CONVOLUTIONAL POLYNOMIAL NEURAL NETWORK
FOR IMPROVED FACE RECOGNITION

Dissertation

Submitted to

The School of Engineering of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree of

Doctor of Philosophy in Engineering

By

Chen Cui

UNIVERSITY OF DAYTON

Dayton, Ohio

August, 2017
CONVOLUTIONAL POLYNOMIAL NEURAL NETWORK FOR IMPROVED FACE RECOGNITION

Name: Cui, Chen

APPROVED BY:

Vijayan K. Asari, Ph.D.
Advisor Committee Chairman
Professor, Department of Electrical and Computer Engineering

Raul E. Ordonez, Ph.D.
Committee Member
Professor, Department of Electrical and Computer Engineering

Eric J. Balster, Ph.D.
Committee Member
Associate Professor, Department of Electrical and Computer Engineering

Muhammad Usman, Ph.D.
Committee Member
Associate Professor, Department of Mathematics

Robert J. Wilkens, Ph.D., P.E.
Associate Dean for Research and Innovation, Professor
School of Engineering

Eddy M. Rojas, Ph.D., M.A., P.E.
Dean
School of Engineering
ABSTRACT

CONVOLUTIONAL POLYNOMIAL NEURAL NETWORK FOR IMPROVED FACE RECOGNITION

Name: Cui, Chen
University of Dayton

Advisor: Dr. Vijayan K. Asari

Deep learning is the state-of-art technology in pattern recognition, especially in face recognition. The robustness of the deep network leads a better performance when the size of the training set becomes larger and larger. Convolutional Neural Network (CNN) is one of the most popular deep learning technologies in the modern world. It helps obtain various features from multiple filters in the convolutional layer and performs well in the hand written digits classification. Unlike the unique structure of each hand written digit, face features are more complex, and many difficulties are existed for face recognition in current research field, such as the variations of lighting conditions, poses, ages, etc. So the limitation of the nonlinear feature fitting of the regular CNN appears in the face recognition application. In order to create a better fitting curve for face features, we introduce a polynomial structure to the regular CNN to increase the non-linearity of the obtained features. The modified architecture is named as Convolutional Polynomial Neural Network (CPNN). CPNN creates a polynomial input for each convolutional layer and captures the nonlinear features for better classification. We firstly prove the proposed concept with MNIST handwritten database and compare the proposed CPNN with regular CNN. Then, different parameters in CPNN
are tested by CMU AMP face recognition database. After that, the performance of the proposed CPNN is evaluated on three different face databases: CMU AMP, Yale and JAFFE as well as the images captured in real world environment. The proposed CPNN obtains the best recognition rates (CMU AMP: 99.95%, Yale: 90.89%, JAFFE: 98.33%, Real World: 97.22%) when compared to other different machine learning technologies. We are planning to apply the state-of-art structures, such as inception and residual, to the current CPNN to increase the depth and stability as our future research work.
To my husband, who has stood by me

To my family, who has supported me

And to my friends, who have cheered me up.
ACKNOWLEDGMENTS

This is only the beginning of my journey to chase my academic ideals. Without the support of so many individuals, I would not have finished this dissertation by the end of my PhD program.

I still remember the day when I first talked with Dr. Vijayan K. Asari - he told me about the significance and spirit of academic research. I could hardly have completed this dissertation without his professional guidance, patient explanations, and continuous encouragement. I would like to thank all of the professors who taught me over my six years at University of Dayton; I sincerely appreciate the efforts of Dr. Eric J. Balster, Dr. Raul Ordonez and Dr. Muhammad Usman, who spent time commenting on my dissertation and attending my defense.

I would like to thank my friends and all of the members of Vision Lab who helped me review my dissertation and whose insights helped me improve the performance of my research. I really appreciate their help, and I am very touched by their support.

Thanks to my husband whose endless love gives me the courage to chase my research. With the support and inspiration of my parents, I had the chance to visit and study in the United States. They led me to the engineering world, and I will do my best to continue to achieve this dream.
# TABLE OF CONTENTS

ABSTRACT ................................................................................................. iii
DEDICATION .................................................................................................. v
ACKNOWLEDGMENTS ...................................................................................... vi
LIST OF FIGURES ........................................................................................... ix
LIST OF TABLES .............................................................................................. xi
NOMENCLATURE ........................................................................................... xii

I. INTRODUCTION ......................................................................................... 1
   1.1 Specific Objectives .................................................................................. 3

II. BACKGROUND ............................................................................................ 5
   2.1 Single Layer Perceptron (SLP) ............................................................... 6
   2.2 Multilayer Perceptron (MLP) ................................................................. 7
   2.3 Support Vector Machine (SVM) ............................................................. 9
   2.4 Radial Basis Function (RBF) ............................................................... 10
   2.5 Hopfield Network .................................................................................. 12

III. CONVOLUTIONAL NEURAL NETWORK (CNN/ConvNet) ...................... 14
   3.1 Architecture ......................................................................................... 14
   3.2 Forward Pass ....................................................................................... 17
   3.3 Back Propagation ................................................................................. 19
   3.4 Discussion ............................................................................................ 21
<table>
<thead>
<tr>
<th>IV. CONVOLUTIONAL POLYNOMIAL NEURAL NETWORK (CPNN)</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Polynomial Expansion</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Proposed Structure</td>
<td>25</td>
</tr>
<tr>
<td>4.3 Forward Pass</td>
<td>27</td>
</tr>
<tr>
<td>4.4 Back Propagation</td>
<td>28</td>
</tr>
<tr>
<td>4.5 Discussion</td>
<td>31</td>
</tr>
<tr>
<td>V. EXPERIMENTAL RESULTS</td>
<td>32</td>
</tr>
<tr>
<td>5.1 Performance with Face Databases</td>
<td>36</td>
</tr>
<tr>
<td>5.1.1 Parameter Evaluation</td>
<td>36</td>
</tr>
<tr>
<td>5.1.2 Performance Evaluation</td>
<td>40</td>
</tr>
<tr>
<td>5.2 Discussion</td>
<td>47</td>
</tr>
<tr>
<td>VI. CONCLUSION AND FUTURE WORKS</td>
<td>48</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>50</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.1 A basic structure of a neural network ................................................. 5
2.2 Illustration of SLP neural network .................................................... 6
2.3 Illustration of MLP neural network .................................................... 8
2.4 Illustration of SVM ........................................................................... 10
2.5 Illustration of RBF neural network .................................................... 11
2.6 Illustration of Hopfield network ....................................................... 12
3.1 Illustration of CNN ................................................................. 16
3.2 Illustration of average pooling and max pooling ................................. 17
4.1 An illustration of the polynomial expansion ........................................ 24
4.2 The modified framework for the proposed CPNN ............................... 26
5.1 The learning curve of CNN ............................................................ 33
5.2 The learning curve of the proposed CPNN ......................................... 34
5.3 The comparison between the learning curves of CNN and the proposed CPNN ......................................................... 35
5.4 An example from CMU AMP facial expression database ...................... 37
5.5 Accuracy of different numbers of filters in each convolutional layer ........ 38
5.6 The error rate with different number of poly-terms ............................. 40
5.7 An example of Yale database ........................................... 42
5.8 An example of the images in JAFFE .............................. 44
5.9 An example of the images captured in real world .......... 45
LIST OF TABLES

5.1 The comparison between the error rates of CNN and the proposed method . . . . . . . . 35
5.2 The accuracy with different number of filters in each convolutional layer . . . . . . . . 39
5.3 The average accuracy with different methods (CMU AMP) . . . . . . . . . . . . . . . 41
5.4 The average accuracy with different methods (Yale) . . . . . . . . . . . . . . . . . . . . 42
5.5 The average accuracy with different methods (JAFFE) . . . . . . . . . . . . . . . . . . . 44
5.6 The average accuracy with different methods (Real World) . . . . . . . . . . . . . . . 46
NOMENCLATURE

\( \alpha, \beta \) Parameter

\( \delta \) Difference

\( a, b \) Filter in the Convolutinal Layer

\( b_y, b_z \) Output of Sigmoid Function

\( d_y, d_z \) Output of Subsampling Layer

\( f \) Input of Fully Connected Layer

\( l_x \) Training Label

\( b \) Bias

\( b_f \) Final Output

\( f_y \) Output of Fully Connected Layer

\( g \) Gaussian Function

\( L \) Output of Loss Function

\( M \) Filter Size of \( a \)

\( N \) Image Size
$s$  Value of Summation

$w_{ij}$  Weight Node

$x_i$  Input Node

$y$  Value of Activation Function

$z_{ij}$  Weight Node

$\mu$  Mean

$\sigma$  Standard Deviation
CHAPTER I

INTRODUCTION

Face recognition has been developing for decades under the auspices of different research centers and individuals. Robust face recognition could have various applications in our daily life, such as securing restricted areas by using face information, logging into personal accounts without entering any character, and helping catch a person of interest by the police. With the help of face recognition, people can live in a more secure and safe environment. However, many difficulties exist in face recognition of current research field, such as the variations of lighting conditions, poses, aging, etc. In order to tackle multiple issues at once, pre-processing methods may be performed, like image enhancement. Also, features in different conditions need to be obtained to train the face recognition system.

Deep learning is wildly used in modern pattern recognition area as well as in face recognition. Traditional machine learning methods use the same type of layers to perform feature extraction and classification, whereas, deep learning methods include multiple structures to extract a better representation of the nonlinear input patterns. Convolutional Neural Network (CNN) is one of the most popular deep learning technologies that obtains a good performance in the hand written digit classification. In previous research work, we focus on the local contrast algorithm - Enhanced Local Binary Patterns (ELBP) [1] to tackle the feature extraction. However, the limitation of the local binary patterns [2, 3] based features is obvious when the size of training data becomes larger.
and larger. The classification is usually performed by the distance measurement algorithms which are sometimes not robust enough. Also, the features obtained from the ELBP are extracted directly without any back propagation to validate whether the extracted features are useful or not. Thus, this dissertation considers a development of a new deep learning strategy to improve the performance of the face recognition system.

CNN encodes the features obtained from the original input patterns with convolutions and classifies them using the regular machine learning layers. It requires three main layers which are the convolutional layer, subsampling/pooling layer, and fully connected layer to perform the feature extraction and classification. In the convolutional layer, it extracts the information from the input patterns with different trained filters. Then, the outputs are subjected to a subsampling/pooling layer to reduce the dimensionality. Finally, the reduced sub features are classified by a fully-connected layer. The feature extraction for handwritten digits recognition is very successful and widely used.

However, face features are much more complex than the features of handwritten digits. The shapes of different digits have a large variation, but the faces of different people do have a similar structure. Face images contain structural information as well as pixel information, so they must be represented by much higher dimensional feature sets. If only the linear hypothesis of the regular CNN is considered, the training system creates a linear boundary to classify the high dimensional feature sets. This linear hypothesis may cause under fitting issue which means the learning curve cannot be fit well to the training data.

It is difficult to obtain better accuracy when under fitting occurs in the training stage. If the training stage is not perfectly trained, then the testing accuracy is decreased. Also, the linear boundary is a “hard” boundary that performs the classification without flexibility. In order to create a better feature fitting curve for face recognition, we introduce a polynomial expansion concept to
each convolutional layer to accomplish the non-linearity for appropriate distinctions of the original input face images. This method is named as Convolutional Polynomial Neural Network (CPNN).

CPNN is a multilayer back propagation neural network based on Convolutional Neural Network (CNN) [4] to learn high dimensional features from a large quantity of training examples, which are then analyzed by a fully-connected network to perform the recognition. In the CPNN, there are three different types of functional layers: the convolutional layer, subsampling/pooling layer, and fully-connected layer. The polynomial expansion is applied to the input patterns before the convolutional layer so that the single data is in the format of a polynomial term. After that, the polynomial term is convolved with multiple filters in the convolutional layer, followed by a subsampling/pooling layer to reduce the dimensionality. These sub-features are sent to the fully connected layer to be classified. In other words, the original input patterns are decomposed by different filters with polynomial expansion, and the obtained features with non-linearity are classified by the nonlinear boundaries. Therefore, the proposed CPNN encodes the non-linearity to the regular CNN structure to obtain a better classification result.

1.1 Specific Objectives

The specific objectives involved in this dissertation research are the following:

- To analyze the extracted features from the regular CNN to determine the optimal structure for developing a new network architecture;
- To develop a polynomial expansion concept to accomplish non-linearity
- To develop an appropriate structure for the polynomial inputs and test the effectiveness of the proposed CPNN with MNIST database;
• To evaluate the performance of the regular CNN and the proposed CPNN, and compare the recognition results with the MNIST database;

• To select challenging face databases for effectiveness evaluation of the proposed CPNN;

• To evaluate the best parameters for face recognition using the proposed CPNN;

• To compare recognition results of the proposed CPNN and other machine learning techniques on different face databases.

This dissertation is organized as follows. In Chapter 2, a literature survey of different machine learning technologies is reviewed in detail. In Chapter 3, the regular CNN is reviewed step by step with explanation. Chapter 4 presents the main contribution of this dissertation that analyzes the proposed CPNN. Chapter 5 presents the experiments with the MNIST, CMU AMP, Yale, and JAFFE databases, as well as the images captured in real world environment. The conclusion and future works are included in Chapter 6.
CHAPTER II

BACKGROUND

Many techniques based on the artificial neural networks were developed in recent years. The conception of the model is based on the biological neural networks, or the brain systems of animals [5]. Although different networks may have different training structures and initial conditions, they all share some similar characteristics: a set of training parameters to lead the input vectors to a particular class, and the ability to process high dimensional input data. Figure 2.1 shows a basic structure of a neural network.

![Figure 2.1: A basic structure of a neural network](image)
A basic neural network is formed by different types of layers: input layer, hidden layer(s) (optional) and output layer. Mostly, the input layer contains the original input data, and each number in the input data is associated with part or all of the neurons in the hidden layer. Depending on different properties of the input data and effectiveness of the system structures, varying numbers of neurons are applied, as well as a number of hidden layers. In the final output layer, each input data transforms into a prediction vector that contains only binary values, therefore being assigned to a particular class based on the result from the prediction vector. Some popular machine learning algorithms are reviewed below.

2.1 Single Layer Perceptron (SLP)

Perceptron is a supervised learning technology of binary classification [6]. SLP [7, 8] creates linear boundaries to classify the input patterns. Multi-class classification is performed by a “one versus all” strategy. In other words, the label of a particular class is one and the labels for the other classes are zero. SLP contains one input layer and one output layer. Each input unit is fully-connected with the node(s) in the output layer. Each node in the output layer takes a weighted sum of all the input units. Figure 2.2 is an illustration of SLP.

![Figure 2.2: Illustration of SLP neural network](image-url)
In Figure 2.2, \( x_1 \) to \( x_n \) are the input units and \( w_1 \) to \( w_n \) are the corresponding weights. \( s \) and \( y \) are calculated as:

\[
s = \sum_{i=1}^{n} w_i x_i, \tag{2.1}
\]

\[
y = \begin{cases} 
1 & \text{if } \text{sgn}(s) \geq \text{threshold} \\
0 & \text{if } \text{sgn}(s) < \text{threshold}
\end{cases} \tag{2.2}
\]

where \( \text{sgn()} \) represents a sigmoid function. The corresponding weights are randomly selected at the beginning and trained from forward propagation and back propagation. For the forward propagation, it follows the weighted sum strategy and calculates the output for the output layer neurons. Then, the back propagation starts from the output layer to minimize the error function between the output and the actual label. SLP is a simple-structured neural network, so the training process is straight forward. Also, it is applied for simple testing tasks to prove the viability in some cases due to the quick processing stage. However, because of the complexity of the input data, the single layer classification is limited in its abilities to distinguish the input to different categories.

### 2.2 Multilayer Perceptron (MLP)

MLP [9, 10, 11] is an extension of SLP that includes one or more hidden layers in the structure. It creates multiple linear boundaries to assign input data to different groups. Each input unit is connected to the nodes of the first hidden layer, and each node of the first hidden layer is connected to the nodes of the second hidden layer. In each hidden layer, there are particular weight matrices to extract features from the output of the previous layer. Figure 2.3 is an illustration of one-hidden-layer MLP.
In Figure 2.3, $w_{11}$ to $w_{1l}$ are the related weights of the first input unit, $w_{n1}$ and $w_{nl}$ are the related weights of the $n$th input unit. $s_1, ..., s_l, t$ and $y$ are calculated as:

$$s_1 = \sum_{i=1}^{n} w_{i1} x_i, \quad (2.3)$$

$$s_l = \sum_{i=1}^{n} w_{il} x_i, \quad (2.4)$$

$$t = \sum_{i=1}^{l} z_i s_i, \quad (2.5)$$

$$y = \begin{cases} 1 & \text{if } \text{sgn}(s) \geq \text{threshold} \\ 0 & \text{if } \text{sgn}(s) < \text{threshold} \end{cases} \quad (2.6)$$

where $\text{sgn}()$ represents the sigmoid function. The forward propagation is similar to that of SLP, but the back propagation is slightly different. There is no ground truth information from the output in each hidden layer, thus the error calculation of this hidden layer is obtained from the connected hidden layer. The efficiency of the training process depends on the number of layers and neurons of each layer. It involves a time-consuming process when compared with SLP, but the accuracy payoff...
is more promising in most cases. Although the classification result is better than that of SLP, MLP still creates linear boundaries and combines the linear boundaries to create complex boundaries.

2.3 Support Vector Machine (SVM)

Unlike the perceptron neural networks, SVM [12, 13, 14] creates the maximum margin boundaries between two different classes [15]. In contrast with the hypothesis of the general neural networks, it calculates the similarity between the data point and landmarks. The maximum margin is obtained by the selected similarity function and defined by the support vectors. In other words, SVM creates a hyperplane to split the input patterns into two classes. As shown in Figure 2.4, 

\[ W^T x - b \geq 1 \]

defines the boundary of the first class, \( W^T x - b \leq -1 \) defines the boundary of the second class. \( x \) is one of the input vectors. \( \frac{b}{||W||} \) decides the offset of the hyperplane from the origin along the normal vector \( W \). There are a variety of choices of the similarity functions to create linear or non-linear boundaries depending on the properties of the input data. Multiclass SVM can classify more than two classes of input patterns. The strategy is to reduce the single multiclass problem into multiple binary classification problem.
SVM is a robust machine learning method for the binary class classification with a high recognition rate and low false positive rate. It is widely used even for modern recognition tasks. Although the multiclass SVM could be performed by multiple regular SVM, the limitation of reducing the error rates still appears when the number of classes is increased. The complexity of the training process rises as well.

2.4 Radial Basis Function (RBF)

RBF [16, 17, 18] is a particular neural network which applies a similarity computation in its hidden layers. It creates non-linear boundaries to classify inputs into different classes. Different from the weighted summation strategy in the hidden layer of MLP, RBF uses a series of non-linear functions to approximate the relationship between the inputs and the centroid of different clusters.
in the hidden layer. The weightage strategy is applied from the hidden layer to the output layer. Figure 2.5 shows an example of RBF.

![Figure 2.5: Illustration of RBF neural network](image)

In Figure 2.5, \( g_1 \) to \( g_l \) are the non-linear functions to approximate the distances between the input data and the centroid of clusters. There are multiple options of the non-linear functions, but Gaussian is the most popular one, shown as Equation 2.7.

\[
g_i = e^{-\frac{||x - \mu_i||}{2\sigma_i^2}}
\]  

(2.7)

where \( \mu_i \) is the centroid of the \( i \)th cluster, and \( \sigma_i \) is the standard deviation of the \( i \)th cluster. The final output \( y \) is the weighted summation of \( g_1 \) to \( g_l \). The similarity function facilitates increase in robustness compared to other regular neural networks, and the training stage of RBF is much more efficient than that of MLP. However, the proper non-linear functions need to be selected to tackle different recognition problems.
2.5 Hopfield Network

Hopfield [19, 20] network connects each input unit with a trainable weight matrix to obtain the local minimum of the input data. It is applied to the binary input data (noisy) to remove or reduce noise. The weight matrix is a square matrix because it calculates the relationships within the input data. Figure 2.6 shows an illustration of Hopfield network.

![Illustration of Hopfield network](image)

There are two enforced conditions for the Hopfield network:

- $w_{ii} = 0$, which means no self connection.
- $w_{ij} = w_{ji}$, which means connected symmetrically.

Several learning rules are developed for the Hopfield network, and the Hebbian learning rule - which was introduced by Donald Hebb in 1994 [21] - is one of the most basic rules. Equation 2.8 illustrates the weight matrix of the Hebbian rule.
\[ w_{ij} = \frac{1}{u} \sum_{\mu=1}^{u} m_i^{(\mu)} m_j^{(\mu)} \] (2.8)

where \( m_i^{\mu}, m_j^{\mu} \) are the \( i \)th, \( j \)th unit of the binary pattern \( \mu \), and \( u \) is the number of units of the related binary pattern. Each unit is updated by Equation 2.9.

\[
s_i = \begin{cases} 
1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i \\
-1 & \text{otherwise}
\end{cases} \quad (2.9)
\]

where \( s_j \) is the state of \( j \)th unit, and \( \theta_i \) is the threshold of \( i \)th unit.

Nevertheless, neural networks obtain a good performance in the object detection/recognition field, deep learning is the new trend in artificial intelligence. In the next chapter, one of the most popular techniques in our research area - CNN - is discussed.
CHAPTER III

CONVOLUTIONAL NEURAL NETWORK (CNN/ConvNet)

Deep learning is a class of machine learning based algorithms used to tackle high-dimensional data via several functional layers and complex structures [22]. CNN is one of the most popular deep learning algorithms for artificial intelligence. The concept of CNN is inspired by the animal visual system, which contains complex arrangements of cells called “receptive fields”, which are used to detect the low lighting and overlapping visual areas [5]. CNN is similar to the original neural networks, and it is also constructed to resemble number of neurons with trainable weights and biases. Moreover, CNN extracts different feature sets from the original data with multiple filters, encoding various properties in each layer.

3.1 Architecture

As with the neural networks mentioned in Chapter II, the units/nodes of each layer are fully-connected with their next related layer. For example, if there is a group of training images, and each image size is $28 \times 28$, then the number of weights to each node in the first hidden layer is $28 \times 28 = 784$. If the image size is much larger than the example above, then the dimension of the weight matrix will be tremendous. For CNN, it constrains the size of the inputs with multiple layers of filters before the fully-connected layer. The weight-sharing strategy minimizes the number of weights, encoding a variety of features to the output layer.
There are three types of layers in CNN: the convolutional layer, subsampling/pooling layer, and fully-connected layer [23]. For the convolutional layer, inputs convolve with a group of filters whose size (e.g. $5 \times 5$) is much smaller than the one of inputs, then go through a sigmoid function to increase the nonlinearity. The subsampling/pooling layer follows the convolution layer and reduces the dimensionality of the outputs from the convolutional layer. Usually, there is more than one pair of convolutional and subsampling/pooling layers. After the last subsampling/pooling layer, a fully-connected layer processes the obtained data to generate a prediction.

If the size of each input image is $28 \times 28$ (gray scale image), and there are two pairs of convolutional and subsampling/pooling layers, then the number of filters (size is $5 \times 5$) in the first convolutional layer is one, and the number of filters in the second convolutional layer is two. These inputs are assigned to 10 classes so that the details for this CNN are:

- First convolutional layer: the number of weights is $5 \times 5 \times 1 = 25$; the size of output is $24 \times 24 \times 1$ (no zero padding).

- First subsampling/pooling layer: the data after convolution is downsampled to $12 \times 12$.

- Second convolutional layer: the number of weights is $5 \times 5 \times 2 = 50$; the size of output is $8 \times 8$.

- Second subsampling/pooling layer: the data after convolution is downsampled to $4 \times 4 \times 2$.

- Fully-connected layer: the number of weights is $4 \times 4 \times 2 \times 10 = 320$.

The total number of weights for this CNN is $25 + 50 + 320 = 395$. Compared to the number of weights in the fully-connected neural networks (mentioned above), CNN has much fewer weights.

Figure 3.1 shows the structure of a regular CNN.
The convolutional layer is the main functional layer for CNN. The set of filters shared with the entire image are trainable. In the regular fully-connected networks, the system connects every unit in the input data, whereas the convolutional layer of CNN connects the local units, e.g. $5 \times 5$ pixel block if the input data is an image. The shared weights reduce the complexity of the connection of each local pixel. Also, each filter obtains one type of sub-features so that the high-dimensional data is decomposed into different feature vectors.

The subsampling/pooling layer provides the most significant contribution to reducing the complexity of the original data. There are many pooling algorithms \cite{24, 25} to downsample the data from the convolutional layer, such as average pooling and max pooling. Firstly, the input feature map is divided into several equal and non-overlapped sub-regions whose size is $2 \times 2$. Then, the pooling algorithm is applied to each sub-region. For average pooling, the total value of the sub-region is calculated, then divided by the size of the sub-region which is $2 \times 2 = 4$. The average value is the final representation of this sub-region. Max pooling finds the max value in the sub-region and represents the related region with the max value. Figure 3.2 illustrates the average pooling and max pooling.
3.2 Forward Pass

The details of feed forward process are explained here. First, the input image $x (N \times N)$ is convolved with filter $a_i (M \times M, i \text{ is the } i\text{th filter})$, as Equation 3.1

$$y_i = x \ast a_i$$  \hspace{1cm} (3.1)

Then, $y_i$ passes through a sigmoid function [26]:

$$by_i = \text{sigmoid}(y_i + \text{bias}_{1i})$$  \hspace{1cm} (3.2)

where $\text{bias}_{1i}$ is the bias vector for the filters in the first convolutional layer. For the first subsampling/pooling layer, $by_i$ is downsampled by average pooling:

$$dy_i(n, m) = \frac{1}{4} (by_i(2(n-1) + 1, 2(m-1) + 1) + by_i(2(n-1) + 1, 2(m-1) + 2)$$

$$+ by_i(2(n-1) + 2, 2(m-1) + 1) + by_i(2(n-1) + 2, 2(m-1) + 2))$$  \hspace{1cm} (3.3)
where \( \mathbf{d}y_i \) is the subsampled feature matrix, and \( n, m \) are the coordinates so that \( \mathbf{d}y_i \) is the downsampled matrix of \( \mathbf{b}y_i \). For the second convolutional layer, \( \mathbf{d}y_i \) is convolved with another set of filters \( \mathbf{b}_{ij} \):

\[
\mathbf{z}_j = \sum_i \mathbf{d}y_i \ast \mathbf{b}_{ij}
\]  
(3.4)

After that, the output is processed by the sigmoid function:

\[
\mathbf{b}z_j = \text{sigm}(\mathbf{z}_j + \mathbf{bias}_{2j})
\]  
(3.5)

where \( \mathbf{bias}_{2} \) is the bias vector for the filters in the second convolutional layer. Then, followed by the second subsampling layer:

\[
dz_j(n', m') = \frac{1}{4}(\mathbf{b}z_j(2(n' - 1) + 1, 2(m' - 1) + 1) + \mathbf{b}z_j(2(n' - 1) + 1, 2(m' - 1) + 2) \\
+ \mathbf{b}z_j(2(n' - 1) + 2, 2(m' - 1) + 1) + \mathbf{b}z_j(2(n' - 1) + 2, 2(m' - 1) + 2))
\]  
(3.6)

where \( \mathbf{d}z_j \) is the subsampled feature matrix, and \( n', m' \) are the coordinates. The outputs from the subsampling layer with different filters are concatenated to a vector to represent the original data.

\[
f = [\mathbf{d}z_1(:); \mathbf{d}z_2(:); ...]
\]  
(3.7)

For the fully-connected layer, each input unit is fully connected with the nodes of the output layer. Here, SLP is the system fully-connected layer. Consider the weight matrix of the output layer is \( \mathbf{w} \):

\[
\mathbf{f}y = \mathbf{w} \times f + \mathbf{f}bias
\]  
(3.8)
where \( fbias \) is the bias, then, \( f_y \) is also passed by a sigmoid function:

\[
bf = \text{sigm}(f_y)
\]  

(3.9)

### 3.3 Back Propagation

The forward pass begins from the first convolutional layer, whereas the back propagation begins from the fully-connected layer. The cost function of the fully-connected layer is:

\[
L = \frac{1}{2} \sum (bf - lx)^2
\]  

(3.10)

where \( lx \) is the label vector (contains only zeros and one). In order to minimize the cost function, we take the partial derivative of the cost function with respective weights, and the result of the partial derivative would be zero:

\[
\frac{\partial L}{\partial w(k, l)} = 0 \forall k = 1, 2, ..., T, l = 1, 2, ..., \text{length}(f)
\]  

(3.11)

where \( T \) is the number of classes. From Equation 3.11, the \( \delta_w \) is calculated as follows:

\[
\delta_w = \text{diag}(\text{diag}((bf - lx)bf^T)(1 - bf)^T)
\]  

(3.12)

The update of \( w \) is:

\[
\Delta = \delta_w f^T
\]  

(3.13)

So, the new weights are:

\[
w_{new} = w_{old} - \eta \Delta
\]  

(3.14)
where $\eta$ is the learning rate. The updated $f_{\text{bias}}$ is:

$$\Delta_{f_{\text{bias}}} = \text{mean}(\delta_w)$$

(3.15)

So, the new bias is:

$$f_{\text{bias}}_{\text{new}} = f_{\text{bias}}_{\text{new}} - \eta \Delta_{f_{\text{bias}}}$$

(3.16)

For the second subsampling/pooling layer, the chain rule would be:

$$\frac{\partial L}{\partial z_j(p, q)} = \frac{\partial L}{\partial b_{f(k)}} \frac{\partial b_{f(k)}}{\partial f_y(k)} \frac{\partial f_y(k)}{\partial b_{z_j}(p, q)} \frac{\partial b_{z_j}(p, q)}{\partial z_j(p, q)}$$

(3.17)

$$\frac{\partial L}{\partial b_{z_j}(p, q)} = \frac{\partial L}{\partial b_{f(k)}} \frac{\partial b_{f(k)}}{\partial f_y(k)} \frac{\partial f_y(k)}{\partial b_{z_j}(p, q)} = \sum_k \delta_w(k, 1)w(k, (j-1)N'^2 + \left\lceil \frac{p}{2} \right\rceil N' + \left\lceil \frac{q}{2} \right\rceil)$$

(3.18)

$$\frac{\partial b_{z_j}(p, q)}{\partial z_j(p, q)} = (1 - b_{z_j}(p, q))b_{z_j}(p, q)$$

(3.19)

where $p$ and $q$ represent the position of related vector, and $N'$ represents the size of $d_{z_j}$ ($N' \times N'$). Let the final representation of $\frac{\partial L}{\partial z_j(p, q)}$ be:

$$\alpha_{z_j}(p, q) = \frac{\partial L}{\partial z_j(p, q)}$$

(3.20)

For the second convolutional layer:

$$\beta_{dy_i} = \frac{\partial L}{\partial dy_i} = \sum_j \alpha_{z_j} \ast \text{flip}(b_{ij})$$

(3.21)
where \( \text{flip}(b_{ij}) \) represents the flip of \( b_{ij} \). The first convolutional layer and subsampling/pooling layer follow the same strategy as explained above, but the \( \sum_k \delta_w(k, 1) w(k, (j - 1) N'^2 + \lceil \frac{g}{2} \rceil N' + \lceil \frac{q}{2} \rceil) \) is replaced by \( \beta_{dy_i} \). After the calculations from all the convolutional and subsampling/pooling layers are completed, the increment of the filters in the convolutional layer must be determined:

\[
\Delta_{a_i} = x * \alpha_{y_i} \quad (3.22)
\]

\[
\Delta_{b_{ij}} = dy_i * \alpha_{z_j} \quad (3.23)
\]

So, the filters for the new training iteration would be:

\[
a_{new_i} = a_{old_i} + \theta_1 \Delta_{a_i} \quad (3.24)
\]

\[
b_{new_{ij}} = b_{old_{ij}} + \theta_2 \Delta_{b_{ij}} \quad (3.25)
\]

where \( \theta_1 \) and \( \theta_2 \) are learning rates. For the training part, combining the feed forward and back propagation together to train the system until the result from the cost function is almost zero.

### 3.4 Discussion

Neural network mimics the brain system with large numbers of nodes and trained weights to assign target inputs into different classes. It aims to connect each unit of inputs and weight them in a particular way which helps the entire system to learn the differences between various targets. However, the fully-connected structure is not efficient for all the cases, especially for complex inputs. Some experiments processed by neural networks with digit databases, such as NIST [27] and MNIST, obtain promising recognition results, whereas for the face recognition, the results with
such systems are not satisfactory. This is because that the types of features are too numerous to be connected. Some connections may be useless and cause confusion to a recognition system constructed by the regular neural networks. In the contrast of the regular neural networks, CNN is a structure that contains multiple neural networks used to decompose the original data into several sub-features. It also optimizes the relationship between the units in each local region, rather than connecting all of the units. Compared with the regular neural networks, as discussed in Section 3.1, the number of weights is much smaller and more manageable. Also, the shared weights save the storage space in the training process. Although CNN performs better and is more adaptive than the regular neural networks, we would like to explore one more step to enhance the obtained features with different orders. In the next chapter, the proposed method - Convolutional Polynomial Neural Network (CPNN) is introduced, and its effectiveness is demonstrated in the experimental results.
CHAPTER IV

CONVOLUTIONAL POLYNOMIAL NEURAL NETWORK (CPNN)

In the convolutional layer of CNN, input units are weighted linearly by filters. As mentioned in Chapter III, linear combination may cause some errors when processing complex data. We would like to apply the CPNN to face recognition by adding non-linearity to the CNN structure and introducing the polynomial expansion of the input patterns of each convolutional layer. The entire structure is distinct from the regular CNN.

4.1 Polynomial Expansion

The polynomial expansion is introduced to each convolutional layer, thereby leading to more exact nonlinear fitting of data dependencies. Unlike the simple number data, image data contains structural information as well as pixel information so that a 2D image is represented by much higher dimensionality feature sets. If only the linear hypothesis is considered, the training system creates a linear boundary to classify the high dimensionality feature sets. This linear hypothesis may cause an under fitting issue, which refers to a learning curve that does not fit well with the training data. Better accuracy was not achievable when the under fitting occurred in the training stage. The training stage is not perfectly trained, therefore the testing accuracy will be decreased. Also, the linear boundary is a “hard” boundary that performs the classification without flexibility. The polynomial expansion creates a nonlinear boundary that fits the nonlinear features much better than the linear boundary. It
also creates a more flexible fitting curve, hence, a better weights could be obtained from the training stage, which in turn facilitated the extraction of the useful information regarding the input image, and improved the accuracy in the testing stage. Consider that there are two hypotheses $y_1$ and $y_2$ which are linear and polynomial, the R-squared value is calculated by:

$$R^2_j = 1 - \sum_i (l - y_j)$$

(4.1)

where $l$ is the actual label of the input data and $j = 1, 2$. Then, the R-squared value is compared by F test:

$$F = \frac{df_2 - df_1}{df_1} \times \frac{R^2_2 - R^2_1}{R^2_1}$$

(4.2)

where $df_1$ and $df_2$ are the degree of freedom of $y_1$ and $y_2$. From the result of the F test, the p-value could be derived from the F table [28]. If the p value is larger than a critical value defined in the F table, $y_2$ is better than $y_1$, and vice versa. In [29], the curve fitting is improved when the order of the polynomial term is increased. However, a perfect fitting curve in training does not equate to perfect performance in testing. In order to obtain the best number of orders, the parameters must be evaluated later. Figure 4.1 illustrates the polynomial expansion.

![Figure 4.1: An illustration of the polynomial expansion](image)

Figure 4.1: An illustration of the polynomial expansion
Firstly, the inputs are calculated into $D$ different orders. These calculated inputs are the actual inputs for the first convolutional layer. In each order, there is a particular group of related filters to perform the convolution. Then, these filtered outputs are added together to obtain the polynomial expansion of the original input data. Compared to Equation 3.1, the polynomial expansion modify this equation as:

$$y_i = x * a_{i1} + bias_{1i1} + x^2 * a_{i2} + bias_{1i2} + ... + x^D * a_{iD} + bias_{1iD} \quad (4.3)$$

where $a_{i1}$ is the $i$th filter for the first order image, $a_{iD}$ is the $i$th filter for the $D$th order image, $bias_{1i1}$ is the bias for first order, and $bias_{1iD}$ is the bias for $D$th order. The need for the polynomial expansion is to accomplish the nonlinear fitting of the pixels’ relationships with their neighborhoods [30].

### 4.2 Proposed Structure

In the proposed method, the filters between each order are different; in other words, the filters are all independent and self-learned by iteration. With the help of the polynomial expansion in Equation 4.3, the framework is updated with the new architecture. Figure 4.2 illustrates the modified framework of the proposed method.
First, the input image is calculated in different orders, then, the convolution with multiple filters is applied to each order. The group of filters applied to each order is totally disjoint with those applied to the other orders. After the convolution, the filtered images obtained from the same order of filters are added together and downsampled using the average pooling method. The second convolutional layer follows right after the first subsampling/pooling layer. Before the second convolutional layer, the outputs from the first subsampling/pooling layer are calculated into different orders, then, apply the convolution with the appropriate filters. The filtered images are added together depending
on the order of filters, and are then downsampled by the average pooling method. Finally, the downsampled features are reshaped into one large vector to be classified by the fully-connected layer.

The mathematical steps are explained in the next sections.

4.3 Forward Pass

Given that the size of the input image $x$ is $N \times N$, the output of the first convolutional layer is calculated with Equation 4.3. Then, $y$ passes through a sigmoid function:

$$by_i = \text{sigm}(y_i)$$ (4.4)

The sigmoid function encodes the nonlinearity to the output matrix, but the result is different from the one by the polynomial function (the addition of different orders’ inputs with respective weights). In other words, the sigmoid function is more like a control function to limit the trend and value of the inputs. After the sigmoid function, the result is downsampled by average pooling:

$$dy_i(n, m) = \frac{1}{4} (by_i(2(n - 1) + 1, 2(m - 1) + 1) + by_i(2(n - 1) + 1, 2(m - 1) + 2)$$
$$+ by_i(2(n - 1) + 2, 2(m - 1) + 1) + by_i(2(n - 1) + 2, 2(m - 1) + 2))$$ (4.5)

Before the second convolutional layer, the $D$ orders of $dy_i$ must be obtained. Then, calculate the convolution with their respective filters $b_{ij1}$ to $b_{ijD}$:

$$z_j = \sum_i (dy_i * b_{ij1} + \text{bias}_{2i1} + dy_i^2 * b_{ij2} + \text{bias}_{2i2} + ... + dy_i^D * b_{ijD} + \text{bias}_{2iD})$$ (4.6)

where $\text{bias}_{2i1}$ is the bias for first order, and $\text{bias}_{2iD}$ is the bias for $D$th order. Then, $z_j$ passes the sigmoid function:
\[ \mathbf{b}_z_j = \text{sigm}(\mathbf{z}_j) \]  

Next, \( \mathbf{b}_z_j \) is downsampled by the second subsampling/pooling layer:

\[
dz_j(n', m') = \frac{1}{4}(\mathbf{b}_z_j(2(n' - 1) + 1, 2(m' - 1) + 1) + \mathbf{b}_z_j(2(n' - 1) + 1, 2(m' - 1) + 2) \\
+ \mathbf{b}_z_j(2(n' - 1) + 2, 2(m' - 1) + 1) + \mathbf{b}_z_j(2(n' - 1) + 2, 2(m' - 1) + 2))
\]  

All of the \( \mathbf{dz}_j \) are reshaped and concatenated to one vector to serve as the inputs for the fully-connected layer:

\[
f = [\mathbf{dz}_1(:); \mathbf{dz}_2(:); ...]
\]  

SLP is applied to be the fully-connected neural network so that:

\[
\mathbf{f} = \mathbf{w} \times \mathbf{f} + \mathbf{f}_{\text{bias}}
\]  

Followed by the sigmoid function:

\[
\mathbf{b}_f = \text{sigm}(\mathbf{f}_y)
\]  

4.4 Back Propagation

Similarly, as discussed in Section 3.3, the cost function of the fully-connected layer is:

\[
L = \frac{1}{2} \sum (\mathbf{b}_f - \mathbf{l}_x)^2
\]  

The partial derivative of the cost function with respective weights for the fully-connected layer must be zero:
\[ \frac{\partial L}{\partial w(k,l)} = 0 \forall k = 1, 2, ..., T, l = 1, 2, ..., \text{length}(f) \]  

\[ (4.13) \]

So, we can obtain the \( \delta_w \) from the partial derivative function as:

\[ \delta_w = \text{diag}(\text{diag}((bf - lx)bf^T)(1 - bf)^T) \]  

\[ (4.14) \]

With the help of \( \delta_w \), the update of \( w \) would be:

\[ \Delta = \delta_w f^T \]  

\[ (4.15) \]

And the new weights are updated depending on \( \Delta \) with a learning rate \( \eta \):

\[ w_{\text{new}} = w_{\text{old}} - \eta \Delta \]  

\[ (4.16) \]

Also, the bias of the fully-connected layer must be updated as:

\[ \Delta_{\text{bias}} = \text{mean}(\delta_w) \]  

\[ (4.17) \]

And the new bias is calculated as:

\[ fbias_{\text{new}} = fbias_{\text{new}} - \eta \Delta_{\text{bias}} \]  

\[ (4.18) \]

As explained in Equations 4.12 to 4.18, there is no difference between the CNN and CPNN processes for the back propagation of the fully-connected layer. The main differences appear in the convolutional layer in the feed forward (Equation 4.3 and Equation 4.6) and back propagation. For the second subsampling/pooling layer, the chain rule (also the same as the one in CNN) is described as:
\[ \frac{\partial L}{\partial z_j(p, q)} = \frac{\partial L}{\partial bf(k)} \frac{\partial bf(k)}{\partial fy(k)} \frac{\partial fy(k)}{\partial bz_j(p, q)} \]  

(4.19)

\[ \frac{\partial L}{\partial bz_j(p, q)} = \frac{\partial L}{\partial bf(k)} \frac{\partial bf(k)}{\partial fy(k)} \frac{\partial fy(k)}{\partial bz_j(p, q)} = \sum_k \delta_w(k, 1) w(k, (j - 1) N'^2 + \left\lfloor \frac{p}{2} \right\rfloor N' + \left\lfloor \frac{q}{2} \right\rfloor) \]  

(4.20)

\[ \frac{\partial bz_j(p, q)}{\partial z_j(p, q)} = (1 - bz_j(p, q)) bz_j(p, q) \]  

(4.21)

Let \( \frac{\partial L}{\partial z_j(p, q)} \) be:

\[ \alpha_{z_j}(p, q) = \frac{\partial L}{\partial z_j(p, q)} \]  

(4.22)

Because \( dy_i \) is constructed by \( D \) in different orders, the chain rule of the second convolutional layer is obtained from these \( D \) orders:

\[ \beta_{dy_i} = \frac{\partial L}{\partial dy_i} + \frac{\partial L}{\partial dy_i^2} + \ldots + \frac{\partial L}{\partial dy_i^D} = \sum_j \alpha_{z_j} * \text{flip}(b_{ij1}) + \sum_j \alpha_{z_j} * \text{flip}(b_{ij2}) + \ldots + \sum_j \alpha_{z_j} * \text{flip}(b_{ijD}) \]  

(4.23)

The first subsampling/pooling layer follows the same steps as the one described in the second subsampling/pooling layer. For the second convolutional layer, \( \sum_k \delta_w(k, 1) w(k, (j - 1) N'^2 + \left\lfloor \frac{p}{2} \right\rfloor N' + \left\lfloor \frac{q}{2} \right\rfloor) \) is replaced by \( \beta_{dy_i} \). After the back propagation of all the subsampling/pooling and convolutional layers, the increment of the filters in each convolutional layer must be calculated:

\[ \Delta_{a_{i,h}} = x^h * \alpha_{y_i} \]  

(4.24)

\[ \Delta_{b_{ij,h}} = dy_i^h * \alpha_{z_j} \]  

(4.25)
where $h$ indicates the order number. The filters for the new training iteration would be:

$$
a_{ih}^{new} = a_{ih}^{old} + \theta_{1h} \Delta a_{ih} \quad (4.26)$$

$$
b_{ijh}^{new} = b_{ijh}^{old} + \theta_{2h} \Delta b_{ijh} \quad (4.27)$$

For the training process, it follows the structures of the feed forward and back propagation to train the system by iterations until the error (the result of the cost function) converges to a number almost equal to zero.

### 4.5 Discussion

Unlike the regular CNN, the proposed method encodes the nonlinear properties to the inputs of each layer. It calculates the relationships between the pixels in each local region, as well as the nonlinearity of each local region. The weights sharing strategy only appears in the same order of each convolutional layer. This is because each unit from each order has inner-connections with respective neighbor units. In contrast, the features of the units between different orders are not connected by the filters so that the features obtained by different orders are fused by adding the features together. The update of the filters in different orders is calculated separately depending on the obtained increments in different orders. The experimental results in the next chapter compares the effectiveness of CNN and the proposed method.
To test the effectiveness of the proposed method, we first test its performance with the MNIST database of handwritten digits (0 to 9). There are 60,000 training images and 10,000 testing images in this database. The size of each digit is normalized and centered in a fixed-size image ($28 \times 28$). In our experiments, all the 60,000 training images and 10,000 testing images are used to train and test respectively. First, we test the result with the regular CNN. For the first and second convolutional layers, we set 6 and 20 filters ($5 \times 5$) respectively. Figure 5.1 shows the mean squared error (MSE) of each iteration in the training process.
For the proposed CPNN, we also use all of the images in the training set to train the proposed system, and all of the images in the testing set to test the accuracy. Two orders are applied to each convolutional layer. The number of filters is set at 6 and 20 for the first and second convolutional layers. Figure 5.2 depicts the MSE of each iteration in the proposed training process.
In order to compare CNN and the proposed method, the learning curves are plotted together. Figure 5.3 illustrates the comparison between the learning curves obtained from CNN and the proposed method.
Figure 5.3: The comparison between the learning curves of CNN and the proposed CPNN

It is clear that the MSE of each iteration for the proposed method is smaller than that of the CNN. Table 5.1 contains the recognition results obtained by CNN and the proposed method on the MNIST database.

Table 5.1: The comparison between the error rates of CNN and the proposed method

<table>
<thead>
<tr>
<th>Method</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>0.98%</td>
</tr>
<tr>
<td>CPNN</td>
<td>0.96%</td>
</tr>
</tbody>
</table>
5.1 Performance with Face Databases

The error rate of the proposed method is less significant than that of CNN with 10,000 testing images. The training and testing processes were run for multiple times, and the training images were ordered randomly. For the error rate of CNN, it ranges from 0.98% to 1.13%, whereas, the error rate of the proposed method is roughly 0.96%. This result indicates that the proposed method is more stable than CNN when the training order of the images are selected randomly. The comparison between the regular CNN and the proposed method demonstrates the effectiveness of using the proposed method to classify the handwritten digits.

To test the performance of the proposed system with face databases, three well-known databases and the images captured in real world environment were selected for evaluation. The three databases are listed as follows: CMU AMP facial expression [31], Yale [32] and Japanese Female Facial Expression (JAFFE) [33].

5.1.1 Parameter Evaluation

Before the performance evaluation, some parameters of the proposed algorithm were adjusted. In order to obtain the best parameters, several tests were conducted with CMU AMP facial expression database. Five images per individual were used for training, and the rest of the images (70 per person) were used for testing. For each evaluation, training images were randomly selected from 75 images of each individual. Figure 5.4 illustrates an example from the database.
The average recognition accuracy based on the same group of parameters is calculated from multiple mutually disjoint tests. The evaluated types of parameters are listed as follows:

- Number of convolutional and subsampling layers
- Number of filters of each convolutional layer
- Best order number of polynomial inputs

To determine the best number of convolutional and subsampling layers, the analysis began with two pairs of convolutional and subsampling layers. For the two pairs’ network, 12 filters were applied for the first convolutional layer, and 40 filters were applied for the second layer. For the three pairs’, 12, 40 and 80 filters were applied for the first, second, and third convolutional layer respectively. Assuming that there are $n$ pairs of convolutional and subsampling layers in one network, the number of filters in each convolutional layer would be: 12, 40, 80, 160, ..., $(n - 1) \times 40$. The average error rate of the two pairs is 0.35% and the three pairs’ error rate is 0.27%. These average
error rates are calculated from 20 tests where training and testing images are randomly selected. The standard deviation of error rates is much higher when the number of pairs is increased to four, therefore the system may not be stable with more convolutional and subsampling layers. To avoid the high standard deviation and obtain a better recognition rate, we choose three pairs to evaluate the other tests.

To set the best number of filters of each convolutional layer, we test multiple groups of parameters based on their layer order. Although we would like to test the performance affected by each convolutional layer only, the results show that there is a hidden effect between the connected layers. Figure 5.5 shows the comparison between different numbers of filters of each convolutional layer.

![Accuracy with different numbers of filters in different convolutional layer](image)

Figure 5.5: Accuracy of different numbers of filters in each convolutional layer
In Figure 5.5, it is difficult to distinguish the plots from different numbers of filters. Therefore, we calculated the average accuracy (Table 5.2) of the 20 rounds in each test. f-s-t represents the number of filters in the first, second and third convolutional layer. According to Table 5.2, we choose 12, 45, 70 filters in the first, second and third convolutional layer.

Table 5.2: The accuracy with different number of filters in each convolutional layer

<table>
<thead>
<tr>
<th>f-s-t</th>
<th>Average Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-20-40</td>
<td>99.48%</td>
</tr>
<tr>
<td>10-15-40</td>
<td>99.67%</td>
</tr>
<tr>
<td>10-20-40</td>
<td>99.68%</td>
</tr>
<tr>
<td>10-25-40</td>
<td>99.62%</td>
</tr>
<tr>
<td>10-30-40</td>
<td>99.63%</td>
</tr>
<tr>
<td>10-25-50</td>
<td>99.32%</td>
</tr>
<tr>
<td>12-30-60</td>
<td>99.60%</td>
</tr>
<tr>
<td>12-40-60</td>
<td>99.68%</td>
</tr>
<tr>
<td>12-35-70</td>
<td>99.81%</td>
</tr>
<tr>
<td>12-40-70</td>
<td>99.85%</td>
</tr>
<tr>
<td>12-45-70</td>
<td><strong>99.91%</strong></td>
</tr>
<tr>
<td>12-40-80</td>
<td>99.73%</td>
</tr>
<tr>
<td>12-45-80</td>
<td>99.83%</td>
</tr>
<tr>
<td>12-50-80</td>
<td>99.70%</td>
</tr>
</tbody>
</table>

After the best number of filters was determined, we tested the optimal order number of the polynomial inputs. Because the comparison between the one-order input and the two-order input was obtained by tests using the MNIST database, order number is started from two. Figure 5.6 illustrates the comparison among different order numbers of polynomial inputs after 10 runs.
From Figure 5.6, the number of poly-terms is not the more the better. More terms may obtain a lower training error, whereas, the testing accuracy may not be increased. This is because of the over fitting for the hyper dimensionality hypothesis. Consider about obtaining a better accuracy and avoiding the over fitting, the tests were ceased once the number of poly terms equaled 5, and we found 4 to be the best number of poly terms. Finally, the parameters for the performance evaluation were as follows: three conv + sub layers, 12-35-70 convolutional filters and 4 poly terms.

5.1.2 Performance Evaluation

The performance evaluation was tested using three different databases: CMU AMP, Yale and JAFFE, as well as the images captured in real world environment. All images were re-sized to 64 by 64.
CMU AMP

The first database is the CMU AMP facial expression database, which consists 75 images for each of 13 individuals. This database contains various facial expressions, such as smiling, crying, scared, angry, etc., with normal lighting conditions. These images were taken on the same day, and all the participants are male individuals. In our experiment, five images were randomly selected per person to be used for training, and all the other images for each person were for testing; thus there were 65 images for training and 910 images for testing. Table 5.3 presents the comparison among different machine learning technologies after 10 runs.

Table 5.3: The average accuracy with different methods (CMU AMP)

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLP</td>
<td>84.41%</td>
</tr>
<tr>
<td>MLP</td>
<td>72.35%</td>
</tr>
<tr>
<td>RBF</td>
<td>77.61%</td>
</tr>
<tr>
<td>SVM</td>
<td>98.48%</td>
</tr>
<tr>
<td>CNN</td>
<td>98.74%</td>
</tr>
<tr>
<td>CPNN</td>
<td>99.95%</td>
</tr>
</tbody>
</table>

From Table 5.3, it appears that the proposed CPNN obtained the best recognition rate at 99.95%. Compared to CNN, the recognition rate of the proposed CPNN is 1.21% higher. Compared to other regular machine learning methods, the performance of the proposed method is much better, especially in comparison to SLP, MLP, and RBF.

Yale

The second database is Yale database. There are 15 individuals, and each individual has 11 images with different expressions and illuminations. All the participators are males and all the
images are taken in the same day. For Yale database, we randomly apply 5 images per individual for training and all the other images per individual for testing, therefore there are 75 training images and 90 testing images. Figure 5.7 shows an example from Yale database.

![Figure 5.7: An example of Yale database](image)

The face images are the front images that include multiple expressions such as, happy, angry, sad, neutral, etc. The lighting conditions vary slightly depending on the angle of the light source. Also, some images have shadows in the background due to the lighting source. In order to increase the difficulty of the face recognition, Yale database collected images of the same individual with and without eyeglasses. Table 5.4 presents the rate of accuracy among different methods used on the Yale database.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLP</td>
<td>66.89%</td>
</tr>
<tr>
<td>MLP</td>
<td>67.56%</td>
</tr>
<tr>
<td>RBF</td>
<td>64.22%</td>
</tr>
<tr>
<td>SVM</td>
<td>83.33%</td>
</tr>
<tr>
<td>CNN</td>
<td>89.78%</td>
</tr>
<tr>
<td>CPNN</td>
<td><strong>90.89%</strong></td>
</tr>
</tbody>
</table>
With several difficulties of the images in Yale database, the accuracy is lower than that seen in recognition of the CMU AMP database. However, the proposed CPNN exhibited the best accuracy among different machine learning technologies. The recognition rate of the proposed CPNN is 1.11% higher than that of the regular CNN, which is similar to the performance disparity with the CMU AMP database. The CPNN performance is significantly better than SVM in the Yale database, particularly as compared to the results using the CMU AMP database. Also, the results of the regular machine learning methods are less competitive than that of the proposed CPNN.

**Japanese Female Facial Expression**

The third database is Japanese Female Facial Expression (JAFFE) which contains 10 different individuals. Different from the previous two databases, all the participants are females with different expressions. The face images include the entire head and neck information as well as part of the shoulder information. The background is very simple, but with some textures. The expressions in JAFFE vary more extremely than the expressions in the previous two databases. For each female, there are 20 images in total. In the training stage, 5 images per female were randomly selected for training. In the testing stage, all the other images per female were used for testing. There are totally 50 training images and 150 testing images to be evaluated by various methods. Figure 5.8 illustrates an example of images in JAFFE.
Table 5.5 shows the recognition results with different algorithms via JAFFE.

Table 5.5: The average accuracy with different methods (JAFFE)

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLP</td>
<td>76.40%</td>
</tr>
<tr>
<td>MLP</td>
<td>86.13%</td>
</tr>
<tr>
<td>RBF</td>
<td>83.60%</td>
</tr>
<tr>
<td>SVM</td>
<td>96.13%</td>
</tr>
<tr>
<td>CNN</td>
<td>97.60%</td>
</tr>
<tr>
<td>CPNN</td>
<td>98.33%</td>
</tr>
</tbody>
</table>

As evinced in Table 5.5, the proposed CPNN obtained the best performance with a recognition rate 0.73% higher than that of the regular CNN. Compared to other methods, the accuracy of the proposed CPNN is far superior. The previous two databases demonstrate the capacity of the proposed method with the face recognition in male face images, and the JAFFE database tests its
proficiency with female face images. A more complex environment is necessary to test the robustness of the proposed CPNN; thus, a fourth experiment which evaluates these methods in a real world environment was performed.

**Testing in Real World**

The last experiment is performed with the images captured in real world. There are 9 individuals including 6 males and 3 females involved in this experiment, and each individual has around 20 images in different lighting conditions and poses. These images are captured from video sequences taken in the lobby of a working center (The Vision Lab). People entered the lobby and walked to the working area normally; in other words, they were not looking at the camera while being captured. Figure 5.9 exemplifies the images captured in real world.

![Figure 5.9: An example of the images captured in real world](image)

From Figure 5.9, we could see the toughness and multiple issues. The camera position was fixed, therefore the sizes of the captured face images exhibit significant variation between each other due to the varying distance from the face to the camera. The lighting condition also varied when the same person was captured walking in different positions, so the contrast among different images of the same person was noticeable. Another issue was the different poses of the faces from image to image. We may capture some frontal images as well as the profile images. Also, some images were blurred due to the movement of the captured individual. Even though the camera position was fixed, the face detection is performed frame by frame; thus, the background information may still differ from frame to frame. In our experiment, 10 images per individual were randomly selected for training, while the other 10 images per individual were used in testing. There are 90 training images and 90 testing images in total. Table 5.6 compares the accuracy of different methods in processing the images captured in real world environment.

Table 5.6: The average accuracy with different methods (Real World)

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLP</td>
<td>73.89%</td>
</tr>
<tr>
<td>MLP</td>
<td>75.67%</td>
</tr>
<tr>
<td>RBF</td>
<td>63.55%</td>
</tr>
<tr>
<td>SVM</td>
<td>92.33%</td>
</tr>
<tr>
<td>CNN</td>
<td>94.67%</td>
</tr>
<tr>
<td>CPNN</td>
<td>97.22%</td>
</tr>
</tbody>
</table>

The performance of the proposed CPNN is the best among these methods. Compared to CNN, the accuracy of the proposed CPNN is 2.55% higher. Among the regular machine learning methods, SVM obtains the highest accuracy, but the performance of SVM is much worse than that of the
proposed CPNN. Even though, the recognition test with the images captured in real world environment is demonstrably more difficult than tests performed on database images, the proposed CPNN obtained a promising recognition results.

5.2 Discussion

Due to the complexity of the deep learning neural networks, evaluation of different parameters in the proposed CPNN took an extended period of time. The nonlinear curve fitting obtains a better feature representation of the original face images, whereas, the processing stage is tremendously time-consuming. In order to obtain the best structure for face recognition, both of the under fitting and over fitting issues must be avoided. The parameter evaluation illustrates that the sheer number of filters in each convolutional layer and the number of polynomial terms are directly proportional to the network’s accuracy. In other words, fewer parameters and a simpler network may cause under fitting; however, more parameters in a more complex network may cause over fitting. To obtain the optimal parameters, we spent most of the time to test via various choices. Even though these parameters are originated from the CMU AMP database only, they still work well with other databases for face recognition. From Table 5.3 to Table 5.6, it can be seen that the proposed CPNN obtain the best accuracy among different machine learning methods for face recognition with all different databases. Compared to the classic machine learning algorithms, the proposed CPNN improved the accuracy dramatically. While SVM can also be robust in various conditions. However, the accuracy illustrates the limitation of the performance. The comparison between the proposed CPNN and the regular CNN demonstrates the improvement of the recognition rate, which aligns with the performance in respect of the MNIST handwritten database. The proposed CPNN can be applied to other areas as well, such as the object detection and other pattern recognition tasks. The suitable parameters for these tasks are obtainable via the parameter test.
CHAPTER VI

CONCLUSION AND FUTURE WORKS

In this dissertation, we reviewed several machine learning technologies and proposed a new method based on the Convolutional Neural Network (CNN) which is called Convolutional Polynomial Neural Network (CPNN) for face recognition. Unlike the simple number data, image data possesses higher dimensionality and more nonlinear properties. In order to increase the non-linearity and perform a better decision region fitting, the polynomial expansion is introduced to the regular CNN. The polynomial expansion extends the input patterns for each convolutional layer with different polynomial terms, therefore the linear curve fitting is modified by the nonlinear curve fitting technique. We first proved our concept with MNIST database, compared the results of CPNN from the loss function of the training stage and the recognition accuracy of the testing stage with the regular CNN. Then, the proposed CPNN was tested with three face databases: CMU AMP, Yale and JAFFE, as well as the images captured in real world environment. There are many challenges presented by these face images, such as different expressions, lighting conditions, poses, motion blur, etc. The recognition rates of the proposed CPNN are 99.95%, 90.89%, 98.33% and 97.22% respectively. Despite the challenging image sets, the proposed CPNN still obtains the best accuracy among different methods via different face databases and the images captured in real world environment.
In our future effort, we will concentrate on improving the accuracy of face recognition with the pre-processing technologies, such as the image enhancement. Also, the current deep learning neural networks need to be improved with multiple structures. In order to increase the depth and improve the performance of the current network, future inclusion may consist of adding the state-of-art architectures, such as the residual [34] and the inception [35], to the proposed CPNN to strive for further robustness and more accuracy network. The proposed CPNN could be applied to other areas, such as object detection and other pattern recognition tasks. More parameter tests may be performed to obtain the optimal sets for these areas.
BIBLIOGRAPHY


