An experimental study of human action recognition system using machine learning

Bi, T.

Award date:
2016
An Experimental Study of Human Action Recognition System Using Machine Learning

Master Thesis
by
Tianyu Bi
0978479

Master of Science in Embedded Systems

Supervisors:
Tanir Ozcelebi (TU/e)
Dragan Sekulovski (Philips Lighting)

Committee:
Mike Holenderski (TU/e)
Nicola Zannone (TU/e)

Eindhoven August 10, 2016
Abstract

Human action recognition is an important branch of computer vision and is getting increasing attention from researchers. It has been applied in many areas including surveillance, healthcare, sports and computer games. This thesis focuses on designing a human action recognition system for a human interaction dataset. Literature research is conducted to determine suitable algorithms for action recognition.

In this thesis, three machine learning models are implemented as the classifiers for human actions. An image processing method and a projection-based feature extraction algorithm are presented to generate training examples for the classifier. The action recognition task is divided into two parts: 4-class human posture recognition and 5-class human motion recognition. Classifiers are trained to classify input data into one of the posture or motion classes. Performance evaluations of the classifiers are carried out to assess validation accuracy and test accuracy for action recognition.

In the experiment, key parameters of machine learning models are tested to find the optimal configuration. Modifications are conducted on the neural network model to implement it in a distributed way. The modified classifier shows a high performance for human action recognition. The architecture designs for the centralized and distributed recognition systems are presented. Later these designed architectures are simulated on the sensor network to evaluate feasibility and recognition performance. Overall, the designed classifiers show a promising performance for action recognition. The distributed recognition system can be improved for better performance in the future work.

Key words: Human Action Recognition, Machine Learning, Neural Networks.
Acknowledgements

First of all, I would like to thank Tu/e and Philips Lighting for the opportunity to study and work in the area of embedded systems. I would like to express my heartfelt gratitude to my supervisors Dr. Tanir Ozcelebi and Dr. Dragan Sekulovski, for their encouragement and great help. I would like to thank Dr. Ramon Clout, for his insight and guidance. I would like to thank my colleagues and classmates, Mengye Li, Pingping Lin and Di Liu, for their kindly support. Next, I would like to express my gratitude to Xiaochen Huai and Yiting Xu for their help during my master project. I would like to thank to Dr. Mike Holenderski and Dr. Nicola Zannone for being my exam committee. Finally, I am particularly indebted to thank my parents for their love and moral support.

Tianyu Bi

August 10, 2016
# Table of Contents

Abstract ........................................................................................................................................... 2  
Acknowledgements ....................................................................................................................... 3  
List of abbreviations .................................................................................................................... 7  
List of Figures .................................................................................................................................. 8  
List of Tables .................................................................................................................................... 10  
1 Introduction .................................................................................................................................. 11  
  1.1 Background ............................................................................................................................... 11  
  1.2 Natural Elements ...................................................................................................................... 12  
  1.3 Motivation .................................................................................................................................. 13  
  1.4 Research Questions .................................................................................................................. 13  
  1.5 Thesis layout ............................................................................................................................ 14  
2 Problem Description and Related Works .................................................................................... 15  
  2.1 Problem Statement .................................................................................................................... 15  
  2.2 Machine Learning Models ....................................................................................................... 16  
  2.3 Problem Formalization ............................................................................................................ 18  
    2.3.1 Mapping Function ............................................................................................................... 18  
    2.3.2 Machine Learning Model ................................................................................................... 19  
    2.3.3 Hypothesis and logistic function ....................................................................................... 20  
    2.3.4 Cost Function .................................................................................................................... 21  
    2.3.5 Gradient Descent ................................................................................................................ 21  
    2.3.6 Stochastic Gradient Descent ............................................................................................ 21  
    2.3.7 Artificial Neural Networks ............................................................................................... 22  
    2.3.8 A performance measure: accuracy ..................................................................................... 23  
  2.4 Related work ............................................................................................................................ 23  
3 Human Action Recognition System ............................................................................................ 25  
  3.1 An overview of data flow .......................................................................................................... 25  
  3.2 Data preprocessing .................................................................................................................... 26  
  3.3 Feature extraction ...................................................................................................................... 28  
    3.3.1 Posture separation method ............................................................................................... 28  
    3.3.2 Motion History Image ....................................................................................................... 29  
    3.3.3 Projection-based feature extraction ................................................................................... 30  
  3.4 Implementation of machine learning methods ......................................................................... 33
3.4.1 Determine the model of MLR................................................................. 33
3.4.2 Determine the model of SGD-MLR...................................................... 34
3.4.3 Determine the model of ANN.............................................................. 35
3.4.4 Implementation of optimization techniques ....................................... 36
3.4.5 Implementation of TensorFlow platform ............................................ 36
3.5 Modification for Neural Networks .......................................................... 37
3.5.1 Modifications..................................................................................... 38

4 Architecture Designs.................................................................................. 41
4.1 Functional Requirements........................................................................ 41
4.2 Extra-functional Requirements............................................................... 42
4.3 Architecture 1: Centralized classifier on Intel i5 platform ..................... 42
4.3.1 Scenario view..................................................................................... 43
4.3.2 Process view.................................................................................... 44
4.3.3 Logical view.................................................................................... 45
4.3.4 Development view.......................................................................... 45
4.3.5 Deployment view............................................................................ 46
4.4 Architecture 2: Multi-classifier on wireless sensor network.................. 46
4.5 Architecture 3: Distributed classifier on wireless sensor networks ......... 48
4.6 Comparison of three architectures......................................................... 51

5 Experiments and Results............................................................................ 52
5.1 Experiment on Intel i5 platform............................................................. 52
5.1.1 Outline of experiments.................................................................... 52
5.1.2 Experiment process......................................................................... 53
5.1.3 Experiment Results......................................................................... 54
5.2 Simulation on embedded platform ......................................................... 57
5.2.1 Simulation tool and node selection.................................................... 57
5.2.2 Network setup................................................................................. 58
5.2.3 Simulation design........................................................................... 59
5.2.4 Simulation tasks............................................................................. 59
5.2.5 Simulation Results........................................................................ 60

6 Conclusion and Future work..................................................................... 63
6.1 Conclusion.......................................................................................... 63
6.2 Future work......................................................................................... 63

Bibliography................................................................................................. 65
**List of abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Networks</td>
</tr>
<tr>
<td>ELM</td>
<td>Extreme Learning Machine</td>
</tr>
<tr>
<td>FPS</td>
<td>Frame Per Second</td>
</tr>
<tr>
<td>GD</td>
<td>Gradient Decent</td>
</tr>
<tr>
<td>ILI</td>
<td>Intelligent Lighting Institute</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>LR</td>
<td>Logistic Regression</td>
</tr>
<tr>
<td>MLR</td>
<td>Multinomial Logistic Regression</td>
</tr>
<tr>
<td>MHI</td>
<td>Motion History Image</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptron</td>
</tr>
<tr>
<td>RBFN</td>
<td>Radial Basis Function Network</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Descent</td>
</tr>
<tr>
<td>SGD-MLR</td>
<td>Stochastic Gradient Descent based Multinomial Logistic Regression</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1 Natural Elements ........................................................................................................ 12
Figure 2 Depth Images ............................................................................................................... 12
Figure 3 Human action classifier .............................................................................................. 15
Figure 4 A example for mapping input data to predetermined posture classes ....................... 15
Figure 5 Workflow for Supervised Learning .............................................................................. 17
Figure 6 Simplified ANN Structure [45] ..................................................................................... 18
Figure 7 How learning algorithm works (h denotes the learned hypothesis function) [32] ........ 19
Figure 8 The curve of logistic function [32] ................................................................................ 20
Figure 9 A simple example of ANN architecture ......................................................................... 22
Figure 10 Data Flow Structure .................................................................................................. 26
Figure 11 Generated dataset for human interaction .................................................................... 26
Figure 12 Four class human postures .......................................................................................... 27
Figure 13 Five class human motions ............................................................................................ 27
Figure 14 Software design of labelling tool .................................................................................. 28
Figure 15 Connectivity of pixels [34] ........................................................................................ 29
Figure 16 An example of human body separation: (a) depth image; (b) a color map for connected
regions; (c) the region separated by bounding box; (d) image resized to 24x20 ............................. 29
Figure 17 An example of generating a Motion History Image (MHI) ........................................... 30
Figure 18 An example of extracting projection-based features .................................................... 31
Figure 19 X, Y projection features for “Left Arm Raised” and “Right Arm Raised” postures .......... 31
Figure 20 X, Y projection features for “Single Person Standing” and “Two People Standing” postures.
.................................................................................................................................................. 32
Figure 21 X, Y projection features for “Waiving Up” and “Waiving Down” motions ..................... 32
Figure 22 X, Y projection features for “Standing”, “Moving Left” and “Moving Right” motions .......... 33
Figure 23 The structure of a MLR classifier ................................................................................ 33
Figure 24 Structure of SGD-MLR classifier ................................................................................. 34
Figure 25 Structure of 3-layer neural network classifier ............................................................... 35
Figure 26 Sparse propagation of activations ............................................................................... 35
Figure 27 Architecture of development environment [39] ........................................................... 37
Figure 28 An example of modified neural networks ................................................................. 38
Figure 29 4-channel 3-layer neural network .............................................................................. 39
Figure 30 8-channel 4-layer neural network .............................................................................. 40
Figure 31 8-to-4-channel 4-layer neural network ....................................................................... 40
Figure 32 Kruchten’s 4+1 view model[40] ..................................................................................... 41
Figure 33 Scenario of architecture 1 ............................................................................................ 43
Figure 34 Activity diagram of architecture 1 ............................................................................... 44
Figure 35 Logical view of architecture 1 ...................................................................................... 45
Figure 36 Development view of architecture 1 ............................................................................ 45
Figure 37 Deployment view of architecture 1 .............................................................................. 46
Figure 38 Logical view of architecture 2 ...................................................................................... 47
Figure 39 Development view of architecture 2 ............................................................................ 48
Figure 40 Deployment view of architecture 2 .............................................................................. 48
Figure 41 Logical view of architecture 3 ...................................................................................... 49
Figure 42 Development view of architecture 3 ............................................................................ 50
Figure 43 Deployment view of architecture 3 .............................................................................. 50
Figure 44 Impact of training steps on the performance of the classifier ...................................... 54
Figure 45 Impact of learning rate on the test accuracy ........................................... 55
Figure 46 Impact of beta on the performance of the SGD-MLR classifier .................. 56
Figure 47 Impact of dropout on the performance of 3-layer neural networks ................ 56
Figure 48 Structure of Rime stack ........................................................................ 58
Figure 49 Initialization of a Sky node ...................................................................... 59
Figure 50 Topology of the sensor network .............................................................. 60
Figure 51 Recognition results received from four classifiers ..................................... 61
Figure 52 Confusion matrix for simulation task 2 .................................................... 61
Figure 53 Confusion matrix for simulation task 3 ..................................................... 62
Figure 54 Human action visualization tool .............................................................. 68
Figure 55 An efficient labelling tool ........................................................................ 68
List of Tables

Tabel 1 Configurations for modification I to V ................................................................. 40
Tabel 2 Results of modification I to V .................................................................................. 40
Tabel 3 Availability of architecture 1, 2 and 3 ...................................................................... 51
Tabel 4 Configuration of datasets ........................................................................................ 52
Tabel 5 Description of experiments ....................................................................................... 53
Tabel 6 Default configuration for parameters ....................................................................... 53
Tabel 7 Comparison of various learning rates on test accuracy ........................................... 55
Tabel 8 Comparison of various batch sizes on validation and test accuracy ......................... 56
Tabel 9 The performance for neural networks with various hidden layers and hidden notes ...... 57
Tabel 10 Configuration of Tmote Sky .................................................................................... 58
Tabel 11 The result of simulation of task 1 ........................................................................... 60
Tabel 12 The result of simulation task 1 ................................................................................ 62
1 Introduction

Human action recognition is an important branch of computer vision and has been widely applied in various areas. This chapter generally describes the background of human action recognition and relevant applications. A lighting installation is introduced followed by the motivation of this master project. Finally, the outline of the thesis is presented.

1.1 Background

Human activity analysis, which is one of the most active research topics of computer vision, is getting increasing attention from computer vision researchers. The goal of human activity analysis is to automatically analyze ongoing activities from a video or a sequence of image frames [1]. It could be categorized into single person activity analysis and multiple people activities analysis [2]. Single person activity analysis can be also called human action recognition. The process of analysis consists of detecting the human body, tracking movements and recognizing the action. An action refers to a posture or a motion of a human. To be specific, a posture is a static state of a person which is conducted by a single person. A typical posture can be “raised arms”, “sitting” or “standing”. A motion, composed by a sequence of postures, shows the process of a movement which could be “walking”, “jumping” and “waving arms”. Human action recognition is based on posture recognition and motion recognition.

Human action recognition has a variety of applications which require automated recognition of human behaviors. By using motion recognition, movements of a person in image frames can be detected and regenerated into a 3D model, and this can be useful for sports experts to analyze the performance of athletes [3]. Besides, it can be implemented in surveillance system which can automatically monitor human presences and behaviors at public areas like shopping mall, train station and airport to detect abnormal or suspicious activities. The capability to understand the meaning of human interactions by machines enables the development of advanced human-computer interfaces which can be used for computer games or designs. Apart from that, analysis of human activities can also be used in video conferencing, healthcare monitoring, and virtual reality.

Much research has been done in the field of human action recognition. Before 2000, researchers paid more attention to body tracking and posture recognition. Bharatkumar et al. [4] proposed a model-based approach by which stick figures are obtained from ordinary images to derive a model of the lower limbs. Iwasawa et al. [5] introduced an approach to estimate human postures by infrared images in real-time. Space-time methods have been widely used which can model 3D volumes of human actions by combining 2D images in a space-time dimension. Ke et al. [6] introduced a spatio-temporal approach which can recognize a wide range of actions by using over-segmented videos with correlation techniques. To recognize human activities from image sequences, state-space [7, 8] and template matching [9, 10] are also applied by many researchers.

Traditional approaches usually need a series of specifically designed algorithms to preprocess the images and extract feature sets. Designing of such complicated algorithms can be quite time consuming. Besides, these manually designed algorithms cannot be easily reproduced for new use cases. Compared with traditional approaches, machine learning methods do not need complex preprocessing algorithms and only depend on good training datasets. Apart from that, when more patterns need to be considered, modification of the classifier can be easily applied on machine learning methods. This is why traditional approaches are gradually replaced by machine learning methods.

The purpose of our project is to design a human action recognition system for a specific interaction dataset. The designed system should be first implemented on host machine for the task of posture recognition and motion recognition. After that, the system needs to be modified so that it can be implemented in a distributed manner. Besides, the system should be able to classify predefined human actions. Therefore, supervised machine learning methods are desirable and applicable for our project.
1.2 Natural Elements

Glow Next [11] is a lighting exhibition in Eindhoven, a world of light, where new lighting projects and experiments are shown for the very first time. Natural Elements [12] is an interactive installation developed by Philips lighting and TU/e Intelligent Lighting Institute (ILI). This interactive installation is able to simulate natural light reflection and patterns. During Glow Next, researchers conducted experiments on Natural Elements to explore the visual principles of natural light phenomena. Our project is inspired by Natural Elements and the data used in our experiments are collected by this interactive installation during Glow exhibition.

As can be seen in Figure 1, this installation can produce images of natural elements on a low resolution LED display. Impressions of the classical elements including earth, air, fire and water are represented by simulations of mathematical models learned from video recordings of natural light phenomena. These natural lights can be dynamically generated on the installation and are influenced by human interactions. The lights change in shape depending on the position and movements of people in relation to the installation. The lighting installation can capture the movements, extract useful information, process the information by algorithms and produce new images on the screen. People can notice the changes of visualization and play with the lights by different actions in real time. In this way, interaction between human and machine can be achieved.

In order to capture human movements, Kinect cameras are used on the installation. Actually, the installation does not use the full capabilities of the Kinect camera and the generated data is low resolution binary images. Each Kinect camera contains a depth sensor that can produce depth images. Depth image can save information relating to the distance of objects from a viewpoint [13]. The depth image used for this installation is binary-value black and white images instead of gray-scale images. Figure 2 shows the depth images in which the human body is white color and the background is black color.

In order to produce dynamic visuals to interact with the human, the installation operates a visualization program which simply loads the depth images, renders them by visual lights and displays them on the screen in order. This program works well and enables simple interactions between human and the installation. However, it can only conduct binary classifications on presence of human or absence of human. To analyze human actions in the depth images, a better solution needs to be found.

![Figure 1 Natural Elements](image1)

![Figure 2 Depth Images](image2)
1.3 Motivation
As mentioned in section 1.2, the current solution of Natural Elements cannot be used to analyze human actions in the interaction dataset. Thus, we need to design a better solution that takes depth images as inputs and recognizes the underlying human actions. According to the literature study, designing a human action recognition system is exactly what such a solution is.

However, the data collected by Natural Elements cannot be easily used with existing methods. First, the resolution of the image frame is quite low and only upper part of the human body is included, which means that the shape of the human is considerably rougher and details of gestures can be scarcely obtained. Second, the data is a depth image so no texture or color can be used for feature extraction. Therefore, a suitable method needs to be found depending on this specific data.

As discussed in section 1.1, supervised machine learning methods can be used to solve this problem. Thus, the goal of this master project is to design a classifier for human action recognition by using supervised learning methods. Recognition accuracy is calculated to evaluate the performance of the classifier. The key parameters of the classifier should be adjusted to reach the optimal performance. Thus, the thesis conducts experiments to test configurations of parameters and analyzes the results. Since the classifier is trained and operated on a host machine, this solution can be considered as a centralized recognition system.

Indeed, the centralized system is sufficient and acceptable for action recognition of the interaction dataset. However, the goal of this thesis is not only developing a feasible centralized recognition system, but also discovering a distributed recognition system that can distribute the data flow and the computation cost on sensor networks. Thus, the thesis presents various modifications of the classifier and evaluates their performance. Architectures of centralized and distributed recognition systems are designed in the thesis. Simulations of different architectures are conducted and the results are evaluated. In this project, the recognition result is not being used for actuation. That will be the task for future work.

1.4 Research Questions
The research of this thesis focuses on designing and optimizing a centralized action recognition system and exploring modifications of the system to implement it on sensor networks. The research questions of this thesis is shown below:

- **Question1**: Which machine learning method can be used for the classifier?
  This question is answered by Chapter 2 which describes the selection of machine learning algorithms.

- **Question2**: What are the image processing steps to get high quality training dataset?
  This question is answered by Chapter 3 which explains the methods used to process the depth images.

- **Question3**: Which parameters are vital to the optimization of the classifier and what are the optimal values of the parameters?
  Chapter 2 and Chapter 5 answer this question. In Chapter 2, the quality of a classifier is defined by problem formalization. Chapter 5 designs the experiments to test parameters and find the optimal configuration.

- **Question4**: How can the machine learning model be modified to develop a distributed recognition system?
  This question will be answered in Chapter 3 which explains the method to implement machine learning models in a distributed way.

- **Question5**: How can the distributed recognition system be deployed on sensor networks?
  The architecture design of action recognition systems in Chapter 4 can answer this question.
1.5 Thesis layout

The thesis is organized as follows: Chapter 2 presents the problem description, formalizes the problem and briefly introduces the related works. Chapter 3 describes image processing methods and machine learning methods and presents several modification approaches for distributed solutions. Chapter 4 proposes architecture designs of centralized and distributed human action recognition systems. Chapter 5 explains experiments and simulation tasks for human action recognition system and analyzes the results. Chapter 6 concludes the thesis and discusses the future works.
2 Problem Description and Related Works

As discussed in Chapter 1, the key task of this thesis is to develop a human action recognition system for Natural Elements. The system should be able to classify a single image to one of the postures and classify an image sequence to one of the motions. Thus, the core problem of designing such a system focuses on how to establish a mapping between unknown images and predefined human actions.

This chapter presents the problem statement, and explains machine learning models that are selected to build the classifier. The details of the problem are defined and formalized into mathematical expressions. At the end of this chapter, some related works are introduced.

2.1 Problem Statement

Figure 3 illustrates the simplified structure of human action recognition system. The input data are depth images retrieved from the interaction dataset of Natural Elements. Output values are limited to the predefined values representing different human postures or motions. The classifier maps the input data to the predefined classes.

As an example, we use four single images as inputs of the classifier. The classifier uses a learned “hypothesis” for mapping inputs to predetermined classes. Figure 4 shows an example for mapping input data to predefined posture classes.

To be specific, the classifier implements a mathematical model to read in input data, transform the data, and compare the result with standard values to determine which action it belongs to. As the original data is not labelled, it cannot be directly used as input to the classifier in the learning stage. Before recognition, the original data needs to be visualized to investigate what kinds of human actions can be found. The recognizable human actions need to be classified and each action is noted by a unique label. Then the images are processed by algorithms and manually labelled to generate training examples for the classifier.
However, it is still not easy to construct such a model that can precisely recognize human actions. The hypothesis which can approximate the relationship between input data and label is unknown to us at the moment. Thus, the model should be modifiable to adjust the parameters of the function and get to the optimal configuration. According to the literature study, machine learning methods can provide such kinds of models. The desired classifier can be realized by training the machine learning model using human action images.

2.2 Machine Learning Models

Machine learning [14] has been widely used in spam detection, speech recognition, robot control, object recognition and many other domains. It is a type of artificial intelligence technology which enables computers to learn without being explicitly programmed [15]. Machine learning can be considered as the development of a system that can process a large volume of data, extract meaningful and useful information and exploit such information in practical problems [16]. Machine learning has two common learning styles: supervised learning and unsupervised learning.

Supervised learning [17] is usually used for regression, classification and ranking. In case of classification, the task of supervised learning is to establishing a relationship function from training data which consists of labelled training examples to their respective labels. Each training example contains input data and a corresponding labelled output value. The output of the relationship function is a logical value used for classification. The trained function should be able to predict the correct value of any input data. By analyzing each pair of training examples, the relationship function is produced and adjusted step by step until the underlying relationship between inputs and outputs can be appropriately expressed by the function. Common approaches for supervised learning includes logistic regression, Bayesian statistics and artificial neural networks [16].

Unsupervised learning [18] can be used for clustering, anomaly detection and dimensionality reduction. Unlike supervised learning, the task of this machine learning method is discovering the hidden structure of unlabeled training data [19]. As the training data is unlabeled, unsupervised learning does not have such a reward system to evaluate predicted output values. It mainly focuses on exploring hidden patterns or intrinsic structure of training data. The intrinsic structure can be used to organize these unlabeled data into similarity groups, which are also called clusters. K-means, hierarchical clustering and hidden Markov models are common cluster algorithms for unsupervised learning.

The goal of this thesis is to develop a human action recognition system based on the interaction dataset. The algorithm used for this system should be able to recognize each human action which needs to be predefined before training. Thus, compared with unsupervised learning, supervised learning which can use labelled training set for classification tasks is the optimal solution for designing such a recognition system.

Figure 5 shows the workflow of supervised learning methods. For classification tasks, the classifier is developed by three phases: training, validation and performance test. Feature extraction is the first step in the training phase. Raw data are preprocessed by feature extraction algorithms to get the feature matrix. Then the correct outputs are created for the derived feature data. After that, feature data together with correct outputs are randomly selected and separated into three datasets: training set, validation set and test set. The training set is fed to the model iteratively to train the parameters of the model. The validation set is used to determine when to stop the training by estimating the performance of the model during the training process. The test set is a set of examples that never take part in the training process, which means that is totally new data for the trained model and is finally used to evaluate the quality and the generalizability of the trained model by calculating the prediction accuracy on it.
According to the literature study, some supervised learning methods are selected for our classifier. The reason why these methods are suitable for our project is discussed as follows.

Logistic Regression (LR) is one of the most commonly used method for applied statistics and discrete data analysis and often works surprisingly well as a classifier [20]. When the target outcome has only two types, it can be called a binary classification problem. For binary classification, this algorithm learns from the relationship between the target outcome and a given set of predictors by estimating probabilities using the logistic function. When the target outcome has more than two types, it is called a multiclass problem. Multinomial Logistic Regression (MLR) is a classification method which generalizes logistic regression to multiclass problems. We implement the MLR model as the classifier for human action recognition tasks.

Stochastic Gradient Descent (SGD) [21] is a popular algorithm which can optimize the training process for a wide range of machine learning methods like multinomial logistic regression, Support Vector Machine (SVM) and neural networks. This algorithm can reach convergence much faster than standard gradient descent algorithm due to more frequent weight updates [22]. For a large training set, gradient descent requires expensive computation cost to calculate the gradient for all training examples and this process could be quite slow. Unlike gradient descent, SGD only computes the gradients of a mini-batch (a small set) of training examples in each iteration and updates the weights of the model by these gradients. This simple algorithm can usually train a good set of parameters surprisingly quickly in contrast with other sophisticated optimization methods [23]. Thus, this algorithm can be implemented in the training phase of our classifier to speed up the training process. The MLR model which is based on SGD optimizer can be called SGD-MLR. It is also implemented as the classifier for action recognition tasks.

Artificial Neural Networks (ANN) [24] have been used in many fields, such as regression analysis, data processing, robotics, computer vision, and pattern recognition, especially for speech recognition, face recognition and handwriting recognition tasks. ANN, which is inspired by biological neural networks, is composed by layers of interconnected computing units [25]. A simplified structure of ANN is shown in Figure 6 [45]. In the simplified ANN model, input examples are summed up using weight parameters and the output is computed by using the activation function. For supervised learning applications, the error between output results and target values is calculated by a cost function, and learning algorithms like back-propagation are used to train the model by adjusting the weights. When the cost function reaches the optimal value, the training process of the weights is finished and the model is ready for predictions.
Neural networks have been successfully implemented in a wide variety of tasks performing better than many other algorithms. Although neural networks can be applied for prediction, data filtering, interpretation and control, the most successful applications are categorization and pattern recognition. Therefore, ANN is quite suitable for our human action recognition problems. As for distributed recognition system, the algorithm should be modified and applied on sensor nodes. As the layered structure of ANN is similar to a layered sensor network, it is possible to deploy neural networks on the sensor network and each sensor serves as a group of neurons of ANN. In this way, the computation procedure can be separated into sub-processes and operated on a group of sensors. When the depth images are collected by the sensor networks, the recognition of human actions can be conducted immediately by the distributed neural networks. Thus, ANN is a promising solution for the distributed recognition system. The neural network model can be first designed for the centralized system running on the host machine, and then modified and deployed on sensor networks based on different architectures.

2.3 Problem Formalization

According to Section 2.1, the problem of our project is to develop a classifier which can map sequences of images with correct human actions. The classifier can be developed by using a machine learning model which can learn from training examples and tune parameters of objective function to reach an optimal mapping between inputs and target outputs. In this way, the trained machine learning model is able to classify input images to one of the human action classes. It is important to establish a quantitative representation for each step in solving this problem. Thus, the problem will be formalized in this section. The formalization will start from high level abstractions and then move to low level abstractions. The definition of each term will be given and corresponding parameters will be discussed.

2.3.1 Mapping Function

The human action classifier can be defined as a mapping function that takes the feature vector of each image as the input and takes the label vector as the output. Let us assume that the mapping function is trained by using n pairs of training examples. The mapping function is given by

$$\mathbf{y}^{(i)} = f(\mathbf{x}^{(i)}), \ i = 1, 2, \ldots, n \quad (1)$$

where $\mathbf{x}^{(i)}$ is the feature vector of the $i^{th}$ training example, $\mathbf{y}^{(i)}$ is the label vector of the $i^{th}$ training example. Assume we use pixels of each image as features. The resolution of an image is $(r, c)$, so there are $k = r \times c$ features in each image. Then, the feature vector $\mathbf{x}^{(i)}$ can be denoted as
\[ \mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_k^{(i)} \end{bmatrix}, \quad i = 1, 2, \ldots, n \]  

where \( x_j^{(i)} \) is the \( j^{th} \) feature in the vector. Then we assume \( m \) is the number of human action categories, so the label vector \( \mathbf{y}^{(i)} \) is given by

\[ \mathbf{y}^{(i)} = \begin{bmatrix} y_1^{(i)} \\ y_2^{(i)} \\ \vdots \\ y_m^{(i)} \end{bmatrix}, \quad i = 1, 2, \ldots, n \]  

where \( y_l^{(i)} \) denotes the label of the \( l^{th} \) class of human action. In the label vector \( \mathbf{y}^{(i)} \), only one element can be equal to 1 as the valid label, and others are equal to 0. For example, when \( y^{(i)} \) is related to the first class of human action, only the element \( y_1^{(i)} \) should be equal to 1.

As we can see above, the mapping function can convert the input vector \( \mathbf{x}^{(i)} \) which is a \( k \)-dimensional vector to the output vector \( \mathbf{y}^{(i)} \) which is an \( m \)-dimensional vector. This could be achieved by conducting a series of linear and non-linear transformations on the input vector.

### 2.3.2 Machine Learning Model

As discussed above, the mapping relationship between inputs and target outputs can be represented by a mapping function. We use machine learning algorithms to guess a “hypothesis” function that approximates the unknown mapping function. The goal of learning process is to find the final hypothesis function that best approximates the unknown mapping function.

We establish the notation of machine learning methods based on Andrew Ng’s online course [32]. As defined before, we have \( n \) training examples and the \( i^{th} \) training example is a pair of vectors \( (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \) in which \( \mathbf{x}^{(i)} \) is the input vector and \( \mathbf{y}^{(i)} \) is the output vector. The training set can be represented as \( \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\} \) which contains \( n \) pairs of training examples. In the machine learning model, the mapping relationship can be represented by the hypothesis. Figure 7 [32] illustrates a simple principle of how the learning algorithm works. By learning from the training set, the learning algorithm trains a hypothesis function \( h(\cdot) \) that can map from input data to output labels.

![Figure 7 How learning algorithm works (h denotes the learned hypothesis function) [32]](image)

Basic concept of the machine learning model including the hypothesis function, logistic function, cost function, and gradient decent are explained below.
2.3.3 Hypothesis and logistic function

As mentioned before, the hypothesis function \( h(\cdot) \) is used to approximate the mapping function. Depending on what learning algorithm we use, the hypothesis function can be represented by linear or non-linear function. To introduce the hypothesis function, we take the logistic regression model as an example. When logistic function is used for logistic regression, the hypothesis can be given by

\[
h_\theta(x) = g(\theta^T x + b)
\]

where \( \theta \) is a weight vector, \( b \) is a bias, \( x \) is an input vector and \( g(\cdot) \) is the logistic function. \( \theta \) and \( b \) are the parameters of logistic regression used for linear transformation on the input data. The vector \( \theta \) is denoted as

\[
\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_k \end{bmatrix}
\]

where \( k \) as defined before is the number of features for a training example. It can be updated during learning process to best approximate the mapping relationship. The logistic function is the most commonly used sigmoid function that is used in logistic regression to introduce non-linearity in the model. The logistic function is given by

\[
g(z) = \frac{1}{1 + e^{-z}}
\]

where \( e \) is “Euler’s number”, a mathematical constant. This function converts the value of \( z \) from \((-\infty, +\infty)\) to \((0, 1)\) and the curve is shown in Figure 8.

![Figure 8 The curve of logistic function [32]](image)

The converted value can be considered as the probability of a prediction. We take binary classification for example, two classes are represented by 0 and 1, so \( y \) can be 0 or 1. When \( z \) is larger than 0.5, it will be predicted as \( y=1 \); otherwise, it will be predicted as \( y=0 \). Thus, the probability of \( y=1 \) and \( y=0 \) can be given by

\[
P(y = 1|x; \theta) = g(\theta^T x + b)
\]

\[
P(y = 0|x; \theta) = 1 - g(\theta^T x + b)
\]

For multi-class classification, the softmax function [35] is introduced as a generalization of logistic function, used in multinomial logistic regression. The softmax function is denoted by:

\[
\sigma(z_l) = \frac{e^{z_l}}{\sum_{m=1}^{m} e^{z_l} \ l = 1, \ldots, m.
\]
where \( z_l \) is the \( l^{th} \) element in the vector \( z \), and \( m \) as defined before is the number of classes. The softmax function is used to convert the \( m \)-dimensional vector \( z \) to an \( m \)-dimensional vector \( \sigma(z_l) \) of values in range of \((0, 1)\) and the values add up to 1. Thus, \( \sigma(z_l) \) can be considered as the probability that an input example can be predicted as an output class \( l \). Then, the output vector \( \sigma(z) \) can be considered as a probability distribution of all possible classes. When \( \sigma(z_l) \) gets the largest value in the probability distribution, the input example should be predicted as class \( l \). For multi-class classification problems, the softmax function works better than the logistic function \([43]\). Thus, we select the softmax function as the sigmoid function in our machine learning models.

### 2.3.4 Cost Function

In order to determine how well the learning model performs in mapping input examples to target outputs, a cost function is introduced. A typical cost function is represented as

\[
J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})
\]

where \( h_\theta(x^{(i)}) \) is the hypothesis for the \( i^{th} \) input data and \( y^{(i)} \) is the correct output. The function \( \text{Cost}(\cdot) \) is used to calculate the error between the predicted result (hypothesis) and the correct value of a training example. The \( f(\theta) \) function is the average error for the predictions of the training set. For binary classification, \( \text{Cost}(\cdot) \) based on cross-entropy \([36]\) is given by

\[
\text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} 
-\log(h_\theta(x^{(i)})) & \text{if } y^{(i)} = 1 \\
-\log(1 - h_\theta(x^{(i)})) & \text{if } y^{(i)} = 0 
\end{cases}
\]

And then, the function \( J(\theta) \) can be given by

\[
J(\theta) = -\frac{1}{n} \left[ \sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]
\]

It is a convex function which enables an optimization algorithm to find the global minimum of the cost. The optimization algorithm updates the weight parameter \( \theta \) to minimize the cost function.

### 2.3.5 Gradient Descent

As discussed above, the weight parameter \( \theta \) should be trained to get the minimum value of cost function. The question is how to find the optimal setting of the weight parameter. According to Raschka, S. \([33]\), the optimization algorithm – gradient descent (GD) can be applied to find the minimum of the cost function. During the learning process, the cost function will be optimized by updating the weight parameter using gradient descent.

The principle of gradient descent can be described as climbing down a hill until a global minimum is reached \([33]\). The gradient descent algorithm repeatedly updates the parameters in this way

\[
\text{Repeat} \left\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \right\}, \quad j = 1, 2, ..., k
\]

Where \( \theta_j \) is the \( j^{th} \) weight in the weight vector \( \theta \), \( \frac{\partial}{\partial \theta_j} J(\theta) \) is the partial derivative of \( J(\theta) \) for \( \theta_j \), and \( \alpha \) is the notation for learning rate which determines how much the weight \( \theta_j \) updates in each step. By using gradient descent, the minimum of the cost function can be reached and the optimal setting of the weight parameter can be found.

### 2.3.6 Stochastic Gradient Descent

The gradient descent algorithm calculates the cost of all training examples in a single step and then updates the weight parameters by partial derivatives. When the training set has a large number of training examples or there are quite a lot features for each training item, computing the derivatives using gradient descent can be very expensive and kind of slow.
Different from normal gradient descent algorithms, stochastic gradient descent works in a different way. In each iteration, it can just look at one training example and update the parameters by the partial derivatives of its cost. The way SGD updates the parameters can be given by

\[
\text{Repeat } \left\{ \theta_j - \alpha \frac{\partial}{\partial \theta_j} \text{Cost}(h_\theta(x^{(i)}, y^{(i)})) \right\}, \ j = 1, 2, \ldots, k
\]

(4)

where \( \frac{\partial}{\partial \theta_j} \text{Cost}(h_\theta(x^{(i)}, y^{(i)})) \) is the partial derivative of the cost of the \( i^{th} \) training example \((x^{(i)}, y^{(i)})\). This algorithm updates much faster than gradient descent. It could be applied on multinomial logistic regression and neural network models for parameter optimization.

2.3.7 Artificial Neural Networks

According to Figure 6, the basic ANN component, neuron can be considered as a logistic regression model if the activation function is the logistic function. Figure 9 shows a simple example of ANN architecture in which each yellow circle is a single neuron connected with other circles by activation functions.

In an artificial neural network, each neuron uses a sigmoid function. There are three types of layers: input layer, hidden layer and output layer. Each layer is constructed by a set of neurons. Training examples are fed to the input layer and the hypothesis is calculated on the output layer. To get the predicted output, ANN uses a feed-forward algorithm which calculate the activations on each layer and use them as new inputs for next layer until we get the final hypothesis. The hypothesis of ANN is denoted by \( h_\theta(x^{(i)}) \) where \( \Theta \) denotes the weight parameter. The cost function of ANN can be derived from logistic regression and given by

\[
J(\Theta) = -\frac{1}{n} \left[ \sum_{i=1}^{n} \sum_{j=1}^{m} y_j^{(i)} \log(h_\theta(x^{(i)})) + (1 - y_j^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right) \right]
\]

(5)

where \( n \) is the number of training examples, and \( m \) is the dimension of hypothesis \( h_\theta(x^{(i)}) \) with respect to the number of action classes.

To minimize the cost function, ANN uses a back-propagation algorithm that first runs a feed-forward algorithm to get the hypothesis and then calculates partial derivatives on each layer. The calculation of error terms starts from the output layer then goes back to the previous layers. Optimization algorithms
like gradient descent can be used to update weight parameters iteratively based on the error terms and activations until a minimum of cost function is reached.

2.3.8 A performance measure: accuracy

In this section, a performance measure - accuracy is defined to evaluate the machine learning model for human action recognition. The accuracy is given by

$$\eta = \frac{1}{n} \sum_{i=1}^{n} \frac{p_i}{n} \times 100\%$$

where $n$ is the number of training examples and $p_i$ is the prediction result of each training example. If the prediction of the $i^{th}$ training example is correct, $p_i$ is set to 1; otherwise, $p_i$ is set to 0. Thus, $\eta$ shows is prediction accuracy of the trained classifier. In our experiments, human action images are labelled manually and then grouped in three datasets: training set, validation set and test set. Machine learning models learn from the training set and optimize the parameters to reach minimum cost value. The training accuracy can be denoted by $\eta_{training}$ which represents the prediction accuracy on the training set. The generalizability of the classifier can be evaluated by the validation accuracy $\eta_{validation}$ which is the prediction accuracy on validation set. Similarly, test accuracy which is denoted by $\eta_{test}$ is used to evaluate the performance of the classifier when the training process is finished.

2.4 Related work

Aaron F. Bobick, et al. [27] present Motion History Image (MHI) to represent human motion. An MHI is a kind of temporal template which can be created by recording the motion in an image sequence. Human postures in the image sequence are accumulated in a way that records the corresponding motion history. MHI can be used in our project to extract motion features from humans. Aaron F. Bobick, et al. use a matching based method to recognize predefined motions by comparing new templates with labelled MHI instances. This matching approach performs well because of the high quality of motion examples. Training examples for different motions are designed and created by the algorithm developers as reference. However, we want to make the classification algorithms suitable for training and test datasets that are randomly collected, with unpredictable variations for each kind of motion. Our algorithms will classify human actions by learning from data without the need of standard templates created by experts.

Alexandros, et al. [28] apply ANN and Bayesian framework for human action representation and classification on a multi-camera setup. Action recognition is achieved by using Multilayer Perceptron (MLP) which is a feed forward neural network model. The ability of this ANN based method to correctly classify human actions is shown by the experiments on multi-view database in which the highest recognition rate is 94.87%. The proposed method demonstrates the capacity of ANN in human action representation and recognition and shows the effectiveness of ANN model in challenging experimentations. Thus, we envision that ANN is a good candidate classifier for human action recognition.

Haitham, et al. [29] present a vision-based technique for static hand gesture recognition. Multilayer neural networks with back-propagation algorithms are used to recognize gesture features into predefined classes. The neural network based method performs well in the testing experiment and reaches sufficient recognition accuracy. What we can learn from this paper is how they analyze the problems and determine proper techniques for each step of recognition process. They divide the process into four steps: data gathering, data processing, feature extraction and classification. This type of work flow can be applied in the development of algorithms. Apart from that, this paper gives a detailed introduction of how to train the multilayer neural networks and how to design the experiments to test the classifier.

R. Venkatesh Babu, et al. [30] present a neural network approach for subject independent human action recognition. The features of human actions are extracted by using 2D optical flow over depth image frames and trained on a Radial Basis Function Network (RBFN) which is a kind of ANN. It can be
observed from the result that the RBFN classifier performs better than Extreme Learning Machine (ELM) and Support Vector Machine in terms of recognition accuracy of human actions. The result also demonstrates the ability of this method to generalize human actions accurately. This paper proposes three performance measures of classification efficiency which can be used in this thesis to evaluate the performance of general action recognition.

Chen, et al. [31] propose a real-time human action recognition system based on fusion features of depth images and inertial signals. The system is trained by a public human action dataset and evaluated for real-time and offline recognition performance. The recognition accuracy is more than 97% which demonstrates the effectiveness of the system. This paper gives us a good example of data analysis in which bar charts, tables and confusion matrix are used to analyze different aspects of recognition performance.

Mike, et al. [46] conduct investigation on different types of distributed neural networks in terms of communication cost and memory usage. They propose centralized, horizontal and vertical decomposition approaches for distributing neural networks in a Wireless Sensor Network (WSN). Compared with vertical decomposition, horizontal decomposition gets a more promising result for communication costs. This article gives a good example of how neural network can be modified to a distributed structure.
3 Human Action Recognition System

In this chapter we give an overview of data flow, introduce data preprocessing methods as well as a visualization tool and a labelling tool. Feature extraction algorithms are presented followed by implementation details of training a classifier for action recognition. Finally, the neural network model is modified to a multi-channel structure which can be implemented in a distributed way.

As we discussed in Chapter 2, machine learning models are used to design classifiers for human action recognition. According to the requirement of training examples, original data should be processed to fit the format of input layer of the classifier. For human posture recognition, still images are used as posture representation. For human motion recognition, a motion history image algorithm is applied to generate a motion representation from successive posture images. Pixel values of generated depth images can be directly used as features to train on machine learning models. The classifier based on pixel features performs well on Intel Core i5 platform but it is not applicable on embedded platform which is limited on computational workload and memory capacity. Thus, it needs feature extraction methods to extract higher level human action features from depth images. A projection-based feature extractor is presented to generate smaller feature matrix for the classifier used on embedded platforms. Machine learning models are built on TensorFlow platform. MLR is used for action recognition tasks followed by the SGD-MLR as a contrast method. A multi-layer ANN is also implemented as the classifier which uses flexible configuration of parameters to optimize recognition performance. This ANN model is then modified to a distributed structure which can be deployed on sensor networks.

3.1 An overview of data flow

To establish a training dataset used on machine learning models, the raw data needs to be processed in a few steps. The data flow chart is shown in Figure 10. The raw data of human interactions are converted to strings of binary values and saved in text files as the new dataset. Depth images are created during visualization to investigate human posture and motion patterns. Postures can be represented by a single depth image while motions are represented by motion history images which contain motion features extracted from successive depth images. When common postures and motions are determined, we manually label the training examples by using the labelling tool.

A posture separation method is designed to detect posture of a single person in the image, separate it out and resize the image to a predefined size. The resized posture images are used as training examples for the human posture recognition. Pixel values are extracted as features of human postures. The feature matrix created by all feature vectors of training examples will be used as the training dataset for machine learning models. Since large matrix computation is hard to implement on embedded devices, a projection-based feature extraction method is used to extract high level features from pixel features. By using this feature extraction method, the size of the motion feature vector shrinks from 1728 to 84 and the size of posture feature shrinks from 480 to 44. Thus, this method can reduce computation workload and memory capacity of machine learning models and satisfy the requirement of embedded devices.

After data processing, four training datasets are created – two for human posture recognition and two for human motion recognition. These datasets can be easily used on machine learning models for training, validation and testing. The detail of data processing and feature extraction are explained in the following sections.
At the beginning of the implementation phase, raw data should be transformed to a format which can be easily processed and readily retrieved. Data preprocessing methods are applied to solve this problem. The raw data are zipped log files which contain depth values of human interactions. By using data preprocessing methods, these zipped log files are converted to binary value depth images and saved in text files. Pixel values are extracted from each frame and written in a single line of a text file. Timestamps related to the frames are saved in another text file as index of frames. All empty frames are filtered and only valid frames are kept. Compared with raw dataset, the new dataset has better data structure and enables easier and faster read/write operations. As shown in Figure 11, a text file is named by the timestamp of corresponding recording time of the log file.

![Figure 10 Data Flow Structure](image1)

3.2 Data preprocessing
At the beginning of the implementation phase, raw data should be transformed to a format which can be easily processed and readily retrieved. Data preprocessing methods are applied to solve this problem. The raw data are zipped log files which contain depth values of human interactions. By using data preprocessing methods, these zipped log files are converted to binary value depth images and saved in text files. Pixel values are extracted from each frame and written in a single line of a text file. Timestamps related to the frames are saved in another text file as index of frames. All empty frames are filtered and only valid frames are kept. Compared with raw dataset, the new dataset has better data structure and enables easier and faster read/write operations. As shown in Figure 11, a text file is named by the timestamp of corresponding recording time of the log file.

![Figure 11 Generated dataset for human interaction](image2)
Since the data is randomly collected and unlabeled, the action information hidden in it should be investigated to determine the most common postures and motions. This process could be time consuming if we simply plot each frame of the dataset and recognize them one by one. Therefore, we developed a visualization tool which can be used to conduct the investigation process. The user interface of visualization tool is shown in Appendix A.

With the help of the visualization tool, each new found human action including postures and motions is recorded and the set of actions are classified into categories. There are some postures that rarely appear or motions which are not representative and have a low number of examples. Since the generalization is essential to our human action recognition system, these examples that rarely appear were filtered out and only the most common examples were selected. Four classes of postures and five classes of motions were determined as patterns. Human postures including “Left Arm Raised”, “Right Arm Raised”, “Single Person Standing” and “Two People Standing” are shown in Figure 12. Besides, five common motions are chosen: “Moving Left”, “Moving Right”, “Waving Up”, “Waving Down” and “Standing”. These motions are represented by MHIs and shown in Figure 13.

![Figure 12 Four class human postures](image1)

![Figure 13 Five class human motions](image2)

According to problem description in Chapter 2, supervised learning methods should learn from labelled training examples. Each example needs to be selected from the dataset and labelled manually. To guarantee the generalizability of the classifier, at least a few hundreds of examples should be selected for each human pattern. Thus, the manual labelling process can be considerably time-consuming. To solve this problem, we designed a labelling tool which can read through the entire dataset, filter blank images and images of noise points or incomplete parts. The remaining frames can be efficiently classified into one of the action categories by simple operations. The software design of the labelling tool is shown in Figure 14 and the UI of the labelling tool is shown in Appendix A.

By using the labelling tool, 450 examples were labelled for each posture class so the training dataset has 2200 examples for human posture recognition. To build up a human motion classifier, successive depth frames are converted to motion history images and 1200 MHIs are selected as training examples. With thousands of training examples, the posture and motion datasets used in the project are sufficient for machine learning models to train an effective classifier which can accurately classify human postures or motions.
3.3 Feature extraction

As mentioned before, features need to be extracted from labelled images. The classifier is first trained and used on Intel Core i5 platform and then transplanted to embedded platform which has limited computation capacity. Thus, feature extraction methods that require complex computations should not be applied. Since raw data has already been converted to binary value depth images, those methods designed for the RGB image are not applicable in our project. According to the literature study, learning methods can directly learn from pixel values of images. Thus, pixel values of each training example are extracted as the feature of human action patterns.

Since the size of a depth image is 36x48, the corresponding feature vector can be denoted as $x[1, k]$ ($k = 1728$) where $k$ represents the number of features. When we have 1000 training examples for posture recognition, the input matrix can be denoted as $X[n, k]$ ($n = 1000, k = 1728$) where $n$ represents number of examples. If we use MLR as the classifier, the weight parameter used in LR is denoted by $W[k, m]$ ($k = 1728, m = 4$) where $m$ represents the number of output classes. The output can be denoted by $h(WX)$ where $h(\cdot)$ is the hypothesis. During the training process, partial derivatives of cost function are calculated to update the weight parameter. The scale of feature matrix affects the speed of matrix operation and the larger feature matrix needs longer computation time. The cost for iteratively updating such big matrices can be quite high and the training process is correspondingly time-consuming.

3.3.1 Posture separation method

In order to reduce computational workload and accelerate convergence of the objective function, we introduce a posture separation method that can detect a human body and cut it out by a bounding box. Only pixel values in the detected area are considered as effective features. This method can be achieved by image processing algorithm based on the scikit-image library [34]. First, connected regions in the depth image are set to different colors. As can be seen in Figure 15, connectivity refers to the maximum distance between neighbors. In our project, connectivity is set to 2 which means that the pixels with same value in 2 hops can be considered as neighbors. A connected region is a complete set of neighbors in which pixels are linked in range of connectivity. As shown in Figure 16 (b), connected regions in the image are set to different colors.
After coloring the connected regions, the number of pixels in each filled area is calculated and the largest region (except the background) is considered as the target region. The target region is then detected and cut out by a bounding box vector \((\text{min}_\text{row}, \text{min}_\text{col}, \text{max}_\text{row}, \text{max}_\text{col})\). The separated image cannot be used by learning models because they vary in size and contain different number of pixels. To solve this problem, we resize the separated image to 24x20 which is the best resolution compatible for most images. Pixel values of resized images can now be extracted as features. By applying this method, the size of the feature vector is reduced from 1728 to 480. Thus, the training workload on new feature matrix is much less than using full size images as features. Apart from that, the classifier trained by separated images only learns from posture information of single person and ignores the location information. This can increase the accuracy of human posture recognition.

### 3.3.2 Motion History Image

Depth images only contain static information of human postures which is not sufficient for motion representation. According to paper [27], motion history image, a temporal template, can be used for motion feature representation. MHI not only records the presence of motion but also saves the history of a movement from the start frame to the end frame in sequence of images. MHI is created by past successive images using a weighted sum algorithm. Latest image contributes most and produces the brightest part of the MHI. The algorithm is shown below:

\[
H_\tau(x, y, t) = \frac{\tau}{\tau} \max(0, H_\tau(x, y, t-1) - 1) \quad \text{if} \ D(x, y, t) = 1 \\
\text{otherwise.}
\]  

where \(D(x, y, t)\) is a depth image, \(x\) and \(y\) are locations of pixels, \(t\) is the index of frame, \(\tau\) is the duration, and \(H_\tau(x, y, t)\) is the generated MHI. Basically, MHI is a vector-image which contains direction information of a motion by combining and vanishing past images step by step. In order to create desirable MHIs which have clear borders and complete motions, key parameters of the algorithm should be determined. After several trials, the best configuration is found:

- Sampling rate is set to 3: each third image is captured in the frame sequence.
- The duration \(\tau\) is set to 5: MHI is created by 5 frames.
- Intensity of MHI decays from 1 to 0: the change in intensity shows the gradient of motion.
Figure 17 shows five depth images which are selected by the algorithm to generate a new MHI. As shown in the last figure, the motion history is quite clear and the moving process is easy to distinguish. It demonstrates the feasibility of this configuration for MHI representation. By applying this method, a large number of MHIs were generated to build up a dataset for human motion. Five common motions are found during investigation by using the visualization tool and examples of these motions are shown in Figure 13.

![Figure 17 An example of generating a Motion History Image (MHI)](image)

The generated MHIs are classified by the labelling tool to one of the five classes. Pixel values are extracted as the features of MHI. Different from posture features which are binary values, the value of a motion feature can be in the set \{0, 0.2, 0.4, 0.6, 0.8, 1\}. A feature matrix is created by concatenating the feature vector of each MHI. It can be denoted as \(F[n, k] (n = 1000, k = 1728)\) where \(n\) is the number of training examples and \(k\) is the number of features. Machine learning models can be trained by this feature matrix to classify the human motions.

### 3.3.3 Projection-based feature extraction

When using pixel features, the neural network is connected by large matrices as weight parameters of each layer. Each element in the weight matrix is a floating point variable so the matrix multiplication during the training process requires high computation capacity. Although this level of computation can be easily handled by a computer which has powerful CPU units, it is unrealistic to do such heavy computation on embedded devices which have memory and computation limitations. Therefore, we present a projection-based feature extraction method which calculates summations of pixel values along \(x\) axis and \(y\) axis of each image as features. The classifier which learns from this feature requires less computation capacity to update parameters during the training process.

Figure 18 shows an example of how this feature extraction method works. The input data is a depth image which has been processed by image separation method. This image is separated from a full-size image and resized to 24x20. To extract projection features by Python scripts, the image is read to a 2D numpy array and pixel values are converted from \{0, 255\} to \{0, 1\}. The projection algorithm first loops along \(x\) axis to sum up pixel values vertically as \(X\)-projection features and then loops along \(y\) axis to sum up pixel values horizontally as \(Y\)-projection features. The extracted \(X\)-projection and \(Y\)-projection features are visualized in the bar charts on Figure 3-9. When there are more pixels in one dimension, it gets a larger value on that dimension. The value of the feature represents the change of shape in human body. As shown in Figure 3-9, the \(Y\)-projection feature is quite similar to the human posture plotted on the left. When both \(X\)-projection and \(Y\)-projection features are extracted from the image, they will be merged into a feature vector which can be finally fed to the training model. The \(X\), \(Y\)-projection features extracted from four common postures are shown in Figure 19 and Figure 20.
Figure 18 An example of extracting projection-based features

Figure 19 X, Y projection features for “Left Arm Raised” and “Right Arm Raised” postures.
The sizes of the X-projection and Y-projection feature vectors are 20 and 24 and the size of the merged X, Y projection feature vector of human posture is 44 which is much smaller than the size of the pixel feature vector (480). When using this method on MHIs, the size of projection feature vector is only 84 while the size of the previously used pixel feature vector is 1728 more than 20 times larger. Figure 21 and Figure 22 show the X, Y projection features extracted from MHIs of five common motions.
As an effective feature extraction method, the feature should contain enough information of corresponding postures or motions. Compared with the pixel feature, the projection feature which only contains a few numbers might be insufficient to train a classifier with high recognition performance. To compare the effectiveness of the two different features, both of them were tested in the experiment of posture and motion recognitions. Although the prediction performance of the classifier using projection features is a little bit lower than the one using pixel features, it is still promising as discussed in Chapter 5. Considering the tiny size of this feature vector, the projection-based feature extraction method is expected to reduce the computation workload and memory footprints. This method gives a promising solution for human action recognition system on embedded platforms.

3.4 Implementation of machine learning methods

When training examples are well prepared and converted into specified data format, they can be fed to machine learning models for training. The implementation of machine learning models can be classified into three steps:

- Determine the model of MLR, SGD-MLR and ANN
- Apply optimization techniques to improve performance
- Implement the classifier using Python scripts.

3.4.1 Determine the model of MLR

Multinomial logistic regression is a predictive analyzer which learns to describe the relationship between training examples and correct predictions. The basic structure of a MLR classifier is shown in Figure 23. Training examples in form of feature matrix are used as input data. To conduct prediction on training examples, we multiply the feature matrix with the weight matrix, add the product with the bias vector and use a softmax function on the output layer to predict target outputs. The class with the highest probability will be selected as the prediction result.

![Figure 22 X, Y projection features for "Standing", "Moving Left" and "Moving Right" motions.](image)

![Figure 23 The structure of a MLR classifier](image)
As we mentioned in the problem formalization, logistic function is also a common sigmoid function. It works well for binary classification problems. However, for multi-class classification problems, softmax function works better [43]. It can convert the values of the output vector in range of (0, 1) and the summation of all values is 1. As converted values of the logistic function are not constrained to add up to 1, each value in the output vector could be 0.99 at the same time which means that when you update the probability of one output class, it might not affect the probabilities of other outputs. Since human action recognition is multiclass problem, the softmax function is a better choice as the predictor of our classifier. Thus, we select the softmax function at the output layer of MLR model for classification. The MLR using the softmax function can be also called softmax regression [44].

To train the MLR model, we need to define the objective function that determines how well the model works and can be optimized during the training process. In machine learning, the objective function is typically called cost function which can calculate the error between the predicted value and the true value. We use cross-entropy as the cost function to measure the distance between the estimated probability distribution of predefined action classes and the correct distribution. To minimize the cost function, we use the gradient descent algorithm. As discussed in Chapter 2, gradient descent is a simple optimization procedure that updates the parameters in negative direction of the gradient to reduce the cost. This optimization procedure works iteratively until the minimum cost is reached.

Learning rate \( \alpha \) is an important parameter of gradient descent. It determines how much we update the parameters in each step. In our implementation, learning rate is optimized to reach the optimal performance for classification. The value of the learning rate \( \alpha \) is selected from the set \{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\} for each classification task and the setting is determined by the optimal performance.

### 3.4.2 Determine the model of SGD-MLR

We implement SGD-MLR based on the MLR model. Softmax predictor and cross-entropy cost function are also used in SGD-MLR model. The difference between SGD and gradient decent is the optimization procedure used during the training process. Different from gradient descent which uses all training examples in each iteration, SGD calls a mini-batch (subset) of training examples to calculate cost and update parameters. This optimization procedure is more efficient than normal gradient descent. The structure of a SGD-MLR classifier is shown in Figure 24. Instead of using all training examples, the classifier is trained by a mini-batch of training examples in each iteration.

![Figure 24 Structure of SGD-MLR classifier](image)

In our implementation, the batch size is tested in the range of \{5, 10, 20, 50, 100, 200\} to see how it affects the performance. The mini-batch selection procedure is shown below:

\[
\begin{align*}
\text{offset} &= \text{step} \cdot \text{batch size} \% (n - \text{batch size}) \\
\text{batchData} &= \text{trainDataset}[\text{offset}:(\text{offset}+\text{batch size}):]
\end{align*}
\]

where step denotes the number of training iterations, \( n \) is the size of training dataset and \( \text{batch size} \) is the predefined batch size. The offset can be considered as a random position of the mini-batch in the training set. In each iteration, SGD calculates the value of offset and then generates a mini-batch of data for training.
3.4.3 Determine the model of ANN

Multilayer perceptron (MLP) is implemented as a feedforward ANN classifier for human posture and human motion recognition tasks. In our implementation, the MLP which consists of multiple layers of neurons uses backward error propagation with stochastic gradient descent for the training process. The features of randomly selected mini-batches are fed to the input layer of MLP. Hidden layers connect the input layer and the output layer by linear transformations and non-linear activation functions. The output layer is a fully connected layer which uses softmax function to predict results of training examples. Figure 25 shows the structure of the classifier based on a 3-layer neural network.

![Figure 25 Structure of 3-layer neural network classifier](image)

In our MLP neural networks, the neuron is a Rectified Linear Unit (ReLU) which employs the rectifier as a non-linear activation function. This rectifier function can be donated by

\[
\text{Relu}(x) = \max(0, x)
\]

where \(x\) is the input of a neuron and the output is the maximum value between 0 and \(x\). This means that it only uses positive values as activations and sets all negative values to 0. Using rectifier activation function, the neural network can obtain a sparse representation that is more biologically plausible and easier for mathematical investigation [37].

The MLP based on ReLU neurons can be called rectifier networks. The feed-forward on rectifier networks is a sparse propagation of activations which is shown in Figure 26. The path selection of sparse propagation shows the non-linearity of the network that only a part of neurons are set to active depending on certain input data. The computation is linear on the selected neurons and thus the gradients flow well on the active paths [37]. As we know, the most commonly used activation functions for neural networks is the logistic function. Compared with logistic function which has exponential operations, rectifier function needs much cheaper computations. Apart from that, the rectifier function does not have the gradient vanishing problem which sigmoid function suffers from. Since the rectifier networks can reach optimal performance on purely supervised learning tasks, we select it as the target model for our human action recognition system.

![Figure 26 Sparse propagation of activations](image)
The number of layers of MLP is an important parameter which can be changed for specific tasks to create an optimal model. In our implementation, we test supervised tasks on 3-layer, 4-layer and 5-layer neural networks. When using more layers in an MLP, the accuracy could be higher but the configuration of parameters is more complex. Besides, it requires more time for recognition computation on more layer networks. In fact, the classifier using 3-layer neural networks can already get high performance on human posture and motion recognitions. This is the reason why we did not choose deeper networks. The number of nodes of the hidden layer is another important parameter for MLP. We test it in experiments to see which configuration can give us the highest accuracy.

Some other parameters need to be set for this model. Step determines how many iterations need to be conducted during the training process. It needs to be set properly to avoid under-fit and over-fit problems. Learning rate affects the learning speed of the network. It is tested in the same way as we do for MLR. For initialization of variables, we use a standard deviation method to initialize weight matrix, and set bias to zero.

The training dataset which is prepared well in the data processing phase is divided into three subsets: training set, validation set and test set. The training set is fed to the neural network for training; the validation set is used to validate the prediction accuracy during the training process; the test set is used to evaluate the final performance of the trained classifier.

### 3.4.4 Implementation of optimization techniques

Fitting the training set too well is a problem for the classifier. When the classifier is over-fitting to the training examples, it will decrease the prediction accuracy on new examples. To solve this problem, we use regularization to prevent over-fitting and improve the generalization of the classifier. Regularization is one of the most common optimization techniques. It adds a penalty term associated with weight parameters to the cost function of hypothesis. In this way, it makes a tradeoff between weight shrinking and minimum cost to find the model which has optimal prediction performance on all possible input examples. For logistic regression, the cost function with L2-regularization can be denoted by

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} Cost(h_{\theta}(x^{(i)}), y^{(i)}) + \frac{\beta}{2n} \sum_{l=1}^{m} \theta_l^2$$

Where $\theta_l$ is the $l^{th}$ weight in the vector $\theta$, $n$ is the number of training examples, $m$ is the number of target classes, and $\beta$ is the coefficient of regularization. When beta is too large, the cost function can never converge so it is under-fitting; when beta is too small, it can not prevent overfitting problem. We test beta in our experiment to find the best setting for specific supervised tasks.

Apart from regularization, dropout can be used to optimize the generalization of neural networks. During feedforward procedure, activations of neurons are calculated from front to the end. By using dropout on neural networks, it randomly chooses a set of activations on each layer as the input of next layer. In this way, the neural network never relies on any given set of activations. Gamma is the parameter of dropout that determines how much of activations we use in computation. It can be set in range of (0,1], where 1 means all activations will be used. We also evaluate dropout technique in our experiments.

### 3.4.5 Implementation of TensorFlow platform

TensorFlow [38] is a machine learning platform which provides Python APIs for various supervised learning methods. We implement TensorFlow platform on a 64-bit Ubuntu OS which is running in a Docker container. Docker [39] provides a complete filesystem with required applications, libraries, settings and dependencies and isolates a Linux operating system on windows machine for application development. TensorFlow can be installed as a Docker image which simplifies the setting of dependencies. A Docker container is created for TensorFlow image which holds everything needed for running. When running the Docker container, it launches a Jupyter Notebook which is used to edit and execute Python scripts via a web browser. Figure 27 illustrates the architecture of our development environment. By using Docker, software version control can be easily handled by creating Docker images.
images for different versions. It provides an effective and reliable development environment for our human action recognition system.

![Docker Host](image)

**Figure 27 Architecture of development environment [39]**

In TensorFlow, mathematical operations of machine learning are represented by nodes in a dataflow graph. A machine learning model can be defined as a graph which contains operation objects which represent computation units and Tensor objects which represent data units [38]. To create a machine learning model, we call a default graph and define data, variables and operations in the graph. We load training, validation and test data into constants as the input data for MLR. For SGD-MLR and ANN, the training set and label set are loaded in a placeholder which is fed with a mini-batch of data in each step of training. Then, weight and bias variables are created and initialized as modifiable tensors. For MLR and SGD-MLR, we create a training computation unit that multiples inputs with the weight variable and adds bias to it. For ANN, a forward propagation function is defined to calculate activations on each layer. The training computation unit of ANN is the forward propagation running of the training set. The cost of a model is calculated based on the training computation unit by using softmax and cross entropy functions. A L2 regularizer is added to the cost to prevent over-fitting problems. To minimize the cost of the model, an optimizer is defined based on stochastic gradient descent techniques. For ANN, the backward propagation algorithm is automatically applied on the optimizer. Prediction operations are defined for training, validation and test sets based on softmax predictor which computes the probability distribution for each dataset. To evaluate the performance of the model, we define an accuracy function that compares output results with correct labels and returns the percentage of accurate predictions.

To start the training process, we launch the model in a session which initializes all variables and schedules the running of operations. The step parameter can be set to determine how many iterations need to be done for training. In each iteration, the session runs on the optimizer, gets a mini-batch of data, calculates the cost and updates weight and bias variables. The loss and training accuracy are presented for every 100 steps followed by validation accuracy to estimate how well the model performs on prediction. When the training process is completed, test accuracy is obtained to evaluate the performance of the trained model. To investigate how the parameters like learning rate and beta affect the performance of the model, the cost and training, validation and test accuracies are recorded in numpy arrays and visualized by using matplotlib library. The learning curves of different configurations are plotted into different colors. Validation and test accuracy are analyzed to find the optimal configuration of parameters.

**3.5 Modification for Neural Networks**

By using the TensorFlow platform, machine learning models can be built and trained by labelled examples. We conducted several experiments to test the key parameters of models and try to find the
best configuration for specific learning tasks. The learning model which is properly configured and well trained can be used as the classifier for human posture or motion recognition. The created classifier running on Intel x86 platform serves as a centralized recognition system. As we mentioned in Chapter 1, Kinect cameras used on the lighting installation can be replaced by sensors which record depth images and recognize human actions. Thus, we should modify the neural network model to a distributed structure to fit on the sensor network. This is an interesting research that investigates the feasibility and performance of the distributed recognition system.

The neural network needs to be modified depending on different structures of sensor networks. To distribute data transmission and computation, the input layer needs to be separated into several parts with the same number of features. Similarly, each hidden layers is divided into smaller parts corresponding to each input parts. An input part is connected to a single part of the first hidden layer followed by more hidden layers of single parts connected with each other. Weight and bias matrices should be separated to smaller parts to fit on the separated layers. In this way, the neural network is divided into several channels of MLPs. Each channel is constructed by part of the neurons on each hidden layer. The input layer of each channel takes a part of an image as features. On the last hidden layer, hidden nodes in different channels are concatenated to a complete part fully connected with the output layer. Figure 28 shows an example of the modified neural network.

![Modified Neural Network Diagram](image)

**Figure 28** An example of modified neural networks

This is a 3-channel 5-layer neural network which has three independent channels of MLPs. Activations of neurons are propagated along each single channel using the feed-forward algorithm. On the third hidden layer, three channels are concatenated into a single channel. Then, this hidden layer is fully connected to the output layer.

We designed several modified neural network models and implement the modifications on four-class human posture recognition. We evaluate the recognition performance of each modification to verify the feasibility of these multi-channel neural networks. The modifications and evaluations are discussed in section 3.5.1.

### 3.5.1 Modifications

If four sensor nodes are used for human action recognition, the neural network model needs to be modified to a four-channel MLP structure. Figure 29 illustrates a 4-channel 3-layer neural networks based on modification I. As each sensor generates 1/4 part of the depth image, features are extracted separately from each part of training examples. Four feature matrices are created as input data to be fed
to four single MLP channels. Weight and bias matrices used between input and hidden layer are modified to four smaller size weight and bias matrices. A rectifier function is used on each channel for activation computation. The separated hidden nodes are then concatenated into one complete hidden layer which is fully connected to the output layer. During the training process, computations flow through four channels and converge into the output node. In each iteration of backward propagation, weight and bias parameters of four individual channels and the fully connected channel are all updated by partial derivatives of the cost function.

Key parameters of the modified model needs to be configured before the training process. In modification I, the number of hidden nodes is set to 16 for each channel. The regularization coefficient – beta is set to 0.005 and the learning rate is 0.5. After 10000 steps of training, it gets 100% for training accuracy, 94.5% for validation and 94.8% for test evaluation.

In the same manner, we modify the model to different structures. The configuration of different modifications are illustrated in Table 1. As can be seen in the table, these modified models have the same number of hidden nodes (64) in the hidden layers. Recognition performance of all five modifications are shown in Table 2. Modification II is designed for an 8-channel 3-layer neural networks while modification III is designed for a 16-channel 3-layer neural network. The result shows a high performance of modification II and modification III. We notice that with the same number of nodes in hidden layer, modification II with 8 channels gives higher validation and test accuracy. Modification IV is a 8-channel 4-layer neural network with 32 nodes on hidden layer 1 and 32 nodes on hidden layer 2. Figure 30 shows the structure of this neural network. After 10000 steps of training, it gets 93.2% for test accuracy which is lower than the test accuracy of modification II. In modification V, we tests a 4-layer neural network with 8 channels on input layer and 4 channels on hidden layer 1. Figure 31 shows the structure of this neural network. Each two channel on hidden layer 1 are combined to one channel and all four channels are concatenated into a fully connected channel on hidden layer 2. The result of modification V is 95.0% for validation and 94.5% for test accuracy.

We also test the performance of the original neural network model for each modified one. Each original model uses the same configuration of parameters with the modified one. After 10000 steps of training, the original 3-layer neural network model gets 95.3% for test accuracy and the original 4-layer neural network model gets 94.7% test accuracy. The result shows that the performance of the modified model is only a little bit lower than the performance of the original model. In general, the modified models show a quite good performance for human posture recognition. It demonstrates the feasibility to implement the neural network model in a distributed way.
Tabel 1: Configurations for modification I to V

<table>
<thead>
<tr>
<th></th>
<th>Modification I</th>
<th>Modification II</th>
<th>Modification III</th>
<th>Modification IV</th>
<th>Modification V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channels</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Layers</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>No. of nodes in hidden layer 1</td>
<td>4x16</td>
<td>8x8</td>
<td>16x4</td>
<td>8x4</td>
<td>8x4</td>
</tr>
<tr>
<td>No. of nodes in hidden layer 2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8x4</td>
<td>4x8</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Regularization</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>Steps</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
</tr>
</tbody>
</table>

Tabel 2: Results of modification I to V

<table>
<thead>
<tr>
<th></th>
<th>Modification I</th>
<th>Modification II</th>
<th>Modification III</th>
<th>Modification IV</th>
<th>Modification V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Accuracy</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Validation Accuracy</td>
<td>94.5%</td>
<td>94.8%</td>
<td>93.5%</td>
<td>92.8%</td>
<td>95.0%</td>
</tr>
<tr>
<td>Test Accuracy</td>
<td>94.8%</td>
<td>94.7%</td>
<td>93.0%</td>
<td>93.2%</td>
<td>94.5%</td>
</tr>
</tbody>
</table>
4 Architecture Designs

In this chapter, different architectures are designed for our human action recognition system. Functional and extra-functional requirements are explained at first. Three architectures are proposed based on Kruchten’s 4+1 view model [40] which describes a software architecture by five concurrent views: logical view, development view, process view, physical view and scenarios. As shown in Figure 32, the views address specific concerns of different stakeholders like end users, programmers, system integrators and system engineers. In addition, the scenario is used to define the functionality of the system as an additional view. The proposed architectures based on different recognition methods are compared at the end of this chapter.

4.1 Functional Requirements

The functional requirement describes what a system should do which can be data processing, calculations, communications or other system behaviors. In our case, the proposed system should be able to collect binary depth images, process data to extract features and classify them into human action categories. Functional requirements of our system are given below:

Data collecting and processing

The proposed system should collect depth images as the input data for action recognition. A Kinect camera or depth sensor can be used independently for image production. Apart from that, the sensor networks can be used as a distributed system in which each sensor only records a part of the image. The generated images cannot be directly used on the classifier. Thus, a data processing program is operated by the system to convert images to the target format. Then, a feature extractor is used to extract posture features or motion features which can be used by the classifier for recognition.

Calculation and recognition

Human action recognition is the key function that the system offers to end users. The classifier of the system takes feature vectors as input data and computes the output using the machine learning model. The machine learning model has been trained by examples previously. After computation, it outputs a probability distribution and recognize the input as belonging to one of the human action categories. The classifier can be a single program running independently or a group of programs that process the input data in a distributed way. The system which applies the first classifier can be called a centralized recognition system and the system which uses the second one can be called distributed recognition system. Different architectures are designed for these two kinds of systems.
4.2 Extra-functional Requirements

The non-functional requirement describes how the system should behave. It specifies the quality attributes of a system which can be used to judge how well a system works. For a human action recognition system, the non-functional requirements are:

Performance

Performance can be evaluated by accuracy, response time and throughput. According to Section 2.3.8, accuracy is used to evaluate the prediction efficiency of a classifier. The classifier is tested by a group of examples and the accuracy is the ratio of right predictions to all predictions. A classifier with high prediction accuracy can be considered as a reliable classifier. Response time is the amount of time the system takes from receiving a request to sending out the prediction result. It consists of the time used on image loading, data processing, recognition and transmission. A shorter response time represents a faster reaction of a system to a given input. It is usually measured in milliseconds (ms). Throughput represents the rate of processing or the rate of successful packet transmitted. A high throughput guarantees a consistent user experience on human action recognition. It is measured by number of recognition results delivered per second in our architecture design.

Availability

Availability is the proportion of time that the system is in a functioning condition. For human action recognition system, functioning conditions can be image producing, data processing, data transmission, action recognition and message delivery. Availability is usually represented by the ratio of the total time that the system is capable of being used during a given interval to the length of the interval. Specifically, the equation of availability is \( \frac{MTTF}{MTTF+MTTR} \) where MTTF is the mean time to failure and MTTR is the mean time to repair. The term unavailability is expressed as \( \frac{MTTR}{MTTF+MTTR} \) which is 100% minus availability. Typically, availability is used to evaluate the mission capable rate of the system. “Five nines” metric which equals to 99.999% is an example for high availability.

Scalability

Scalability represents the capability of a system to handle an increasing amount of work or the possibility to be enlarged to accommodate the increase [41]. For our system, the scalability can be classified in two categories:

a) Horizontally scaling: It means to add more nodes in a system to increase the computation capacity of the system. For example, adding more sensor notes to the sensor network as additional computing units.

b) Vertically scaling: It means to add resources to a single node of a system. For example, increasing the memory size or using a more powerful CPU on the host machine.

4.3 Architecture 1: Centralized classifier on Intel I5 platform

The first architecture is designed for a centralized human action recognition system running on an Intel I5 platform. It consists of three parts: a data acquisition unit, an action recognition system and a visualization tool.

- Data acquisition is handled by Kinect cameras which generate depth images and save them in a database. The database is connected to the host machine so that the depth images can be retrieved and recognized by the system.
- Action recognition system is responsible for classifying input images to common postures and motions. Data processing methods and feature extraction algorithms are specifically designed for the depth image and implemented as independent components of the system. Machine learning models created on the TensorFlow platform are used as classifiers of the system. These models are
trained by labelled examples and operated on Intel I5 platform as centralized classifiers. All recognition results are sent to the visualization tool for analysis.

- Visualization tool is dedicated to monitoring human actions and illustrating recognition results. Depth images are fetched from the database periodically and visualized on the screen with recognized action information.

When people enter the data acquisition unit, the Kinect camera extracts depth information from their interactions and creates depth images continuously. These images are stored locally in a database, and meanwhile, sent to the host machine. The host machine receives successive frames from the database and processes data into target format. The system user should set the mode to posture recognition or motion recognition and determines whether pixel feature or projection feature should be extracted from the data. Then the extracted features are fed to the classifier which conducts transformations and makes predictions. Finally, the visualization tool plots the frames on the screen and shows the recognition results.

This architecture applies a line topology where the data acquisition unit is directly connected to the host machine. The action recognition system is running on Intel I5 platform, the host machine, which provides large computation capacity and memory space. Components including the data processing program and the classifier are all accommodated by the host machine and all recognition tasks are conducted internally in the host machine. Thus, architecture 1 can provide high processing speed and low context switching time. The “4+1” views of architecture 1 are shown in figures 33 through 37.

4.3.1 Scenario view

The scenario illustrates a set of use cases as the description of an architecture. It describes the interactions between use cases and actors. As shown in Figure 33, the stakeholders of the system are participants, system engineers and system users. System engineer are responsible for management and maintenance of the system. They deal with processing algorithms, model configurations and the training process. Participants can interact with the system while system users are able to check image frames and conduct action recognitions.
4.3.2 Process view

Figure 34 shows the process view which deals with dynamic aspects of the system. It uses activity diagram to explain the system processes and illustrates the runtime activities of the system. Before running the system, the user should decide that either the posture or the motion is selected as the recognition target, which feature to use, and which classifier to apply.

![Activity diagram of architecture 1](image-url)

*Figure 34 Activity diagram of architecture 1*
4.3.3 Logical view

The logical view uses a sequence diagram to illustrate the functionality that the system provides to users. As shown in Figure 35, the objects of the system are the people, the Kinect camera, human action recognition system, and the visualization tool.

![Logical view of architecture 1](image)

Figure 35 Logical view of architecture 1

4.3.4 Development view

The development view which is also called implementation view is related to software management of the system. It uses component diagram to illustrate the system from a software programmer’s point of view. The system consists of depth sensor, data processing program, classifier, and visualization component. Figure 36 shows the development view of architecture 1.

![Development view of architecture 1](image)

Figure 36 Development view of architecture 1
4.3.5 Deployment view

The physical view, or deployment view, describes the system from a system engineer’s perspective. It shows the topology of software components and the communication between these components. As shown in Figure 37, the system contains data acquisition unit, and host machine for action recognition.

![Figure 37 Deployment view of architecture 1](image)

4.4 Architecture 2: Multi-classifier on wireless sensor network

The second architecture is designed for multiple classifiers running on embedded devices. This architecture can be considered as a modified version of architecture 1. The architecture 2 provides action recognition for several different places by using multiple classifier nodes. A classifier node can be considered as a single “host machine”. These classifier nodes are physically deployed in different rooms and each one is responsible for an independent recognition task. Each classifier node can get full-size depth image and conduct recognition on this image independently. A central node is used to analyze the recognition results of these rooms and make an actuation decision. This architecture contains two main parts: a central node and four classifier nodes. We decide to use four classifier nodes as a basic architecture for multi-classifier. It can be scaled up by adding more classifier nodes as well as the “central” nodes. In that case, these “central” nodes are actually intermediate nodes and each of them is connected with a few number of classifier nodes. Then, a new central node is used to communicate with intermediate nodes. The functionality of the central node and classifier nodes are explained below:

1. The central node, which is the master node in the sensor network, can communicate with classifier nodes and control the operation on each classifier node. At some point, the central node sends a request to a classifier node to start action recognition on that node. In this way, the central node sends requests to all classifier node and then waits until all recognition results are successfully received by the central node. These recognition results are stored in a database. When the results of all four classifier nodes are received, the actuation module makes a decision. This decision can be used to control other devices like intelligent lighting which uses the decision to change lighting modes.

2. Classifier nodes are slave nodes which are controlled by the central node for image generating and action recognition. They cannot communicate with each other and thus all recognition tasks are conducted independently on each single node. When a classifier node receives a request from the central node, it retrieves a group of depth images, executes data processing and feature extraction, classifies the images to predefined human postures or motions, and finally sends back the recognition results to the central node.
This architecture implements a star topology, so each classifier node is directly connected with the central node by unicast communication. Since sensor nodes are small, lightweight and portable, they can be easily deployed in many different environments. Besides, this sensor network is a scalable system that can be extended by adding more classifier nodes to increase recognition areas. As the process view and scenario are same with architecture one, only deployment view, logical view and development view are redesigned for this architecture. These three architecture views are shown in Figure 38, Figure 39 and Figure 40.
Architecture 3: Distributed classifier on wireless sensor networks

The third architecture is designed for distributed classifier running sensor networks. Each feature node conducts part of the feature extraction work and the classifier node uses the results from the feature nodes for action recognition. The system consists of two parts: four feature nodes and one classifier node. The classifier node is able to connect with more feature nodes. If the feature extraction work is distributed to more feature nodes, the transmission time is getting longer. Thus, we choose to use four feature nodes in this system as a basic architecture for distributed classifier. This architecture could be modified in the future work by using more sensor nodes. The communication mechanism used in this architecture can also be improved in the future. The functionality of the feature node and the classifier node is explained below:
(1) Feature nodes are slave nodes which are controlled by the classifier node. The feature node has three main components: depth sensor, data processing program and feature extractor. Each feature node is responsible for image acquisition and feature extraction of a part of the area. The feature extraction task is distributed to four channels of MLP which are deployed on four feature nodes. When a feature node receives a request from the classifier node, it conducts feature extraction and sends back the feature vector to the classifier node.

(2) The classifier node is the master node which communicates with feature nodes and classifies the retrieved data to one of the action categories. Each feature node is connected with the classifier node using wireless communications. The classifier node periodically sends requests to features nodes, and in each period, features of separated images from each feature node are received by the classifier node. Then, the distributed features are combined to one complete feature matrix and fed to the neural network classifier for action recognition.

*Figure 41 Logical view of architecture 3*
This architecture also uses star topology where all feature nodes are directly connected with the classifier node by unicast communications. Different from architecture 2 in which slave nodes operate as independent classifiers, this architecture distributes recognition tasks on the slave nodes. In this way, the size of recognizable area can be enlarged by adding more feature nodes to the sensor network. However, the classifier node should make sure that all received features from different nodes are related to the same image. Thus, communication between the classifier node and feature nodes should be carefully scheduled to guarantee the consistency. The scenario and process view are same with architecture 1 which are shown in Figure 33 and Figure 34. Deployment view, logical view and development view of this architecture are illustrated in figures 41 through 43.
4.6 Comparison of three architectures

The three presented architectures are designed for different action recognition systems. Architecture 1 is for centralized recognition system which is based on Intel I5 platform. It provides a high throughput for recognition tasks. Data processing, feature extraction and recognition can be executed quickly by powerful computing units. Besides, the waste of time on context switches is quite low, so it guarantees a short response time. Compared with the other two architectures which use embedded platforms, it is more suited to high speed recognition.

When there is a failure or system crash, it needs a certain amount of time for recovery. We assume that the software error of the system comes once a day and the recovery time for the host machine is 10 min while the recovery time for each classifier node is 5 min. It is only a simple assumption in which the recovery time of these devices can be different values in different situations. According to the equation, we can calculate the unavailability of architectures 1:

\[
\frac{10 \text{ min} \times 365 \text{times}}{365 \text{ days}} = 0.6944\% 
\]

Thus, the availability of the first architecture is 1 - 0.6944% = 99.3056%. As architecture 2 contains 4 classifier nodes, the unavailability is:

\[
\frac{4 \text{ nodes} \times 5 \text{ min} \times 365 \text{times}}{365 \text{ days}} = 1.3888\% 
\]

The availability of the second architecture is 1 – 1.3888% = 98.6112%. Since the third architecture only has one classifier node, the unavailability is:

\[
\frac{5 \text{ min} \times 365 \text{times}}{365 \text{ days}} = 0.3472\% 
\]

And its availability is 1 - 0.3472% = 99.6528%. As shown in Table 3, architecture 3 has the highest availability which is 1.0416% more than architecture 2.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Architecture 1</th>
<th>Architecture 2</th>
<th>Architecture 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>99.3056%</td>
<td>98.6112%</td>
<td>99.6528%</td>
</tr>
</tbody>
</table>

For scalability, the architecture 1 for centralized recognition system has better scalability on vertically scaling by increasing the memory size or using more powerful CPU on the host machine to get higher throughput on recognition tasks. Architecture 2 and 3 are suited to horizontally scaling as we can add more sensor nodes to the system and this can enlarge the size of recognition areas. However, when a system is scaled up with more classifier nodes, the system gets a lower availability because the overall recovery time for system can be longer. Thus, there is a tradeoff between availability and scalability of the system.
5 Experiments and Results

This chapter explains the experimental methods that were to find an optimal classifier and evaluate the results of the experiments conducted using different classifiers. Key parameters of these the classifiers are tested on Intel I5 platform to find the best configuration. To verify the modified classifiers based on architecture 2 and 3, the neural network model is redesigned in the c language and simulated on the sky notes by using Cooja. Performance evaluation is conducted for the simulated classifiers.

5.1 Experiment on Intel I5 platform

In order to train an optimal classifier for human posture and human motion, we design a series of experiments. Seven experiments were conducted on the Intel I5 platform to search for best parameter configurations for the classifier. In each experiment, the classifier is created using TensorFlow model and are trained on Intel x86 64 bit platform (Intel Core i5-4200M CPU @ 2.50 GHz) with 8GB RAM.

5.1.1 Outline of experiments

This section gives a brief introduction for experiments and explains the setups and measurements. The experiments were tested on 4-class posture recognition, 3-class motion recognition and 5-class motion recognition. Examples of four classes of postures and five classes of motions are shown in Figure 12 and Figure 13. The 3-class motion recognition is a simplified version of 5-class motion recognition. We consider “Moving Left” and “Moving Right” as the same class – “Moving”, and consider “Waving Up” and “Waving Down” as the same class – “Waving”. The third class of motion is “Standing”.

At the beginning of the experiment, labelled examples of human actions are grouped into three parts: training set, validation set and test set. The training set is a set of examples used to tune the parameters of the classifier. The validation set is a set of examples used to estimate the performance of the model during the training process. The test set is a set of examples used to assess the performance of a fully-trained classifier. Table 4 illustrates the configuration of datasets for each recognition tasks. For 3-class and 5-class human motion recognition, the training/validation/test set ratio is 80:10:10 which is a commonly used settings. For 4-class human posture recognition, as there exists big variances between different examples of the same posture, to guarantee the generalizability of the classifier, the training dataset is divided with more validation and test examples. The description of experiments and the default configuration of parameters are shown in Table 5 and Table 6.

<table>
<thead>
<tr>
<th>Recognition Tasks</th>
<th>Actions</th>
<th>No. of Examples In the Training Set</th>
<th>No. of Examples In the Validation Set</th>
<th>No. of Examples In the Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-Class Human Posture Recognition</td>
<td>Left Arm Raised</td>
<td>200</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Right Arm Raised</td>
<td>200</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Single Person Standing</td>
<td>200</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Two People Standing</td>
<td>200</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td>3-Class Human Motion Recognition</td>
<td>Moving</td>
<td>320</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Waving</td>
<td>320</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Standing</td>
<td>320</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>5-Class Human Motion Recognition</td>
<td>Moving Left</td>
<td>160</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Moving Right</td>
<td>160</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Waving Up</td>
<td>160</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Waving Down</td>
<td>160</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Standing</td>
<td>160</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>
### Tabel 5 Description of experiments

<table>
<thead>
<tr>
<th>Experiment No.</th>
<th>Brief Description</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>Train the classifier by different training steps to see how it affects the performance</td>
<td>Training, validation, and test accuracy</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>Train the classifier by different learning rates to see how it affects the performance</td>
<td>Test accuracy</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>Train the classifier by different regularizations to see how it affects the performance</td>
<td>Validation, and test accuracy</td>
</tr>
<tr>
<td>Experiment 4</td>
<td>Train the classifier by different dropout values to see how it affects the performance</td>
<td>Validation, and test accuracy</td>
</tr>
<tr>
<td>Experiment 5</td>
<td>Adjust batch size to see how it affects performance</td>
<td>Validation, and test accuracy</td>
</tr>
<tr>
<td>Experiment 6</td>
<td>Adjust number of hidden nodes in neural network and see how it affects the performance</td>
<td>Validation, and test accuracy</td>
</tr>
<tr>
<td>Experiment 7</td>
<td>Adjust number of layers of neural network and see how it affects the performance</td>
<td>Validation, and test accuracy</td>
</tr>
</tbody>
</table>

### Tabel 6 Default configuration for parameters

<table>
<thead>
<tr>
<th>Steps</th>
<th>Learning Rate</th>
<th>Beta</th>
<th>Batch Size</th>
<th>No. of Hidden Nodes</th>
<th>No. of Hidden Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>0.5</td>
<td>0.005</td>
<td>20</td>
<td>50</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.1.2 Experiment process

The details about how these experiments were conducted are explained below:

**Experiment 1**: It was conducted to test how the number of training steps affects the performance of the classifier. In this experiment, we trained the classifier using 3-layer neural networks and use different setting of training steps for each model. The steps were selected from the set \{0, 400, 800, 1200, 1600\}. Other parameters are set based on the default configuration. The performance of different settings are compared in terms of training accuracy, validation accuracy and test accuracy.

**Experiment 2**: As we know, the learning rate (alpha) determines how much the weight and bias parameters are updated in each step. In experiment 2, we train the classifiers including MLR and 3-layer neural networks by the same number of steps with different learning rate to evaluate how learning rate affects the performance. The learning rate is selected from the set \{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}. Regularization is not applied in this experiment to avoid interference. In this experiment, the performance of different settings are compared in terms of test accuracy.

**Experiment 3**: Overfitting is a serious problem that affects the generalizability of the classifier. Regularization is widely used in the fields of machine learning to solve overfitting problems. Usually, a regularization term is added to the cost function and beta controls the importance of this regularization term. Experiment 3 is conducted on the SGD-MLR classifier to test the coefficient of regularization (beta). The beta is selected from the set \{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\} and set other parameters based on the default configuration. Performances are compared to evaluate the influence of beta.

**Experiment 4**: We apply dropout as the optimization technique instead of regularization. Dropout can be used on the 3-layer neural network to prevent overfitting problems. It reduces overfitting by preventing complex co-adaptations on training data. In the Tensorflow model, we create a placeholder for the probability (lambda) that a neuron’s output is kept during dropout. We turn on dropout during training and turn it off for testing. The parameter lambda is tested for the values in the set \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}. Recognition performance of corresponding classifiers are compared.
**Experiment 5:** We build a graph with variable batch sizes for training. We create placeholders with different setting of batch sizes and train a 3-layer neural network by using different placeholders as input parameters. The batch size is selected from the set {1, 5, 10, 20, 50, 75, 100, 150, 200, 300, 400, 500}. The classifier with different settings are compared to see how batch size affects the performance.

**Experiment 6:** It is conducted to figure out how the number of hidden nodes affects the recognition performance. In this experiment, we apply a 3-layer neural network which has one input layer, one hidden layer and one output layer. The dimensions of the input layer and output layer are determined by the number of features and number of classes respectively. However, it leaves a question that how to determine the size of the hidden layer. According to some empirically-derived rules, the optimal size of a hidden layer is always between the size of input and the size of output layer. Thus, we test the number of hidden nodes from the set {5, 10, 20, 50, 75}.

**Experiment 7:** As we know, the neural network model with more hidden layers can get higher performance on prediction. In this experiment, we test the neural network model with different number of layers and compare the performance of each well trained neural networks. The experiment is conducted on 3-layer, 4-layer and 5-layer neural networks by using optimal configuration of parameters. The result is analyzed to verify if it improves the performance by adding more layers.

5.1.3 Experiment Results

In this section, the results of the experiments are presented by using graphs and tables. We measure validation and test accuracy as the performance and discuss how the key parameters affect the performance of the classifier.

Figure 44 shows the result of experiment 1. We choose pixel values as feature of training examples and test the number of training step on 4-class posture recognition task. As can be seen in the figure, when the number of steps increases, the classifier achieves higher validation accuracy and test accuracy. Training accuracy is fluctuating because it is affected by the regularization term which tradeoff training accuracy to guarantee the generalizability. When the training step is larger than 1600, the test accuracy stops increasing and stays around 94.8%. Thus, for this specific task, 1600 steps is sufficient for training the classifier.

![Image](image-url)

*Figure 44 Impact of training steps on the performance of the classifier*

In experiment 2, learning rate is tested on 4-class human posture recognition. Figure 45 shows the test accuracy of MLR model with different learning rates. The test accuracy increases faster when a higher learning rate is used. When the learning rate is set to 0.5, the performance reaches 87.5% at 2000 steps, almost keeps stably in the next 10000 steps, and finally reaches 88.0%. However, when the learning rate is set to 0.001, the performance increases slowly and only gets 76.0% at 12000 steps. The result indicates that the classifier set by smaller learning rate needs more steps for training. After that, we apply the 3-
layer neural network model to find the optimal value of the learning rate. The resulting test accuracy for each learning rate is listed in Table 7. When the learning rate (alpha) is set to a large value like 0.5, the model is unable to find the optimal spot so the resulting test accuracy keeps stably at 90.7%. When the alpha is set to a small value like 0.001, the training process is quite slow and needs more than 12000 steps. We find that 0.1 is a good value for learning rate that reaches the best test accuracy (91.2%).

![Impact of learning rate on the test accuracy](image)

**Figure 45 Impact of learning rate on the test accuracy**

<table>
<thead>
<tr>
<th>Training Step</th>
<th>Alpha = 0.5</th>
<th>Alpha = 0.1</th>
<th>Alpha = 0.05</th>
<th>Alpha = 0.01</th>
<th>Alpha = 0.005</th>
<th>Alpha = 0.001</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>90.0%</td>
<td>89.2%</td>
<td>87.5%</td>
<td>86.5%</td>
<td>86.2%</td>
<td>75.3%</td>
</tr>
<tr>
<td>4000</td>
<td>90.5%</td>
<td>91.0%</td>
<td>87.5%</td>
<td>86.3%</td>
<td>86.3%</td>
<td>81.2%</td>
</tr>
<tr>
<td>6000</td>
<td>90.5%</td>
<td>91.2%</td>
<td>87.3%</td>
<td>86.7%</td>
<td>86.3%</td>
<td>81.3%</td>
</tr>
<tr>
<td>8000</td>
<td>90.7%</td>
<td>91.2%</td>
<td>87.3%</td>
<td>86.7%</td>
<td>86.3%</td>
<td>81.8%</td>
</tr>
<tr>
<td>10000</td>
<td>90.7%</td>
<td>91.2%</td>
<td>87.3%</td>
<td>86.8%</td>
<td>86.3%</td>
<td>81.8%</td>
</tr>
<tr>
<td>12000</td>
<td>90.7%</td>
<td>91.2%</td>
<td>87.3%</td>
<td>86.8%</td>
<td>86.3%</td>
<td>81.8%</td>
</tr>
</tbody>
</table>

In **experiment 3**, we test the parameter of regularization (beta) on 3-class human motion recognition by using SGD-MLR model. Motion history image and projection feature extraction method are used on the training examples. Figure 46 shows the test accuracy of the classifier using different beta values. When beta is set to a large value like 0.5, the classifier suffers from under-fitting problem so the test accuracy is only 75.8%. If the beta is a small value like 0.001, it can hardly prevent the over-fitting problem and only reaches 78.3%. We find 0.005 is an optimal value for beta which leads to 85.8% test accuracy on 3-class motion recognition.

Figure 47 illustrates the result of **experiment 4**. The parameter of dropout (lambda) is tested on 4-class human posture recognition task by using a 3-layer neural network with pixel features. When lambda is too small like 0.2 or 0.1, the classifier is under-fitting to the training examples. Thus, it results in low validation and test accuracy. When lambda is set to 0.7, it reaches the best performance – 95.3% for test accuracy.

The result of **experiment 5** is listed in Table 8. Batch size is tested on 3-layer neural network for 3-class motion recognition problem using pixel features. When the batch size is set to an extremely low value like 1, the classifier is under-fitting and only gets 33.3% for test accuracy. For other batch sizes, the classifier get similar performance about 95% ~ 97%. We find 10 is a good value for batch size that leads to the optimal test accuracy (97.5%) on this specific recognition task.
Figure 46 Impact of beta on the performance of the SGD-MLR classifier

Figure 47 Impact of dropout on the performance of 3-layer neural networks

Table 8 Comparison of various batch sizes on validation and test accuracy

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Validation Accuracy</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.3%</td>
<td>93.3%</td>
</tr>
<tr>
<td>5</td>
<td>94.2%</td>
<td>95.0%</td>
</tr>
<tr>
<td>10</td>
<td>95.0%</td>
<td>96.7%</td>
</tr>
<tr>
<td>20</td>
<td>95.0%</td>
<td>95.0%</td>
</tr>
<tr>
<td>50</td>
<td>95.0%</td>
<td>95.0%</td>
</tr>
<tr>
<td>75</td>
<td>95.0%</td>
<td>94.2%</td>
</tr>
<tr>
<td>100</td>
<td>94.5%</td>
<td>95.8%</td>
</tr>
<tr>
<td>150</td>
<td>94.5%</td>
<td>95.8%</td>
</tr>
<tr>
<td>200</td>
<td>95.0%</td>
<td>95.8%</td>
</tr>
<tr>
<td>300</td>
<td>95.0%</td>
<td>95.8%</td>
</tr>
<tr>
<td>400</td>
<td>95.0%</td>
<td>95.8%</td>
</tr>
<tr>
<td>500</td>
<td>95.0%</td>
<td>95.8%</td>
</tr>
</tbody>
</table>

The table above shows the comparison of various batch sizes on validation and test accuracy. The validation and test accuracy values are given for different batch sizes ranging from 1 to 500.
Experiment 6 and experiment 7 are conducted on 5-class human motion recognition to test the optimal number of hidden nodes and the optimal number of hidden layers in neural network model. The result for these experiments are listed in Table 9. As can be seen from the result, the neural network with more hidden nodes can get higher performance. For a 4-layer neural network, the best configuration is [75,50] for hidden nodes which leads to the optimal performance – 88% test accuracy. However, when we add one more hidden layer to the neural network, the performance does not improve a lot. In addition, tuning a 5-layer neural network is much tougher because of the various possible combinations of parameters. Thus, the 4-layer neural network is sufficient for this recognition task.

<table>
<thead>
<tr>
<th>No. of Hidden Layers</th>
<th>No. of Hidden Nodes</th>
<th>Validation Accuracy</th>
<th>Test Accuracy</th>
<th>No. of Hidden Layers</th>
<th>No. of Hidden Nodes</th>
<th>Validation Accuracy</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>75.0%</td>
<td>72.0%</td>
<td>1</td>
<td>50</td>
<td>89.0%</td>
<td>83.0%</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>84.0%</td>
<td>81.0%</td>
<td>1</td>
<td>75</td>
<td>90.0%</td>
<td>84.0%</td>
</tr>
<tr>
<td>2</td>
<td>10, 5</td>
<td>80.0%</td>
<td>76.0%</td>
<td>2</td>
<td>50, 20</td>
<td>87.0%</td>
<td>86.0%</td>
</tr>
<tr>
<td>2</td>
<td>20, 10</td>
<td>79.0%</td>
<td>83.0%</td>
<td>2</td>
<td>75, 50</td>
<td>84.0%</td>
<td>88.0%</td>
</tr>
<tr>
<td>3</td>
<td>20, 10, 5</td>
<td>77.0%</td>
<td>78.0%</td>
<td>3</td>
<td>75, 20, 10</td>
<td>85.0%</td>
<td>86.0%</td>
</tr>
<tr>
<td>3</td>
<td>50, 20, 10</td>
<td>82.0%</td>
<td>85.0%</td>
<td>3</td>
<td>75, 50, 20</td>
<td>87.0%</td>
<td>88.0%</td>
</tr>
</tbody>
</table>

By doing these experiments, we learn how the parameters of the machine learning model affect the training process and the resulting performance. Generally, using more number of steps for training, the classifier can get higher performance. Learning rate is an important parameter that determines the learning speed and affects the final performance of the classifier. For 4-class human posture recognition, the optimal learning rate is different for different machine learning models. For the coefficient of regularization (beta), we find 0.5 as an optimal value for both posture and motion recognition tasks. Differently, the coefficient of dropout (lambda) has a wider range of good values so it is much easier to be tuned. To some extent, adding more hidden layers and using more hidden nodes can increase the potential performance of the neural network model. However, a deeper network needs to be tuned more elaborately to achieve performance improvement. For our recognition tasks, 3-layer and 4-layer neural network models are sufficient for high recognition performance.

5.2 Simulation on embedded platform

In this section, we conduct the simulation of human action recognition system on embedded platforms. The neural network classifier is modified and implemented in the c programming language and deployed on sensor nodes. Different architectures are verified in simulation including architectures 2 and 3 from Chapter 4. This chapter introduces the tools and notes used in the simulation, explains the simulation setup and tasks, and analyzes the results of simulation tests.

5.2.1 Simulation tool and node selection

To simulate a Wireless Sensor Network (WSN) for the classifier, simulators like Cooja, NS2, NS3, etc can be used. As sensor nodes should be able to transmit real data and load data from files, Cooja is the best choice for us. Cooja is a WSN simulator based on Contiki OS [42] which is an open source operating system for the Internet of Things (IoT). Contiki provides low-power Internet communications. Contiki applications are written in standard C and the development of these applications is easy and fast. Applications can be emulated on the Cooja simulator before burned into hardware. Thus, the classifiers designed in our simulation can be implemented and tested on real devices in the future work.

For nodes that has an external memory chip, Contiki provides a lightweight flash file system – the coffee file system. It supports applications to open, read from and write to files on the external flash. In order to deploy the neural network model on the sensor nodes, the node should be able to load weight and bias parameters from files. Besides, the node should be supported Cooja simulator. Therefore, we select Tmote Sky as the target node for simulation. Tmote Sky is a low power wireless node which is the only
kind of node that is supported by both Cooja and coffee file system. The configuration of Tmote Sky is listed in Table 10.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>MSP430 8MHz</td>
</tr>
<tr>
<td>RAM</td>
<td>10k</td>
</tr>
<tr>
<td>Flash memory</td>
<td>48k</td>
</tr>
<tr>
<td>Transceiver</td>
<td>CC2420</td>
</tr>
<tr>
<td>Radio</td>
<td>IEEE 802.15.4</td>
</tr>
<tr>
<td></td>
<td>250kbps 2.4GHz</td>
</tr>
</tbody>
</table>

5.2.2 Network setup

There are two communication stacks in Contiki: uIP and Rime. Applications can use either or both of them. Rime is a set of lightweight networking protocols for low-power wireless networks. It provides primitives ranging from single-hop broadcast, single hop unicast to network flooding. The Rime stack supports simple operations like sending a message to a specified neighbor by using unicast. As we apply star topology in the architecture design, one sender and multiple receivers are used in the sensor networks. We decide to connect these sensor nodes by unicast communications. To be specific, the unicast adds a destination address to the packets on the sender side. If a receiver does not match the packet’s destination address, the packet is discarded. So only the receiver assigned by the same address can successfully receive the message. In this way, the synchronization is simplified by scheduling the order of unicasts on the master node.

The structure of the Rime communication stack is shown in Figure 48. As we choose to use the Sky node, cc2420 is selected for the physical layer. The Radio Duty-Cycle (RDC) layer handles sleep period of nodes. It decides when a packet is transmitted and makes sure that a node is awake to receive an incoming packet. We decide to use “nullrdc” that always keeps the radio transceiver awake and never switches it off. Compared with the default setting, the “contikimac”, it reduces the reaction time of the nodes so that the system can get higher throughput on recognition. Contiki OS has two types of MAC layer protocols: “csma” and “nullmac”. The second one is a simple pass-through protocol that simply calls RDC functions. In our simulation, we use the default setting “csma” which supports collision avoidance on MAC layer. Figure 49 shows the initialization of the Sky node. Information including MAC, RDC, radio channel, node address and id are printed out.

![Figure 48 Structure of Rime stack](image-url)
In our simulation, the transmission range and the interference range are both set to 20 meter. The success ratio for transmission and reception are set to 100%. Therefore, a message sent from a node can be successfully received by the target node if there is no collision in the radio.

5.2.3 Simulation design

In order to test the performance of the classifier on embedded platforms, we design simulation tasks for each specific architecture of the classifier. These architectures are implemented by using modified neural network models and tested on 4-class human posture datasets. First, we try to create a neural network model on sensor nodes using pixel features as input data. The weight matrix which contains a large number of float variables requires large memory space on the sensor node. However, Sky node has limited memory space and breaks down because of memory overflow. Therefore, we decide to use projection feature instead of pixel feature. The size of the weight matrix is much smaller than the previous one and is able to be saved on the sensor nodes now.

The neural network model needs to be trained before deploying on the sensor networks. The parameters of neural networks should be stored after training. A 3-layer neural network used for the first three simulation tasks uses four matrices as parameters: weights1 and bias1 for hidden layer, and weights2 and bias s2 for output layer. We save these matrices in different files. By using the coffee file system, the float variables of a parameter can be read from the file and are saved in a float array. In this way, the neural network model is successfully deployed on the sensor nodes. Apart from that, the rectifier function and the predictor of the neural network are implemented as well as the data processing and feature extraction algorithms by using C language.

5.2.4 Simulation tasks

This section introduces the simulation setup and process for each simulation task.

**Simulation task 1:** It is designed for a simple case: one-to-one architecture. In this architecture, a slave node is used as the classifier and a master node is used to control the recognition process. At the beginning of simulation, the slave node loads the neural network model from files by using the coffee file system. Then it is set to ready mode waiting for messages from the master node. Each time the master node gets command from the user, it sends a message by unicast to the slave node. Then, the slave node initiates the recognition task. To be specific, it reads a depth image from the flash memory, conducts feature extraction, and makes a classification using neural networks. After that, the recognition result is loaded to packet buffer and sent back to the master node. When the master node receives the result, it prints it out and send a new request to the slave node. This process runs iteratively until the recognition task is finished on the slave node.

**Simulation task 2:** It is designed for architecture 2 proposed in chapter 4. This architecture uses four slave nodes as single classifiers and one master node as the coordinator in the sensor network. The topology of this sensor network is shown in Figure 50. At the beginning of the simulation, all four slave nodes load neural network models from files. When the master node gets a command from the user, it sends a unicast to one of the slave nodes. The target node recognizes a depth image in one of posture categories and sends back the result. Then, the master node prints out the received result and send a new request to another slave node. The master node uses a timer scheduler function to control the operation of each slave node. The simulation keeps running until recognition tasks on all slave nodes are finished.
Simulation task 3: It is designed for architecture 3 proposed in chapter 4. This architecture uses four slave nodes as feature extractors and one master node as the classifier. It applies star topology on the sensor network, the same with Figure 50. At the beginning of the simulation, the master node loads all parameters from files. When it gets a command from the user, it sends requests to all slave nodes. Then, each slave node reads one fourth part of a depth image, conducts feature extraction and sends back the features to the master node. When the master node receives features from all slave nodes, it combines them into one feature matrix, conducts the recognition tasks and prints out the result. After that, it sends requests again to all slave nodes. This process is controlled by a time scheduler, and it continues until the recognition task is finished.

5.2.5 Simulation Results
In this section, the results of simulation tasks are presented by graphs and tables. We analyze the results and evaluate the performance and quality attributes of each architecture.

The result of simulation task 1 is listed in Table 11. We can see that the setup time for the classifier is 10.138 s and the average operation time for a single recognition task is about 114 s. The average response time for each frame is approximately 0.69 s which means that the recognition speed is about 1.45 Frame Per Second (FPS). The speed can be higher by using the sensor node with more computation capacity. As the people may not moves a lot within one second, the recognition speed at 1.45 FPS is quite acceptable. We get high recognition performance for each class posture: 95.3% for “Right Arm Raised”, 92% for “Left Arm Raised”, 98% for “One Person Standing” and 96.7% for “Two People Standing”. The average performance is 95.5% on the entire test set.

<table>
<thead>
<tr>
<th>Test set</th>
<th>No. of Examples</th>
<th>Right Predictions</th>
<th>Wrong Predictions</th>
<th>Accuracy</th>
<th>Total Recognition Time (s)</th>
<th>Time Per Frame (s)</th>
<th>Frame Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right Arm Raised</td>
<td>150</td>
<td>143</td>
<td>7</td>
<td>0.95333333</td>
<td>103.971</td>
<td>0.6931</td>
<td>1.4427</td>
</tr>
<tr>
<td>Left Arm Raised</td>
<td>150</td>
<td>138</td>
<td>12</td>
<td>0.92</td>
<td>107.414</td>
<td>0.7161</td>
<td>1.3965</td>
</tr>
<tr>
<td>One Person Standing</td>
<td>150</td>
<td>147</td>
<td>3</td>
<td>0.98</td>
<td>94.914</td>
<td>0.6328</td>
<td>1.5802</td>
</tr>
<tr>
<td>Two People Standing</td>
<td>150</td>
<td>145</td>
<td>5</td>
<td>0.96666667</td>
<td>108.42</td>
<td>0.7228</td>
<td>1.3835</td>
</tr>
<tr>
<td>Average</td>
<td>150</td>
<td>143.25</td>
<td>6.75</td>
<td>0.955</td>
<td>103.680</td>
<td>0.6912</td>
<td>1.4468</td>
</tr>
<tr>
<td>Total</td>
<td>600</td>
<td>573</td>
<td>27</td>
<td>-</td>
<td>414.719</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

In simulation task 2, the recognition result is printed out and shown in Figure 51. As we can see from the figure, node 5.0 is the master node which sends requests in each 4s and node 1.0, 2.0, 3.0, 4.0 are

![Figure 50 Topology of the sensor network](image-url)
slave nodes which conduct recognitions in each 1s. The slave nodes are tested by four test sets and the performance is analyzed by confusion matrix, as shown in Figure 52.

The diagonal items in the confusion matrix indicate the number of correctly recognized postures. The sum of each row is the total number of test examples which is 150 for each test set. It is interesting to note that the classifier can perfectly differentiate “Right Arm Raise” from “Left Arm Raised” which means that an example of “Right Arm Raise” may never be recognized as “Left Arm Raised”. This also works on “One Person Standing” and “Two People Standing”. We notice that “One Person Standing” gets the highest test accuracy (98%) but at same time it get 16 wrong predictions for other test sets. We think the reason is that when it is ambiguity between “Left” and “Right” it is more likely to be predicted as “One”. Finally, the overall performance for this architecture is 95.5% and the recognition time for each frame is 0.99s. The speed can be higher by reducing the waiting time of the node. However, to ensure that all results can be printed out successfully, we deliberately set a longer waiting time. The result shows that this architecture is quite feasible that created classifiers can get high performance on posture recognition tasks with acceptable response time.

In simulation task 3, we test a distributed classifier that uses four nodes for data acquisition and feature extraction. The simulation result is listed in Table 12 and the performance of the classifier is shown in Figure 53.
According the result in Table 14, we notice that the recognition time for 150 examples is longer than the time used in simulation task 1 and task 2. This is because that it costs longer time to send the intermediate data. The average response time for each frame is about 1.12s and the corresponding throughput is about 0.89 FPS. The result shows that when we only distribute feature extraction on slave nodes, the average recognition speed is a little bit lower than the single classifier simulated in task 1. From the confusion matrix, we can see that “Right Arm Raised” and “Left Arm Raised” get the highest test accuracy (96.67%). We also notice that “One Person Standing” and “Two People Standing” are more likely to be wrongly predicted to “Left Arm Raised” which is 15 for “One” and 11 for “Two”. The average performance is 93.83% on the entire test set.

Overall, the simulated classifiers have a high performance for human action recognition tasks. Architecture 1 and 2 reach 95.5% average test accuracy and architecture 3 reaches 93.8%. The average response time for architecture 1 and 2 is about 1s for each training example. This is quite acceptable considering the low computation capacity of the sensor nodes. However, architecture 3 needs to be improved in terms of response time. This could be achieved by using more powerful sensor nodes or applying better communication mechanisms in the future work. The simulation result shows that the simulated action recognition systems basically fulfill the functional and extra-functional requirements of the proposed architectures. These simulation tasks provide a good example for the future work with respect to how to choose simulation tools and nodes, how to configure the network, and what performance measures can be used.
6 Conclusion and Future work

This chapter concludes the thesis work, evaluates the proposed action recognition systems, discusses our observations during the project, and finally gives some recommendations for the future work.

6.1 Conclusion

The main contributions of this thesis are:

- Design efficient image processing and feature extraction methods to build training datasets
- Implement machine learning models to create classifiers for human action recognitions
- Modify the neural network model and implement it in a distributed way
- Design architectures for centralized and distributed recognition systems
- Simulate the classifiers on the sensor network and evaluate the performance

The first task of the thesis work is to investigate human action patterns and process training examples. By using the visualization tool, the most common postures and motions are found. Examples of postures and motions are quickly selected and labelled by using the labelled tool. An image processing method is designed to improve the performance of posture recognition. The MHI is implemented to properly represent the human motion. For feature extraction, a projection based feature is presented to efficiently extract high level features from pixel values. Compared with the pixel feature, it reduces the computation workloads and the memory footprints of recognition process.

Secondly, the classifiers based on machine learning models are successfully created and trained by using the TensorFlow platform. For both posture and motion recognitions, the trained classifiers show high performance in terms of validation accuracy and test accuracy. Seven experiments are conducted on Intel I5 platform and the optimal configurations of the key parameters are found based on each specific recognition tasks. Several modifications are designed for the neural network model to implement the classifier in a distributed way. We evaluate the performance of the modified neural network models, and it shows a high recognition accuracy for human posture and motion recognitions.

This thesis presents the architecture designs for centralized and distributed recognition systems and evaluate them in terms of extra-functional requirements including the performance and scalability. The designed architectures are simulated on the sensor network using the Cooja simulator and the Sky nodes. We evaluate the simulation tasks by using the confusion matrix and the result shows a quite good recognition accuracy for the distributed recognition system. However, as the Sky node has limited computation capacity, the distributed recognition system has lower throughput and longer response time compared with the centralized one. This problem can be solved in the future work by using more powerful devices and applying better communication mechanisms in the system.

To recapitulate, the experiments, modifications and simulations in this thesis serves as a precursor to developing a real-time high-scalable and power efficient action recognition system.

6.2 Future work

There are some ways to improve the system in the future work:

- The proposed system can classify input data to predefined posture and motion categories. In our implementation, four classes of the human posture and five classes of the human motion are defined as the most common patterns. This is because that other human postures are rarely appeared and human motions like “Sitting” and “Raising Legs” cannot be captured due to the deployment of Kinect cameras. In the future work, we recommend the researchers to design more typical human postures and motions, and create training examples by themselves.
- In our experiment, the pixel feature and the projection based feature are used as the feature of the human posture and motion. Complex feature extraction methods are not applied because of the
specific data of the given dataset. Besides the binary-value depth image that is used in our project, the gray-scale depth image and the RGB image can be also used as the input data. Apart from that, investigation on new features of the posture and motion can be conducted to design better methods for feature extraction.

- The action recognition system is tested for off-line recognition tasks. The result shows a promising performance for the human action recognition. In the future work, this system can be improved for real-time recognition tasks which require high throughput, low latency and fast processing speed. Some more powerful CPUs or GPUs can be used as the platform of the centralized classifier. Powerful sensor nodes can be used to establish distributed classifier.

- In this thesis work, three architecture are designed for centralized and distributed recognition systems. In the future work, the researchers can design new architectures which has more sensor nodes as distributed computing units with more complex network topology. For example, the proposed 16-channel 4-layer neural network model can be deployed on larger sensor network which uses 16 slave nodes for low-level activation calculation, uses 4 intermediate nodes for mid-level activation computation and uses a master node connected with intermediate nodes as the classifier of the system.

- In the simulation tasks, the sensor network is configured based on the Rime protocol. Sensor nodes use unicast to communicate with each other. In the future work, different communication mechanisms can be implemented and compared on the sensor network to find the optimal network configuration. Different network protocols like Ethernet or WIFI and different MAC protocols like CSMA/CD or CSMA/CA can be tested on the sensor network. Apart from simulations, the researchers can also use real devices to test the scalability and power efficiency of the system.
Bibliography


Appendix A

Figure 54 Human action visualization tool

Figure 55 An efficient labelling tool