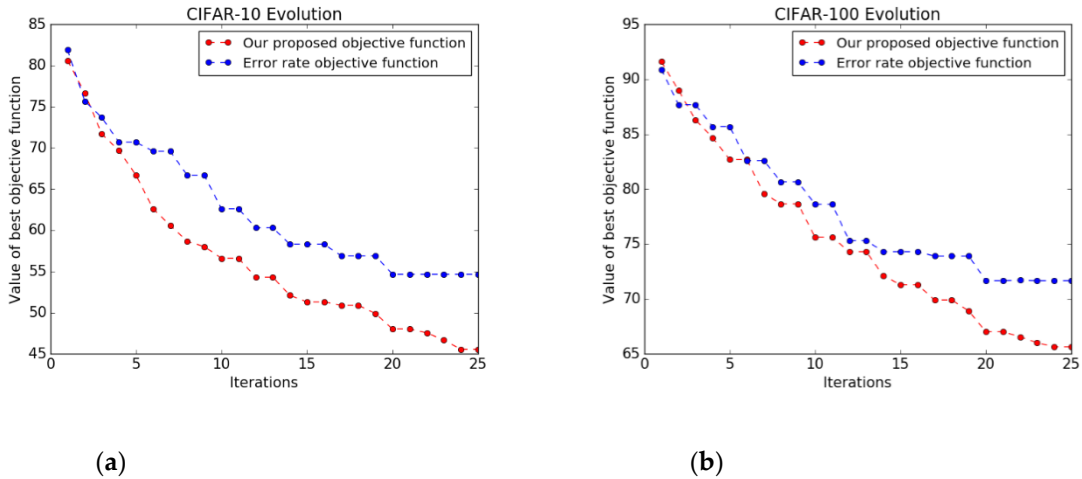


Figure 4.2. Objective functions progress during the iterations of NMM. (a) CIFAR-10; (b) CIFAR-100.

We further investigated the characteristics of the best CNN architectures obtained by both objective functions using NMM on CIFAR-10 and CIFAR-100. We took the average

of the hyperparameters of the best CNN architectures. As shown in Figure 4.3 (a), CNN architectures based on the proposed framework tend to be deeper compared to architectures obtained by the error rate objective function alone. Moreover, some convolutional layers are not followed by pooling layers. As a result, we found that reducing the number of pooling layers shows a better visualization, and results in adding more layers. On the other hand, the architectures obtained by the error objective function alone result in every convolutional layer being followed by a pooling layer.

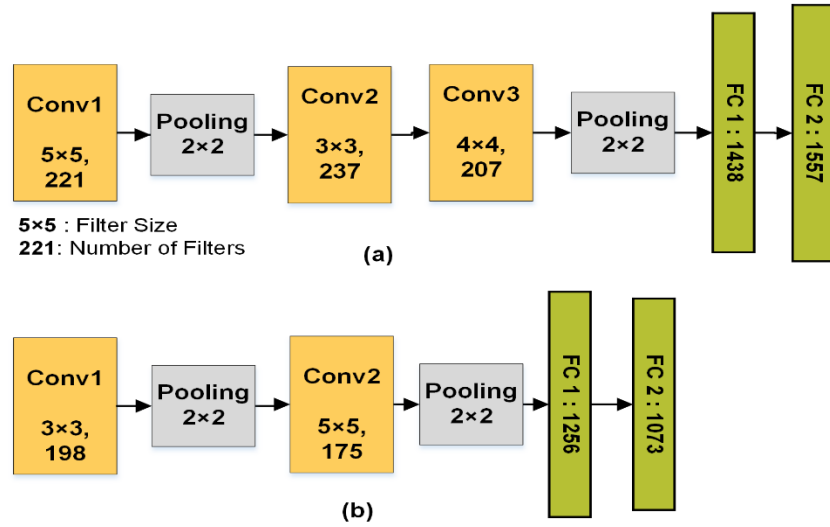


Figure 4.3. The average of the best CNN architectures obtained by both objective functions. (a) The architecture averages for our framework; (b) The architecture averages for the error rate objective function.

We compare the run-time of NMM on a single computer with a parallel NMM algorithm on three distributed computers. The running time decreases almost linearly ($3\times$) as the number of workers increases, as shown in Table 4.4. In a synchronous master-slave NMM model, the master cannot move to the next step until all of the workers finish their jobs, so other workers wait until the biggest CNN architecture training is done. This creates a minor delay. In addition, the run-time of *Corr* based on Equation 3.1 is 0.05

second; when based on FFT in Equation 3.2, it is 0.01 second.

Method	Single Computer Execution	Parallel Execution Time
	Time	
CIFAR-10	48 h	18 h
CIFAR-100	50 h	24 h
MNIST	42 h	14 h

Table 4.4. Comparison of execution time by serial NMM and parallel NMM for Architecture Optimization.

We compare the performance of our approach with state-of-the-art results and recent works on architecture search on three datasets. As seen in Table 4.5, we achieve competitive performance on the MNIST dataset in comparison to the state-of-the-art results, with an error rate 0.42%. The results on CIFAR-10 and CIFAR-100 are obtained after applying data augmentation. Recent architecture search techniques [61-64] show good results; however, these promising results were only possible with substantial computation resources and a long execution time. For example, GA [63] used genetic algorithms to discover the network architecture for a given task. Each experiment was distributed to over 250 parallel workers. Ref. [61] used reinforcement learning (RL) to tune a total of 12,800 different architectures to find the best on CIFAR-10, and the task took over three weeks. However, our framework uses only three workers and requires tuning an average of 260 different CNN architectures in around one day. It is possible to run our framework on a single computer. Therefore, our approach is comparable to the state-of-the-art results on CIFAR-10 and CIFAR-100, because our results require significantly fewer resources and less processing time. Some methods such as Residential

Networks (ResNet) [105] achieve state-of-the-art results because the structure is different than a CNN. However, it is possible to implement our framework in ResNet to find the best architecture.

Method	MNIST	CIFAR-10	CIFAR-100
Maxout [106]	0.45%	9.38%	38.57%
DropConnect [107]	0.57%	9.32%	-
DSN [108]	0.39%	8.22%	34.57%
ResNet(110) [105]	-	6.61%	-
CoDeepNEAT [62]	-	7.3%	-
MetaQNN [64]	0.44%	5.9%	27.14%
RL [61]	-	6.0%	-
GA [63]	-	5.4%	23.30%
Our Framework	0.42%	10.05%	35.46%

Table 4.5. Error rate comparisons with state-of-the-art methods and recent works on architecture design search. We report results for CIFAR-10 and CIFAR-100 after applying data augmentation and results for MNIST without any data augmentation.

To test the robustness of the selection of the training sample, we compared a random selection against RMHC for the same sample size; we found that RMHC achieved better results by about 2%. We found that our new objective function is very effective in guiding a large search space into a sub-region that yields a final, high-performing CNN architecture design. Pooling layers and large strides show a poor visualization, so our framework restricts the placement of pooling layers: It does not follow every convolutional layer with a pooling layer, which shrinks the size of the strides. Moreover, this results in increased depth. We found that the framework allows navigation through a large search space without any restriction on depth until it finds a

promising CNN architecture. Our framework can be implemented in other problem domains where images and visualization are involved, such as image registration, detection, and segmentation.

4.1.4 Statistic Power Analysis

Power analysis can be used to calculate the required minimum sample size. We use the power analysis to estimate that our sufficient sample size of the data can attain adequate power. We used the result of our first experiment to conduct the power analysis. The power analysis also allows us to examine the alternative hypothesis.

There are two kinds of statistical hypotheses: the null hypothesis (H_0) and the alternative hypothesis (H_1). The result of the exam will be: Accept H_0 or reject H_0 . The statistical hypothesis of our work is that our proposed objective function performs better than the error rate objective function (Equation 3.3). There are several types of tests, we used the two-tailed test.

The sample size (S) is computed as follows:

$$S = \frac{N * P(1 - p)}{\left[N - 1 * \left(\frac{d^2}{z^2} \right) \right] + p(1 - p)} \quad (4.2)$$

Where d is the effect size, p is the required power (commonly 80%), z is the t -value of $\alpha/2$. N is the total number of the training set. In our case, the values of N is 50,000 for CIFAR-10, $p=80\%$, $d=3\%$, Significance (α) = 5%. Level of confidence ($1 - \alpha$) = 95%, $z=1.96$. According to Equation 4.2, the sample size (S) is calculated as follows:

$$S = \frac{50,000 * 0.8(1 - 0.8)}{\left[50,000 - 1 * \left(\frac{0.03^2}{1.96^2}\right)\right] + 0.8(1 - 0.8)} = \frac{8,000}{1.4615} \approx 5474$$

According to the result, the size of 5474 is adequate. Then, the parameters below are calculated in order to make the decision of the hypothesis.

- Mean (\bar{X}) = $\frac{\sum x}{n} = 0.8473$
- Variance (σ^2) = $\frac{\sum(x-\bar{x})^2}{n} = 0.1293$
- Standard Deviation (σ) = $\sqrt{\sigma^2} = 0.3597$
- Significance (α) = 0.05
- Level of confidence ($1 - \alpha$) = 95%
- Critical $t = 1.653$
- Standard Error (SE) = $\frac{\sigma}{\sqrt{S}} = \frac{0.3597}{\sqrt{5474}} = 0.00486$
- Hypothesis (H_0) = 50%
- t value = $\frac{\bar{X} - H_0}{SE} = \frac{0.8473 - 0.5}{0.00486} = 71.46$

The H_0 is rejected when the t value > Critical t . In our case, the t value (71.46) > critical t (1.653). Thus, it means Reject H_0 and Accept H_1 . Furthermore, the power analysis shows that a sample set of 5474 is sufficient and can prove that our proposed method is efficient.

4.2 Analysis of optimized instance selection algorithms on large datasets with CNNs

4.2.1 Dataset

We evaluate instance selection methods on CIFAR-10 with the same network architecture. We split training set into two parts: 40000 images for training and 10000 images for validation. The preprocessing step only normalizes pixels values between [0, 1].

4.2.2 Training methodology

Most of training parameters are similar Krizhevsky et al. [1]. We trained our network using stochastic gradient descent (SGD) with mini-batch size of 32 instances, weight decay is 0.0005, we initialized learning rate at 0.01 for iteration < 100000 , otherwise 0.001 (iteration = number of training batches \times epoch + mini batch index). We initialized the weights of all layers with mean-zero normal distribution with standard deviation 0.05 and all biases equal to 0. Dropout rate is 0.5 for fully-connected layers and ReLU activation function for all layers. The total number of epochs is 120 for training the network. All inputs were normalized between [0, 1]. Finally the last layer is softmax. The code is written by python for different instance selection methods and CNNs.

The CNN architecture of our implementation is summarized in Table 4.6. This architecture is uniform for all obtained subsets.

Layer No.	Layer type	Feature map size	Kernel size
1	Color image	32×32×3	-
2	Convolutional	28×28×30	5×5
3	Max pooling	14×14×30	2×2
4	Convolutional	10×10×40	5×5
5	Max pooling	5×5×40	2×2
6	Full connected	200×1	-
7	Softmax layer	10×1	-

Table 4.6. Uniform CNN architecture summary.

4.2.3 Experimental results and discussion

Table 4.7 shows the accuracy of different instance selection methods. Results are obtained from the average of five experiments with the same architecture as shown in Table 4.6. The reduction rate is based on the technique. ConNN gives the best accuracy compared to other instance selection algorithms. It retains noisy instances; this means that CIFAR-10 contains a lot of noisy and border instances. For this reason, the reduction rate is only 25%. In contrast, the reduction rate in ENN is high, but the accuracy is low (48.16%). ENN makes a smooth boundary between the classes.

Method	# instances in S	Reduction%	Acc. %
Entire Dataset	40000	0 %	73.02%
Random subset	8632	78.42 %	58.21%
CNN	30243	24.39 %	69.36%
ENN	8632	78.42 %	48.16%
All kNN	19750	50.62 %	62%
RMHC	8632	78.42 %	59.42
Clustering	8632	78.42 %	59.42

Table 4.7. Illustrates the accuracy of different instance

We compared the running time of the original RMHC with our improved approach. Table 4.8 shows the running time of one iteration (replacement) for both approaches. As shown, the running time of our approach takes 7.9 seconds and 150.86 seconds on the original RMHC.

Method	Time in Seconds
Original RMHC	150.86
Our proposed approach of RMHC	7.9

Table 4.8 Running time comparison between original RMHC and our proposed approach of RMHC for one iteration.

The running speed can be computed in the following equation:

$$speed = \frac{|T_s| \times |T_{TS}| \times N_{iter}}{4 \times k \times |T_s| \times N_{iter}} = \frac{|T_{TS}|}{4k} \quad (4.3)$$

Suppose $k=100$, the size of testing set is $|T_{TS}|= 10,000$, $|T_s|=3,000$, and $N_{iter} = 200$. The total number of operations for RMHC= $|T_{TS}| \times |T_s| \times N_{iter}=10,000 \times 3,000 \times 200=6 \times 10^9$ while the total number of operations for our method $= 4 \times k \times |T_s| \times N_{iter}= 4 \times 200 \times 3,000 \times 100 = 24 \times 10^7$. Based on Equation 4.3, the running speed of our approach is $25 \times$ faster than the original RMHC. We investigated further to find the relationship between the running speed and the value of k . As shown in Figure 4.4, we found that increasing the value of k reduces the running speed. The running speed of our approach is equal to the original RMHC when the value of $k=2,500$. Logically, it is correct because $4 \times k = 4 \times 2,500 = 10,000$, which is equal to the testing set size $|T_{TS}|$.

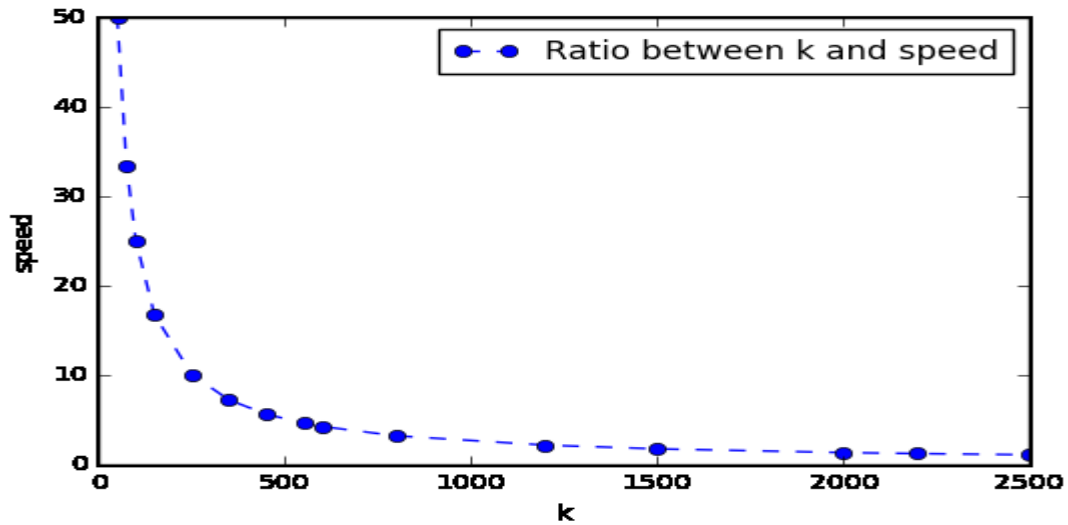


Figure 4.4. The running speed with different values of k .

We evaluated different types of instance selection methods in reducing the training set size in order to shorten the training time. Based on techniques that are evaluated, condensed nearest neighbor gave acceptable result compared to result of the entire dataset. We conclude from the comparison that retaining noisy instances is more important to get good accuracy.

4.3 A Deep Architecture for Face Recognition Based on Multiple Feature Extractors

4.3.1 Datasets

The ORL (Olivetti Research Labs) database of faces [109] is a set of grayscale images, each with a size of 112×92 pixels. It consists of 400 images for 40 people. The images were taken under different lighting conditions, facial expressions and different

details, such as the subject wearing glasses. In our experiment, we specified the first 5 images of each person for training and the remaining 5 for testing.

The AR face database [110] has color images for 126 individuals (70 men and 56 women). The images have high variation in facial expressions, illumination, and occlusion with scarves and sunglasses. The images were taken in two sessions separated by 2 weeks. Each session contains 13 images for each person. The first session was meant for training, whereas the second session for testing. We used the same images that were used in [111], in which all images were cropped and resized with 120×165 pixels for 50 randomly-selected men and 50 randomly-selected women.

4.3.2 Training setting

The system was written in Python. We used gradient descent to train the neural networks, SA, and SSA. The value of the learning rate is set to 0.01 for all networks, with a weight decay of 0.0005 and momentum of 0.9. We initialized the weights according to the Xavier initialization technique [54], and biases are set to zero. The total number of epochs is 200. We apply sigmoid activation function to the layers. The number of features (k) is 150 and 250 on the ORL and AR databases respectively.

In some classifiers, such as PCA and LBP, recognition is done by calculating the Euclidean distance (L_2) between feature vectors in each testing image in the testing set with all training images. The minimum distance will be chosen as the recognized face. The activation function in our proposed approach is a sigmoid function (sigmoid in the range $[0, 1]$). Therefore, the outputs of PCA and LBP in the first stage will be normalized

between 0 and 1 before being forwarded into the NN or SSA.

4.3.3 Experimental Results and Discussion

Tables 4.9 and 4.10 show the results from a series of experiments designed to evaluate the effectiveness of our proposed system of combining MC with SSA. Table 4.9 shows the results of different individual classifiers and MC methods. The results clearly show that our approach provides the highest accuracy compared to other techniques. After extracting feature vectors with length $3k$, we also investigated whether we could replace some popular classifiers such as NN, Support Vector Machine (SVM) [112], and logistic regression [113] with SSA as a classifier in the last stage. This comparison is necessary to show the effectiveness of SSA as a classifier in the last phase compared to other classifiers. It can be seen in Table 4.10 that SSA outperforms other classifiers with an accuracy of 98% and 81.67% on the ORL and AR databases respectively. Figure 4.5 summarizes Tables 4.9 and 4.10 by comparing our approach to all individual classifiers and MC systems that have been implemented and tested [114,115].

Classifying test images based on the joint feature vector extracted from three different extraction techniques using Euclidean distance shows low accuracy. This means joint feature vectors do not provide diverse features between different classes. Thus, SSA can detect new features from these joint vector features in a hierarchical way to make the recognition task work effectively. The advantage of SSA is 1) unsupervised pre-training by sparse autoencoder extracts the more useful features from joint feature vectors and initializes the weights, and 2) supervised training in the next step modifies the boundaries

between classification classes and minimizes the classification error.

Technique name	ORL	AR
PCA	89%	56.74%
LBP	95%	60.52%
LBP+PCA	93.50%	61.37%
LBP+NN	95.50%p	73.60%
Our system (MC+SSA)	98%	81.67%
Joint feature vector	85.50%	63.38%

Table 4.9. Performance of different classifiers on the ORL and AR databases, including individual classifiers and MC systems.

Technique name	ORL	AR
MC+ Logistic regression	97%	74.08%
MC+NN	96.5%	67.61%
MC+ SVM	86.0%	68.30%
Our system (MC+SSA)	98%	81.67%

Table 4.10. Performance of classifiers with the replacement of classifiers with SA in the last stage.

While previous techniques combine the results of individual classifiers, we propose combining the features obtained from diverse approaches into a deep architecture for final classification. Even though we present results obtained from combining features from PCA and LBP approaches, our technique and implementation are open to accepting features from any new or existing feature extraction approach. The key is to concatenate the features from individual techniques in such a way that no single technique biases the result in its favor. We accomplish this by ensuring that features from all individual

techniques have the same dimensionality and are scaled between 0 and 1. The combined features are fed to a stacked sparse autoencoder to classify the results correctly. If a feature extraction technique results in a relatively larger (or smaller) number of features, we employ a preprocessing NN to match it to the number of features that other extractors have provided. In our detailed testing on face datasets, we accomplish better results than any of the previous works.

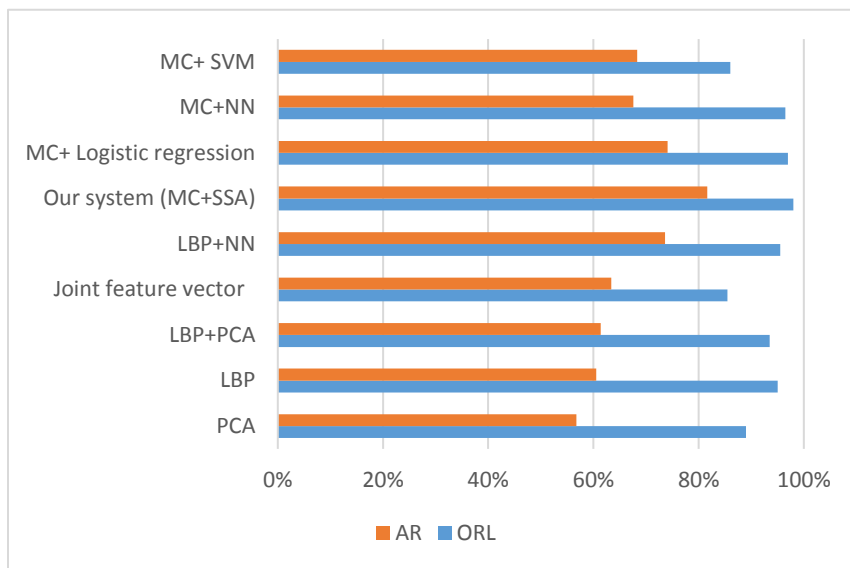


Figure 4.5. Performance of all proposed classifiers in Tables 4.9 and Table 4.10.

CHAPTER 5: CONCLUSIONS

Convolutional neural networks (CNNs) have achieved impressive results in variety applications of computer vision and speech processing. Despite this, it is still difficult to optimize an architecture appropriately for a given problem. This dissertation addresses this issue by constructing a framework that automatically discovers a high-performing architecture without hand-crafting it.

In previous work on optimization, the error rate alone is used as the objective function; i.e., the error rate on the validation set is used to decide whether one architecture design is better than another. In this work, we present a new objective function that utilizes information from the error rate on the validation set as well as the quality of the feature visualization obtained from deconvnet. The objective function aims to find a CNN architecture that minimizes the error rate and provides a high level of feature visualization. We demonstrate that exploiting the visualization of feature maps via deconvnet in addition to the error rate in the objective function produces a superior performance. The advantage of the proposed objective function is that it does not get stuck in local minima easily as compared to the error rate objective function. Further, our new objective function results in much faster convergence towards a better architecture. Consequently, our results are better, as shown in the results section.

To address high execution time in exploring numerous CNN architectures during the optimization process, we parallelized the NMM in conjunction with an optimized training sample subset. Our framework is limited by the fact that it cannot be applied

outside of the imaging or vision field because part of our objective function relies on visualization via deconvnet. We also investigated the effectiveness instance selection methods in reducing the training set size in order to shorten the running time. In addition, we proposed a new approach that makes RMHC works quickly with the same accuracy as the original RMHC.

In our future work, we plan to implement our framework on different problems in the context of the images. We will also explore cancellation criteria to discover and avoid evaluating poor architectures, which will accelerate the execution time.

REFERENCES

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, Lake Tahoe, Nevada, pp. 1097-1105, 2012.
- [2] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915-1929, 2013.
- [3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [4] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.
- [5] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.
- [6] A. Conneau, H. Schwenk, Y. Le Cun, and L. Barrault, "Very deep convolutional networks for text classification," *arXiv preprint arXiv:1606.01781*, 2016.
- [7] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215-243, 1968.
- [8] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

- [9] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1-127, 2009.
- [10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009*, Miami, FL, USA, pp. 248-255, 2009.
- [11] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.
- [12] Y. Liu *et al.*, "Application of Deep Convolutional Neural Networks for Detecting Extreme Weather in Climate Datasets," *arXiv preprint arXiv:1605.01156*, 2016.
- [13] A. Esteva *et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115-118, 2017.
- [14] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, Zurich, Switzerland, Springer 2014, pp. 818-833, 2014.
- [15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv preprint arXiv:1512.03385*, 2015.
- [16] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in Neural Information Processing Systems*, Montreal, Quebec, Canada, pp. 2368-2376, 2015.

- [17]K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Boston, Massachusetts, pp. 5353-5360, 2015.
- [18]J. Gu *et al.*, "Recent advances in convolutional neural networks," *arXiv preprint arXiv:1512.07108*, 2015.
- [19]X. Z. Alice and B. Mikhail, "Lazy paired hyper-parameter tuning," in *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 978-1-57735-633-2, ed. Beijing, China: AAAI Press, 2013, pp. 1924-1931, 2013.
- [20]J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, Granada, Spain, pp. 2546-2554, 2011.
- [21]J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, Lake Tahoe, Nevada, USA, pp. 2951-2959, 2012.
- [22]B. Wang, H. Pan, and H. Du, "Motion Sequence Decomposition-Based Hybrid Entropy Feature and Its Application to Fault Diagnosis of a High-Speed Automatic Mechanism," *Entropy*, vol. 19, no. 3, p. 86, 2017.
- [23]S. Albelwi and A. Mahmood, "A Framework for Designing the Architectures of Deep Convolutional Neural Networks," *Entropy*, vol. 19, no. 6, p. 242, 2017.
- [24]L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *arXiv preprint arXiv:1603.06560*, 2016.

- [25] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," presented at the Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, Austin, Texas, 2015.
- [26] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, p. 436-444, 2015.
- [27] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329, 2017.
- [28] L. Deng, "Three classes of deep learning architectures and their applications: a tutorial survey," *APSIPA Transactions on Signal and Information Processing*, 2012.
- [29] Y. LeCun *et al.*, "Efficient BackProp," presented at the Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop, 1998.
- [30] F. Schilling, "The Effect of Batch Normalization on Deep Convolutional Neural Networks," Master thesis, Computer science and engineering KTH Royal Institute of Technology in Stockholm, Sweden, 2016.
- [31] I. Chakroun, T. Haber, and T. J. Ashby, "SW-SGD: The Sliding Window Stochastic Gradient Descent Algorithm," *Procedia Computer Science*, vol. 108, pp. 2318-2322, 2017.
- [32] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.

- [33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.
- [34] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," ed: Citeseer, 2009.
- [35] D. Jia, D. Wei, R. Socher, L. Li-Jia, L. Kai, and F.-F. Li, "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009*, pp. 248-255, 2009.
- [36] T. Chen, R. Xu, Y. He, and X. Wang, "A gloss composition and context clustering based distributed word sense representation model," *Entropy*, vol. 17, no. 9, pp. 6007-6024, 2015.
- [37] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-2010)*, Haifa, Israel, pp. 807-814, 2010.
- [38] A. Aldhaheri and J. Lee, "Event detection on large social media using temporal analysis," in *7th IEEE Annual on Computing and Communication Workshop and Conference (CCWC), 2017*, Las Vegas, USA, 2017, pp. 1-6: IEEE.
- [39] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2, pp. 1137-1145: Stanford, CA, 1995.
- [40] R. Schaer, H. Müller, and A. Depeursinge, "Optimized Distributed Hyperparameter Search and Simulation for Lung Texture Classification in CT Using Hadoop," *Journal of Imaging*, vol. 2, no. 2, p. 19, 2016.

- [41] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281-305, 2012.
- [42] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*, Lille, France, Springer 2011, pp. 507-523, 2011.
- [43] I. Murray and R. P. Adams, "Slice sampling covariance hyperparameters of latent Gaussian models," in *Advances in Neural Information Processing Systems*, Vancouver, British Columbia, Canada, pp. 1732-1740, 2010.
- [44] M. A. Gelbart, "Constrained Bayesian Optimization and Applications," Doctoral dissertation, Graduate School of Arts & Sciences, Harvard University, 2015.
- [45] I. Loshchilov and F. Hutter, "CMA-ES for Hyperparameter Optimization of Deep Neural Networks," *arXiv preprint arXiv:1604.07269*, 2016.
- [46] J. Luketina, M. Berglund, K. Greff, and C. T. Raiko, "Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters," *arXiv preprint arXiv:1511.06727*, 2015.
- [47] L.-W. Chan and F. Fallside, "An adaptive training algorithm for back propagation networks," *Computer speech & language*, vol. 2, no. 3-4, pp. 205-218, 1987.
- [48] J. Larsen, C. Svarer, L. N. Andersen, and L. K. Hansen, "Adaptive regularization in neural network modeling," in *Neural Networks: Tricks of the Trade*, Springer 1998, pp. 113-132, 1998.
- [49] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *ArXiv e-prints* vol. 1602, ed, 2016.

- [50] C.-C. Yu and B.-D. Liu, "A backpropagation algorithm with adaptive learning rate and momentum coefficient," *Proceedings of the IEEE International Joint Conference on Neural Networks. IJCNN'02.*, Honolulu, HI, USA, vol. 2, pp. 1218-1223, 2002.
- [51] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.
- [52] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *NIPS*, Denver, CO, USA, pp. 402-408, , 2000.
- [53] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE international conference on Acoustics, speech and signal processing, icassp 2013*, Vancouver, BC, Canada, 2013, pp. 6645-6649, 2013.
- [54] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," presented at the Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Sardinia, Italy, 2010.
- [55] B. A. Garro and R. A. Vázquez, "Designing artificial neural networks using particle swarm optimization algorithms," *Computational intelligence and neuroscience*, vol. 2015, p. 61, 2015.
- [56] K. Chau and C. Wu, "A hybrid model coupled with singular spectrum analysis for daily rainfall prediction," *Journal of Hydroinformatics*, vol. 12, no. 4, pp. 458-473, 2010.

- [57] W.-c. Wang, K.-w. Chau, D.-m. Xu, and X.-Y. Chen, "Improving forecasting accuracy of annual runoff time series using ARIMA based on EEMD decomposition," *Water Resources Management*, vol. 29, no. 8, pp. 2655-2675, 2015.
- [58] R. Taormina and K.-W. Chau, "Data-driven input variable selection for rainfall–runoff modeling using binary-coded particle swarm optimization and Extreme Learning Machines," *Journal of hydrology*, vol. 529, pp. 1617-1632, 2015.
- [59] J. Zhang and K.-W. Chau, "Multilayer Ensemble Pruning via Novel Multi-sub-swarm Particle Swarm Optimization," *J. UCS*, vol. 15, no. 4, pp. 840-858, 2009.
- [60] P. Kulkarni, J. Zepeda, F. Jurie, P. Pérez, and L. Chevallier, "Learning the Structure of Deep Architectures Using L1 Regularization," in *British Machine Vision Conference BMVC 2015*, SWANSEA, UK, pp. 23.1-23.11, 2015.
- [61] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.
- [62] R. Miikkulainen *et al.*, "Evolving Deep Neural Networks," *arXiv preprint arXiv:1703.00548*, 2017.
- [63] E. Real *et al.*, "Large-Scale Evolution of Image Classifiers," *arXiv preprint arXiv:1703.01041*, 2017.
- [64] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing Neural Network Architectures using Reinforcement Learning," *arXiv preprint arXiv:1611.02167*, 2016.
- [65] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

- [66] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Columbus, Ohio, pp. 580-587, 2014.
- [67] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *IEEE International Conference on Computer Vision (ICCV), 2011*, Barcelona, Spain, pp. 2018-2025, 2011.
- [68] S. Albelwi and A. Mahmood, "Automated Optimal Architecture of Deep Convolutional Neural Networks for Image Recognition," in *15th IEEE International Conference on Machine Learning and Applications, (ICMLA) 2016*, Anaheim, CA, USA, pp. 53-60, 2016.
- [69] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1-19, 2011.
- [70] M. Grochowski, "Simple Incremental Instance Selection Wrapper for Classification," in *Artificial Intelligence and Soft Computing*, Berlin, Heidelberg, Springer 2012, pp. 64-72, 2012.
- [71] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, "A review of instance selection methods," *Artificial Intelligence Review*, vol. 34, no. 2, pp. 133-143, 2010.
- [72] H. Liu and H. Motoda, "Data Reduction via Instance Selection," in *Instance Selection and Construction for Data Mining*, H. Liu and H. Motoda, Eds. Boston, MA: Springer US, pp. 3-20, 2001.
- [73] Z. Zhang, F. Eyben, J. Deng, and B. Schuller, "An agreement and sparseness-based learning instance selection and its application to subjective speech phenomena," in

- Proc. 5th Int. Workshop Emotion Social Signals, Sentiment, Linked Open Data, Satellite of LREC 2014*, Reykjavik, Iceland, pp. 21-26, 2014.
- [74] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine learning*, vol. 38, no. 3, pp. 257-286, 2000.
- [75] X. Sun and P. K. Chan, "An Analysis of Instance Selection for Neural Networks to Improve Training Speed," in *13th IEEE International Conference on Machine Learning and Applications*, Detroit, MI USA, pp. 288-293, 2014.
- [76] S. Albelwi and A. Mahmood, "Analysis of instance selection algorithms on large datasets with Deep Convolutional Neural Networks," in *IEEE Long Island Systems, Applications and Technology Conference (LISAT), 2016*, Farmingdale, New York, pp. 1-5, 2016.
- [77] P. Hart, "The condensed nearest neighbor rule (Corresp.)," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 515-516, 1968.
- [78] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man and Cybernetics*, no. 3, pp. 408-421, 1972.
- [79] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 6, pp. 448-452, 1976.
- [80] D. B. Skalak, "Prototype and feature selection by sampling and random mutation hill climbing algorithms," presented at the Proceedings of the Eleventh International Conference on International Conference on Machine Learning, New Brunswick, NJ, USA, 1994.

- [81] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.
- [82] A. F. Abate, M. Nappi, D. Riccio, and G. Sabatino, "2D and 3D face recognition: A survey," *Pattern recognition letters*, vol. 28, no. 14, pp. 1885-1906, 2007.
- [83] C. Y. Suen and L. Lam, "Multiple Classifier Combination Methodologies for Different Output Levels," in *Multiple Classifier Systems*, Berlin, Heidelberg, Springer 2000, pp. 52-66, 2000.
- [84] M. Woźniak, M. Graña, and E. Corchado, "A survey of multiple classifier systems as hybrid systems," *Information Fusion*, vol. 16, pp. 3-17, 2014.
- [85] T. K. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, , vol. 16, no. 1, pp. 66-75, 1994.
- [86] L. Lam, "Classifier Combinations: Implementations and Theoretical Issues," in *Multiple Classifier Systems*, Berlin, Heidelberg, Springer 2000, pp. 77-86, 2000.
- [87] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.
- [88] X. Lu, Y. Wang, and A. K. Jain, "Combining classifiers for face recognition," in *Proceedings. IEEE International Conference on Multimedia and Expo*, Baltimore, MD, USA, 2003, vol. 3, pp. III-13-16 vol. 3, 2003.
- [89] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98-113, 1997.

- [90] A. Eleyan and H. Demirel, "PCA and LDA based face recognition using feedforward neural network classifier," in *International Workshop on Multimedia Content Representation, Classification and Security*, Istanbul, Turkey, Springer 2006, pp. 199-206, 2006.
- [91] M. A. Lone, S. Zakariya, and R. Ali, "Automatic face recognition system by combining four individual algorithms," in *IEEE International Conference on Computational Intelligence and Communication Networks (CICN) 2011*, Gwalior, India, 2011, pp. 222-226, 2011.
- [92] V. E. Liong, J. Lu, and G. Wang, "Face recognition using Deep PCA," in *9th IEEE International Conference on Information, Communications and Signal Processing (ICICS) 2013*, Tainan, Taiwan, 2013, pp. 1-5, 2013.
- [93] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," *arXiv preprint arXiv:1506.02078*, 2015.
- [94] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *University of Montreal*, vol. 1341, p. 3, 2009.
- [95] P. Ahlgren, B. Jarneving, and R. Rousseau, "Requirements for a cocitation similarity measure, with special reference to Pearson's correlation coefficient," *Journal of the American Society for Information Science and Technology*, vol. 54, no. 6, pp. 550-560, 2003.
- [96] A. Dragomir *et al.*, "Acoustic Detection of Coronary Occlusions before and after Stent Placement Using an Electronic Stethoscope," *Entropy*, vol. 18, no. 8, p. 281, 2016.

- [97] K. Katoh, K. Misawa, K. i. Kuma, and T. Miyata, "MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform," *Nucleic acids research*, vol. 30, no. 14, pp. 3059-3066, 2002.
- [98] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308-313, 1965.
- [99] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Maui, HI, USA, pp. 586-591, 1991.
- [100] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, , vol. 28, no. 12, pp. 2037-2041, 2006.
- [101] O. A. Vătămanu and M. Jivulescu, "Image classification using local binary pattern operators for static images," in *2013 IEEE 8th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, Romania,, pp. 173-178, 2013.
- [102] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," In: Tech. Rep. University of Toronto, 2009.
- [103] J. Bergstra *et al.*, "Theano: A CPU and GPU math compiler in Python," in *Proc. 9th Python in Science Conf*, Austin, Texas, pp. 1-7, 2010.
- [104] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning

- Curves," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, Buenos Aires, Argentina, pp. 3460-3468, 2015.
- [105] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *ArXiv e-prints* vol. 1512, ed, 2015.
- [106] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.
- [107] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, Atlanta, GA, USA, pp. 1058-1066, 2013.
- [108] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Artificial Intelligence and Statistics*, vol. 2, no. 3, pp. 562–570, 2015,
- [109] A. T. L. Cambridge. (1992-1994). *The ORL Database of Faces*. Accessed on: May. 1, 2017. [Online].: <http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>
- [110] A. Martinez and R. Benavente, "The AR face database," *Rapport technique*, vol. 24, 1998.
- [111] A. M. Martínez and A. C. Kak, "Pca versus lda," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, , vol. 23, no. 2, pp. 228-233, 2001.
- [112] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121-167, 1998.
- [113] D. W. Hosmer Jr and S. Lemeshow, *Applied logistic regression*. John Wiley & Sons, 2004.

- [114] M. Daneshzand, J. Almotiri, R. Baashirah, and K. Elleithy, "Explaining dopamine deficiency of spiking neurons based on quantum superposition theory," *NeuroQuantology*, vol. 13, no. 2, 2015.
- [115] M. Daneshzand, M. Faezipour, and B. D. Barkana, "Hyperbolic Modeling of Subthalamic Nucleus Cells to Investigate the Effect of Dopamine Depletion," *Computational intelligence and neuroscience*, vol. 2017, 2017.