Artificial Neural Network Design Approaches to Multi-Channel Information Analysis

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by

Jaehoon Cha

October 2020

Abstract

In recent years, a large amount of *multi-channel data* has been collected due to advances in technology such as with computers and the Internet. However, obtaining and labelling data are still laborious and time-consuming. Yet another issue that adds to the difficulty is finding important channels and features from *multi-channel data* since having enough channels alone does not guarantee designing efficient algorithms due to scalability problems. In this thesis, a generative model and hierarchical learning models are introduced to deal with the aforementioned issues.

First, the learning process of Variational Autoencoders is analysed. Taking into account the role of the mean and the standard deviation, which are used in the reparameterization trick, we propose a new generative model. The proposed model is modified from the original Autoencoder architecture which is used for dimensionality reduction. The model preserves the architecture of the Autoencoder by removing the reparameterization trick and becomes a generative model by extension of the mapping of the decoder from a discrete latent space to a continuous latent space. The model is compared with VAE and MMD on three benchmark datasets: MNIST, Fashion-MNIST and SVHN datasets. The experimental results show that the difference of the accuracy of the test set when training ANNs using synthetic data generated by the proposed model is less than 10% when training it using the original training set in MNIST and Fashion-MINST datasets. In addition, further experiments are carried out to investigate the impact of the number of the training set when training generative models. The results show that the accuracy of the test set decreases less than 10% when the number of the training set decreases in the NNIST and the Fashion-MNIST dataset.

Second, two types of hierarchical learning models are proposed. Designing these models began with the idea of utilizing an innate hierarchy of targets. The first type of model, HAL, is proposed when targets are discrete. This model involves inserting the auxiliary block to output the auxiliary scores from the coarse classes. These scores are distributed based on the corresponding coarse classes. Although the model improves the accuracy of a test set, it has the disadvantage of requiring the coarse classes at the test phase. The second type of models are proposed when targets are continuous. C-FNNs and HADNNs are proposed to perform the regression task by utilizing the coarse classes. C-FNNs and HADNNs are evaluated on three benchmark indoor localization datasets, examples of *multi-*

channel data. Results show that C-FNNs increase the floor accuracy by 30% at least and 60% at most in the three datasets. However, C-FNNs require more than three times the parameters than the baseline. HADNNs achieve better accuracy than C-FNNs and require 1.2 times the parameters than the baseline at most.

Third, human motion data is analysed in order to show the importance of the relationship between sensor locations and motion types when identifying motion types. The data were gathered from patients and students in Inha University Hospital, Korea. Twenty-three subjects participated in the experiment and all had to perform nine motion types. Forty-eight total measurements were obtained from eight different body parts. The motion type detection algorithm is divided into five steps and is evaluated based on four metrics: recall, precision, accuracy and F-measure. The proposed detection algorithm has 0.8986 average recall, 0.9071 average precision, 0.9739 average accuracy and 0.8977 average F-measure. The detection algorithm outperforms PCA, which is a popular method in feature extraction. This shows the importance of feature extraction based on the relationship between channels and targets in *multi-channel data*.

Finally, the motion type detection process is proposed by integrating the proposed models. The process is divided into three: generation, labelling and classification. In generation, the proposed generative model is used to generate synthetic data. In labelling, SVM and PCA are used to label synthetic data. In classification, ResNet with C-FNNs and with HADNNs for a classification task are trained using the combination of the labelled synthetic data and the original training set, and the neural networks are used to detect motion types. The process is evaluated using InhaMotion and nine open source human motion datasets. The results show that training ANNs with synthetic data prevents overfitting, and the proposed generative model outperforms VAE, β-VAE and MMD. In addition, the combination of ResNet and C-FNNs increase the accuracies of the test sets when coarse classes are available during the training phase. Since C-FNNs do not require coarse classes at the test phase, it is practical to use in daily life problems where hierarchy of targets should be considered.

Acknowledgements

This PhD thesis is the greatest present to me in the last four years. I would like to thank the people who helped me in completing this present.

Thank you to my supervisor, Dr. Sanghyuk Lee, who I am sincerely indebted and grateful to. He created the perfect environment for my successful PhD. I might have failed to settle in the field of engineering if not for his help. I will never forget everything he gave me.

Thank you to Dr. Kyeongsoo Kim, who taught me the qualifications and responsibilities of researchers. His advice and encouragement were of great assistance to completing my thesis. I will forever keep all the lessons I've learnt from him in mind.

Thank you to Dr. Thiyagalingam Jeyarajan, for encouraging me to continue with my topic when I was on the verge of changing it. The conversations I had with him in the early stages of my PhD significantly affected the direction of my thesis. His endless help always removed my anxiety and worries. It's always a pleasure to work with him.

I would also like to take this opportunity to express my thanks to my co-supervisors, Dr. EngGee Lim and Dr. Angel Garcia Fernandez. Their guidance and warm heart broadened my perspectives on life and my research.

I especially appreciate the outstanding committee members, Dr. Ka Lok Man, Dr. David O'Connor, and Dr. HeuiSeok Lim, for finding the time to assess my thesis. All their pointed questions, comments, and advice were of immense help in the improvement and outcome of my thesis. I thank them for conducting remote viva under unprecedented circumstances.

I also wish to thank my IPAP members, Dr. Mark Leach and Dr. Danushka Bollegala, for reviewing my annual report. Their questions during our annual IPAP meetings contributed to the successful progress of my research on this thesis.

I also wish to thank Dr. Kaizhu Huang, for the opportunity to attend his group seminar about machine learning which opened my eyes to what it is capable of. To Dr. Sangmin

Lee, Professor, Department of Electronic and Electrical Engineering, Inha University, for allowing me access to the human motion dataset. And Dr. Keunho Ryu, retired professor and now honorary professor in Chungbuk National University for providing me the opportunity to join his group to learn the basics of machine learning.

Thank you to Dr. Jinhae Park, Professor, Chungnam National University, because of whom I began pursuing research. During my master's studies, he introduced me to studying abroad that, until then, had never crossed my mind and guided me to achieve it. I might not have tried to study abroad if I did not have his guidance.

Thanks you to Dr. Changhyun Jun, who I would like to say is my mentor. He has always motivated me and influenced me positively. I've been really luck to have met him during my PhD.

I also wish to thank Dr. Hangyun Chu, for informing me of the opportunity to study engineering, and Dr. Hyungchul Chung and Dr. Moonkeun Kim, for advising me when needed during my PhD.

Thanks to my PhD friends, Bing, Shufei, and Haochuan, who helped make my life in China easier. I quickly got used to and adjusted to China thanks to them. Also, to Inho, who was my only Korean friend in China. I might have felt lonely hadn't I met him. I have many good memories of China thanks to him. I also thank my PhD friend Jason. He gave me good memories in the United Kingdom.

Special thanks to Mayet, my teacher and old friend. Thank you for always listening to my anxiety and worries. I would like to tell you that you played a huge role in completing my PhD.

Thank you to my parents, who gave me their unending support. They have always believed in me and assured me that I could do anything. Knowing that they always trust and support me no matter what, I was able to confidently go my way.

Last but not the least, I would like to thank my life partner, Sohyun, for standing by me. She shared all of my emotions during my PhD. She was happy when I was happy, sad when I was sad, and annoyed when I was annoyed. I owe everything to her and I am ready to start a new phase of my life with her.

I also would like to extend my gratitude to everyone who has, knowingly or unknowingly, directly or indirectly, helped me in the successful completion of my PhD.

Jaehoon Cha

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the development of technologies such as computers and the Internet, a vast range of data has become more accessible and manageable within various fields. This widespread availability allows researchers to design equations and models based on a large number of data characteristics [5, 41, 89]. Nevertheless, some researchers still choose to extract key information to design these equations and models. However, rather than extracting key information, it makes more sense to combine independent information and heterogeneous datasets so that we may benefit from the abundance of available data [36]. Throughout this thesis, the term *multi-channel data* is used to refer to data obtained from multiple channels or sources that are independent. Scientific research on human motion type detection, indoor localization, recommendation system and other research areas are carried out by processing *multi-channel data* [15, 101, 73]. For human motion type detection, measurements come from the head, hands, and legs, which are considered different channels. These measurements can be used to determine how much care a patient requires. Another example of a *multi-channel data* application is research on indoor localization using Wi-Fi signals. The importance of localization inside buildings has been further emphasized by the increasing number of multiplex buildings. The characteristics of the Global Positioning Systems (GPS) do not allow for accurate indoor positioning, making the use of GPS impractical. Wi-Fi routers, however, are positioned within buildings, making localization using Wi-Fi signals a more sensible indoor localization alternative. These routers work independently from one another, thereby making the derived data *multi-channel data*. Nevertheless, data processing by integrating independent channels presents difficulties because the significance of each channel is different. This can be solved by assigning a value,

which we will refer to as weight henceforth, to each channel according to its significance. Even so, another challenge still remains in designing functions which can integrate the weighted channel.

As mentioned before, even if a large amount of data is accessible, some researchers still choose only a few key pieces of information to solve problems to reduce uncertainty and conflict from probably irrelevant information. With the advent of machine learning methods, it becomes practical to use as much information as possible. Machine learning is the methodology used to find the optimal function of given datasets. The process of finding the optimal function is called the training phase. Parameters in machine learning algorithms are randomly set before the training phase, and the goal of machine learning is to tune the parameters, referred to as learnable parameters, to get the optimal function based on a training set, a set consisting of samples used during the training phase. The methods to achieve this goal generally consist of two major tasks, one of which is feature extraction and the other could be classification, regression or clustering using the extracted features. Features are extracted by removing redundant information. On the other hand, classification and regression are types of supervised learning where the targets are known, *i.e.*, data in the training set are composed of pairs of inputs and targets. The goal of supervised learning is to predict targets corresponding to its inputs. Nonetheless, classification and regression differ in their type of targets. The former has discrete targets, while the latter has continuous targets. Meanwhile, data clustering is considered to be unsupervised learning because it does not have any targets. Data clustering takes into account the similarity of inputs to create clusters. These two major tasks can be combined using the cutting-edge machine learning method called Artificial Neural Networks (ANNs). These ANNs are inspired by the function of the human brain and consist of a minimum of three layers [24]. The first layer is called an input layer and the last layer is called an output layer. Multiple layers, which are called hidden layers, exist between the input layer and output layer. Data are fed to the input layer, pass through the hidden layers, and reach the output layer. Hidden layers have learnable parameters and are tuned to represent features. Tuned parameters contain the important features of a training set and enough information to perform a given task. ANNs have been applied to various fields including computer vision, speech recognition, recommendation system and medical diagnosis, leading to exceptional results in each of the fields.

However, there are still some difficulties in applying ANNs to fields where the amount of data is inadequate. Because training ANNs relies absolutely on a training set, it is

important to collect a sufficient amount of data [24]. Furthermore, the performance of machine learning including ANNs is evaluated based on a test set, a set that is not used during the training phase. ANNs should use the training set to learn the general features of the training set. With a small number of training samples, ANNs alongside all the other methods included in machine learning, will learn only a few parts of the important features. To address this, generative models can be used as they are powerful tools when only a small number of training samples are provided.

A generative model is a type of unsupervised learning that does not require pairs of inputs and targets. Generative models are specialized for dimensionality reduction, feature extraction and new sample generation [29]. The parameters obtained after the training phase represent the hierarchical level representation of the inputs and are able to generate synthetic data which are convincingly drawn from the training set. Because it is difficult to make pairs of inputs and targets when a tremendous amount of data is given, in Chapter 3, we design a new generative model and propose an algorithm to make a set of labelled synthetic data.

Another issue of conventional ANNs is that they consider all targets equally, *i.e.*, the similarities between targets are not considered and all targets are equally distributed. In other words, traditional ANNs do not consider the hierarchy of targets and this hierarchy cannot be learnt during the training phase. However, *multi-channel data* may have their own innate hierarchical structure depending on channel information, which should be considered at the test phase. Considering hierarchy of targets makes ANNs logical by tuning parameters to represent features of inputs and to capture hierarchy of targets at the same time. ANNs can learn the hierarchy of targets by either taking hierarchical information as inputs or adjusting the loss function with respect to the hierarchy. In Chapter 4, hierarchical learning models based on hierarchical information are studied and new types of models are proposed.

In Chapter 5, before analysing *multi-channel data* with the use of the proposed models, we analyse human motion data as an example of *multi-channel data*. This data is a typical example of *multi-channel data* because the data is collected from multiple channels, *i.e.*, sensor locations in human motion data, and is difficult to obtain due to both the complexities in the approval process and the gathering of subjects. We selected human motion data as an example of *multi-channel data* because improving the health care system for the elderly and the sick is an exceedingly important social issue. Furthermore, the evaluation of the physical ability of patients, those who have difficulties with mobility, is

required prior to rehabilitation treatments. Without standardized evaluation methods, the ability level of a patient is given a grade through a subjective evaluation by one of many varying methods or people. It goes without saying that overestimation of ability can cause injuries and underestimation can result in a delay in rehabilitation [52]. In this regard, it is necessary to objectively detect and evaluate physical ability. In order to explore motion type detection using sensors, human motion data obtained from Inha University Hospital, Korea are analysed in Chapter 5. Eight sensors are attached to each human subject and these subjects are required to perform nine motion types with respect to their abilities. We hierarchically group these nine motion types based on their relationship with the locations of the sensors. In Chapter 6, this hierarchical group information is applied to the hierarchical learning models proposed in Chapter 4.

Finally, the proposed models in Chapter 3 and Chapter 4 are applied to human motion data as examples of *multi-channel data* in Chapter 6. In order to overcome the issue of difficulty gathering a sufficient amount of data to deal with a variety of samples formed by a combination of multiple channels, the generative model proposed in Chapter 3 is used to generate synthetic data. The proposed model is trained on a training set and generates synthetic data which are plausibly drawn from the distributions of the training set. Subsequently, they are labelled using Support Vector Machine (SVM) and Principal Component Analysis (PCA). The labelled synthetic data are used together with the training set when training ANNs, which take measurements of human motion data as inputs and detect motion types. Successfully trained generative models are able to generate synthetic data quite similar to the data in the training set. These synthetic data allow ANNs to capture important features and to avoid bias towards the training set. In addition, human motion types can be grouped according to their relationships with multiple channels. The hierarchical group structure of human motion types, studied in Chapter 5, is used to better identify motion types by use of hierarchical learning models. To validate the proposed models in motion type detection, human motion data obtained from Inha University Hospital, Korea and nine open source human motion datasets are used. We carry out four experiments with or without synthetic data and with or without hierarchical grouping of *multi-channel data*.

This thesis is organized as follows. First, the previous works related to this thesis are introduced in Chapter 2. In Chapter 3, generative models are analysed in detail and a new generative model is proposed. These models are tested on three benchmark datasets and further evaluated on a portion of the datasets in order to investigate the impact

of the amount of data on the performance of the models. In Chapter 4, two types of hierarchical learning models are introduced to utilize an innate hierarchy of targets. The first type of the proposed model shows the importance of learning hierarchy of targets. It leads us to propose the second type of models which can be used for *multi-channel data*. The proposed second type of models are evaluated with indoor localization datasets as examples of *multi-channel data*. In Chapter 5, human motion data are analysed to study the relationship between sensor locations and motion types. In Chapter 6, a total of ten human motion datasets, including the human motion data analysed in Chapter 5, are used with the proposed models from Chapter 3 and Chapter 4. Finally, conclusions and future research interests are summarized in Chapter 7.

Chapter 2

Literature Review

Since we have emphasized the importance of the research on *multi-channel data* analysis and capturing features of data through machine learning, we briefly summarize previous research in related fields. This thesis studies three subject matters. First, generative models are studied to generate more various samples that are not included in a training set. Second, hierarchical learning models are proposed in order to utilize an innate hierarchy of targets. Third, the relationships between motion types and the locations of sensors attached to different human body parts are studied.

2.1 Generative models

There are two large families of generative models in ANNs. One type is Variational Autoencoders (VAEs) and the other is Generative Adversarial Networks (GANs). There are two models of GANs: the generator model which provides new samples, and the discriminator model which classifies samples as either real (from a training set) or fake (generated). Although GANs are able to generate synthetic images which are indistinguishable from real images, VAEs have drawn the attention of many researchers due to a key drawback in the functional training of GANs [39, 93]. This difficulty in training GANs arises because of the zero-sum game between its two models, *i.e.*, when one model reduces its loss, the loss of the other increases [25]. VAEs circumnavigate this issue by introducing a probabilistic encoder and a probabilistic decoder [28, 40, 71]. The probabilistic encoder is a model which compresses given data x into z with respect to ϕ, which is denoted by $q_\phi(z|x)$. This model is referred to as the encoder model. On the other hand, the probabilistic decoder is a model

which reconstructs x from z with respect to θ and is denoted by $p_\theta(x|z)$. This model is referred to as the decoder model. The architecture of VAEs is based on Autoencoders (AEs), the goal of which is dimensionality reduction and the architecture of which consists of an encoder and a decoder. In AEs, the encoder reduces the dimensions of the samples, *i.e.*, the encoder maps $\mathbb{R}^n$ to $\mathbb{R}^m$ where $n \gg m$, and the decoder operates in reverse.

Two big differences between AEs and VAEs are in how they deal with the latent space, the space between the encoder and the decoder. The first difference is that VAEs apply Kullback–Leibler divergence (KL divergence) denoted by $D_{KL}(\cdot||\cdot)$, to compare the similarity between two distributions: $q_\phi(z|x)$, the distribution of the latent space (*i.e.*, the outputs of the encoder), and $p_\theta(z|x)$, the prior distribution (*i.e.*, given distribution before the training phase). $D_{KL}(\cdot||\cdot)$ plays a crucial role as a regularizer to ensure that outputs of the encoder map onto the prior distribution. The second difference is that VAEs use the reparameterization trick to learn sampling, the random selection from the distribution of the latent space to generate data. The encoder model learns the mean and the standard deviation of the distribution of the latent space which are required for sampling. Noting the fact that KL divergence is non-negative, the variational lower bound on the marginal likelihood of x is proposed by

$$- D_{KL}(q_\phi(z|x)||p_\theta(z|x)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)], \tag{2.1}$$

where $\mathbb{E}[\cdot]$ is the mean. Finally, ϕ and θ are learnt by maximizing the variational lower bound or minimizing the negative of the variational lower bound, which is a loss function of VAE, to approximate ($q_\phi(z|x)$ to $p_\theta(z|x)$. The advantage of maximizing the variational lower bound compared to optimizing loss functions of GANs is more reliable [39].

With the great success of the VAE proposed in [39], many new VAE variants with improved performance have emerged. These variants attempt to find a complicated and flexible distribution of the latent space that corresponds to the distribution of the samples.

In this section, we explore some of the variants of the VAE in [39]. For example, Improved Variational Inference with Inverse Autoregressive Flow attempts to find flexible distributions [40]. One idea of normalizing flow is to start off with an initial random variable with a relatively simple distribution, then apply a chain of invertible parameterized transformations to find a more flexible distribution. In order to start off with a simple distribution, the initial encoder model outputs μ_0 and σ_0, which are the mean and standard deviation of the latent space, respectively. Then, latent variables are drawn with the

reparameterization trick:

$$z_0 = \mu_0 + \sigma_0 \odot \epsilon, \tag{2.2}$$

where $\odot$ is an element-wise multiplication operator and $\epsilon \sim \mathcal{N}(0, I)$. The flow consists of a chain of T of the following transformations:

$$z_t = \mu_t + \sigma_t \odot z_{t-1}, \tag{2.3}$$

where $t = 1, \cdots, T$. Because $\frac{\partial \mu_t}{\partial z_{t-1}}$ and $\frac{\partial \sigma_t}{\partial z_{t-1}}$ are triangular with zeros on the diagonal and $\frac{\partial z_t}{\partial z_{t-1}}$ is triangular with σ_t on the diagonal, the determinant of $\frac{\partial z_t}{\partial z_{t-1}}$ becomes $\Pi_{i=0}^{D} \sigma_{t,i}$ where D is the dimension of the latent space. Finally, the density after the final iteration is:

$$\log q(z_t|x) = -\sum_{i=1}^{D} (\frac{1}{2}\epsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T} \log \sigma_{t,i}). \tag{2.4}$$

Apart from the learning strategy used to learn more flexible distributions, a new variational lower bound is proposed to enforce meaningful representation learning in the latent space. β-VAE introduces a single hyper-parameter β to improve the degree of disentanglement in the learnt representation in the latent space [28]. The difference between β-VAE and the VAE is the multiplication of β to $D_{KL}(q_\phi(z|x)||p_\theta(z|x))$ and so the variational lower bound becomes:

$$-\beta D_{KL}(q_\phi(z|x)||p_\theta(z|x)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] \tag{2.5}$$

and this new variational lower bound encourages the model to learn the most efficient representation of the data. The improved performance of β-VAE is shown with the disentanglement metric proposed in [28]. β-VAE shows that the simple modification of the variational lower bound improves the ability of VAEs in terms of representation in the latent space regardless of the flexible distribution. In addition, the problem of the variational lower bound of the VAE is tackled in [105]. The variational lower bound of the VAE itself causes amortized inference of failures and the information preference property, which both lead to an imbalance in learning between the distributions of the samples and of the latent space. In order to overcome this problem, a new objective of variational lower bound is proposed in [105]:

$$-\lambda D_{KL}(q_\phi(z)||p_\theta(z)) - \mathbb{E}_{q_\phi(z)} D_{KL}(q_\phi(x|z)||p_\theta(x|z)) + \alpha I_q(x; z), \tag{2.6}$$

where $I_q(x;z)$ is the mutual information between x and z under distribution $q_\phi(x,z)$. In order to efficiently optimize the objective the equivalent form is derived using Maximum-Mean Discrepancy (MMD), which measures the distance between two distributions [105]. Finally, the neural network is trained to optimize the following objective

$$\begin{aligned}\mathbb{E}_{P_D(x)}\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - (1-\alpha)\mathbb{E}_{P_D(x)}D_{KL}(q_\phi(z|x)||p(z)) \\ - (\alpha+\lambda-1)D_{MMD}(q_\phi(z)||p(z)),\end{aligned} \tag{2.7}$$

where $P_D(x)$ is the true distribution and $\alpha < 1$, $\lambda > 0$ and

$$\begin{aligned}D_{MMD}(q||p) =& \mathbb{E}_{p(z),p(z')}[k(z,z')] \\ &- 2\mathbb{E}_{q(z),p(z')}[k(z,z')] + \mathbb{E}_{q(z),q(z')}[k(z,z')]\end{aligned} \tag{2.8}$$

with any positive definite kernel, $k(\cdot,\cdot)$. Henceforth, MMD is referred to as MMD-VAE for the convenience.

In Chapter 3, we introduce a new generative model similar to VAEs, as well as analyse and compare existing models with image datasets.

2.2 Hierarchical learning

Conventional ANNs for classification previously had not considered the structure of target classes despite the fact that most target classes in real life problems have a hierarchy. For example, images from daily life can be divided into living organisms and inanimate objects, drugs can be structured by parent/child relationships and locations can be grouped under big or small areas. Unlike conventional ANNs, hierarchical learning is proposed in a way that allows for the use of additional information, also called auxiliary information, to utilize the hierarchy of targets. The auxiliary information can be easily drawn from either original data or an innate hierarchy of targets and this is done by means of defining the hierarchical classes of the targets.

In this section, techniques that utilize the hierarchy of targets as auxiliary information or by architectures of neural networks are introduced. Before introducing the previous techniques, we define the term coarse class and fine class. Throughout this thesis, we call the original targets which are either discrete or continuous as fine classes. Sets containing the fine classes are referred to as coarse classes if the sets are disjoint and their union contains all fine classes. For example, let Y be a set of original targets. A partition of Y

can be defined by taking subsets of Y, $C_1, \cdots, C_n$, such that

(1) $C_i \neq \phi$ for all $i = 1, \cdots n$,

(2) $C_i \cap C_j = \phi$ if $i \neq j$,

(3) $\cup_{i=1}^{n} C_i = Y$.

Then, we refer to element in Y as fine classes and to elements in the partition as the coarse classes. The coarse class can be defined with multiple levels as shown in Figure 2.1. In stage-wise learning, coarse to finer images which are subsampled from original images

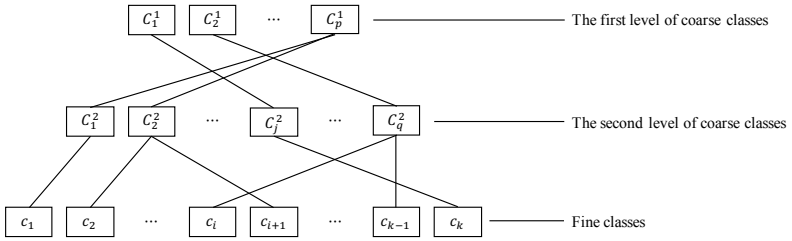


Figure 2.1: Relationships between coarse classes with two levels and fine classes when fine classes are discrete targets. The node is a link between a superior class and an inferior class.

are fed to a neural network step by step to enhance the learning process [3]. At the beginning stage, only coarse image information is given to this neural network, forcing its first few layers to learn the global structures of the samples in the training set. Fine image information is provided to this neural network as the stages progress. This strategy forces the neural network to learn features more hierarchically, which sometimes neural networks fail to achieve due to hyper-parameters.

The residual auxiliary block architecture introduces an auxiliary block which can perform multiple tasks, outputs of which improve the performance of a main task [56]. This auxiliary block extracts useful information from samples in a training set and inserts the information into a neural network as an input of the main task [56]. Apart from this main task, the neural network is able to learn more meaningful representations of the training set.

Approaches which directly utilize hierarchical class have also been studied. For example, [8] designs a sequence of multi-layer perceptrons (MLPs). Each MLP sequentially learns coarse classes through its rear layer taking outputs of the preceding layer. Because

the neural network consists of a number of MLPs proportional to its number of coarse classes, scalability problems are encountered. In addition, as the number of MLPs increases, *i.e.*, the number of layers increases, the information of the first few layers rarely reaches the last layer.

Another approach that can be taken is that coarse category layers and multiple fine category layers follow shared layers [98]. These shared layers learn the common properties of samples in a training set. After the training phase, weights of the shared layers are used for the initialization of the coarse category layers and multiple fine category layers. In order to build these layers, labels are grouped manually and the number of groups is set to the number of elements within the coarse category. Then, the coarse category layers learn about the coarse category. Following the coarse category layers, multiple fine category layers are built. Each fine category corresponds to one element in the coarse category. Because multiple fine category layers are connected in parallel, the number of layers from an input layer and an output layer is much smaller than the proposed network in [8]. However, this model still has a scalability issue since the number of fine category layers is proportional to the number of elements in the coarse category.

B-CNN outputs coarse classes in the middle of the architecture of a neural network [106]. The layers which outputs coarse classes are called branches and the outputs of the branches are compared with target coarse classes. If there are more than two levels in the coarse classes, the branch closest to the input layer learns the first level of the coarse classes and the branch closest to the output layer learns the lowest level of the coarse classes. The final loss function is then defined as the weighted sum of the losses over all branches, and afterwards the neural network is forced to learn hierarchically according to the structure of the hierarchy of targets.

Besides changing neural network architectures, a new loss function is proposed to utilize the hierarchy of targets. In [96], an ultrametric tree is proposed based on the semantic meaning of all targets and uses the hierarchical target information. The probability of each node of the ultrametric tree is the sum of the probabilities of its leaves and all its nodes that are on the path from the leaves to the node.

Auxiliary inputs are used to check logical reasoning in [88]. Auxiliary inputs based on human knowledge are provided to a neural network to allow it to learn logical reasoning. The neural network verifies the logical information with the auxiliary inputs first and then proceeds to the next stage.

All of these hierarchical techniques have not only been applied to image classification,

but also to many other fields where the hierarchy of targets can be easily obtained [104, 38]. In Chapter 4, we use indoor localization as an example of *multi-channel data* because it allows for an easier application of a hierarchical approach in indoor localization when applying ANNs.

2.3 Machine learning application to motion type analysis

Motion types can be captured from various sources such as from videos and Inertial Measurement Unit (IMU). Research on motion type detection can benefit a number of diverse fields. For example, improvements can be made in the performance of athletes, the rehabilitations of patients and the prevention of falls in the elderly [15, 67].

The relationship between body parts and mobility has been studied for a long time. Several tests have been designed to measure human mobility based on the movements of body parts [82, 6]. In these tests, subjects are required to perform a few motion types such as standing up, sitting down and walking. Experts in these tests then score the subjects' mobility. Diagnoses and advice are given by these experts based on the subjects' scores. These approaches help with the rehabilitation of patients. Nevertheless, because the score is subject to experts and tests, efforts to find a universal approach for motion type detection are continuing to develop. In order to find this universal approach, standard ways to describe motion types and automatic motion type detection methods should be established. As opposed to the fact that standard ways to describe motion types have already been proposed [12], designing the automatic motion type detection methods remains an ongoing challenge.

In the early stage of research on automatic motion type detection methods, basic statistical values of measurements from IMU sensors were used to detect motion types [85]. Motion types such as standing, sitting, lying, walking, ascending stairs, descending stairs and cycling are detected using IMU sensors attached to the sternum and thighs. The sensors measure three accelerometers and three gyroscopes. The mean and the standard deviation of the measurements were used to detect motion types. Due to a lack of complicated computational methods, but a great development in sensor techniques, the authors in [12] manually separated motion types and set thresholds in order to distinguish motion types from one another. In [59], vertical displacements are computed to discriminate the difference between standing-after-sitting and sitting-after-standing. Both motion types showed the same movements from the back since subjects stretched their backs after

bending. The difference was that sensors on subjects' backs were moving upward when standing-after-sitting and moving downward when sitting-after-standing. Because these actions were completed very quickly, the key was to find the time period where the action happened. The authors in [59] applied the Wavelet Transform (WT) to remove noise and so they could observe clear up and down changes and found the right time period which could be used to measure the height change. In [1], 19 daily and sports motion types (sitting, standing, lying down on the back and on the right side, ascending and descending stairs, standing in an elevator still and moving around in an elevator, walking in a parking lot, walking on a treadmill with a speed of 4 km/hr, running on a treadmill with a speed of 8 km/hr, exercising on a stepper, exercising on a cross trainer, cycling on an exercise bike in horizontal and vertical positions, rowing, jumping and playing basketball) were classified with Bayesian Decision Making (BDM), the Least-Squares Method (LSM), the k-Nearest Neighbor algorithm (k-NN), Dynamic Time Warping (DTW), Support Vector Machines (SVM) and ANNs. Rather than using raw measurements, the authors extracted multiple features: the minimum and maximum values, the mean values, variance, skewness, kurtosis, ten equally spaced samples from the autocorrelation sequence, the first five peaks of the discrete Fourier transform of the signal and the corresponding frequencies. Because subjects wore sensors on five body parts and each sensor produced nine measurements, Principal Component Analysis (PCA) was used to further extract meaningful features. All machine learning algorithms based on features extracted by PCA worked well to distinguish the 19 motion types.

As wearable devices with sensors have become readily available in terms of price, usability and data acquisition, and machine learning algorithms have become easier to use, a lot of research has been carried out on diverse machine learning algorithms in order to detect motion types [15]. The advantage of the use of machine learning algorithms is that researchers do not need to find important measurements and equations. Nevertheless, research on the relationship between sensor locations and motion types will help in understanding human motion data. Therefore, in Chapter 5, human motion data obtained from Inha University Hospital, Korea, is analysed.

Chapter 3

Data Augmentation Approach

The trade-off between bias and variance is one of the biggest challenges in machine learning from the viewpoint of data reliance [4]. Imbalances in handling this trade-off causes either overfitting or underfitting. In overfitting, the trained machine learning algorithms nearly perfectly match with a training set, but show rather poor matching with a test set. Meanwhile, underfitting occurs when the algorithms do not match with both a training set and a test set. The problem of underfitting can be easily solved by increasing the number of parameters, while overfitting cannot be easily solved [4]. This problem was widely studied so that the performance on a test set would improve [45, 70]. Recent research in ANNs has shown that one of the most effective ways to solve overfitting is through data augmentation [14, 77]. This regularizes ANNs making them able to work well with a test set. Data augmentation is additionally essential in fields where big data acquisition is still difficult such as in time series data and medical fields, where data gathering takes lengthy periods of time [19]. In addition, data augmentation can play a crucial role in analysing *multi-channel data*. This is because analysis of *multi-channel data* requires a large amount of data due to a lot of possible samples through a combination of multiple channels. In this chapter, we present generative models which can augment data beyond simple data augmentation techniques such as cropping, flipping and rotating [77]. The chapter is divided into four sections discussing the following: VAEs, the proposed generative model, experiments to verify the proposed model and the summary.

3.1 Variational Autoencoders

In this section, we analyse VAEs, one of the big families under generative models. We begin to cover the architectures and benefits of VAEs compared to GANs. Then, we discuss our observation about the role of reparameterization trick.

3.1.1 Generative models

Generative models can be used to generate new samples that convincingly could have been drawn from a training set through automatically learning patterns of samples in the training set. In fact, these models can learn the distribution of the samples in a latent space allowing us to obtain new samples, *i.e.*, the models learn an unlabeled distribution $P(X|Z)$ where X is the space including the training set and Z is the latent space. These models are popular in computer vision since realistic fake images can be generated. The great success in generating these images is evidenced by the fact that researchers demand fake image detection algorithms [61].

GANs are the second of the big families under generative models in ANNs. GANs have achieved vast successes in generating new images. Nevertheless, the architecture of GANs is only briefly introduced in this chapter. The focus, instead, is on VAEs since the loss function of GANs is difficult to train due to the zero-sum game between its two models mentioned in Chapter 2,whereas the loss function of VAEs is reliable since it has the lower bound [39].

Generative Adversarial Networks (GANs)

There are two models of GANs: the generator model that generates new samples and the discriminator model that attempts to classify samples as either real (from a training set) or fake (generated). The architecture of GANs is shown in Figure 3.1.

The generator model takes in fixed-length vectors which are drawn randomly from the Gaussian distribution as its inputs and generates synthetic samples. These synthetic samples, along with samples from the training set, are provided to the discriminator model and are classified as real or fake. The discriminator model is then updated to improve its skills in telling real from fake, and the generator model is updated based on the performance of the discriminator model. The generator model then generates samples to fool the discriminator model. In other words, both the discriminator and the generator models

are attempting to obtain the upper hand, which explains why training GANs is known as the zero-sum game and why training GANs is difficult [25]. After the training phase, the discriminator model is discarded because the main objective of GANs is to generate new samples, *i.e.*, the generator model is only required during the test phase.

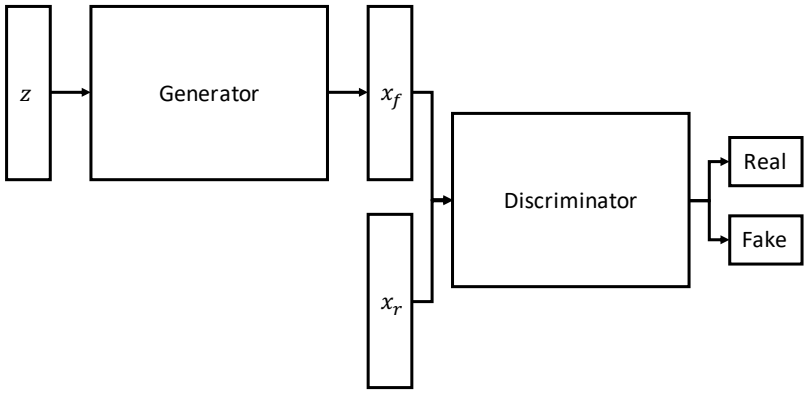


Figure 3.1: Architecture of the Generative Adversarial Networks. z is an fixed length vector, x_f is a fake data and x_r is a real data.

Variational Autoencoders (VAEs)

The architecture of VAEs is similar to that of AEs, which consist of the encoder and the decoder networks. The main objective of AEs is dimensionality reduction [29]. This is done by the opposite roles of the encoder and the decoder networks. The encoder network reduces the dimensions of samples in a training set. The decoder network then enlarges the latent variables, the outputs of the encoder network. By comparing the inputs and outputs of the decoder network, key features are learnt in the latent space. However, the objectives of VAEs are slightly different from those of AEs. The encoder network of VAEs takes samples from the training set and reduces the dimensions in a process similar to that of AEs. The difference is that VAEs learn not only the key features, but also the distribution of the samples in the latent space [39]. Under the assumption that most samples from real life follow the Gaussian distribution, the encoder learns the mean and standard deviation of samples within the latent space. Next, the decoder network generates new samples by sampling based on the learnt mean and standard deviation. The architecture of VAEs is shown in Figure 3.2. In order to apply back-propagation, a technique to reduce the error of a loss function [72], sampling is substituted with the reparameterization trick, which

adds the standard deviation multiplied by the noise to the mean, *i.e.*, the latent variable, z, is obtained by $z = \mu + \sigma \odot \epsilon$, where μ, σ, and ϵ are the mean, the standard deviation, and Gaussian noise, respectively. In order to make the encoder network output the mean and the standard deviation of the distribution of the latent space, KL divergence is used to compare the distribution of the latent space and the prior distribution. Finally, VAEs are trained by optimizing the sum of KL divergence and the discrepancy between inputs and generated outputs. With the lower bound of KL divergence, VAEs are reliably trained and appear theoretically more well-defined than GANs. In the following training phase, the encoder network is discarded and the Gaussian distribution, which is used as the prior distribution during the training phase, is used to sample latent variables.

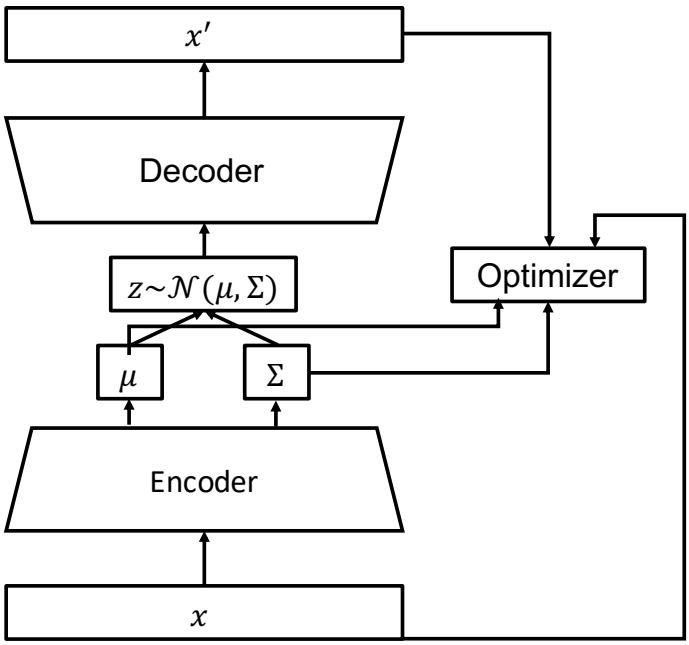


Figure 3.2: Architecture of the Variational Autoencoder. x is a sample in the training set and z is a latent variable. μ and $\sum$ are the mean and standard deviation, respectively

3.1.2 Differences between Autoencoders and Variational Autoencoders

Two points of VAEs differentiate VAEs from AEs, allowing VAEs to be generative models while AEs cannot. The first is that the loss function of VAEs is the sum of the reconstruction loss and KL divergence, whereas the loss function of AEs only has the reconstruction

loss. The reconstruction loss in VAEs plays the same role as in AEs which computes the difference between the outputs of the decoder network and the inputs of the encoder network. The outputs of the decoder network become similar to the inputs of the encoder network, and the learnable parameters in the hidden layers of the encoder network learn key information. At the same time, the decoder network learns to reconstruct inputs based on the key information.

The other metric for VAEs compares the distribution of the latent space with the prior distribution with the end goal of making them similar. The most popular metric for the comparison of the two distributions is KL divergence. In addition, the Gaussian distribution, of which the mean is zero and the covariance matrix is equal to the identity matrix, is used for the prior distribution to improve computational efficiency. A notable effect of using KL divergence to match the distribution of the latent space to the Gaussian distribution is the centralizing of the outputs of the encoder network around the mean.

The second difference between VAEs and AEs appears in the reparameterization trick. In VAEs, generation works by feeding samples, randomly selected from the latent space, to the decoder network. If a random sampling in statistics is used during the training phase, the encoder and the decoder networks are disconnected, which results in a failure to deliver the reconstruction loss to the encoder network by back-propagation [39]. In order to solve this problem, the reparameterization trick was proposed [39]. The reparameterization trick requires the mean and standard deviation to mimic sampling in statistics; hence, the encoder network learns the population parameters of the distribution of the latent space. Since the Gaussian distribution is used as the prior distribution, the encoder network now learns its population parameters: the mean and standard deviation, which are denoted by μ and σ, respectively. Then, a new sample, z, is computed by the reparameterization trick as follows:

$$z = \mu + \sigma \odot \epsilon, \tag{3.1}$$

where $\epsilon \sim \mathcal{N}(0, I)$. The reparameterization trick allows for the delivery of the reconstruction loss to the encoder network by back-propagation. Now, we further cover the necessity of the reparameterization trick.

3.1.3 The role of the Reparameterization trick

The encoder network outputs the mean and standard deviation, both of which are used to perform the reparameterization trick. Let us use x, μ and σ to, respectively, denote an

input, the corresponding mean output and the standard deviation output of the encoder network. If the reparameterization trick is not used, the decoder network reconstructs x from μ.

In AEs, a set of all μs from a training set forms a set of latent variables, which are both dimensionally reduced and are clustered based on the similarity of their inputs. However, KL divergence differentiates between AEs and VAEs. KL divergence forces latent variables to form the standard Gaussian distribution [105, 48]. In other words, latent variables are distributed centered at zero with a standard deviation of one. Because the number of samples in a training set is finite, the number of μs corresponding to all inputs is also finite. This means that there are variables in the latent space which do not correspond to any inputs during the training phase. Thus, it is impossible to reconstruct samples using a decoder network from randomly selected latent variables in the latent space.

VAEs solve this issue by the use of the reparameterization trick. The trick allows VAEs learn population parameter, μ and σ, through back-propagation. In other words, VAEs learn μ to represent all inputs in the latent space. However, we note the fact that each μ has its corresponding input and σ in Eq. (3.1) plays a role in weighting the $\epsilon \sim \mathcal{N}(0, I)$. This implies that VAEs learn μ to represent each input and it is highly possible that $z = \mu + \sigma \odot \epsilon$ is located nearby μ. Nevertheless, VAEs become generative models by comparing the output of the decoder of z with x, which proves that the closer variable z is to μ, the more similar the output of z is to x. From that fact, we observe that the reparameterization trick makes VAEs generative models by mapping latent variables around μ to x rather than by mimicking sampling.

3.2 Analysis on Autoencoder and designing a new generative model

In this section, we show the continuity of AEs and propose a new generative model by extension of the mapping of the decoder network from a discrete latent space to a continuous latent space, one not using the reparameterization trick.

3.2.1 Continuity of the Original Autoencoder

Let us assume that one layer of a neural network consists of a set of a matrix multiplication, an addition and an activation function. We define a function $h : X \to Y$, given by

$$h(x) = f(\boldsymbol{A}x + b) \tag{3.2}$$

where $\dim(X)=n$, $\dim(Y)=r$, $\boldsymbol{A} : \mathbb{R}^n \to \mathbb{R}^r$ is a matrix operator, b is a 1-dimensional vector of r components, and f is an activation function such as Softplus, sigmoid, hyperbolic tangent (tanh) or rectified linear unit (ReLU). Then, $x_n \to x$ implies $h(x_n) \to h(x)$ from the fact that

$$\|h(x_n) - h(x)\| = \|f(\boldsymbol{A}x_n + b) - f(\boldsymbol{A}x + b)\| \tag{3.3}$$

$$\leq \|\boldsymbol{A}(x_n - x)\| \tag{3.4}$$

$$\leq \|\boldsymbol{A}\|\|x_n - x\|, \tag{3.5}$$

where the matrix multiplication is bounded. In addition, all the activation functions considered in ANNs satisfy the following inequality

$$\|f(x_n) - f(x)\| \leq \|x_n - x\|. \tag{3.6}$$

Consequently, the layer consisting of a set of a matrix multiplication, an addition and an activation function is continuous. Due to the fact that a composite of continuous functions is continuous, a composite of these layers is also continuous.

Now, we let $f : X_{in} \to Z$ and $g : Z \to X_{out}$ be the encoder and the decoder networks which are composite functions of layers and Z be the latent space. If $y=g(f(x))$ and $y'=g(f(x'))$ for any $x, x' \in X_{in}$, then

$$\|y - y'\| \leq c_g\|f(x) - f(x')\| \leq c_f\|x - x'\|, \tag{3.7}$$

where c_f and c_g denote the product of the norms of projection matrices in the encoder and the decoder networks, respectively. From Eq. ((3.7)), we can expect that, for any $z, z' \in Z$, if the distance between z and z', is small, then the distance between the resulting outputs of g — *i.e.*, $g(z)$ and $g(z')$ — is also small. Therefore, if $\|z - z'\| \leq \varepsilon$ and $z = f(x)$, then the outputs of z' must be similar to x because $\|g(z) - g(z')\|$ must be small and $y = g(z)$

is a reconstruction of x.

3.2.2 Extending the decoder network from a discrete latent space to a continuous latent space

Note that the outputs of latent variables around z must be similar to x by the continuity of AEs. From this fact and the role of the reparameterization trick mentioned in Section 3.1.3, we aim to design a new generative model by extending the mapping of the decoder network from a discrete latent space to a continuous latent space. In fact, the mapping of the decoder network of latent variables around z by the reparameterization trick is very analogous to the partition of unity, which is defined as follows:

Definition 3.2.1. *[84] A partition of unity of a space X is a set of $\mathcal{S}$ of continuous functions from X to the unit interval $[0, 1]$ such that*

1) the finite number of functions in $\mathcal{S}$ of a neighborhood of x has zero,

2) $\sum_{p\in\mathcal{S}} p(x) = 1$, for all $x \in X$.

In other words, there are finitely many continuous functions, p_i in $\mathcal{S}$, such that $p_i(x) = 0$ for $i = 1, \cdots, N$ and the sum of $p(x)$ for all $p \in \mathcal{S}$ is one. Partition of unity extends a locally defined mapping to a mapping on an entire space. Now, we apply the concept of partition of unity into our problem with AEs. Let us assume that there are N pairs of (x_i, z_i) by the encoder network – *i.e.,* the output of the encoder network of x_i is z_i – and the decoder network of the AE, which we denote by D_{AE}. Assume that D_{AE} perfectly reconstructs the finitely latent variables z_i to x_i. Our goal is to define a new decoder network which can reconstruct any of the latent variables z to one which is convincingly drawn from the distribution of the training set. Now, let us define a function D_{z_i} which maps z to x such that

$$D_{z_i}(z) = \begin{cases} D_{AE}(z_i), & \text{if } z = z_i, \\ \mathbf{0}, & \text{otherwise.} \end{cases} \tag{3.8}$$

where $\mathbf{0}$ is a zero-matrix. Let $\hat{D(z)} = \sum_i D_{z_i}(z)$, then $\hat{D} = D_{AE}|_{z=\{z_1,\cdots,z_N\}}$ completely reconstructs from the latent variables $\{z_1, \cdots, z_N\}$ to samples in the training set. As most samples in real life follow the Gaussian distribution, it is natural that z around z_i happens

following $\mathcal{N}(z_i, \sigma)$ for some $\sigma > 0$. To design a new decoder network with a continuous latent space, we define a weighting function and a neighbor set.

Definition 3.2.2. *A weighting function, w_{z_i}, with respect to z_i, is a function which gives weight to z according to its distance from z_i by*

$$w_{z_i}(z) = e^{-\frac{1}{2}(\frac{z-z_i}{\sigma})^2} \tag{3.9}$$

A weight, $w_{z_i}(z)$, gets closer to one when the z approaches z_i. In contrast, the weight approaches zero when z goes away from z_i. In other words, the higher the weight, the closer z is to z_i. Accordingly, the weight is inversely proportional to the distance between z_i and z.

Definition 3.2.3. *A neighbor set, $M(z)$ is a set of z_k's, the weight of which is the highest with respect to z,*

$$M(z) = \{z_k | w_{z_i}(z) \leq w_{z_k}(z) \text{ for all } z_i\}. \tag{3.10}$$

Finally, we define a new decoder network, D, by

$$D(z) = \sum_i \sum_{z' \in M(z)} D_{z_i}(z') \cdot w_{z_i}(z). \tag{3.11}$$

Theorem 3.2.4. *Let $D(z) = \sum_i \sum_{z' \in M(z)} D_{z_i}(z') \cdot w_{z_i}(z)$ be a mapping from a latent space Z to the space of the output of decoder, Y. Then D satisfies*

1) $D(z) = \hat{D(z)} = D_{AE}(z)$ on $\{z_1, \cdots, z_N\}$,

2) For any small number $\varepsilon > 0$, $\|D(z) - D(z_i)\| < \varepsilon$ if $M(z) = \{z_i\}$ and $\|w_{z_i}(z) - 1\| < \frac{\varepsilon}{\|D_{AE}(z_i)\|}$.

for some $\sigma > 0$.

Proof. 1) For any z_k in $\{z_1, \cdots, z_N\}$,

$$D(z_k) = \sum_i \sum_{z' \in M(z_k)} D_{z_i}(z') \cdot w_{z_i}(z_k) \tag{3.12}$$

$$= \sum_i \sum_{z' \in \{z_k\}} D_{z_i}(z') \cdot w_{z_i}(z_k) \tag{3.13}$$

$$= \sum_i D_{z_i}(z_k) \cdot w_{z_i}(z_k) \tag{3.14}$$

$$= D_{AE}(z_k). \tag{3.15}$$

2) Let $\varepsilon < 0$ and $M(z) = \{z_i\}$, then

$$\|D(z) - D(z_i)\| < \|\sum_j (D_{z_j}(z_i) \cdot w_{z_j}(z) - D_{z_j}(z_i) \cdot w_{z_j}(z_i))\| \tag{3.16}$$

$$= \|D_{z_i}(z_i) \cdot (w_{z_i}(z) - 1)\| \tag{3.17}$$

$$< \|D_{AE}(z_i)\| \cdot \frac{\varepsilon}{\|D_{AE}(z_i)\|} \tag{3.18}$$

$$< \varepsilon. \tag{3.19}$$

□

All of the above show that a decoder network can be built from AEs in order to reconstruct any latent variables into ones which are plausibly drawn from the distribution of the training set. However, it is impractical to compute $M(z)$ and $w_{z_i}(z)$ at the training phase. Noting Theorem 3.2.4 *2*) it is guaranteed that $D(z)$ should be similar to $D(z_i)$ for $z \sim \mathcal{N}(z_i, \sigma)$, so we propose a new generative model based on AEs in the next section.

3.2.3 A new generative model

We propose a new generative model based on AEs, one which does not need the reparameterization trick and extend the mapping of a decoder network from a discrete latent space to a continuous latent space. The key point of the proposed model is that the observed latent variables during the training phase are directly used to extend the latent space of a decoder network to the continuous space. While VAEs use the reparameterization trick, the proposed model explores the new latent variables based on the observed latent variables and given standard deviation. Let x and z be a sample in a training set and its

corresponding latent variable, respectively. Let z' be sampled from $\mathcal{N}(z, \sigma)$. Considering the property of Theorem 3.2.4 *2*) property, the output of the decoder of z' is compared with x to expand the latent space to continuous space. This can be written as:

$$L\left(x, g(z'; \theta)\right), \tag{3.20}$$

where $f(\cdot, \phi)$ and $g(\cdot, \theta)$ are the encoder and decoder networks with respect to parameters sets ϕ and θ, respectively, and $z' \sim \mathcal{N}(f(x; \phi), \sigma)$ and L is a loss function. Now, we let the latent space correspond to a known distribution, specifically the standard normal distribution for the computational efficiency, to sample inputs of the decoder network at the generative phase. As the outputs of the encoder map the standard normal distribution without using the reparameterization trick, we use D_{MMD} instead of KL divergence. Therefore, the loss function of the proposed model becomes

$$L\left(x, g(z'; \theta)\right) + D_{MMD}(f(x; \phi)||\mathcal{N}(z, \sigma)). \tag{3.21}$$

As the decoder network is defined on a continuous latent space, any sample from the distribution has a corresponding output which is plausibly drawn from the distribution of the training set. The architecture of the proposed model is shown in Figure 3.3 (a), and the architecture of the VAE in Figure 3.3 (b). In addition, as the proposed model expands the latent space to a continuous space, it does not depend on the distribution of the latent space, unlike VAE which highly depends on the flexibility of the distribution of the latent space, often resulting in the centralization of latent variables. The reason for mapping the latent space to the standard normal distribution in the proposed model is for the sampling at the generative phase, not for learning distributions.

3.3 Experiments

A number of diverse data samples provide the solution to the problem of overfitting by generalizing neural networks, which results in good performance not only in the training set, but also in the test set. The goal of a generative model is to generate diverse and similar data with samples in a training set to obtain a number of diverse data samples.

In this section, the performance of generative models is evaluated using datasets for a classification task. Since we use datasets for a classification task, the datasets consist of

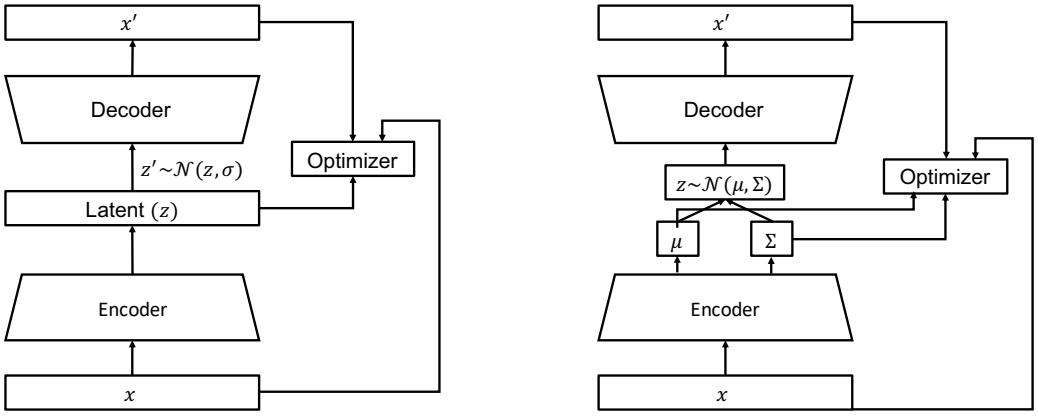


Figure 3.3: Architecture of (a) the proposed generative model and (b) the VAE.

pairs of inputs and targets, which can be referred to as labels in a classification task and are divided into a training set and a test set. The following four steps are performed to evaluate the generative models. First, generative models are trained using inputs of the training set. Second, labels are given to synthetic data generated by the generative models. Third, neural networks for a classification task, which is referred to as classification models, are trained using labelled synthetic data. Last, the performance of the generative models is evaluated based on the accuracy of the test set using the trained classification models. To compare the performance of the proposed model, of VAE and of MMD, we use three benchmark datasets.

Henceforth, the accuracy of the test set is referred to as the accuracy of the test set using the classification model trained by synthetic data generated by generative models.

3.3.1 Dataset

MNIST[1] is one of the most popular benchmark datasets within a classification task due to its small size [97]. It consists of 10 classes of handwritten digits from 0 to 9. There are $60,000$ training and $10,000$ test images of 28×28.

The second dataset is another popular benchmark dataset, called Fashion-MNIST[2]. This dataset has the same size as MNIST and also consists of 10 classes related to fashion: T-Shirt, Trouser, Pullover, Dress, Coat, Sandal, Sneaker, Bag and Ankle boot. Fashion-

[1] Available at `http://www.cs.nyu.edu/~roweis/data.html`.
[2] Available at `https://github.com/zalandoresearch/fashion-mnist`.

MNIST gained popularity by preserving its small size of 28×28, but while also imposing more challenges.

Finally, The Street View House Numbers (SVHN)[3] dataset which is a benchmark dataset consisting of RGB-channel images is used. There are 73,257 training images and 26,032 test images in this dataset, and the size of the images is 32×32×3 [62]. The biggest difference between SVHN and the other two datasets is that images in SVHN have an RGB-channel while the others have grayscale images.

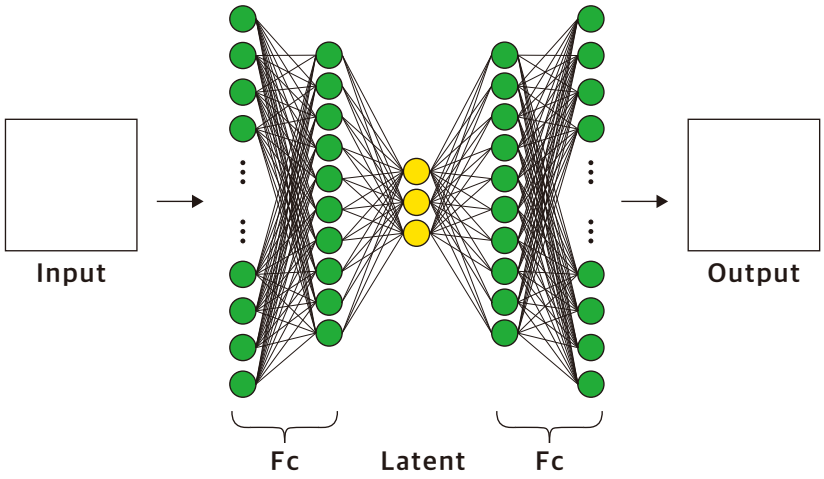


Figure 3.4: Symmetrical architecture of the encoder and the decoder networks for MNIST and Fashion-MNIST datasets. Fc refers to fully-connected layers.

3.3.2 Generative model setup

VAE, MMD and the proposed model all consist of the encoder network and the decoder network. Except for the proposed model, the encoder network has two branches in order to output the mean and standard deviation so that the models can learn the distribution of the latent space. In the models used with MNIST and Fashion-MNIST, with each having one channel, the encoder network and the decoder network are made symmetrical using

[3]Available at `http://ufldl.stanford.edu/housenumbers`.

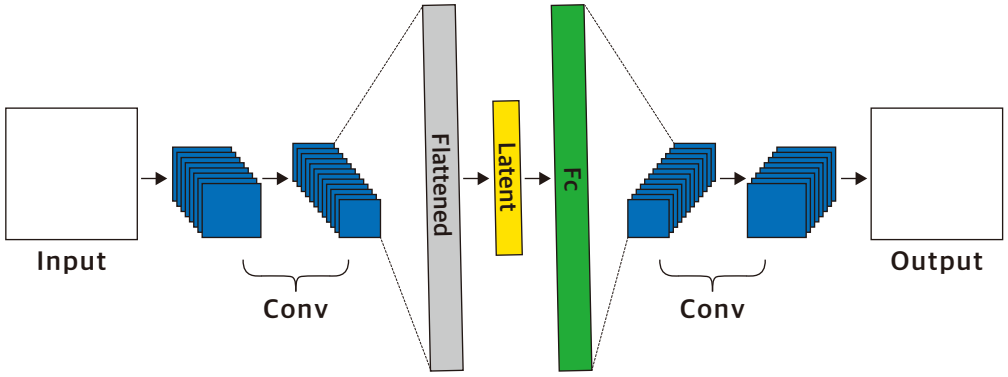


Figure 3.5: Symmetrical architecture of the encoder and the decoder networks for SVHN dataset. Conv refers to convolutional layers and Fc refers to fully-connected layers.

fully connected layers as shown in Figure 3.4. The encoder and the decoder networks are built upon two fully connected hidden layers of 500 nodes with a ReLU activation function. The dimension of the latent space is set to 10, thereby the dimension of the outputs of the encoder network in VAE and MMD is 20 since they have two branches. On the other hand, the proposed model has the dimension of 10 in the output of the encoder network. The architecture with the number of nodes for MNIST and Fashion-MNIST datasets is shown in Figure 3.6 (a). Models are trained with epochs of 500 and a batch size of 128. Adam optimization with a 0.001 learning rate and loss functions corresponding to the models mentioned in Section 2.1 and Section 3.2.3 are set. We choose cross entropy to compare an input x and the corresponding output $\hat{x}$, which is originally used in [39],

$$-\sum_{i=1}^{N} x_i \cdot log(\hat{x_i}), \tag{3.22}$$

where $x = (x_i)_{i=1\cdots N}$ and $\hat{x} = (\hat{x_i})_{i=1\cdots N}$.

In the model used with SVHN, the encoder network and the decoder network are made

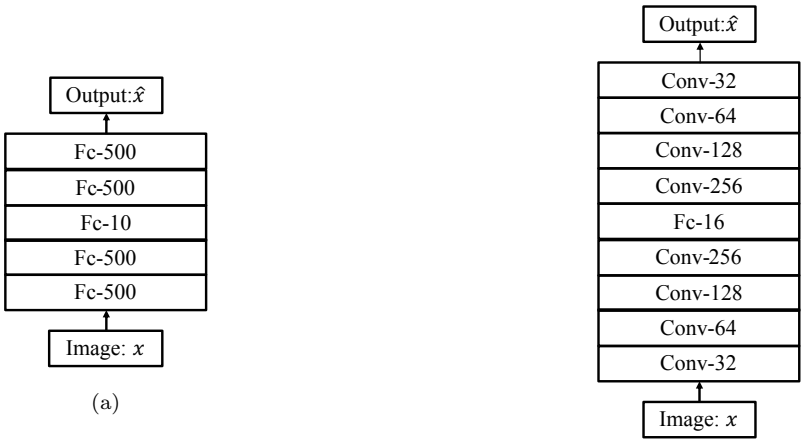


Figure 3.6: Architecture of the baseline of generative models for (a) MNIST and Fashion-MNIST and (b) SVHN datasets

symmetrical using convolutional layers and transposed convolutional layers as shown in Figure 3.5 because the images have three channels. The encoder network is built upon four convolutional layers with ReLU activation function, kernel size of four and stride of two for the first three convolutional layers and stride of one for the last convolutional layer following [28]. The numbers of channels of each convolutional layer are 32, 64, 128, and 256. The decoder network is symmetric to the encoder with the same kernel size and the stride with the encoder network. The architecture with the number of output channels for SVHN dataset is shown in Figure 3.6 (b). The dimension of the latent space is set to 16, thereby the dimension of the outputs of the encoder network in VAE and MMD is 32. The hyper-parameter setups of epochs, batch size, optimizer and the loss functions are the same as the setup for MNIST and Fashion-MNIST datasets, except for the initial learning rate. Here, the initial learning rate is set to 0.0001.

Furthermore, the experiments with the same setup are repeated with $10,000$ randomly selected images, all from the training set, in order to further investigate the performance of the generative models on a small number of data samples.

3.3.3 Labelling synthetic data

Labelling synthetic data is necessary to perform a classification task. We use SVM, a method to find decision boundaries which is also referred to as hyper-planes, to assign

labels to synthetic data in the latent space [74]. The hyper-planes separate data with respect to their labels by maximizing the margin, the smallest distance between the decision boundaries and the data points. SVM uses a kernel function, K, to map the original space into a higher dimensional space. In the experiments, we use the Radial Basis Function (RBF) kernel,

$$K(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2), \tag{3.23}$$

with $\gamma = 0.7$. Assigning labels to synthetic data is done in three steps. First, all inputs in the training set are mapped into the latent space using the encoder network. Second, SVM finds hyper-planes which separate the latent space using pairs of the outputs of the encoder network and their corresponding labels. Finally, labels are assigned to synthetic data using the hyper-planes with respect to the corresponding latent variables of synthetic data. In Figure 3.7, hyper-planes are found based on the variables (white, gray and black points) which are mapped from the training set using the encoder network. When new latent variables (stars) are observed, labels are assigned to the variables based on the hyper-planes.

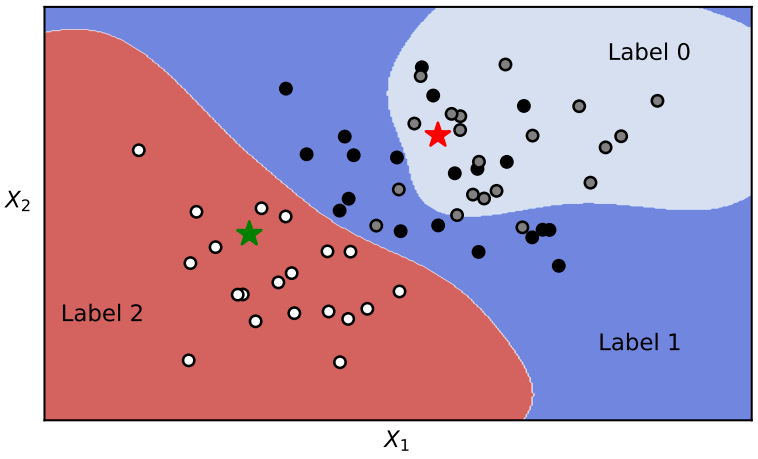


Figure 3.7: Support Vector Machine is used to find the hyper-planes based on observed data (white, grey and black color points) and the hyper-planes are used to assign labels when previously unobserved data (star points) are given. In the figure, label 0 is given to the red star while label 2 is given to the green star.

3.3.4 Classification model setup

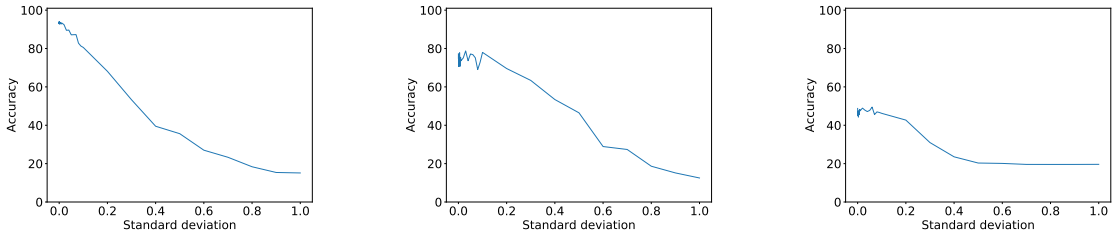
The architecture of a classification model for MNIST dataset consists of one fully-connected layer with 128 nodes, and for Fashion-MNIST consists of two fully-connected layers with 128 and 32 nodes. For SVHN, there are four convolutional layers with the same kernel size, stride and numbers of output channels setup of the generative model. All layers are followed by the ReLU activation function. In order to predict labels with probability distributions proportional to the exponential of the outputs of the last layer, we use the softmax function,

$$\frac{e^{x_i}}{\sum_{i=1}^{M} e^{x_i}} \tag{3.24}$$

where M is the number of target classes and i denotes i-th node in the output layer following the last fully connected layer. Finally, cross entropy is used to compare the outputs and the targets. At every iteration, 128 pairs of labelled synthetic data are generated and the classification models are trained on a total of $10,000$ iterations. Like in the generative model setup, Adam optimizer with 0.001 learning rate is used for all datasets. The evaluation results on MNIST, Fashion-MNIST and SVHN datasets are used in the next section.

3.3.5 Results

The proposed model requires the standard deviation be a hyper-parameter. Therefore, we first investigate the effects of the standard deviation of the proposed model on the accuracy of the test set. For one experiment, we set the standard deviation from 0.001 to 0.009 with an interval of 0.001, another experiment from 0.01 to 0.09 with an interval of 0.01, and the third experiment from 0.1 to 1.0 with an interval of 0.1. In the end, we collate all results. The proposed models with respect to the set of standard deviations are trained using all images in the training set to generate synthetic data. After labelling the synthetic data, they are used to train a classification model to get the accuracy of the test set. As mentioned before, further experiments with the same setup are repeated using a portion of the training set to check the performance of the generative models when a small number of data samples is given.



(a) (b) (c)

Figure 3.8: Relationship between the standard deviation and the accuracy of the test set when all images in (a) MNIST, (b) Fasion-MNIST and (c) SVHN datasets are used.

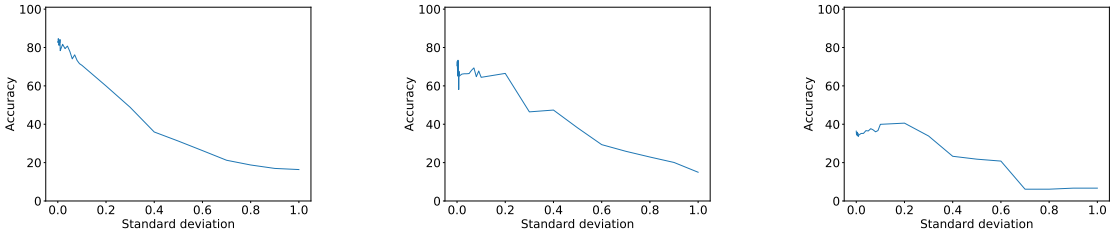
Performance results of the generative models when it is trained on all images of a training set

All the classification performance results using all images of the training sets of the three datasets are shown in Figure 3.8. The results show that the larger the standard deviation is, the worse the performance results will be in all three datasets. This is because when the standard deviation gets larger, the set $M(z)$ in Eq. (3.10) includes latent variables which are not close to z. Therefore, it neglects the property from *2*) in Theorem 3.2.4. The performance results begin to deteriorate at some standard deviation, and the range of the standard deviations that will guarantee a good performance is left for future study.

Next, we compare the best results from the proposed model with the results of VAE and MMD. All models are trained with the same conditions mentioned in Section 3.3.2 and Section 3.3.4. The accuracies of the test set with respect to the models are shown in Table 3.1 and the reconstructed images are shown in Figure A.2. Table 3.1 shows that the highest accuracy is achieved when a classification model is trained on the synthetic data from the proposed model. In other words, the neural network is able to capture the characteristics of the dataset even though it is trained on synthetic data. Note that the difference between accuracies when using the training set and when using the synthetic data from the proposed model is less than 10%. Although the accuracy in the SVHN dataset is very different from the accuracy of the test set when using the training set, it results in a better accuracy than other models. The reconstructed images in Figure A.2 (c) show that VAE fails to reconstruct images in the SVHN dataset, which results in low accuracy in classification tasks.

Table 3.1: The accuracy of the test set when training neural networks using synthetic data from generative models trained by all images in the training set.

Training set from \ Dataset	MNIST	Fashion-MNIST	SVHN
VAE	92.24%	69.55%	19.56%
MMD	91.14%	76.44%	46.23%
The proposed model	**94.13**%	**79.72**%	**49.47**%
Original training set	97.91%	88.64%	88.97%



(a) (b) (c)

Figure 3.9: Relationship between the standard deviation and accuracy of the test set when a portion of (a) MNIST, (b) Fasion-MNIST and (c) SVHN datasets is used.

Performance results of the generative models when it is trained on a portion of the training set

Generative models are proposed so that we may generate synthetic data when it is difficult to obtain a large amount of data. For that reason, we do experiments with the same setup as in Section 3.3.2 and Section 3.3.4 but use only $10,000$ images from each dataset. The images are randomly selected and the sets of selected images are named as: MNIST$_s$, Fashion-MNIST$_s$ and SVHN$_s$. Although we use a smaller number of images, the reconstructed images using the three datasets in Figure A.3 are similar to the reconstructed images using the original datasets in Figure A.1. Accordingly, we again evaluate the models using the accuracy of the test set. Like in the previous results when all images were used, the accuracy decreases when the standard deviation increases, as shown in Figure 3.9. The comparison of the results between the proposed model, VAE and MMD are shown in Table 3.2. The accuracy of the test set when training a classification model using synthetic data from the proposed model also outperforms the other models. The neural network trained on synthetic data generated by the proposed model can still distinguish 84.62% of images from the test set in the MNIST dataset and about 73.37% of the test set from the

Fashion-MNIST dataset without the original training set. The reconstructed images from the three datasets using the proposed model, VAE and MMD are shown in Figure A.4.

Table 3.2: he accuracy of the test set when training neural networks using synthetic data from generative models trained by a portion of the training set.

Training set from \ Dataset	MNIST_s	Fashion-MNIST_s	SVHN_s
VAE	84.13%	46.03%	19.56%
MMD	82.58%	64.72%	37.72%
The proposed model	**84.62**%	**73.38**%	**38.41**%

3.4 Summary

In this chapter, generative models to generate more data have been provided to overcome the overfitting problem. We begin by tackling the role of the reparameterization trick in VAEs, which mimics the sampling in statistics. We notice that each mean in the latent sapce has its corresponding input and the reparameterization trick could make the architecture of autoencoders become generative models by mapping unobserved latent variables around the mean values. This leads us to design a new generative model based on the continuity of the original AEs and by extension of the mapping of the decoder network from discrete latent space to continuous latent space. This is done by mapping samples in the training set into a prior distribution, which is used at the generating phase, and by adding $\varepsilon \sim \mathcal{N}(0, \sigma)$ where σ is given as a hyper-parameter.

Three datasets are used to verify the proposed model. The proposed generative model is trained using one of the datasets. At each iteration in training a classification model, a set of labelled synthetic data from the generative model is fed to the neural network.

The impact of the standard deviation on the accuracy of the test set is empirically shown. The result suggests that a certain range of standard deviations guarantees learning a good representation of the dataset. The accuracy of the test set when training using synthetic data from the proposed generative model is higher than those when training using synthetic data from the other generative models. The impact of the number of training samples on the accuracy of the test set is studied by randomly selecting $10,000$ images from each dataset. As the proposed model performs better than the others in a certain range of standard deviations, it is left for future study to find the optimal range of standard deviations.

The proposed generative model will be applied to *multi-channel data* in a later chapter to generate synthetic data, which are plausibly drawn from the distribution of the training set. This will help in understanding *multi-channel data* as we could consider more possible samples that might arise when multiple channels are combined.

Chapter 4

Hierarchical Learning Approach

Noting the use of conventional ANNs for classification and regression tasks, it is assumed that the relationship between targets are equal (*i.e.*, no hierarchy). ANNs for a classification task built on this assumption cannot take into account an innate hierarchy of targets (e.g., the first and the second floors of Building A belong to Building A in multi-building and multi-floor classification). Moreover, on the same assumption, ANNs cannot utilize additional information that humans know (e.g., numbers larger than five among one-digit numbers), thereby misclassification by ANNs can be recognized more by humans (e.g., ANNs regard the degree of difference between images of cats and images of dogs the same as between images of cats and images of ships). In this chapter, two types of novel neural network architectures are proposed to utilize the coarse classes of targets. The first type directly utilizes coarse classes to boost the training phase and increase performance of a classification task using image datasets as examples. The experiments with the first type show learning coarse classes affects learning fine classes. The second type uses coarse classes to improve the performance of a regression task using indoor localization datasets as examples of *multi-channel data*. The experiments with the second type shows that coarse classes defined based on channels of data can improve the performance of a regression task.

4.1 Hierarchical Auxiliary Learning

Although Convolutional Neural Networks (CNNs) show superior performance in image classification by hierarchically capturing characteristics of images [43, 102, 81, 30, 20], some misclassification results with CNNs are hardly ever explainable [10]. The occurrence

of this misclassification can be reduced by providing coarse classes as auxiliary information to CNNs. In the case of an image classification task, coarse classes can be defined by various conditions. For example, images in real life can be grouped into two classes based on whether or not they are alive. Considering the significance of coarse classes, new neural network architectures which take these coarse classes into account deserve intensive study. To this end, we propose the first type of hierarchical learning models, named as *Hierarchical Auxiliary Learning* (HAL). HAL is a mixed concept composed of *hierarchical classification* [9, 95, 99, 106] and *logical learning by auxiliary inputs* [88].

The goal of HAL is to provide coarse class information at the training phase as an auxiliary input. For example, a set of digits from 0 to 4 and from 5 to 9 can be classified as two different coarse classes, denoted by 0 and 1, respectively. The coarse classes and images are then fed to the neural networks during the training phase. Let (x, y) and x^* be a pair of an image and a fine class and a coarse class, respectively. Triplet (x, y, x^*) is required to train HAL, while the pair of (x, y) is required in the conventional ANNs. During the training phase, x and x^* are fed to the neural network and the output of HAL is compared with y. Therefore, the goal of HAL is to find a function f,

$$\hat{y} = f(x, x^*), \tag{4.1}$$

while the goal of the conventional ANNs is to find a function g,

$$\hat{y} = g(x), \tag{4.2}$$

which minimizes the loss between $\hat{y}$ and y. In this section, an auxiliary block, which takes coarse classes and outputs auxiliary scores, is inserted to CNNs to find the function f.

4.1.1 Convolutional Neural Network

The images of any given dataset are presented using grayscale or color scale m by n matrices. With the conventional feedforward neural networks, this dataset would require $m \times n$ or $m \times n \times 3$ dimensional flatten vectors as its inputs. By flattening the image matrix, we lose spatial information, which would not occur with CNNs as CNNs scan the original m by n matrices [43].

In general, CNNs can be divided into two parts: a series of convolutional layers and of a series of fully-connected layers. In convolutional layers, filters are square arrays which are

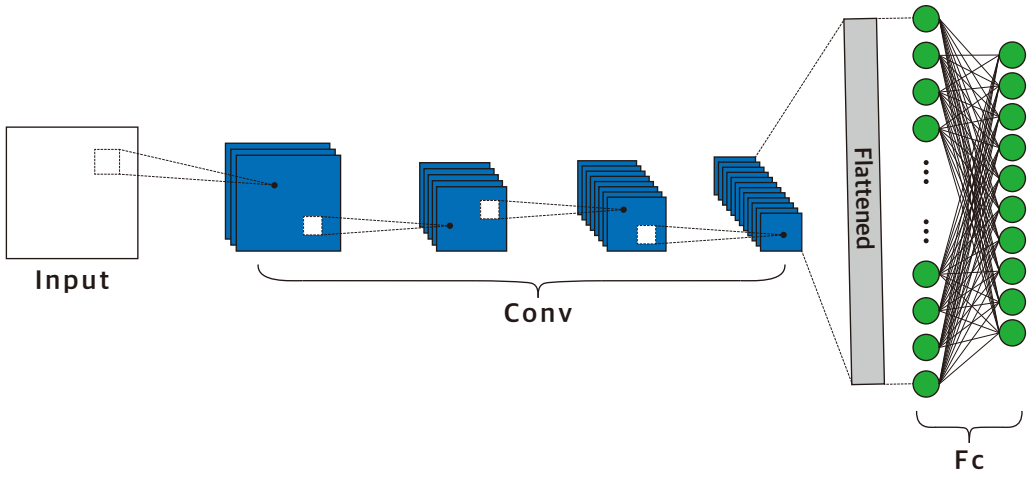


Figure 4.1: Architecture of the Convolutional neural networks. The Convolutional neural networks are comprised of convolutional layers and fully-connected layers

smaller than the inputs and are applied to the inputs by way of element-wise multiplication. The summed outputs of each element-wise multiplication are then collected in a feature map. Once a feature map is created, values in the feature map go through a nonlinear function. Let $h(m)$ and $w(n,m)$ be the m-th input from the previous layer and the m-th array in the n-th filter. Each output of convolutional layer is written as follows:

$$y(n,i,j) = f(z(n,i,j)), \tag{4.3}$$

$$z(n,i,j) = \sum_{m=1}^{d}\sum_{x=1}^{p}\sum_{y=1}^{q} w(n,m,x,y)\cdot h(m,i+x,j+y) + b(n), \tag{4.4}$$

where f is a non-linear activation function, d is the depth of the previous layer, and p and q are width and height of the kernel, respectively. Here, b is a bias and $z(n,i,j)$ denotes (i,j)-element in the n-th feature map. After one or more convolutional layers, there might be a pooling layer. This pooling layer downsamples the previous layers to compress or generalize feature representations and generally reduces the overfitting in CNNs [43].

The advantages of convolutional layers are the extraction of features hierarchically and preservation of spatial information, which is known as translation equivariance [42]. These two advantages allow the outputs of a series of convolutional layers to represent important features of the input images, regardless of position changes.

4.1.2 Learning scheme

In this subsection, we design an auxiliary block which takes coarse classes as inputs and outputs the auxiliary scores. Coarse classes are converted to one-hot encodings as follows to be inputs of the auxiliary block: First, coarse classes are labelled by numbers from zero. Second, if the label, k, is given to coarse classes, one-hot encodings are completed by making a vector with $(k+1)$-th element of 1 and the other elements are 0. For example, let there be 4 coarse classes. The coarse class with label 1 will have the one-hot encoding of $[0, 0, 1, 0]$. Subsequently, the one-hot encodings are fed to the auxiliary block rather than to the input layer as shown in Figure 4.2. This is done to utilize coarse class information with extracted features by a series of convolutional layers rather than utilizing raw images. In other words, the auxiliary block takes one-hot encodings and flattened vectors of the outputs of the last convolutional layer.

This auxiliary block consists of one fully-connected layer, an element-wise subtraction layer, an absolute layer and a summation layer. Nevertheless, the auxiliary block contains learnable parameters only in one fully-connected layer which results in the architecture not suffering from scalability problems unlike the other architectures introduced in Section 2.2. The fully-connected layer outputs vectors whose dimensions are the same as those of the one-hot encoded coarse classes. Passing the other layers in the auxiliary block, the auxiliary score is obtained and is element-wise multiplied by the output of the last convolutional layer. Since the auxiliary block can take any flattened vector of the outputs of the convolutional layers, it can be inserted into any existing pretrained model such as VGGNet, ResNet and so on [78, 27].In this experiment, we insert the auxiliary block after the last convolutional layer, as we want to utilize features extracted by the convolutional layers.

4.1.3 Backpropagation

ANNs use backward-propagation, also referred to as back-propagation, to find the optimal parameters by reducing the loss between the predictions of ANNs and targets [72]. The loss gets smaller by adjusting the learnable parameters of ANNs based on the derivative of the

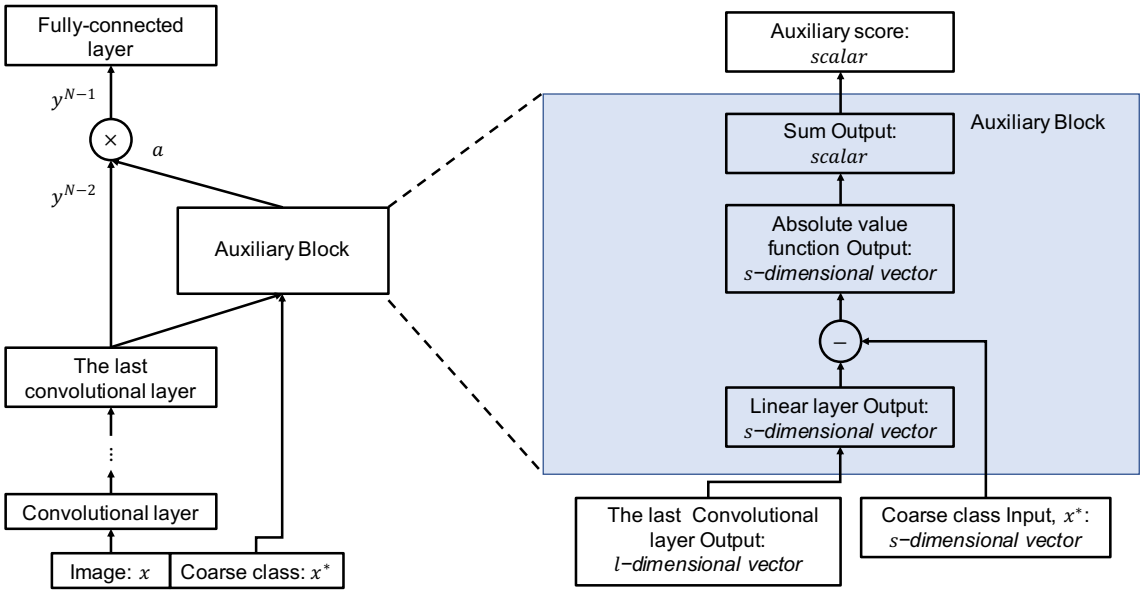


Figure 4.2: Architecture of the Hierarchical Auxiliary learning. The auxiliary block is inserted after a series of convolutional layers.

loss with respect to the learnable parameters, *i.e.*, $\frac{\partial l}{\partial w}$ is the degree of the changes of l with respect to changes of w , where l and w are the loss and a learnable parameter, respectively. However, it is difficult to calculate $\frac{\partial l}{\partial w}$ directly for all learnable parameters since learnable parameters are connected with multiple operations. This difficulty is solved by back-propagation by firstly calculating the derivatives of l with respect to learnable parameters from the last layer to the first layer using the chain rule [72], and secondly adjusting learnable parameters with respect to the derivatives to reduce the loss. The auxiliary block includes learnable parameters in its fully-connected layer and those parameters are also updated through back-propagation, meaning the auxiliary scores are learnt through back-propagation during the training phase.

As back-propagation works well with the flow of the input layer, the convolutional layers, the fully-connected layers and the output layer, we are going to show how to implement back-propagation when the auxiliary block is inserted within this flow. Let us consider a neural network consisting of a series of convolutional layers, one fully-connected layer and the auxiliary block as shown in Figure 4.2. Let y^{N-1}, y^{N-2}, and a be the input to the fully-connected layer, the output of the last convolutional layer and the auxiliary

score, respectively. Compared to conventional CNNs where $y^{N-1} = y^{N-2}$, the input to the fully-connected layers, y^{N-1}, is calculated by

$$y^{N-1} = y^{N-2} \times a \tag{4.5}$$

and

$$a = \sum_j \left| \sum_i w_{ij} \cdot y_i^{N-2} - x_j^* \right|, \tag{4.6}$$

where w_{ij}'s are the learnable parameter of the fully-connected layer in the auxiliary block and $x^* = (x_j^*)$ is the one-hot encoded coarse class. Subsequently, the derivative of l with respect to learnable parameters are calculated as follows:

$$\frac{\partial L}{\partial a} = \sum_i \frac{\partial L}{\partial y_i^{N-1}} \cdot y_i^{N-2}, \tag{4.7}$$

$$\frac{\partial L}{\partial y_i^{N-2}} = \frac{\partial L}{\partial y_i^{N-1}} \cdot a + \sum_j \chi_j \cdot \frac{\partial L}{\partial a} \cdot w_{ij} \tag{4.8}$$

and

$$\frac{\partial L}{\partial w_{ij}} = \chi_j \cdot \frac{\partial L}{\partial a} \cdot y_i^{N-2}, \tag{4.9}$$

where

$$\chi_j = \begin{cases} 1 & \text{if } \sum_i w_{ij} \cdot y_i^{N-2} > x_j^*, \\ -1 & \text{otherwise} \end{cases}. \tag{4.10}$$

This shows that the auxiliary score is updated based on the extracted features from the convolutional layers and the coarse classes using back-propagation. In addition, the auxiliary block transforms the discrete coarse class to the continuous auxiliary scores. Because learnable parameters are included only in one fully-connected layer, the number of learnable parameters of HAL is similar to those of one without the auxiliary block. The result will be verified in the next subsection.

4.1.4 Experiments with three baselines

In this subsection, we investigate the effect of the auxiliary block on the performance of the image classification task. Given the fact that the auxiliary block can be inserted after

Table 4.1: CIFAR-10 dataset label information

Name	airplane	car	bird	cat	deer	dog	frog	horse	ship	truck
Label	0	1	2	3	4	5	6	7	8	9

any series of convolutional layers, three baselines are used on the CIFAR-10 dataset[1]: a) Base A (used in [106]), b) Base B (the pretrained VGG 16) and c) Base C (the pretrained ResNet 34), which are shown in Figure 4.3. The auxiliary block is inserted between the last convolutional layer and the first fully-connected layer in each baseline as shown in Figure 4.4. In CIFAR-10, there are $60,000$ images of 32×32, which belong to the 10 different classes listed in Table 4.1. In order to utilize coarse classes, CIFAR-10 data are grouped into two cases. In the first case, the fine classes are divided into two coarse classes based on whether the class is a form of transportation or an animal. In the second case, the fine classes are grouped into five coarse classes. The relationships of the coarse classes and fine classes for both cases are shown in Figure 4.5. All baselines are trained on the same conditions: the batch size and epoch are set to 128 and 100, respectively. Stochastic gradient descent (SGD) optimizer is used with a step learning schedule: the initial learning rate is set to 0.003 and decreases to 0.0005 after 42 epochs and to 0.0001 after 52 epochs.

The result of the fine class classification task is shown in Table 4.2. The results show that the accuracy of the test set increases in both cases. As the second case has higher accuracies than the first case on all three baselines, we can assume that the coarse class information improves the accuracy of the test set regardless of the architecture of the neural networks.

We discover two notable points of HAL. First, the proposed model can be easily created by inserting an auxiliary block into the existing CNNs. Second, the number of learnable parameters in both the baselines with or without the auxiliary block are very similar. The comparison of the number of the parameters between the baseline with or without the auxiliary block is shown in Table 4.3.

4.1.5 Experiments with three datasets on the selected baseline

We further investigate the effect of the auxillary block on the performance of the image classification task with a new architecture setup modified by Base C using three standard benchmark datasets: MNIST, SVHN and CIFAR-10. ResNet (Base C) is one of the most

[1]Available at `https://www.cs.toronto.edu/~kriz/cifar.html`.

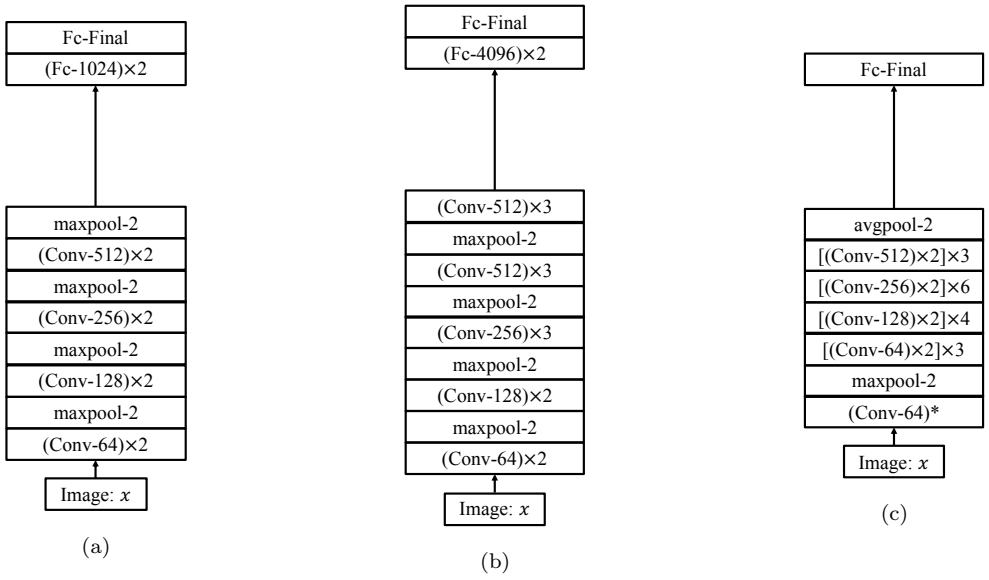


Figure 4.3: Three baselines are used to investigate the effect of the auxiliary block. (a) is proposed in [106], (b) is VGG16 and (c) is ResNet34. Kenerl sizes of all Conv layers are set to 3 and a kernel size of Conv* is set to 7.

Table 4.2: The accuracy of the test set on the three baselines

Case	Base A	Base A with the auxiliary block
	89.15	
Case1		89.57
Case2		91.53
	Base B	Base B with the auxiliary block
	92.24	
Case1		92.47
Case2		94.02
	Base C	Base C with the auxiliary block
	87.64	
Case1		87.99
Case2		89.35

popular architectures in Computer Vision but is designed for a larger image dataset than CIFAR-10 [27]. Accordingly, we reduce the number of residual blocks and the number of filters in each residual block for the three datasets. For the experiments with three datasets, we use three residual blocks with filters of 16, 32 and 64. While images in

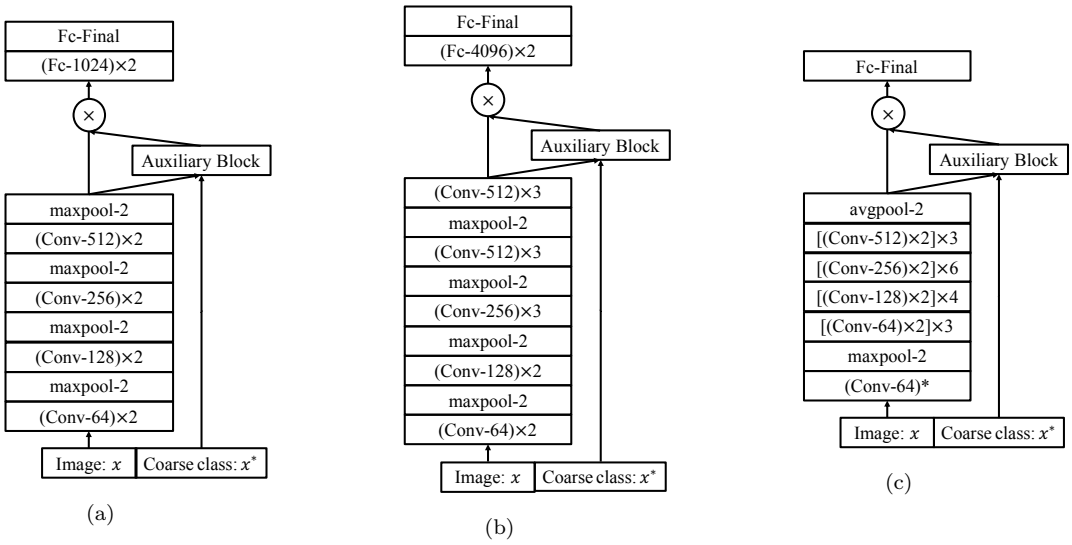


Figure 4.4: Architecture of three baselines with the auxiliary block.

Table 4.3: The number of the learnable parameters on the three baselines

Case	Base A	Base A with the auxiliary block
	7,849,418	
Case1		7,853,516
Case2		7,859,663
	Base B	Base B with the auxiliary block
	39,929,674	
Case1		39,933,772
Case2		39,939,919
	Base C	Base C with the auxiliary block
	21,802,802	
Case1		21,806,900
Case2		21,813,047

SVHN and CIFAR-10 datasets have an RGB-channel, MNIST dataset has grayscale images. Therefore, the number of repetitions in each block is set differently – one for MNIST and four for both SVHN and CIFAR-10. In this experiment, we use cosine annealing with a maximum learning rate of 1.0 and a minimum learning rate of 0 [47]. Batch size is set to 128 and epoch is increased to 250 for all datasets. The learnable parameters are initialized by He initialization [26]. During the training phase, images are augmented by cropping and randomly flipping. All experiments are trained five times, and the mean and the standard

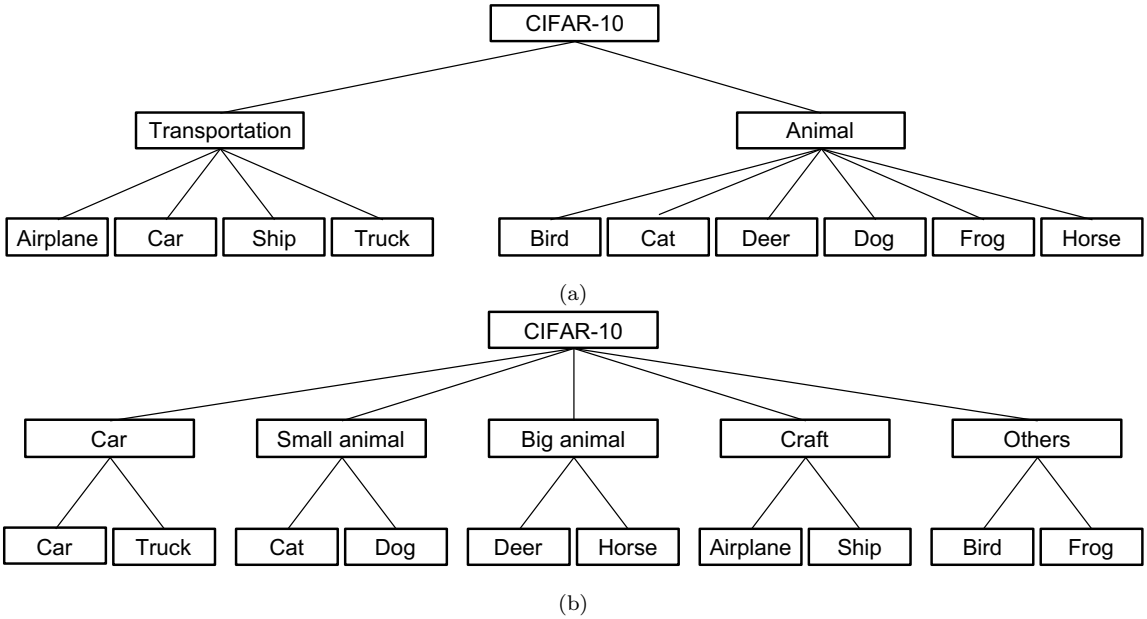


Figure 4.5: Coarse class information of (1) the first case and (2) the second case with CIFAR-10 dataset.

deviation of the accuracies are presented in Table 4.5.

Table 4.4: Coarse classes information

Dataset	Case	Condition	Coarse class
MNIST	Case1	≥ 5	0:{5, 6, 7, 8, 9}, 1:{0, 1 ,2 ,3, 4}
SVHN	Case2	modulo 2	0:{1, 3, 5, 7, 9}, 1:{0, 2, 4, 6, 8}
	Case3	prime	0:{2, 3, 5, 7}, 1:{0, 1, 4, 6, 8, 9}
	Case4	circle/ curve/ straight line	0:{0, 6, 8, 9}, 1:{2, 3, 5}, 2:{1, 4, 7}
CIFAR-10	Case1	transportation/ animal	0:{2, 3, 4, ,5, 6, 7}, 1:{0, 1, 8, 9}
	Case2	car/ small animal/ big animal/	0:{1, 9}, 1:{3, 5}, 2:{4, 7},
		craft/ others	3:{0, 8}, 4:{2, 6}
	Case3	fine class ≥ 5	0:{5, 6, 7, 8, 9}, 1:{0, 1 ,2 ,3, 4}
	Case4	fine class (modulo 2)	0:{1, 3, 5, 7, 9}, 1:{0, 2, 4, 6, 8}

MNIST

We use the MNIST dataset, which was described in Chapter 3, to evaluate HAL. The coarse classes of this dataset are manually drawn based on human knowledge or on the

Table 4.5: The accuracy comparison with three datasets

MNIST	baseline	Case1	Case2	Case3	Case4
Error	99.07	99.57 ± 0.03	99.27 ± 0.06	99.30 ± 0.05	99.31 ± 0.00
SVHN	baseline	Case1	Case2	Case3	Case4
Error	95.95	97.47 ± 0.06	97.36 ± 0.11	97.34 ± 0.07	97.14 ± 0.07
CIFAR-10	baseline	Case1	Case2	Case3	Case4
Error	93.19	93.54 ± 0.08	94.87 ± 0.09	96.7 ± 0.06	94.7 ± 0.14

shape of the digit. First, the dataset is grouped by five into two coarse classes. Second, it is split into even and odd numbers. The third condition is based on whether the digit is a prime number. Finally, the coarse classes are determined by the similarity of the shape of the digits: Circles appear in digits 0, 6, 8 and 9, and lines govern the digits 1, 4 and 7, while the other digits do not have unconnected curves. The four different coarse classes are summarized in Table 4.4 and the results are shown in Table 4.5. While the baseline accuracy is 99.07%, all coarse classes mentioned above affect the improvement of the accuracy of the test set. Loss in the training and test sets of the baseline with or without the auxiliary block during the training phase are compared in Figure B.1. In addition, the auxiliary scores of all training images show that the auxiliary scores can be separated based on the coarse classes. A comparison between Case1 and Case2 shows that a better performance is obtained when the auxiliary scores are well-separated with respect to the coarse classes. A comparison between Case1 and Case4 shows that constructing more coarse classes does not guarantee improvement of the performance.

SVHN

Next, the SVHN dataset introduced in Chapter 3 is used to evaluate HAL. As the fine classes of MNIST and SVHN are the same, we can apply the coarse classes strategy of MNIST to SVHN. The mean and the standard deviation of the errors are shown in Table 4.5. The accuracy of the test set is improved by at least 1.2%. During the training phase, the losses of training and test sets decrease faster when the coarse classes are introduced. Unlike MNIST dataset, the auxiliary scores are well-distributed over all the coarse classes as shown in Figure B.4, thereby the results of all cases are almost identical.

CIFAR-10

We also use CIFAR-10 dataset with four coarse class cases including the first and second case in Section 4.1.4. The new coarse classes, Case3 and Case4, are formed based on their labels shown in Table 4.1. In Case3, if the label of an image is larger than or equal to 5, then it is labeled as 0; otherwise, the label is 1. In Case4, 0 is given if the fine class of an image is an odd number; otherwise, 1 is given. Although the coarse classes are not related to the semantic information of the targets, which makes the coarse classes of Case3 and Case4 meaningless, we still evaluate the proposed model with Case3 and Case4 in order to observe if any coarse classes could improve the overall performance. All coarse class information is described in Table 4.4. As shown in Table 4.5, the meaningless coarse class assignments also improve the accuracy of the test set. Figure B.5 also shows that loss reduction during the training phase with the auxiliary block is faster and more efficient in terms of convergence when coarse classes are provided. Again, a separation of auxiliary scores with respect to coarse classes is related to the accuracy of the test set as shown in Figure B.6 and Table 4.5. Case3 – which shows a clear separation between two coarse classes – outputs a higher accuracy than Case1 – where the two regions of auxiliary scores with respect to coarse classes are overlapped.

4.2 Consecutive Feedforward Neural Networks and Hierarchical Auxiliary Deep Neural Networks

It is clear that HAL performs better by applying coarse classes than conventional ANNs, as seen in Section 4.1. However, HAL still needs coarse classes during the test phase, which are sometimes unavailable. In this section, we derive a new model which utilizes a hierarchy of targets for regression tasks and which does not require coarse classes during the test phase. Conventional ANNs for a regression task only consider the target values, thereby an innate hierarchy of targets cannot be taken into account. In order to deal with this issue, we propose the second type of hierarchical learning models, called *Consecutive Feedforward Neural Networks* (C-FNNs) and *Hierarchical Auxiliary Deep Neural Networks* (HADNNs). Both perform a classification task for coarse classes and a regression task for fine classes. In C-FNNs and HADNNs, coarse classes in a training set are given and the regression task follows the classification tasks. There are two possible cases for the coarse classes: (1) coarse classes have only one level and (2) coarse classes have two levels. At the

test phase, the proposed models predict all coarse classes and a target of the regression task. Both C-FNNs and HADNNs show better performance than conventional ANNs, but C-FNNs require a large number of learnable parameters. In order to balance and scale the loss between coarse classes and fine classes, adaptive loss balancing is introduced in the following subsection.

4.2.1 Adaptive loss balancing

C-FNNs and HADNNs can be regarded as multi-task learning, as they perform coarse class classification tasks as well as a regression task. Learnable parameters in C-FNNs and HADNNs are updated by reducing all the losses of both tasks. The simplest way to deal with multiple losses is to sum all the losses. However, this results in the imbalanced learning of multi-tasks, *i.e.*, a loss of some of the tasks decreases faster than the others. This imbalanced learning causes biased performance, which means having a good performance only in some of the tasks. In this section, we propose adaptive loss balancing in order to deal with this imbalanced learning problem.

Adaptive loss balancing is made up of two strategies: 1) weights of losses are calculated to make the ratio of the losses equal and 2) an additional weight, k, is multiplied by the loss of the regression task to deal with the scale difference of the loss functions between a regression task and a classification task. The procedure is as follows:

1. Calculate each loss, l_{c_1}, l_{c_2}, and l_r.
2. Round up each loss.
3. Take the maximum of the losses, which is denoted by m.
4. Calculate $\frac{m}{l_{c_1}}$, $\frac{m}{l_{c_2}}$, and $\frac{m}{l_r}$.
5. Round them up and denote by a_{c_1}, a_{c_2}, and a_r.
6. Multiply them by the losses.
7. The final loss is $a_{c_1} \cdot l_{c_1} + a_{c_2} \cdot l_{c_2} + k \cdot a_r \cdot l_r$

where l_{c_1}, l_{c_2}, and l_r are losses for the first coarse classes, the second coarse classes, and a regression task, respectively.

4.2.2 Consecutive Feedforward Neural Networks

C-FNNs are a series of pairs of one feedforward neural network and one decoder network. Each feedforward neural network performs a classification task or a regression task. Then, all decoder networks enlarge the outputs of each of the feedforward neural network apart from the output of the last feedforward neural network. This results in the outputs of the decoders having the same dimensions as the inputs of the feedforward neural network. Subsequently, the inputs are added element-wise to the outputs. The process of adding inputs by skipping the feedforward neural network and the decoder is known as skip connection [27, 81]. Next, we illustrate two types of C-FNNs.

C-FNNs for coarse classes with one level

In C-FNNs for coarse classes with one level, there are two feedforward neural networks and one decoder network. The first feedforward neural network performs a coarse class classification task. The decoder network enlarges the output of the coarse class classification task to have the same dimensions as the input of the feedforward neural network. Let FNN_1 be the first feedforward neural network for the input x, then the output of FNN_1 in Eq. (4.11),

$$\hat{y_{c_1}} = FNN_1(x) \tag{4.11}$$

is fed to the first decoder network DEC_1 to enlarge the dimensions. Consequently, the second feedforward neural network, FNN_2, performs the final regression task by taking the sum of the output of the decoder and the input x. This process is expressed in Eq. (4.12).

$$\hat{y_r} = FNN_2(DEC_1(\hat{y_{c_1}}) + x) \tag{4.12}$$

The cross-entropy and the mean squared error are used to calculate the classification loss and the regression loss, respectively. Applying adaptive loss balancing from Section 4.2.1, the loss, l, becomes

$$l = a_{c_1} \cdot C(\hat{y_{c_1}}, y_{c_1}) + k \cdot a_r \cdot M(\hat{y_r}, y_r), \tag{4.13}$$

where C is the cross-entropy, M is the mean squared error, and y_{c_1} and y_r denote the first coarse class target and a regression target, respectively. The architecture is shown in

Figure 4.6.

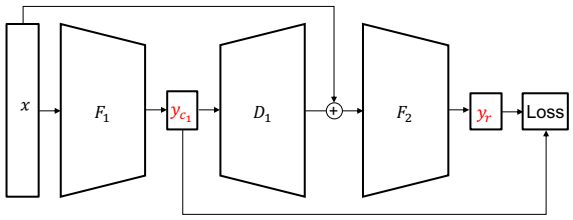


Figure 4.6: Consecutive Feedforward Neural Networks for coarse classes with one level. F and D denote a feedforward neural network and a decoder network, respectively, and y_{c_1}, and y_r denote the first coarse class and a regression output, respectively.

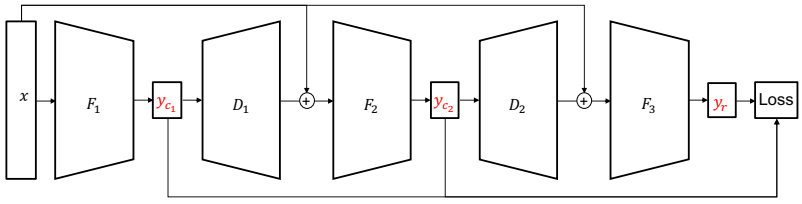


Figure 4.7: Consecutive Feedforward Neural Networks for coarse classes with two levels. F and D denote a feedforward neural network and a decoder network, respectively, and y_{c_1},y_{c_2}, and y_r denote the first coarse class, the second coarse class and a regression output, respectively.

C-FNNs for coarse classes with two levels

C-FNNs for coarse classes with two levels require one more classification task. Hence, the third feedforward neural network, FNN_3, and the second decoder network, DEC_2, are inserted in the architecture in Figure 4.6. In addition, the loss for the second coarse class classification task is added to the loss in Eq. (4.13), and it becomes

$$l = a_{c_1} \cdot C(\hat{y_{c_1}}, y_{c_1}) + a_{c_2} \cdot C(\hat{y_{c_2}}, y_{c_2}) + k \cdot a_r \cdot M(\hat{y_r}, y_r), \tag{4.14}$$

where y_{c_1}, y_{c_2} and y_r denote the first coarse class target, the second coarse class target and a regression target, respectively. The architecture diagram is shown in Figure 4.7.

However, C-FNNs need a large number of learnable parameters due to the use of skip connection. Consequently, we propose an architecture in the next subsection that can

overcome this difficulty.

4.2.3 Hierarchical Auxiliary Deep Neural Networks

Here, we propose a new architecture called HADNNs, which are modifications of C-FNNs by reducing the size of a series of pairs of one feedforward neural network and one decoder network.

HADNNs for coarse classes with one level

In HADNNs, we can reduce the size of a pair of one feedforward neural network and one decoder network because the skip connection is no longer used. The first layer then is followed by two branches: 1) a pair of one feedforward neural network and one decoder network and 2) one feedforward neural network. The first branch performs the coarse class classification task and the second branch directly conveys extracted features from the preceded layer to the next layer. Let us define a layer which has two branches as F. Then, the two branches are denoted as F_1^u and F_1^d. A pair of one feedforward neural network, FNN, and one decoder network, DEC, for a coarse class classification task follows F_1^u, *i.e.*,

$$\hat{y_{c_1}} = FNN(F_1^u(F(x))), \tag{4.15}$$

and DEC extends $\hat{y_{c_1}}$ to have the same dimensions as the outputs of F_1^d, then these are concatenated as follows,

$$\hat{y} = DEC(\hat{y_{c_1}}) \copyright F_1^d(F(x)) \tag{4.16}$$

where © denotes concatenation. $\hat{y}$ is then fed to the next layer. By concatenating the two branches, the latter layers are affected by the information from the input and the extracted features from the previous tasks. The same loss as in Eq. (4.13) is applied and the architecture of HADNNs for coarse classes with one level is shown in Figure 4.8.

HADNNs for coarse classes with two levels

Since there are coarse classes with two levels, we repeat the inserting branches process described in the previous section one more time. We complete the model for coarse classes

with two levels by inserting two branches within F_1^u, as shown in Figure 4.8. The completed model is shown in Figure 4.9. The outputs of the first coarse class information are concatenated with F_2^d. Consequently, the second coarse class information, which is affected by the input x, and the first coarse class information are concatenated with F_1^d. Finally, the regression task is performed based on the information of the first coarse class, the second coarse class, and the input x. By removing skip connection and the conveyance of input information by branches, the proposed network reduces the number of parameters.

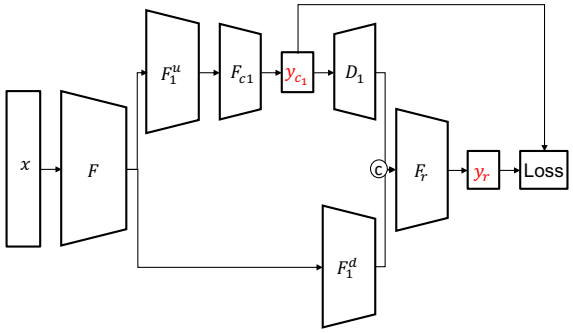


Figure 4.8: Hierarchical Auxiliary Deep Neural Network for coarse classes with one level. F and D denote a feedforward neural network and a decoder network, respectively, and y_{c_1}, and y_r denote the first coarse class and a regression output, respectively.

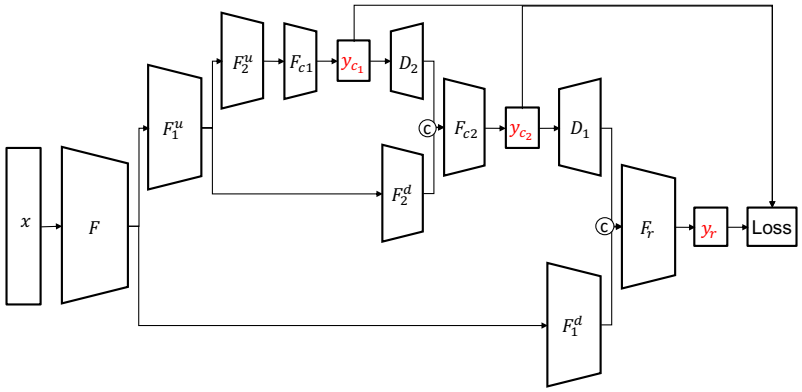


Figure 4.9: Hierarchical Auxiliary Deep Neural Network for coarse classes with two levels. F and D denote a feedforward neural network and a decoder network, respectively, and y_{c_1},y_{c_2}, and y_r denote the first coarse class, the second coarse class and a regression output, respectively.

4.2.4 Comparison between C-FNNs and HADNNs

The most important purpose of C-FNNs and HADNNs is to deliver input information and the extracted information from coarse class classification tasks to a regression task. However, there is a small difference in their architectures.

C-FNNs derive extracted features related to the coarse classes and add them to the inputs. By adding extracted features to the inputs, which is crucial in order to compensate for the information lost at the coarse class classification tasks, all the information together can be delivered to a regression task. Although C-FNNs are able to deliver the information mentioned before to a regression task, C-FNNs still require a lot of parameters because of skip connection.

In HADNNs, extracted features of inputs and of the outputs of the coarse class classification tasks are delivered to a regression task by making two branches. By concatenating, as opposed to adding, the extracted features of inputs and of the outputs, a large number of parameters can be eliminated. In addition, the concatenation of the extracted features allows for the independent connecting of these extracted features to the next layer. On the other hand, by adding the extracted features of inputs and of the outputs of the coarse class classification tasks, separately considering the importance of each extracted feature is difficult. Therefore, concatenation allows for a more flexible process.

4.2.5 C-FNNs and HADNNs setup

C-FNNs and HADNNs are applied to indoor localization datasets as examples of *multi-channel data* since the samples in the indoor localization datasets are obtained from multiple independent Wi-Fi routers. The description of the datasets is in the following subsection. The coarse classes in the indoor localization datasets can be either building and floor and fine classes are location coordinates. The main goal of indoor localization is the estimation of a location coordinate. Because location coordinates are highly related to building and floor information in indoor localization, building and floor information should be considered when training ANNs on indoor localization.

C-FNNs and HADNNs are built based on a baseline of two fully-connected layers with 128 and 68 nodes, respectively, which showed the best result with one of the three datasets in [38, 79]. For the regression task, the outputs of the baseline are set as two continuous targets: the longitude and latitude, which are referred to as location coordinates. Table 4.6 and Table 4.7 summarize the number of nodes of C-FNNs and HADNNs used for the case of

coarse classes with one level and for the case of coarse classes with two levels, respectively. We use a cosine annealing learning rate with a maximum learning rate of 0.01 for the TUT datasets and to 0.03 for the UJIIndoorLoc dataset. In the loss function, defined by adaptive loss balancing, the additional weight, k, is set to 10.

Table 4.6: The number of nodes of the models used for TUT datasets

Methods	Network	The number of nodes
C-FNN	F_1	128 - 64 - N_{c_1}
	F_2	128 - 64 - N_r
	D_1	64 - 128 - N_{input}
HADNN	F	128
	F_1^u, F_1^d, D_1	64
	F_{c1}	N_{c_1}
	F_f	N_r

Table 4.7: The number of nodes of the models used for UJIIndoorLoc dataset

Methods	Network	The number of nodes
C-FNN	$F1$	128 - 64 - N_{c_1}
	F_2	128 - 64 - N_{c_2}
	F_3	128 - 64 - N_r
	D_1, D_2	64 - 128 - N_{input}
HADNN	F	128
	F_1^u, F_1^d, D_1	64
	F_2^u, F_2^d, D_2	32
	F_{c1}	N_{c_1}
	F_{c2}	N_{c_2}
	F_f	N_r

4.2.6 Data description

Three open source datasets are used to evaluate C-FNNs and HADNNs. The first two of the three open source datasets are used to evaluate models for coarse classes with one level and the third of the datasets is used to evaluate models for coarse classes with two levels.

Case study for coarse classes with one level: TUT datasets

The first two of the three open source datasets mentioned above are collected at Tampere University of Technology, Finland as an example of coarse classes with one level. One was published in 2017, denoted by TUT2017 dataset [46], and the other in 2018, denoted by TUT2018 dataset [54]. Both datasets comprise training and test sets and both sets provide Received Signal Strengths (RSSs), floor, location coordinates. RSSs are used as inputs, while floor, longitude and latitude are used as targets. TUT2017 dataset was measured at 5 different floors in a building and TUT2018 dataset was measured at 3 different floors in a building. The distributions of the longitude and latitude of both datasets are shown in Figure 4.10. Since more data were collected for the test sets of both datasets, the distributions of the test sets are denser than that of the training sets. By considering floor and location coordinates predictions, we compute the final score:

$$\text{Final score}(x) = 4 \cdot F + Dist(C). \tag{4.17}$$

where F is the difference between the floor targets and the floor predictions of ANNs and $Dist(C)$ is the Euclidean coordinate mean distance between the location coordinates targets and the location coordinates predictions of ANNs. The score is modified from the proposed score in [57], which was originally suggested for building-floor-coordinate tasks.

Case study for coarse classes with two levels: UJIIndoorLoc dataset

The third of the datasets mentioned above, UJIIndoorLoc dataset [83] was collected from 3 buildings during different time periods. The UJIIndoorLoc dataset consists of trainingData and validationData. TrainingData comprises RSSs, building, floor, location coordinates, spaceID, relative position, userID and PhoneID, whereas validationData only consists of RSSs, building, floor, location coordinates and PhoneID. Since UJIIndoorLoc dataset contains building information which TUT datasets do not contain, we evaluate C-FNNs and HADNNs for coarse classes with two levels by setting the building information as the first level of the coarse classes and floor information as the second level of the coarse classes. Figure 4.11 shows that clusters of floors with respect to the location coordinates are hardly ever separated, while clusters of buildings with respect to the location coordinates are well-distinguished. This implies that the location coordinates regression may not take into account floor classification. Considering all predictions, we compute the final

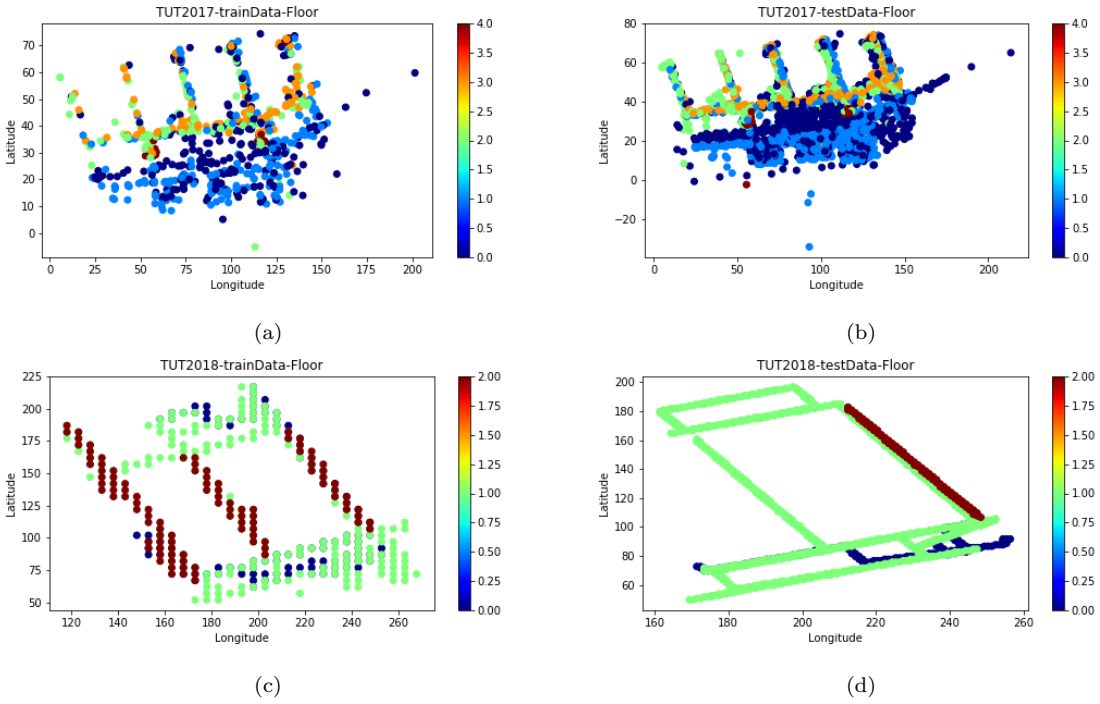


Figure 4.10: Distribution (a) floors in the training set (b) in the test set in the TUT2017 dataset, and (c) floors in the training set (d) the test set in the TUT2018 dataset.

score proposed in the competition [57],

$$\text{Final score}(x) = 50 \cdot B + 4 \cdot F + Dist(C), \tag{4.18}$$

where B is 1 if the building prediction fails and F is the difference between the floor targets and the floor predictions of ANNs, while $Dist(C)$ is the Euclidean coordinate mean distance between the location coordinates targets and the location coordinates predictions of ANNs, which is defined in [57]. The summary of all datasets is listed in Table 4.8.

4.2.7 Experiments

The main target of C-FNNs and HADNNs is the location coordinates prediction by the utilization of coarse classes. C-FNNs and HADNNs employ the coarse classes during the training phase. During the test phase, unlike HAL, they do not require target coarse classes but predict coarse classes and the location coordinates. For the purpose of performance

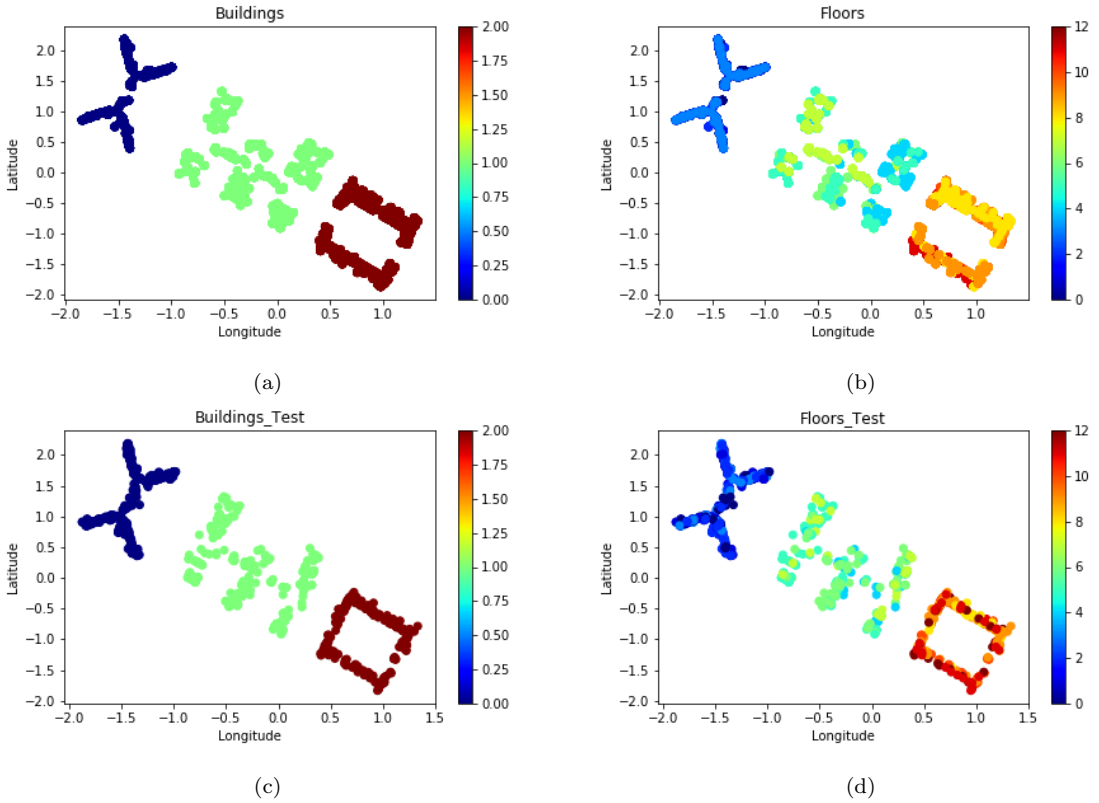


Figure 4.11: Distribution of (a) buildings in the training set (b) floors in the training set (c) buildings in the validation set (d) floors in the validation set in the UJIIndoor dataset.

evaluation, coarse class classification accuracy, fine class regression mean squared error and Final core proposed in subsection 4.2.6 are used. In addition, we compare our results with available methods which predict coarse classes and location coordinates.

TUT datasets

TUT datasets are used to evaluate C-FNNs and HADNNs for coarse classes with one level. All results are shown in Table 4.9 and Table 4.10. Apart from the floor accuracy in TUT2018 dataset, the best performance is obtained with HADNNs. Nevertheless, C-FNNs also achieve an equally good performance as HADNNs although C-FNNs require at least three times the parameters over the baseline while HADNN requires at most 1.1 times.

HADNNs results are compared with the results in [46], which used the TUT2017 dataset. In [46], the smallest Euclidean coordinate mean distance is 8.09,which was ob-

Table 4.8: Summary of datasets

Dataset	UJIIndoor	TUT2017	TUT2018
The number of training data	19937	697	446
The number of test data	1111	3951	982
The number of RSS values	520	992	489
The number of Buildings	3	N/A	N/A
The number of Floors	4 or 5	5	3

Table 4.9: TUT2017 dataset results of C-FNN and HADNN

Methods	Baseline	C-FNN	HADNN
Floor accuracy (%)	38.54	93.12	94.58
Euclidean coordinate mean distance (m)	11.31	9.12	9.05
Final score in Eq. (4.17)	19.14	11.77	11.75
The number of parameters	136,258	410,993	145,233

Table 4.10: TUT2018 dataset results of C-FNN and HADNN

Methods	Baseline	C-FNN	HADNN
Floor accuracy (%)	66.19	97.55	97.45
Euclidean coordinate mean distance (m)	13.88	10.62	9.91
Final score in Eq. (4.17)	19.96	14.33	13.34
The number of parameters	71,874	216,070	80,587

Table 4.11: TUT2017 dataset results comparison with previous works

Methods	Clustering	UJI kNN	RTLS@UM	HADNN
Floor accuracy (%)	90.81	92.26	90.05	94.58
Euclidean coordinate mean distance (m)	8.09	8.45	9.18	9.05

Table 4.12: UJIIndoor dataset results of C-FNN and HADNN

Methods	Baseline	C-FNN	HADNN
Building accuracy (%)	99.09	100.00	100.00
Floor accuracy (%)	32.40	92.34	93.15
Euclidean coordinate mean distance (m)	15.44	11.74	11.59
Final score in Eq. (4.18)	23.95	15.56	14.93
The number of parameters	75,458	382,248	90,584

Table 4.13: UJIIndoor dataset results comparison with previous works

Methods	RTLS@UM	Weighted Centroid	HADNN
Building accuracy (%)	100.00	99.91	100.00
Floor accuracy (%)	94.00	90.19	93.15
Euclidean coordinate mean distance (m)	7.73	9.27	14.93

tained with RSS clustering and the best floor accuracy is 92.26% by use of UJI kNN algorithm. With TUT2017 dataset, HADNNs performs better in all tasks compared with RTLS@UM and in the floor classification task compared with all previous results. Although RSS clustering and the UJI kNN algorithm show better performance in the Euclidean coordinate mean distance than HADNNs, both methods require a training set during the test phase, while HADNNs do not.

UJIIndoorLoc dataset

UJIIndoorLoc dataset is used to evaluate C-FNNs and HADNNs for coarse classes with two levels. A comparison between the baseline and C-FNNs and HADNNs are shown in Table 4.12. The best performance is obtained with HADNNs. C-FNNs also achieve as good performance as HADNNs, but require five times the parameters over the baseline while HADNNs only require 1.2 times. Baseline results show that ANNs designed for only the location coordinates prediction cannot take into account the coarse class of the targets.

The dataset used in the competition in the International Conference on Indoor Positioning and Indoor Navigation (IPIN) 2015 and a number of additional research have been carried out with UJIIndoorLoc dataset due to its large scale [57, 64, 32, 38, 79]. We compare the results of HADNNs with the best result in the competition, which is from the RTLS@UM team, and with the result by ANNs with Weighted Centroid (ANNs with WC) in [38]. All results are shown in Table 4.13. HADNNs perform better than ANNs with

WC, but have a slightly lower performance than RTLS@UM in a floor classification task. Despite this, the advantage of HADNNs is that a training set is not necessary during the test phase.

4.3 Summary

In this chapter, we propose two types of hierarchical learning models to utilize the an hierarchy of targets. They can be applicable to fields where it is easy to obtain a hierarchy of targets such as Computer Vision and indoor localization.

The first type of hierarchical learning models is HAL, a combined idea of auxiliary learning and hierarchical classification. Supposing target classes of images have coarse classes, the coarse class information is fed to the auxiliary block in neural networks. This auxiliary block consists of one fully-connected layer, an element-wise subtraction layer, an absolute layer and a summation layer. Taking outputs of a series of convolutional layers as an input, the auxiliary block transforms discrete coarse classes to the continuous auxiliary scores which can be distributed with respect to their coarse classes, thereby helping in the classification of fine classes.

HAL is tested with three baselines using three benchmark datasets. The accuracy was improved 0.5%, 1.52% and 3.51% at most in MNIST, Fashion-MNIST and SVHN, respectively. The auxiliary block improves the performance and does not have an impact on the number of coarse classes and on the number of parameters of neural networks. This is a powerful advantage as, normally, the number of coarse classes increases when a large amount of data is obtained.

The other type of hierarchical learning models are C-FNNs and HADNNs. They are proposed to deal with indoor localization problems as an example of *multi-channel data*. In C-FNNs and HADNNs, coarse classes of given data are provided and used during the training phase. There are two possible cases in indoor localization. The first case is coarse classes with one level since the data are gathered in one building; hence, the targets are floors and location coordinates. In the second case, coarse classes have two levels since the data are obtained in multiple buildings, meaning the targets are buildings, floors, and location coordinates. C-FNNs perform better than conventional ANNs. Despite this, its number of parameters increases greatly. In contrast, HADNNs, which are modified from C-FNNs, perform as well as C-FNNs and prevent the drastic increase in the number of parameters.

In order to validate C-FNNs and HADNNs, they are compared using three datasets. The C-FNNs increase floor accuracy by 55%, 30% and 60% in TUT2017, TUT2018 and UJIIndoorLoc datasets, respectively. However, C-FNNs require three times and five times the parameters in TUT and UJIIndoorLoc datasets, respectively. On the other hand, HADNNs achieve slightly better accuracy than C-FNNs, and HADNNs also need less parameters than C-FNNs. The biggest advantage of C-FNNs and HADNNs is that they do not require the training set and coarse classes at the test phase.

The results of the second type of hierarchical models show that training C-FNNs and HADNNs with coarse classes based on channels of data can improve the performance of a test set without coarse classes.

Chapter 5

Multi-channel Human Data

Human motion data are obtained from sensors on different body parts such as the head, hands, chest, feet and so on. These sensors are considered as channels in human motion data, which make human motion data a typical type of *multi-channel data* and the sensor locations important information. In addition, the targets of human motion data are motion types. The significance of the relationship between sensor locations and motion types can be identified by the common measurement patterns among subjects. For example, sensors on the back become significant for the motion type standing up because all subjects bend their backs to stand up. In this chapter, we derive characteristic functions which extract key features of human motion data based on the relationship between sensor locations and motion types. Subsequently, an algorithm to detect motion types using the characteristic functions is proposed. This algorithm is divided into five steps: preprocessing, weighting, characteristic functions, thresholds and detection. In the preprocessing step, angular displacement is calculated and noise was removed by WT. In the weighting step, the importance of measurements with respect to motion types is calculated. In characteristic functions, a decision tree of motion types is built following weighting and eight characteristic functions are derived to complete the decision tree. In the threshold step, the ROC curve is used to find the optimal thresholds to differentiate motion types based on the outputs of characteristic functions. Finally, using characteristic functions and the optimal thresholds, characteristic vectors are defined and used to detect motion types. The algorithm is evaluated with subjects and the result is compared with the conventional method, namely, the PCA.

5.1 Human motion data

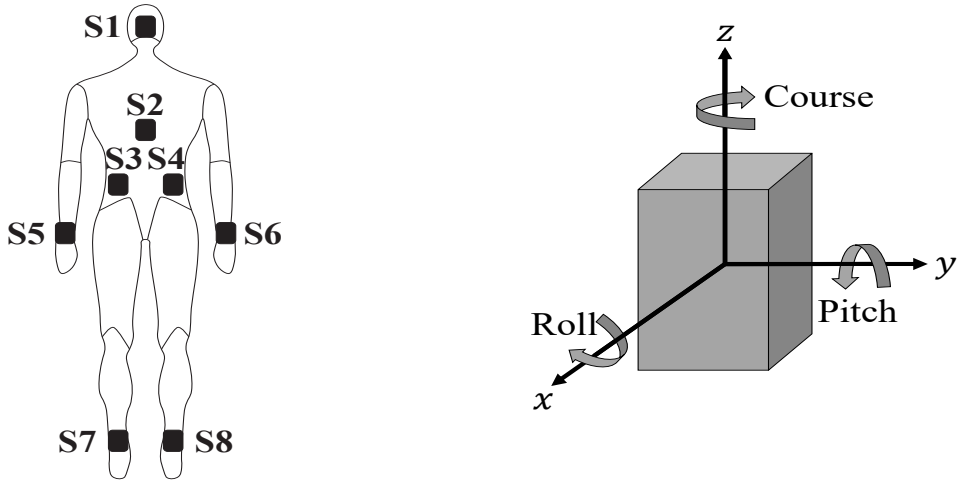
The characterization and quantification of the motion types of people have always been a viable route for assessing mobility. Depending on people's medical conditions, their ability to move different muscle groups will vary greatly, and thus, the exact mobility and resulting motion types may also vary [53]. Motion type detection can help diverse fields related to human health. For example, checking for changes in physical ability daily using motion type detection can prevent problems caused by the degeneration of physical abilities [103]. In motion type detection, the sensor locations play an important role. Many research has attached sensors to different parts of the body to capture various measurements over time [31, 90, 65, 15, 66, 68]. In those research, sensors were attached to some body parts such as the chest (sternum), back, waist, thighs and feet using IMU sensors that can measure three accelerometer measurements and angle measurements with respect to the origin. In this chapter, we introduce five steps of a motion type detection algorithm with experimental human data.

We apply nine different types of motions selected from the Berg Balance Scale Test [6], which is specialized for the determination of people's static and dynamic balance capabilities. The motion types are illustrated as follows:

- $\boldsymbol{M_1}$: Standing
- $\boldsymbol{M_2}$: Sitting
- $\boldsymbol{M_3}$: Sitting-to-standing
- $\boldsymbol{M_4}$: Standing-to-sitting
- $\boldsymbol{M_5}$: Transferring
- $\boldsymbol{M_6}$: Reaching forward (with stretched arms)
- $\boldsymbol{M_7}$: Retrieving an object from the floor
- $\boldsymbol{M_8}$: Turning to look behind
- $\boldsymbol{M_9}$: Turning 360°

where M_1 and M_2 are static motions and are measured on fixed time intervals. The other motion types are dynamic motions and are flexibly measured without time limitation.

In order to help understand the motion types, further descriptions for M_5 to M_9 are provided. Subjects are asked to transfer to a specific chair and back to their origin in M_5. In M_6, subjects are required to lift their arms 90 degrees and to stretch forward as far as possible. In M_7, subjects must pick up an object placed in front of their feet. M_8 asks subjects to turn around to look directly behind themselves, while M_9 requires subjects to turn around in a complete circle.



(a) (b)

Figure 5.1: (a) Sensor placement: the head, upper middle back, lower left back, lower right back, hands, and feet. (b) Directions of x, y, and z axis and orientation angles: roll, pitch, and course.

5.2 Decisive features extraction

Motion types are detected using characteristic vectors which represents motion types. Four steps to derive characteristic vectors including preprocessing, weighting, Characteristic functions and a decision tree and the ROC with threshold are introduce in the following subsections.

5.2.1 Measurement and noise removal with Wavelet Transform

In order to obtain the measurements, sensors are attached to eight different locations on the body: the head, upper middle back, lower left back, lower right back, hands and feet. The locations of the eight sensors are shown in Figure 5.1 (a). Each sensor produces three-axes accelerometer measurements and three angle measurements (Course, Pitch, and Roll) with respect to the origin, which are illustrated in Figure 5.1 (b). In this chapter, we denote a vector represents three accelerometer measurements, $\mathbf{a^i}$ and a vector represents three angle measurements $\mathbf{r}^i$ as follows:

$$\mathbf{a^i} = \begin{bmatrix} a_x^i \\ a_y^i \\ a_z^i \end{bmatrix} \text{ and } \mathbf{r^i} = \begin{bmatrix} r_c^i \\ r_p^i \\ r_r^i \end{bmatrix}$$

where i refers to a sensor location Si ($i = 1, \ldots, 8$) in Figure 5.1 (a), and subscripts (x, y, z) and (c, p, r) denote the three cartesian components of the accelerometer measurements, and the course, pitch and roll components of the angle measurements, respectively. Angular displacements on $\mathbf{r}^i$, which can be drawn from absolute angle measurements, provide meaningful information about motion types, and thus, we calculate the angular displacements in order to check for increases and decreases in the angle, expressed as:

$$\boldsymbol{\theta}^i = \begin{bmatrix} \theta_c^i \\ \theta_p^i \\ \theta_r^i \end{bmatrix}$$

where $\theta_k^i(0) = 0$, $\theta_k^i(t) = r_k^i(t) - r_k^i(t-1)$, $k \in \{c, r, p\}$ and $i = 1, \ldots, 8$. Additionally, the mean and standard deviation are denoted by μ and σ, respectively.

Fourier Transform (FT) is one effective methodology for signal processing [37, 80, 2, 16, 33], but it is limited to analyzing non-stationary signals. As a result, Wavelet Transform (WT) is considered so that we may overcome the limitation as it provides wide resolution over a time interval [50]. In addition, WT also has the ability to delete noise from the signals. Wavelet basis function $\mathcal{T} \in L^2(\mathbb{R})$ has the structure such that $\{D^j T_k \mathcal{T}\}_{j,k \in \mathbb{Z}}$ forms the orthonormal basis of $L^2(\mathbb{R})$, where T_k is the time shifting,

$$T_k(x) = T(x - k),$$

where D is the dilation operator, $D_c : L^2(\mathbb{R}) \to L^2(\mathbb{R})$, hence $D^j f(x) = 2^{\frac{j}{2}} f(2^j x)$. Then, reconstruction from its Wavelet coefficient $< f, D^j T_k \mathcal{T} >$ to a signal is obtained by

$$f = \textstyle\sum_{j,k \in \mathbb{Z}} < f, D^j T_k \mathcal{T} > D^j T_k \mathcal{T}.$$

All basis functions provide scaled time-shifting, same as in the mother wavelet. In this paper, the *db4* wavelet is used to remove noise from the observed signals, which has shown effectiveness in the analysis of IMU sensors [59, 87]. In order to remove noise, the signals are decomposed by the *db4* wavelet and values in the decomposed signals are replaced if they exceed a predefined threshold. Finally, the decomposed signals are recomposed by the *db4* wavelet.

5.2.2 Sensor location decision

Motion types are highly related to movements of certain body parts. For example, the head and hands would be moving for all motion types, and the back would give important information for the standing up and sitting down motion types, since back movement is expected with these motion types. This shows the significance of the knowledge of the relationship between the locations of the sensors and the motion types. The importance of the locations of the sensors is decided by comparing the patterns of measurements from the sensors and motion types. Even though multiple trials are carried out by different subjects, the same motion types should provide the same measurement patterns.

We face two difficulties in determining the importance of the locations of the sensors with respect to the motion types. The first difficulty is called local perturbation. There are sometimes inconsistent measurement patterns when studying the same motion type on different people due to these people's extra subconscious body movements. A local perturbation is changes of measurement between two extremal values where the standard deviation, σ', of the changes is less than the standard deviation, σ, measured for the entire motion.

$$\text{Local perturbation}, \quad \text{if } \sigma' < \sigma$$

We refer to measurement patterns after removing local perturbation as a global pattern. By comparing global patterns, common measurement patterns among subjects are more likely to be captured. The next difficulty is personal dependence. Measurement amplitudes show a large range in variations from different subjects. In this case, it is not possible to apply correlations to measurements from the same locations obtained from different subjects.

In order to solve these two difficulties, angle measurements are used to determine the locations of the sensors. The main reason for this is that angle measurements are less affected than accelerometer measurements, as a result, they suffer less from local perturbation. By comparing the common measurement patterns among subjects with respect to motion types, weights are assigned to a pair of a measurement and a motion type proportional to the number of common measurement patterns among subjects. The results of the weights between the angle measurements and the motion types are illustrated in Figure 5.2.

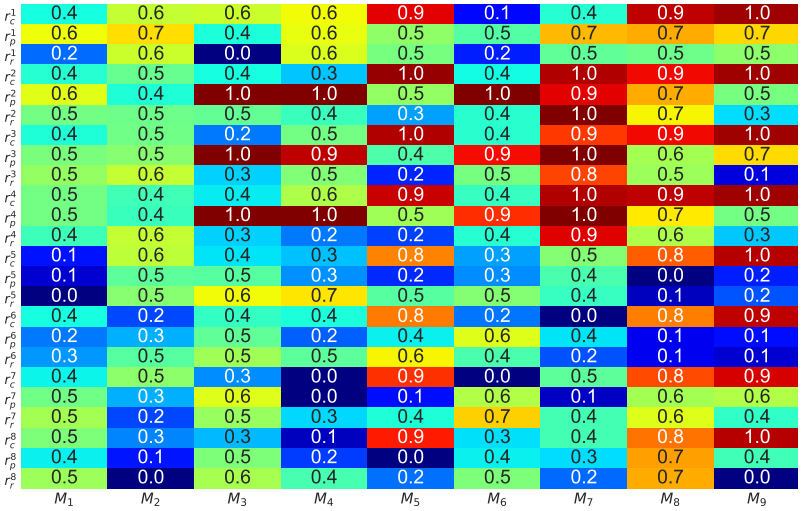


Figure 5.2: Weights for each feature with respect to motion types. Features, have similar patterns in all subjects, have high weights. Otherwise, weights are low.

5.2.3 A decision tree and characteristic functions

In this subsection, a decision tree and characteristic functions are designed based on the weights obtained in subsection 5.2.2.

The nine motion types are divided into a set of hierarchically organized groups, $G_1, \ldots, G_8$ based on the weights of the angle measurements as shown in Figure 5.2. First, motion types are separated into two groups according to the weights. For example, M_1 and M_2 do not have weights over 0.8, while the other motion types have at least one weight above 0.8. We define M_1 and M_2 as G_1 since they show rather little movement. The rest of the motion types are grouped as G_2. Under G_2, M_5, M_7, M_8 and M_9 are included in the same group called G_4 because of their high weights in r_c^2, r_c^3 and r_c^4. The other three motion types are grouped as G_3 according to the weight values of r_p^2. All relationships between groups and motion types are illustrated with hierarchical structure in Figure 5.3.

Identifying motion types in G_1

Now, we analyse each motion type in more detail. The weight of pitch and course measurements from the sensors on the back show a big difference between the G_1 and G_2 from

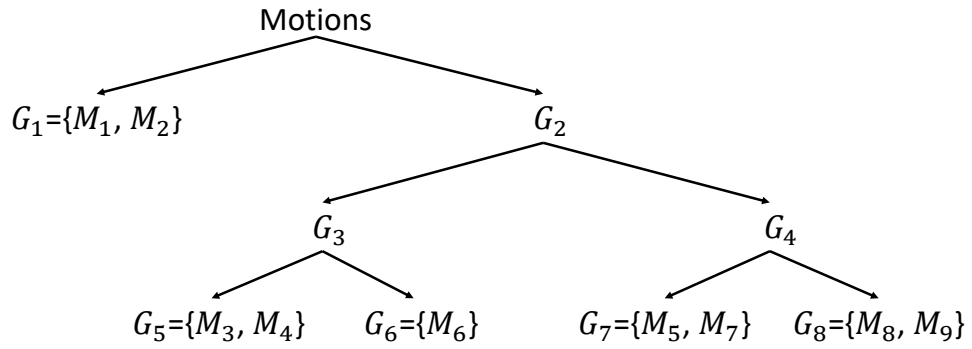


Figure 5.3: A decision tree of motion types.

Figure 5.2. Because motion types in G_1 do not require any movements, pitch and course measurements from the sensors on the back do not show any patterns in motion types for G_1, while they continue to change in motion types for G_2. This means that the standard deviations of pitch and course measurements from sensors on the back can be the boundary values used to differentiate G_1 and G_2. The characteristic function is given by:

$$p_1 = \begin{cases} 1, & \text{if } max(S) \leq \tau \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

where $S = \{\sigma(r_p^2), \sigma(r_p^3), \sigma(r_p^4)\}$ and τ is the required threshold. Eq. (5.1) states that a motion type falls under G_1 when $p_1 = 1$, and G_2 when it does not. We call G_1 a static group. Furthermore, from the fact that the tilt of the trunk varies with each subject [100], the degree of bend of the trunk, β, can be computed by:

$$\beta = E(r_p^2) - E(r_p^k) \tag{5.2}$$

This can be used to discriminate M_1 and M_2, where r_p^k is r_p^3 or r_p^4. When r_p^3 is greater than r_p^4, M_1 can show positive bending, implying forward positioning, and otherwise M_2 can show negative bending. Hence, M_1 and M_2 are discriminated by:

$$p_2 = \begin{cases} 1, & \text{if } \beta \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

where $p_2 = 1$ indicates forward bending, and $p_2 = 0$ is backward bending.

Identifying motion types in G_4

When a motion type is not in G_1, it can be in any of G_5–G_8, which covers motion types M_3–M_9. Among these, M_5 and M_7–M_9 show high values in weights for r_c^2, r_c^3 and r_c^4, primarily resulting from the change in r_c. Hence, these motion types show a strong connection between the change in r_c and G_4. Therefore, we use the standard deviation of r_c to discriminate G_3 from G_4. Thus,

$$p_3 = \begin{cases} 1, & \text{if } mean(S) \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{5.4}$$

where S is $\{\sigma(r_c^2), \sigma(r_c^3), \sigma(r_c^4)\}$ and τ is the required threshold. Eq. (5.4) states that a motion type belongs to the set G_4 when $p_3 = 1$.

Once the motion types in G_3 and G_4 are differentiated, the next logical step is to separate motion types falling under G_7 and G_8. Inside of G_4, the difference between G_7 and G_8 can be noted using the variation of course and pitch measurements. M_5 and M_7 indicate that subjects need to bend their trunk. Thus the standard deviation of the course measurement and the minimum value of pitch measurement can be collectively used as the criterion for G_7 and G_8 detection, which are given by:

$$p_4 = \begin{cases} 1, & \text{if } min\{min(r_p^3), min(r_p^4)\} \geq \tau_1 \text{ and } \sigma(r_c^2) \geq \tau_2 \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

where τ_1 and τ_2 are the thresholds. With Eq. (5.4) and Eq. (5.5), a motion type belongs to G_8 when $p_3 = p_4 = 1$, and to G_7 when $p_3 = 1 \wedge p_4 = 0$.

The weight of course measurements on the feet is high only in M_5. The difference between M_5 and M_7 is in the weight of the course measurement (from the sensor on the feet), as M_5 requires subjects to move from one position to another. In addition, the trunk and feet turn at the same time to transfer subjects to another chair. Therefore, the correlation between r_c^3 and r_c^7 and the correlation between r_c^4 and r_c^8 are both calculated in order to differentiate M_5 and M_7:

$$p_5 = \begin{cases} 1, & \text{if } min\{\rho\{r_c^3, r_c^7\}, \rho\{r_c^4, r_c^8\}\} > \tau \\ 0, & \text{otherwise} \end{cases} \tag{5.6}$$

where ρ measures the correlation between two measurements and τ is the obtained set of thresholds. Here, $p_5 = 1$ means the lower back and feet turn at the same time.

M_8 and M_9 both require turning around the z-axis, which results in both having high weights in course measurements of the sensors on the feet. However, the corresponding weights for M_9 are slightly higher than that of M_8. This means that M_9 requires subjects to do the same movements for the course measurements on the feet. More specifically, M_9 demands a minimum of a 300° turn, ideally a 360° turn, whereas M_8 does not demand any specific range of angle measurements for the turn. Therefore, the maximum difference of r_c^2, r_c^3, and r_c^4 can be calculated to differentiate M_8 and M_9:

$$p_6 = \begin{cases} 1, & if \begin{array}{l} max\{\Theta(r_c^3), \Theta(r_c^4)\} > \tau_1 \text{ and} \\ max\{\Theta(r_c^7), \Theta(r_c^8)\} > \tau_2 \end{array} \\ 0, & \text{otherwise} \end{cases} \tag{5.7}$$

where $\Theta(r_c) = max(r_c) - min(r_c)$ and τ_1, τ_2 are the thresholds. Here, $p_6 = 1$ indicates the motion type results in a turn exceeding the corresponding threshold.

Identifying motion types in group G_3

Given that M_3, M_4, and M_6 require subjects to bend their trunk, these motion types all have the same pattern on the pitch measurement from the sensors attached to the back of their body. The differences between the three motion types M_3, M_4 and M_6 are in the information of the accelerometer measurement about the z-axis. The accelerometer measurement along the z-axis for M_3 and M_4 tend to show radical changes when compared to the corresponding measurements for M_6. This is because the height variations of M_3 and M_4 are larger than that of M_6. Thus, by relying on the standard deviations of the corresponding sensor measurements, G_5 and G_6 can be differentiated.

Let $\Delta t = t(min(r_p^2)) - t(min(\theta_p^2))$ be a time interval to distinguish M_6 from M_3 and M_4, since the trunk will stop at $min(r_p^2)$ when subjects only bends at the trunk. Otherwise, the trunk movement will continue changing. Then, M_6 is detected by:

$$p_7 = \begin{cases} 1, & \text{if } \sigma(a_z^2)_{on\,\Delta t} < \tau \\ 0, & \text{otherwise} \end{cases} \tag{5.8}$$

where τ is the threshold. Here, $p_7 = 1$ means that the change in the measurements from the sensors on the back is small.

Table 5.1: Characteristic vectors corresponding to motion type

Motion	Vector
M_1	[1, 1, $*$, $*$, $*$, $*$, $*$, $*$]
M_2	[1, 0, $*$, $*$, $*$, $*$, $*$, $*$]
M_3	[0, $*$, 0, $*$, $*$, $*$, 0, 1]
M_4	[0, $*$, 0, $*$, $*$, $*$, 0, 0]
M_5	[0, $*$, 1, 0, 1, $*$, $*$, $*$]
M_6	[0, $*$, 0, $*$, $*$, $*$, 1, $*$]
M_7	[0, $*$, 1, 0, 0, $*$, $*$, $*$]
M_8	[0, $*$, 1, 1, $*$, 0, $*$, $*$]
M_9	[0, $*$, 1, 1, $*$, 1, $*$, $*$]

$*$ can be any value.

Now, the final task is to differentiate M_3 and M_4. It is worth noting that the velocity directions of M_3 and M_4 are opposite since the height change of the two motion types are opposing. In fact, there are techniques outlined in literature that utilize time interval calculations in order to differentiate two motion types [58, 59, 23, 60]. Here, we calculate the average velocity of S_2, which denotes the change in height for M_3 and M_4, about the z-axis. This is defined by $\mu[v_z^2]$ over the time interval $[t_1, t_2]$ as follows:

$$\mu(v_z^2) = \frac{\int_{t_1}^{t_2} v_z^2(t)dt}{t_2 - t_1}$$
$$\text{where } \mathrm{v_z^2(y)} = \int_{\mathrm{t_1}}^{\mathrm{y}} \mathrm{a_z^2(t)dt},$$

where t_1 and t_2 are the minimum points of θ_p^2 and r_p^2, respectively. With $\mu(v_z^2)$, M_3 and M_4 are differentiated by:

$$p_8 = \begin{cases} 1, & \mu(v_z^2) > \tau \\ 0, & \text{otherwise} \end{cases} \tag{5.9}$$

where τ is a threshold. Here, $p_8 = 1$ means that the motion type is M_3, and $p_8 = 0$ means that the motion type is M_4.

5.2.4 Decision algorithm

The hierarchical structure in Figure 5.3 provides an effective way to separate the nine motion types. The characteristic vector of each motion type is defined to separate the motion

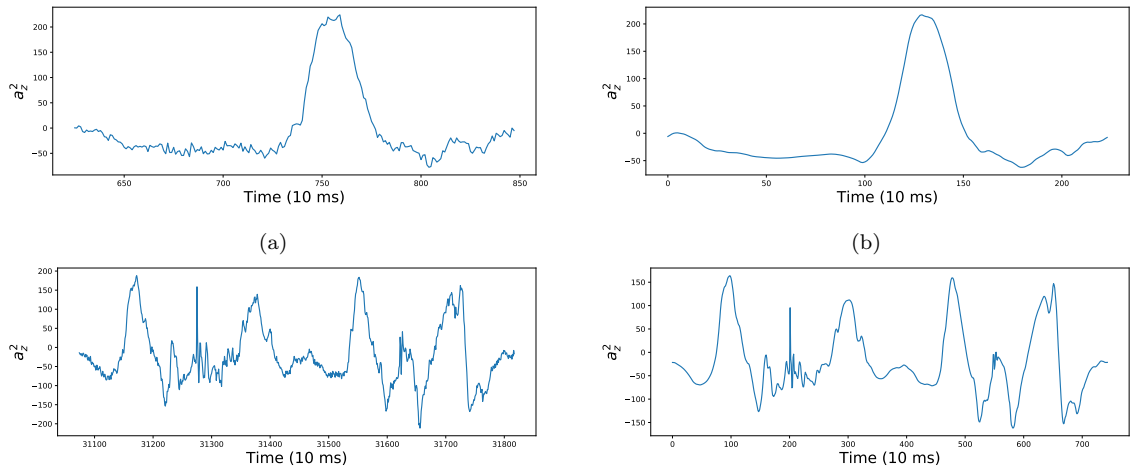


Figure 5.4: (a) Raw a_z^2 reading of a subject when sitting to standing (b) The result of (a) by applying *db4* wavelet. (c) Raw a_z^2 reading of a subject when Transferring (d) The result of (c) by applying *db4* wavelet.

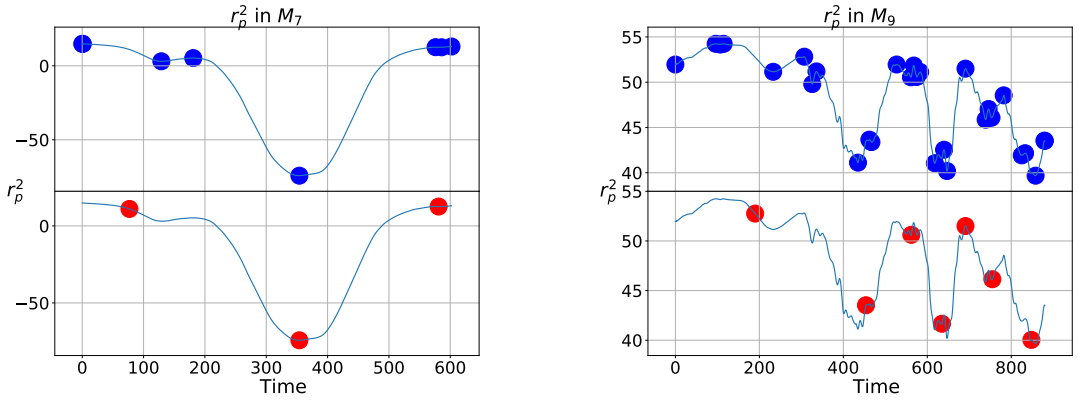


Figure 5.5: Example outputs of the proposed algorithm. The top row contains the inputs, and the bottom row shows the resulting outputs for each case (a) and (b).

types. These vectors are derived from the eight characteristic functions with thresholds and are illustrated in Table 5.1. In this section, we propose an algorithm that can perform motion type detection.

Preprocessing and feature selection

We apply WT with *db4* wavelet to remove noise. Noises in measurements are replaced with the threshold σ_{th}:

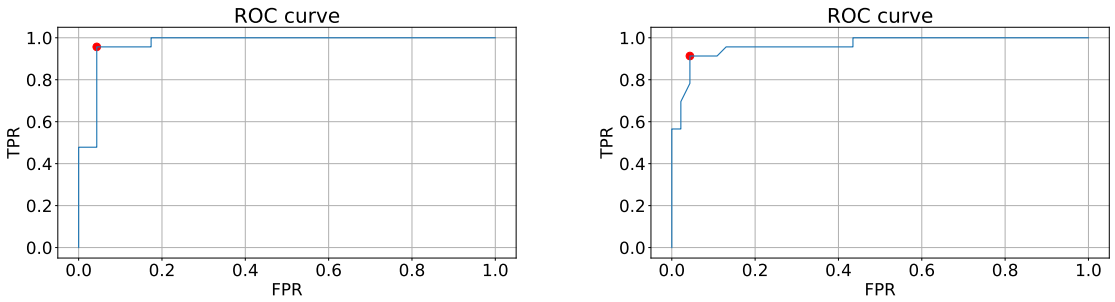
$$\sigma_{th} = \sigma_1 \sqrt{2 \log N},$$

where $\sigma_1 = MAD_1/0.675$ as defined in [34], MAD_1 denotes the median absolute deviation of the detailed component at scale $j = log_2 N$ and N is the number of samples in the measurement. Examples of noise removal are shown in Figure 5.4. In Figure 5.4, measurements on the right side become smooth using the *db4* wavelet. In order to decide the motion type, weights have to be assigned to measurements by exploiting the dominant features of these measurements. Algorithm 2 outlines the steps for computing these weights by comparing global patterns with a specific measurement in a given motion. As shown in Figure 5.5, local perturbations are defined and removed in order to get a good comparison between two measurements. In Figure 5.5, we use Algorithm 2 to show the measurement before (top-row) and after (bottom-row) the processing. The red points obtained after processing turn into binary codes and are then used to obtain weights.

Threshold values

We use a Receiver (or Relative) Operating Characteristic (ROC) curves to find the optimal threshold [21]. Since the 1950s, ROC curves have been used for the purpose of signal detection and analysis based on statistical decision theory [55]. Operating point decisions for instruments were considered with the comparison of true negatives and false negatives and also provided an optimum tradeoff between false positives and false negatives [49]. In the previous research, the optimal threshold values were found by maximizing true positive rate (TPR) + 1 - false positive rate (FPR) [7, 1, 63, 21]. Nevertheless, threshold values for each motion type are determined by maximizing the sum of the true positive (TPR) and true negative (TNR) rates instead of TPR + 1 - FPR in this thesis. In other words, a set of the optimal thresholds, $\{\tau_{th}\}$, is found using the following optimization problem:

$$\underset{\tau_{th}}{\text{maximize}} \ (TP + TN) \tag{5.10}$$



(a) (b)

Figure 5.6: The ROC curves for finding the threshold for the characteristic function (a) p_7 and (b) p_8.

Although the formulation

$$\underset{\tau_{th}}{\text{maximize}} \ (TPR + 1 - FPR) \tag{5.11}$$

was used in the previous research [7, 1, 63, 21]. This is because Eq. (5.10) is more easily calculated and both formulations yield a similar set of results. To show this, let C_1 and C_0 denote the positive and negative groups, respectively. In order to obtain a threshold of interest, τ, we performed a number of simulations covering a range of values for each τ. Then, the relation is satisfied as follows:

$$\begin{aligned} & TPR + 1 - FPR \\ &= \frac{T^+}{T^+ + F^-} + 1 - \frac{F^+}{F^+ + T^-} \\ &= \frac{T^+}{T^+ + F^-} + \frac{T^-}{F^+ + T^-} \\ &= \frac{T^+}{\#\{C_1\}} + \frac{T^-}{\#\{C_0\}} \\ &= \frac{T^+ + T^-}{N}, \text{ for } \#\{\mathrm{C}_1\} \simeq \#\{\mathrm{C}_0\} \simeq \mathrm{N}, \end{aligned}$$

where T^+, T^-, F^+ and F^- are true positive, true negative, false positive and false negative, respectively. Consequently, threshold values that maximize TPR +1 - FPR and TP + FP are the same when two groups, C_1 and C_0, have the same number of elements. Here, the examples of the ROC curve for the threshold values are illustrated in Figure 5.6 and threshold values for each characteristic function are decided as illustrated in Table 5.2.

Table 5.2: Threshold obtained by the ROC curve

Characteristic function	Type of values		Threshold	
	τ_1	τ_2	$\tau_e(\tau_{e1})$	τ_{e2}
p_1	$\sigma(\boldsymbol{r})$	-	2.013	-
p_2	$\boldsymbol{r}$	-	-1.1484	-
p_3	$\sigma(\boldsymbol{r})$	-	13.7673	-
p_4	$\boldsymbol{r}$	$\sigma(\boldsymbol{r})$	-28.0571	25.1001
p_5	correlation	-	0.8651	-
p_6	$\sigma(\boldsymbol{r})$	$\sigma(\boldsymbol{r})$	234.9043	181.5007
p_7	$\sigma(\boldsymbol{a})$	-	13.4068	-
p_8	velocity		97.7670	-

5.3 Experiments

In order to verify that the characteristic functions and algorithms are of practical utility, in collaboration with the INHA University Hospital in South Korea, we evaluated our algorithms using experimental data.

5.3.1 Data description

The experimental data were gathered for all nine different motion types with eight sensors using 23 subjects. The subjects were divided into three distinct groups so that:

- The first and second groups consist of 12 men and seven women who were under medical treatment in INHA University Hospital due to brain diseases such as Cerebral Infarction, Thalamic Infarction and Cerebellar Degeneration.
- In the third group, there was one patient, who was under neurological medical treatment, and three students, all of whom had no history or diagnosis of neurological conditions. The data from this group were collected at a different time period with the first and second groups of subjects.

The subjects in the first group had difficulty in completing all nine motion types. Consequently, they were required to perform seven motion types, from M_1 to M_7. The subjects in the second group were able to perform all motion types, from M_1 to M_9. The subjects in the third group carried out all nine motion types. We used the Mymotion of Noraxon sensors (Noraxon USA Inc, Scottsdale, AZ) to obtain inertial measurements.

5.3.2 Results

For the purpose of performance evaluation, four metrics, recall, precision, accuracy and F-measure are used in this chapter [51, 69]. Recall represents the ratio of true positive motion types to conditioned positive motion types. Precision represents the ratio of true positive motion types to motion types, identified as positive by the algorithm. Accuracy represents the proportion of the sum of true positive motion types and true negative motion types. F-measure represents harmonic mean of the recall and precision. The results are shown in Table 5.3 and Table 5.4.

Table 5.3: Recall and Precision

	Recall			Precision		
Motion	Proposed algorithm	PCA	Percentage increase	Proposed algorithm	PCA	Percentage increase
M_1	0.9130	0.7391	+17.39%	0.8077	0.5484	+25.93%
M_2	0.7826	0.3478	+43.48%	0.9000	0.6154	+28.46%
M_3	0.6957	0.5652	+13.05%	0.9412	0.3095	+63.17%
M_4	1.0000	0.2609	+73.91%	0.7419	0.6667	+7.52%
M_5	1.0000	0.2174	+78.26%	1.0000	0.4545	+54.55%
M_6	0.8261	0.7391	+8.70%	0.8636	0.5152	+34.84%
M_7	0.8696	0.6957	+17.39%	0.9091	0.4706	+43.85%
M_8	1.0000	0.3636	+63.64%	1.0000	0.8000	+20.00%
M_9	1.0000	0.4545	+54.55%	1.0000	1.0000	+0.00%
Mean	0.8986	0.4870	+41.15%	0.9071	0.5978	+30.92%

Table 5.4: Accuracy and F-measure

	Accuracy			F-measure		
Motion	Proposed algorithm	PCA	Percentage increase	Proposed algorithm	PCA	Percentage increase
M_1	0.9617	0.8907	+7.10%	0.8571	0.6296	+22.75%
M_2	0.9617	0.8907	+7.10%	0.8372	0.4444	+39.28%
M_3	0.9563	0.7869	+16.94%	0.8000	0.4000	+40.00%
M_4	0.9563	0.8907	+6.56%	0.8519	0.3750	+47.69%
M_5	0.9945	0.8689	+12.56%	1.0000	0.2941	+70.59%
M_6	0.9617	0.8798	+8.19%	0.8444	0.6071	+23.73%
M_7	0.9727	0.8634	+10.93%	0.8889	0.5614	+32.75%
M_8	1.0000	0.9563	+4.37%	1.0000	0.5000	+50.00%
M_9	1.0000	0.9672	+3.28%	1.0000	0.6250	+37.50%
Mean	0.9739	0.8883	+8.56%	0.8977	0.4930	+40.48%

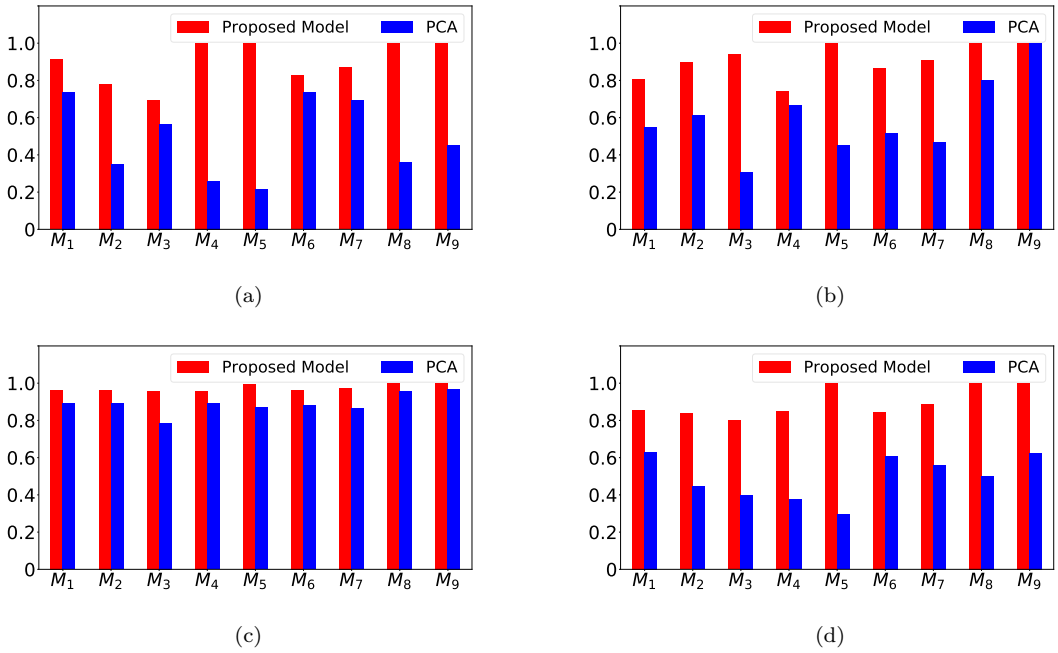


Figure 5.7: Four metrics for a comparison between the proposed method and the Principal Component Analysis: (a) Recall, (b) Precision, (c) Accuracy and (d) F-measure.

In our results, the mean value of the recall across all nine motions is 0.8986. M_3 has the lowest percentage of recall, and M_4, M_5, M_8 and M_9 show 100% recall. The mean value of the precision of the nine motions is 0.9071 and the lowest percentage is found in M_4. M_5, M_8 and M_9 show 100% precision. For accuracy, the mean value of the nine motions is 0.9739. The lowest value is found in M_3 and M_4, and 100% accuracy is found in M_8 and M_9. For F-measure, the mean value of the nine motions is 0.8977. The lowest value is found in M_3, and 100% accuracy is found in M_5, M_8 and M_9. The results show that characteristic vectors consisting of a few features are able to detect motion types accurately.

As the weighing and characteristic functions steps are related to feature extraction, we compare our results with PCA by replacing these steps with PCA, which is widely used for feature extraction [1, 35]. Following the paper [1], we calculate the minimum and maximum values, the mean value, variance, skewness, kurtosis, first five peaks of the discrete Fourier transform of the measurement and the corresponding frequencies. Now, $1,152 (= 72 \times 16)$ features are used to discriminate each motion. With PCA, we reduce the

number of features from 1,152 to 30. Subsequently, the optimal thresholds to differentiate motion types at each node in the decision tree in Figure 5.3 are found. Based on the thresholds, the motion types are differentiated. The result is illustrated in Table 5.3 and Table 5.4, and visualized in Figure 5.7. The mean values of recall, precision, accuracy and F-measure based on PCA are 0.4870, 0.5978, 8883 and 0.4930, respectively. These results are lower than the results based on a few extracted features from the characteristic functions. Specifically, recall, precision and F-measure show much higher values in all motion types in the proposed algorithm than in PCA. In both methods, M_2 and M_3 have lower recall. This is due to most M_2 being detected as M_1, most M_3 as M_4, and some M_3 as M_6. Because M_6 is the midway point of M_3 and M_4, a more detailed study and/or new sensor placements are needed in order to properly detect M_6.

5.4 Summary

In this chapter, relationships between sensor locations and motion types are studied as research on human motion data is a typical example of *multi-channel data*. The data are obtained using multiple sensors, which measure three accelerometer measurements and three angle measurements from different human body parts.

The detection algorithm is divided into five steps. In the first step, preprocessing is done to calculate angular displacement and to remove noise by WT. In the second step, the importance of measurements with respect to motion types is calculated by comparing measurement patterns among subjects. High weights are assigned to measurements which provide the common measurement patterns among subjects. In the third step, a decision tree of motion types is defined based on the weights and the characteristic functions are proposed to differentiate motion types at each node of the decision tree. In the fourth step, the optimal thresholds for the characteristic functions are defined based on the ROC curves. Finally, the motion types are identified based on the characteristic vectors.

In order to evaluate the proposed algorithm, human motion data were gathered from Inha University Hospital, Korea. 23 subjects, including patients in the hospital, participated in the experiment. They were required to perform nine motion types as far as their physical abilities could permit. The nine motion types are standing, sitting, sitting to standing, standing to sitting, transferring, reaching forward, retrieving an object from the floor, turning to look behind and turning 360°. First, the motion types are divided based on their dynamics. Standing and sitting are included in less dynamic groups, which is

referred to as a static group, and the other motion types formed a dynamic group. In the dynamic group, the motion types are further divided into several groups based on pitch and course measurements. Sitting to standing, standing to sitting and reaching forward are highly related to pitch measurements while the other motion types are highly related to course measurements.

Using the characteristic vectors from the characteristic functions and thresholds, the performance of the proposed algorithm was evaluated on recall, precision, accuracy and F-measure. The average of recall, precision, accuracy and F-measure are 0.8986, 0.9071, 0.9739 and 0.8977, respectively and the results outperform PCA.

We can conclude that studying the relationship between channels and targets of human motion data has a significant effect when detecting motion types.

Chapter 6

Multi-Channel Data Analysis with Machine Learning

In this chapter, we detect motion types using the proposed models from Chapter 3 and Chapter 4. The generative models proposed in Chapter 3 generate synthetic data in order to augment the amount of data and prevent overfitting in neural networks. Then, a combination of labelled synthetic data and experimental data is fed to the neural networks. When coarse classes are defined in human motion data, the proposed hierarchical learning models from Chapter 4 are used to utilize these coarse classes. A flowchart diagram illustrated in Figure 6.1 shows the entire process proposed in this chapter. Additionally, nine open source human motion datasets, alongside the dataset studied in Chapter 5, are used to evaluate the proposed process. In this chapter, we refer to the dataset from Chapter 5 as the InhaMotion dataset. In the following sections, we describe the details of each procedure.

6.1 Motion type detection process

In this section, to overcome some issues of the human motion data, the proposed generative model in Chapter 3 and the hierarchical learning models in Chapter 4 are used. The first issue is that human motion data are difficult to obtain due to both the complexities in the approval process and the gathering of subjects. In addition, abundant amounts of human motion data are required in machine learning methods because various human motion data samples will arise when multiple channels are combined. This is addressed by generating

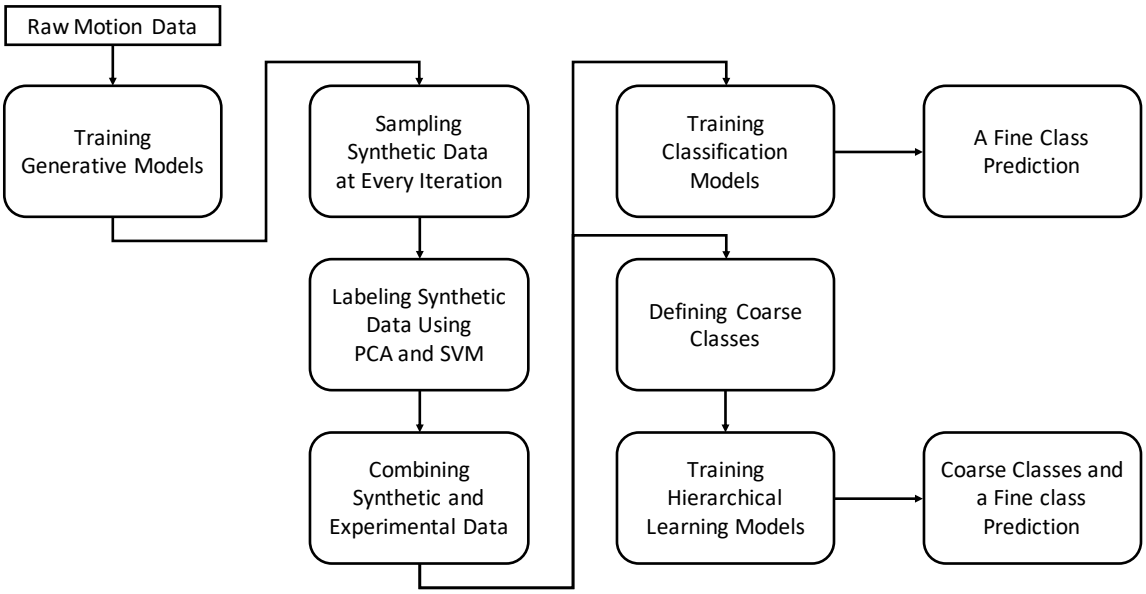
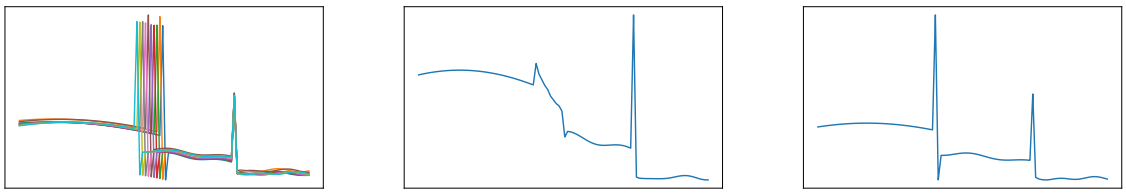


Figure 6.1: A flowchart diagram showing the pipeline to classify motion data with or without coarse classes.

synthetic data. The second issue is that motion types, which are targets of human motion data, can be hierarchically structured based on multiple channels. This is addressed by applying the proposed hierarchical learning models to utilize the hierarchical structure of motion types. To this end, we propose a process to detect motion types as follows: generation, labelling and classification.

6.1.1 Training generative neural networks

Data acquisition is one of the most difficult challenges in training neural networks with human motion data. This can be addressed by data augmentation. For example, in image classification, data augmentation such as cropping, flipping and rotating greatly improves performance of a test set [77]. However, these techniques are not suitable for human motion data – which is a kind of time series data – because cropping, flipping and rotating are designed for data with spatial information [18]. Research on data augmentation in a times series is being conducted [92]. One promising technique is Dynamic Time Warping Barycenter Averaging (DBA), which averages multiple time series data [19]. Although this technique generates better plausible synthetic data than the arithmetic average technique,



(a) (b) (c)

Figure 6.2: A time series synthetic data generation from multiple time series data. (a) Observed multiple time series data, (b) generation by arithmetic average and (c) generation by DBA. (b) fails to generate a pattern of (a) at time around 50 while (c) succeed to generate the pattern.

DBA only generates synthetic data that are very similar to the observed data, as shown in Figure 6.2. Therefore, it is necessary to design a new generative model. We use the proposed generative model in Chapter 3 to generate synthetic human motion data. The synthetic data are used to train a classification model to prevent overfitting. Since training classification tasks require labelled synthetic data, we introduce the labelling synthetic data process in the next section.

6.1.2 Labelling synthetic data

In order to label synthetic data, we modify the labelling process from Chapter 3,which uses only SVM for labelling synthetic data. This modification is needed due to a difference in the dimension of the latent space between images and *multi-channel data*. While the dimension of the latent space of images is fixed, the dimension of the latent space from *multi-channel data* increases proportional to the number of channels. This might cause poor labelling using SVM because machine learning algorithms show poor performance in high-dimensional space, which is the phenomenon called curse of dimensionality [17]. Subsequently, we use PCA to reduce the dimensions of the latent space regardless of the number of channels. In our case, human motion data is mapped into a latent space and the dimension of the latent space is reduced by PCA. Consequently, hyper-planes are found to assign labels to the synthetic data using SVM. This labelling process is outlined in Algorithm 1.

Algorithm 1 Labeling synthetic data

Input: Experimental data
Stage: Fitting PCA and SVM
    Map experimental data into a latent space.
    Fit PCA based on the latent variables.
    Fit SVM based on the output from PCA.
Stage: Labeling synthetic data
    Generate synthetic data
    Map synthetic data into a latent space.
    Transform synthetic data using PCA.
    Obtain labels using SVM.

6.1.3 Motion type detection using the proposed models

The obtained labelled synthetic data are fed to neural networks alongside the original training set. The number of the labelled synthetic data is half of the number of the original training set. This allows the neural networks to focus on the training set more than synthetic data and to prevent overfitting by synthetic data, which results in generalization of neural networks.

In order to observe the effect of coarse classes on a motion type detection, we start by doing an experiment without coase classes. ResNet is used as a baseline because ResNet performs well when applied to time series data classification [91]. ResNet can be divided into two parts: a series of residual blocks and a series of fully-connected layers. The role of a series of residual blocks is to extract important features whereas the role of a series of fully-connected layers is to predict targets based on the extracted features. In human motion type detection, the series of residual blocks extracts important features from human motion data and the series of fully-connected layers predicts motion types. Taking into account this fact, we modify ResNet to utilize coarse classes. As we aim to utilize coarse classes when predicting targets, we replace a series of fully-connected layers with C-FNNs and HADNNs, which are denoted as ResNet with C-FNNs and ResNet with HADNNs, respectively. Because C-FNNs and HADNNs were used when fine classes were continuous in Chapter 4, we need to modify adaptive loss balancing to use C-FNNs and HADNNs in motion type detection where the fine classes are discrete. The new loss scheme is proposed in the next subsection.

6.1.4 Adaptive Loss Weighting

Adaptive Loss Balancing in Section 4.2.1 is proposed to reduce the scale difference between a classification task and a regression task. Since all tasks related to coarse classes and fine classes are classification tasks, we propose a new type of loss scheme, which we name Adaptive Loss Weighing. Its aim is to make neural networks learn the training set starting from coarse classes to fine classes. Adaptive Loss Weighting is derived from a combination of Adaptive Loss Balancing and Branch Training strategy (BT-strategy) in [106]. After balancing all the losses, pre-defined weights are multiplied by losses at every iteration. Weights are defined with the condition that their sum is 1. At the beginning of learning, weights of multiple levels of coarse classes are higher than the weight of the fine classes. A portion of the weight of the fine classes increases as learning progresses. For example, a set of three weights $\{w_{c_1}, w_{c_2}, w_f\}$ is given when two coarse classes are present, where c_1, c_2 and f denote the first level of coarse classes, the second level of coarse classes and the fine classes, respectively. The learning begins by assigning w_{c_1} the highest portion of the set of weights. Consequently, w_{c_1} is reduced to the lowest portion of the set of weights while the portion of w_f increases to the highest.

6.2 Experiments

In order to verify the proposed process, InhaMotion dataset and nine open source datasets[1] are used. InhaMotion dataset is transformed to use them with machine learning algorithms and nine open source datasets, which are originally made for machine learning, are studied.

6.2.1 InhaMotion dataset description

In order to apply InhaMotion dataset to machine learning algorithms, two kinds of preprocessing with respect to the amplitude and the length of a measurement are performed. First, preprocessing with respect to amplitude is addressed by standard normalization. Second, preprocessing with respect to the length of a measurement is addressed by using fast Fourier transform (FFT) and inverse Fourier transform. FFT is used to resample a measurement with a fixed length [11]. Let us denote the fixed length, N, as the length of the resampled measurement and L as the length of the measurement. FFT is applied to the measurement and we compute the minimum number, m, between N and L. The first

[1] All datasets are available at `http://www.timeseriesclassification.com/dataset.php`.

$\frac{m}{2}+1$ FFT coefficients are selected by sampling theorem [75]. Subsequently, the inverse Fourier transform is applied to the selected FFT coefficients and then $\frac{N}{L}$ is multiplied by the result of the inverse Fourier transform to scale the amplitude of the measurement.

The division of a training set and a test set follows the preprocessing step. Since 12 subjects in the first group performed seven motions, and both seven subjects in the second group and four subjects in the third group performed nine motions, $12\times7+(7+4)\times9=183$ samples were collected. From Chapter 5, we chose a set of 18 measurements which are $\boldsymbol{r^2}, \boldsymbol{r^3}, \boldsymbol{r^4}, \boldsymbol{r^7}, \boldsymbol{r^8}$, and $\boldsymbol{a^2}$. In other words, we have 183 samples and each sample has 18 measurements. Now, we make a training set by randomly selecting nine samples for each motion type from InhaMotoin dataset and make a test set with the remaining samples, *i.e.*, the number of samples in the training set and the test set, respectively, is 81 and 102.

Two possible cases for the coarse classes are considered in InhaMotion dataset based on the decision tree in Figure 5.3. First, coarse classes with one level is defined based on G_1, G_5, G_6, G_7 and G_8 and denoted by Case1. Accordingly, C-FNNs and HADNNs for coarse classes with one level are used. Second, coarse classes with two levels are defined and denoted by Case2. The first level consists of G_1 and G_2. The second level consist of G_1, G_3, and G_4.Accordingly, C-FNNs and HADNNs for coarse classes with two levels are used. These two cases are summarized in Table 6.1.

Table 6.1: Coarse class information of InhaMotion dataset

Case	group	the first coarse class	group	the second coarse class
One level coarse class	G_1	0:{0, 1}	N/A	N/A
	G_5	1:{2, 3}	N/A	N/A
	G_6	2:{5}	N/A	N/A
	G_7	3:{4, 6}	N/A	N/A
	G_8	4:{7, 8}	N/A	N/A
Two levels of coarse class	G_1	0:{0, 1}	G_1	0:{0, 1}
	G_2	1:{2, 3, 4, 5, 6, 7, 8}	G_3	1:{2, 3, 5}
			G_4	2:{4, 6, 7, 8}

6.2.2 InhaMotion dataset result

The generative model for InhaMotion dataset comprises a symmetrical architecture of an encoder network and a decoder network introduced in Chapter 3. The encoder and the decoder networks are built upon two fully-connected hidden layers of 500 nodes with the

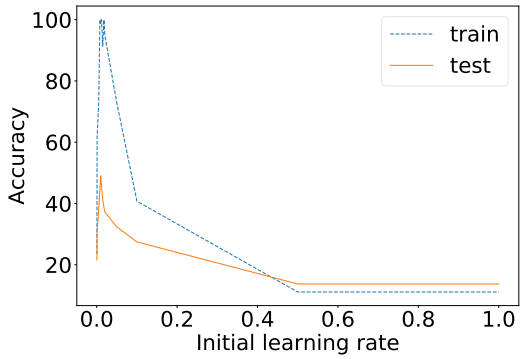
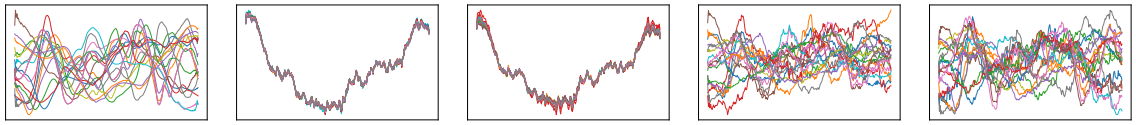


Figure 6.3: Accuracy of a training set and test set of Baseline on InhaMotion dataset with respect to initial learning rate.

dimension of the latent space set to 10. The same configuration of epochs, batch size, activation functions and loss functions with setup in Section 3.3.2 are used to train the generative model. In order to deal with the characteristics of *multi-channel data*, that they consist of multiple channels, the generative model learns one channel at a time instead of learning them altogether, and we assemble them to make *multi-channel data*. Since the proposed generative model requires the standard deviation as hyper-parameter, the model is trained with various standard deviations, which were used in Section 3.3.5.

Next, a classification model without coarse classes is trained following subsections 6.1.2, 6.1.3 and 6.1.4. The baseline is built upon four residual blocks with filters of 32, 64, 128 and 256, and two fully-connected layers with 1024 and 512 nodes. The best learning rate for the baseline is empirically searched as shown in Figure 6.3 and set as 0.01. This learning rate is used when carrying out three classification tasks with the following: when only synthetic data are used, when only coarse classes are used and when both synthetic data and coarse classes are used.

In the first task, the proposed generative model is trained with various standard deviations as shown in Figure 6.5. The highest accuracy is obtained when the standard deviation is 0.6. The results in Table 6.2 show that the accuracy of the training set decreases while the accuracy of the test set increases. It shows that overfitting can be addressed by the use of synthetic data generated by the proposed model. In contrast, the accuracy of the test set decreases when using synthetic data generated by available generative models as



(a) (b) (c) (d) (e)

Figure 6.4: (a) is a sample in the training set of InhaMotion dataset. Visualization of synthetic data generated by VAE is in (b), by β-VAE is in (c), by MMD is in (d) and by the proposed generative model is in (e).

shown in Table 6.3. This is because these models fail to generate synthetic data similar to samples in the training set as shown in Figure 6.4.

In the second task, the test accuracies of both cases decrease except for Case1 with C-FNNs as shown in Table 6.4. These results are compared to the results of the third task in Table 6.5. In the third task, Case1 and Case2 are also used, however their accuracies dramatically increase. The biggest increase happens when using C-FNNs with Case2. This big improvement can be explained by the accuracies of the coarse classes as shown in Table 6.6. Accuracies of the first level of coarse classes and the second level of coarse classes in the third task is much higher than those in the second task. In other words, capturing key features of coarse classes improves the accuracy of the fine classes.

Table 6.2: Accuracy with or without synthetic data of InhaMotion dataset

without synthetic data		with synthetic data		
train acc	test acc	train acc	test acc	difference
98.76	49.01	81.69	51.85	+2.84

Table 6.3: Accuracy of the test set when using synthetic data by various generative models of InhaMotion dataset

Generative models	test acc
VAE	44.44
β-VAE	41.97
MMD	34.56
The proposed model	51.85

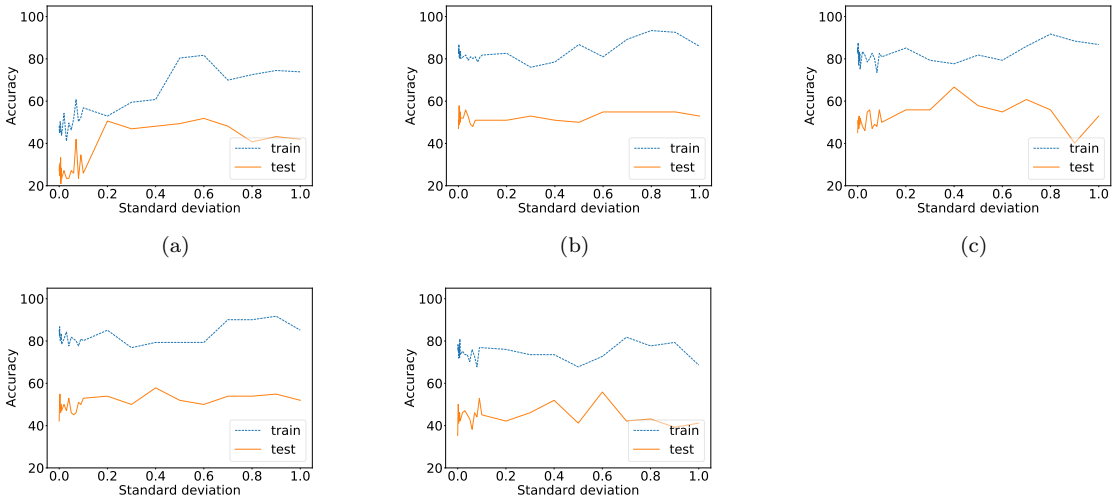


Figure 6.5: Accuracy of a training set and test set of (a) Baseline, (b) Baseline with C-FNN on Case1, (c) Baseline with C-FNN on Case2, (d) Baseline with HADNN on Case1 and (e) Baseline with HADNN on Case2 with respect to standard deviation applied to the generative model proposed in Chatper 3.

Table 6.4: Hierarchical Result without synthetic data of InhaMotion dataset

Case	Baseline		with C-FNN		with HADNN	
	train acc	test acc	train acc	test acc	train acc	test acc
Case 1	98.76	49.01	100.	54.90	100.	46.07
Case 2	98.76	49.01	91.35	38.23	92.59	46.07

Table 6.5: Hierarchical with synthetic data of InhaMotion dataset

Case	Baseline		with C-FNN		with HADNN	
	train acc	test acc	train acc	test acc	train acc	test acc
Case 1	81.69	51.85	80.99	57.84	79.33	57.84
Case 2	81.69	51.85	77.68	66.66	76.85	52.94

Table 6.6: Hierarchical Result without synthetic data of InhaMotion dataset

Synthetic data	Without			With		
Case	1st level	2nd level	test acc	1st level	2nd level	test acc
Case 2	73.52	55.88	38.23	86.27	73.52	66.66

6.2.3 Open source human motion data description

We further test the proposed process with nine open source human motion datasets:

- Data1 measured movements of the tongue and lips during speech. The sensors on the lower lip (LL), tip of tongue (T1) and upper lip produced X, Y and Z positions with a sampling rate of 200 Hz. Targets were 25 words in phonetically balanced (PB) lists from native English speakers [76].

- Data2 measured four basic motions of standing, running, walking and badminton. A smartwatch produced three accelerometer measurements and three gyroscope measurements during each movement.

- Data3 measured the gestures of a Cricket umpire in a game. Three accelerometer measurements were captured from sensors on the wrists of subjects [76].

- Data4 measured four types of movements: seizure mimicking, walking, running and sawing. Each subject wore a sensor on their dominant wrist and the sensor produced three accelerometer measurements [86].

- Data5 measured six types of hand movements, which were gestures to express open fist, two, pointing, ring and grasp. Subjects wore an eRing, which was a sensor for the detection of finger movements, on their thumb, index finger and middle finger [94].

- Data6 measured 15 Brazilian Sign Language (BSL) movements. This consists of swing (curved, horizontal and vertical), arc (anti-clockwise and clockwise), circle, straight-line (horizontal and vertical), zigzag (horizontal and vertical), wavy (horizontal and vertical), curve (down and up) and irregular movement. The movements are captured from a video [13].

- Data7 measured six Naval Air Training and Operating Procedures Standardization (NATOPS) aircraft handling signals. Sensors were attached on the hands, elbows, wrists and thumbs, producing X, Y and Z coordinates [22].

- Data8 measured two motions from badminton and another two motions from squash. Subjects wore a smartwatch, which generated three accelerometer measurements and three gyroscopes measurements.

- Data9 measured eight simple gestures captured by wii, which generated three accelerometer measurements. Gestures included line, circle, rectangular and so on [44].

For the experiments using coarse classes, five datasets are considered. First, coarse classes with two levels are defined in Data3. The first level of coarse classes is divided based on the dynamics of the movements. Fine classes 1, 2, 3, 4, 6, 7 and 11 are defined as dynamic classes, and the other fine classes as static classes. Subsequently, the second level of coarse classes is defined based on the movements of both arms. In the dynamic class, fine classes 2, 6 and 7 require the movement of one arm and 1, 3, 4 and 11 require both arms. In the static class, one arm movement is required for fine classes 0 and 10, and both arms should be moved in fine classes 5, 8 and 9. Second, coarse classes with one level are defined in Data5. The fine classes are divided based on whether the index finger should be bent or not to complete the gestures required. Third, coarse classes with one level are also defined in Data6, but the number of coarse classes are three unlike the case of Data5. The first coarse class consists of vertical movements and the second coarse class consists of horizontal movements. The last coarse class consists of arc movements, which are more dynamic than the others. Fourth, coarse classes with one level are also defined in Data7. The coarse classes are simply defined based on the need for the movements of both arms. Fine classes 0, 1 and 2 require the movement of one arm and the other fine classes require both sides. Lastly, coarse classes with two levels are defined in Data9. The first level of coarse classes is defined based on whether the shape of the movement is a line or not. Subsequently, the second level of the coarse classes is defined as follows: the fine classes related to the line movements are divided into whether the shape of the movement is vertical and horizontal, and the fine classes related to non-line movements are divided into whether the shape of the movement is an arc or not.

6.2.4 Open source human motion data results

These open source human motion datasets are collected for machine learning. Therefore, data preprocessing is not required for all datasets. Generative models are trained with the same configuration as the setup in Section 6.2.2. For the nine open source human motion datasets, the baseline architecture is built upon four residual blocks with filers of 16, 32, 64 and 128, and two fully-connected layers with 512 and 256 nodes. This baseline architecture is found by iterating training with different configurations.

To find the best learning rate corresponding to each dataset, we iterate experiments

with varied learning rates. The best learning rate are taken following Figure 6.7 and illustrated in Table 6.7. The best learning rates are used when carrying out the three tasks mentioned in subsection 6.2.2.

For the first task, the proposed generative model is trained with various standard deviations. The standard deviation with the highest accuracy with respect to each dataset is shown in Table 6.8. As the standard deviations with the highest accuracies are different with respect to the datasets, designing an algorithm to find the best standard deviation is left for future study. Table 6.9 shows all results for the first task. The accuracies of the training sets without synthetic data are 100% in all datasets except for Data7. Comparing the results when using synthetic data generated by the proposed generative model to the results without synthetic data, the accuracies of the training sets decrease, and the accuracies of the test sets increase or stay the same in seven datasets. On the other hand, accuracies of the training sets and test sets decrease in Data6 and Data8. This decrease of the accuracies does not only happen with the proposed generative model but also with other available generative models: VAE, β-VAE and MMD as shown in Table 6.10. As illustrated in Figure 6.6, which shows examples of synthetic data compared to data in the original training set, the accuracies decrease when synthetic data are not similar to the samples in the training set.

In the second task, five datasets are considered by defining their coarse classes following Table 6.14. The accuracies of the test sets all improve when using C-FNNs as shown in Table 6.11, while the accuracies of the test sets decrease when using HADNNs. The decrease of the accuracy in Data6 happens when using both synthetic data and hierarchical learning models in the third task as shown in Table 6.12. From these results, we can deduce that using synthetic data, which are not plausibly drawn from the distribution of the training set, deteriorates the ability of neural networks.

Table 6.7: The best learning rate with respect to each dataset

Data number	1	2	3	4	5	6	7	8	9
Learning rate	0.01	0.015	0.015	0.01	0.01	0.02	0.01	0.05	0.015

Table 6.8: The best standard deviation with respect to each dataset

Data number	1	2	3	4	5	6	7	8	9
Standard deviation	0.8	0.7	0.08	0.08	0.8	0.9	0.06	0.5	0.06

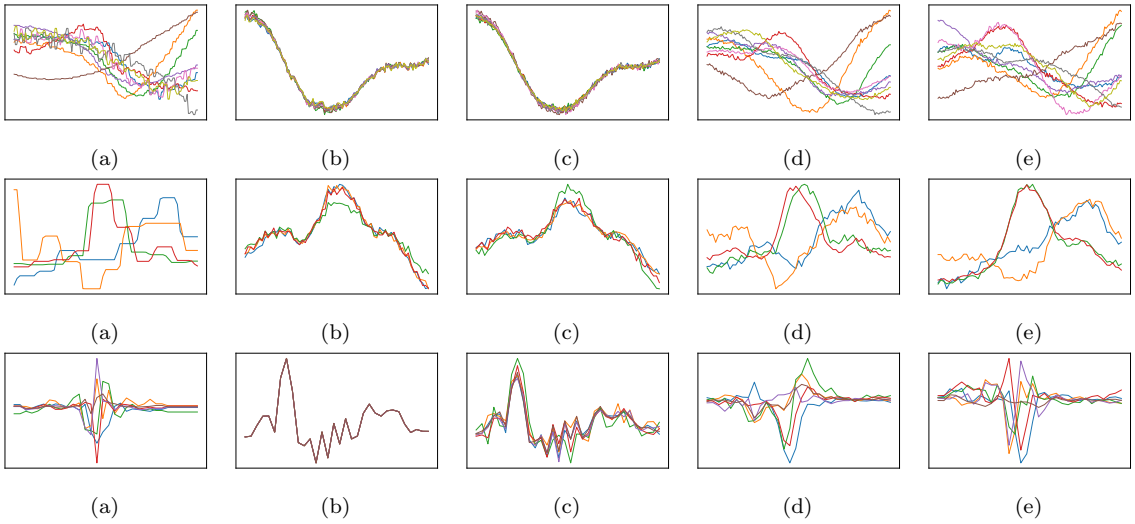


Figure 6.6: Visualization of a sample and synthetic data. The first, second and third rows show data from Data1, Data5 and Data8, respectively. (a) is a sample from each training set. Visualization of the synthetic data corresponding to the dataset generated by VAE is in (b), by β-VAE is in (c), by MMD is in (d) and by the proposed generative model is in (e).

Table 6.9: Accuracy with or without synthetic data of nine open source datasets

Data number	without synthetic data		with synthetic data		difference
	train acc	test acc	train acc	test acc	
1	100.	84.	79.85	89.66	+5.66
2	100.	100.	93.33	100.	+0
3	100.	98.61	71.16	100.	+1.39
4	100.	94.92	96.09	95.65	+0.73
5	100.	78.14	93.33	83.33	+5.19
6	100.	55.	69.62	53.88	-1.12
7	98.88	90.55	80.74	90.55	+0
8	100.	69.07	89.82	66.44	-2.63
9	100.	83.12	78.33	85.	+1.88

6.3 Summary

In this chapter, we apply models proposed in Chapter 3 and Chapter 4 to *multi-channel data* with examples of human motion data. Two or four experiments are carried out per dataset. ResNet is used as a baseline since it performs well with multi-channel time series

Table 6.10: Accuracy of the test set when using synthetic data by various generative models of nine open source datasets

Data number	VAE	β-VAE	MMD	Proposed model
1	86.33	83.66	87.	89.66
2	100.	100.	100.	100.
3	95.83	93.05	94.44	100.
4	92.75	92.08	95.65	95.65
5	81.85	79.25	63.33	83.33
6	52.22	48.88	40.55	53.88
7	88.33	88.88	90.55	90.55
8	63.15	63.81	64.47	66.44
9	80.31	78.13	84.38	85.

Table 6.11: Hierarchical Result without synthetic data of nine open source datasets

	Baseline		with C-FNN		with HADNN	
Data number	train acc	test acc	train acc	test acc	train acc	test acc
3	100.	98.61	100.	100.	100.	95.83
5	100.	78.14	100.	83.33	100.	78.51
6	100.	55.	100.	57.77	99.44	52.22
7	98.88	90.55	100.	92.77	98.88	89.99
9	100.	83.12	100.	83.12	100.	81.18

Table 6.12: Hierarchical with synthetic data of nine open source datasets

	Baseline		with C-FNN		with HADNN	
Data number	train acc	test acc	train acc	test acc	train acc	test acc
3	83.33	100.	85.18	100.	82.71	97.22
5	93.33	83.33	95.55	84.44	93.33	77.77
6	72.22	53.88	67.03	50.55	73.33	50.
7	96.66	90.55	94.81	93.88	94.81	92.22
9	81.11	85.	80.55	85.93	80.	84.06

data.The first experiment is to train the baseline without synthetic data. The second experiment is to train the baseline with synthetic data. If coarse classes are available, two more experiments with hierarchical learning models are carried out.

Because obtaining a large amount of human motion data is time-consuming, we first apply the proposed generative model in Chapter 3 to augment the amount of data. A set of labelled synthetic data is created at every iteration and is used to prevent overfitting

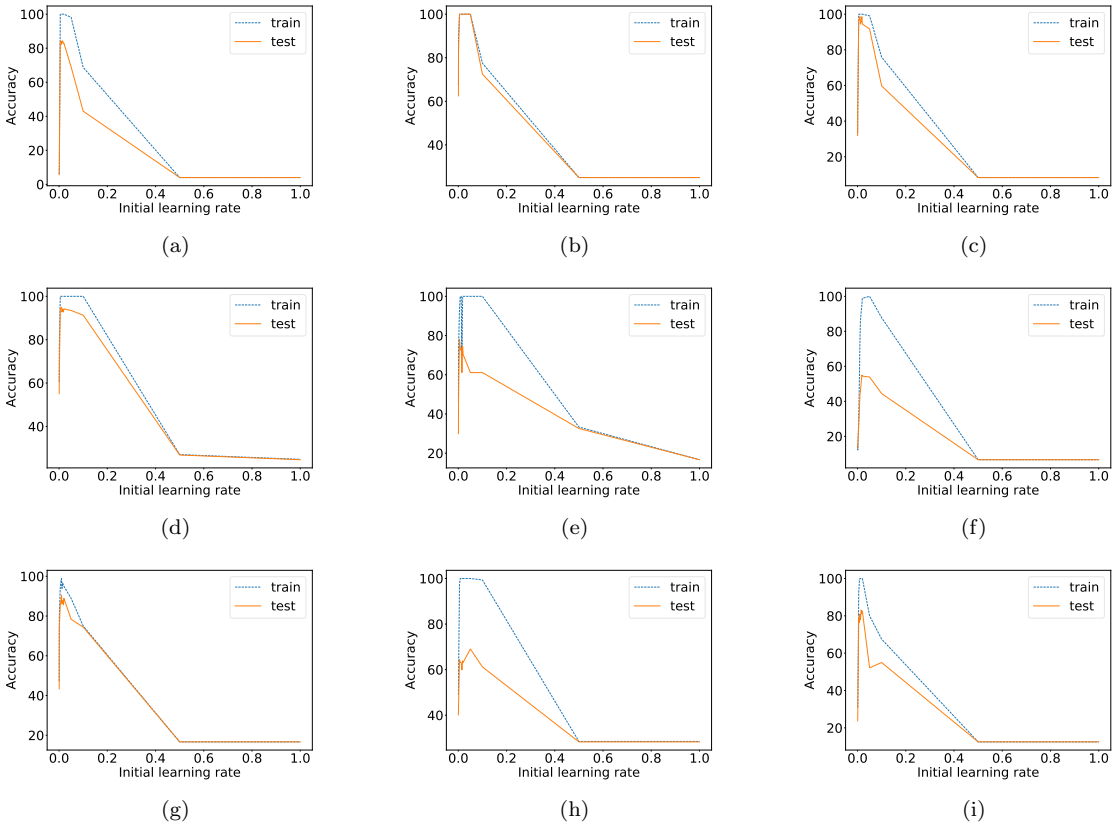


Figure 6.7: Accuracy of a training set and test set of Baseline on (a) Data1, (b) Data2, (c) Data3, (d) Data4, (e) Data5, (f) Data6, (g) Data7, (f) Data8, and (i) Data9 with respect to initial learning rate.

in neural networks. The proposed motion type detection process using synthetic data is evaluated using InhaMotion dataset and nine open source human motion datasets. Training with synthetic data from the proposed generative model results in improvement of the accuracy of the test set in InhaMotion dataset by 2.84%. In the nine open source human motion datasets, training neural networks with synthetic data from the proposed generative model improves the accuracy of the test set in seven datasets and outperforms the other available generative models. This result demonstrates that the proposed model can generate multi-channel time series data which are convincingly drawn from the original training set.

InhaMotion dataset and five open source datasets are used with coarse classes. In order

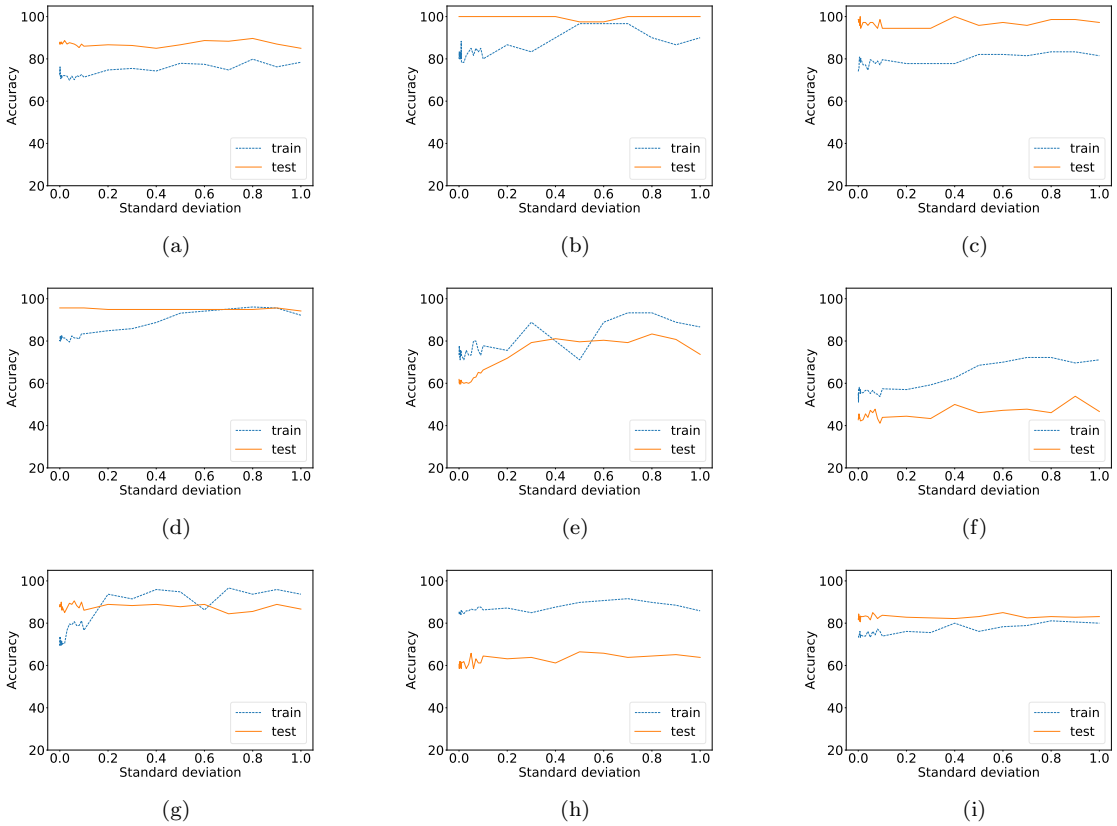


Figure 6.8: Accuracy of a training set and test set of Baseline on (a) Data1, (b) Data2, (c) Data3, (d) Data4, (e) Data5, (f) Data6, (g) Data7, (f) Data8, and (i) Data9 with respect to standard deviation applied to the generative model proposed in Chatper 3.

to take advantage of utilizing coarse classes without providing them at the test phase, C-FNNs and HADNNs are attached to ResNet after removing its fully-connected layers. The accuracies of the test sets of the five open source datasets all improve with C-FNNs. It is worth noting that the accuracy dramatically increases by 17.65% when both synthetic data and coarse classes are used in InhaMotion dataset.

All results show that the proposed generative model and hierarchical learning models can be successfully used in fields where the amount of data is insufficient and where data can be hierarchically structured based on their channels.

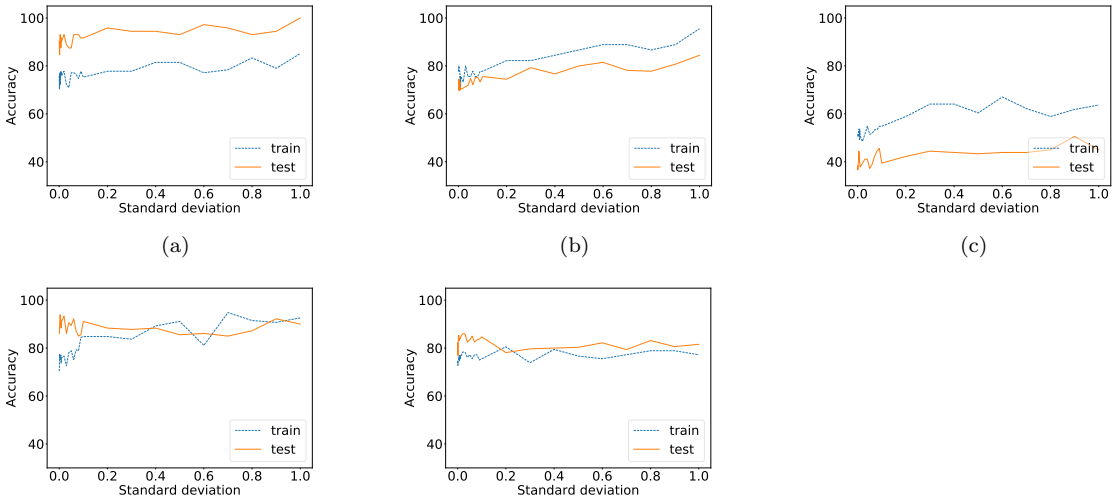


Figure 6.9: Accuracy of a training set and test set of Baseline with C-FNN on (a) Data3, (b) Data5, (c) Data6, (d) Data7, and (e) Data9 with respect to standard deviation applied to the generative model proposed in Chatper 3.

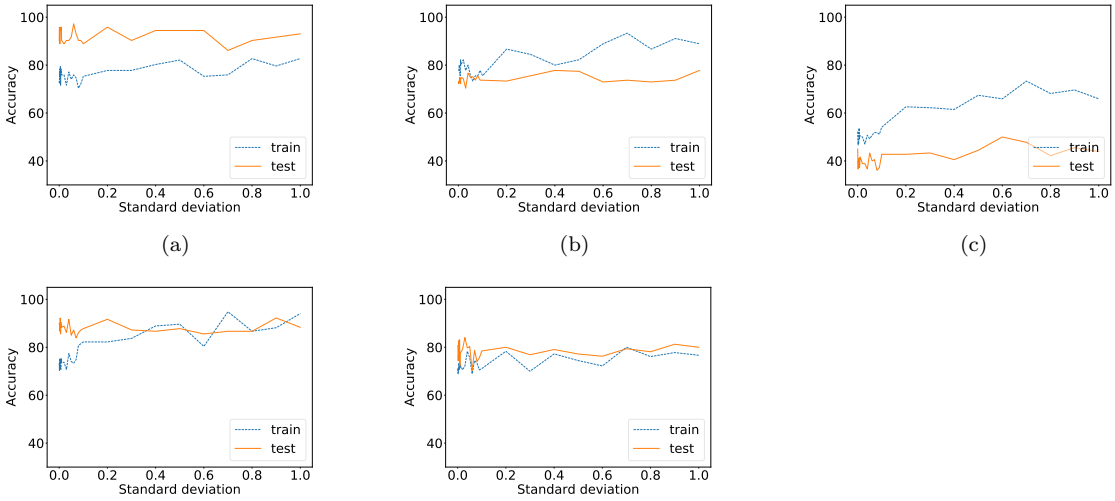


Figure 6.10: Accuracy of a training set and test set of Baseline with HADNN on (a) Data3, (b) Data5, (c) Data6, (d) Data7, and (e) Data9 with respect to standard deviation applied to the generative model proposed in Chatper 3.

Table 6.13: Summary of the nine open source human motion datasets

Summary					
	N(train)	N(test)	N(classes)	N(channels)	length
1	275	300	25	9	144
2	40	40	4	6	100
3	108	72	12	6	1197
4	137	138	4	3	206
5	30	270	6	4	65
6	180	180	15	2	45
7	180	180	6	24	51
8	151	152	4	6	30
9	120	320	8	3	315

Table 6.14: Coarse class information for five open source datasets

Dataset	type	the first coarse class	type	the second coarse class
Data3	dynamic	0:{1, 2, 3, 4, 6, 7, 11}	one arm	0:{2, 6, 7}
			both arms	1:{1, 3, 4, 11}
	static	1:{0, 5, 8, 9, 10}	one arm	2:{0,10}
			both arms	3:{5,8,9}
Data5	not bent	0:{0, 2, 3}	N/A	N/A
	bent	1:{1, 4, 5}	N/A	N/A
Data6	arc	0:{0, 3, 4, 5, 12, 13, 14}	N/A	N/A
	horizontal	1:{1, 6, 8, 10}	N/A	N/A
	vertical	2:{2, 7, 9, 11}	N/A	N/A
Data7	one arm	0:{0, 1, 2}	N/A	N/A
	both arms	1:{3, 4, 5}	N/A	N/A
Data9	line	0:{2, 3, 4, 5}	horizontal	0:{2, 3}
			vertical	1:{4, 5}
	non-line	1:{0, 1, 6, 7}	arc	2:{6, 7}
			non-arc	3:{0, 1}

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we have studied machine learning algorithms to apply them to *multi-channel data*. In order to use machine learning algorithms with data from multiple channels, we propose new models for neural networks in Chapter 3 and Chapter 4. The proposed process in Chapter 6 using machine learning algorithms is mainly divided into two steps: generating a set of labelled synthetic data and classifying motion types with their hierarchy of target classes.

Because *multi-channel data* consist of multiple channels, more various samples can emerge in *multi-channel data* from the combination of multiple channels than in single channel data. Thus, in order to analyse *multi-channel data*, the more the data required, the more channels the data have. Accordingly, we propose a new generative model to augment the amount of data in Chapter 3. Taking into account the role of the mean and the standard deviation in the reparameterization trick of VAEs, we design a new generative model which is built upon the architecture of AEs. Since AEs achieve great success in dimensionality reduction, our goal is to preserve their architecture and to make it become a generative model. This is done by the extension of the mapping of decoder from the discrete latent space to the continuous latent space.

In addition, we propose a validation process to check the performance of generative models with a given dataset, which has a training set and a test set. The process is divided into three: generation, labelling and classification. In generation, generative models are trained on the training set. In labelling, synthetic data generated by the now trained

generative models are labelled using SVM. In classification, a neural network for a classification task is trained using the labelled synthetic data. Subsequently, the generative models are evaluated based on the accuracy of the test set. Because standard deviation is a hyper-parameter of the proposed generative model, we first check the impact of the standard deviation on the accuracy of the test set. The result shows that a certain range of the standard deviations guarantees a good performance on the test set. In other words, the neural networks trained using synthetic data can capture the representations of the training set. The proposed generative model is compared with VAE and MMD. With the comparison, we conclude that the proposed generative model can be used to generate synthetic data.

Multi-channel data may have their own innate hierarchical structure based on channel information. In Chapter 4, two types of hierarchical learning models are proposed. First, Hierarchical Auxiliary Learning (HAL) is proposed to utilize hierarchical structure of target classes in the field of Computer Vision, which is one of the most popular fields in machine learning. HAL takes auxiliary information and transforms them into auxiliary score through the auxiliary block. With the experimental results, we observe four advantages of using the auxiliary block. First, the block can be easily attached to any baselines. Second, the numbers of parameters with or without the auxiliary block are almost the same, which implies that it does not affect computational complexity. Third, any auxiliary information improves the performance in terms of the accuracy of the test set. Last, the auxiliary block transforms discrete coarse class information to continuous auxiliary scores through back propagation. As auxiliary scores still are separable with respect to coarse classes but are different with respect to inputs, the auxiliary scores can contain both coarse class and fine class information. Nonetheless, the disadvantage of the proposed model is that it requires the coarse class information even at the test phase.

By modifying the targets of hierarchical learning models, we propose Consecutive Feedforward Neural Networks (C-FNNs) and Hierarchical Auxiliary Deep Neural Networks (HADNNs) whose targets are continuous and which do not require coarse classes at the test phase. C-FNNs are a series of pairs of one feedforward neural network and decoder network with the skip connection. On the other hand, HADNNs are the modifications of C-FNNs to reduce the number of parameters. The series of pairs of one feedforward neural network and decoder network are reduced and inserted in the middle of an architecture of HADNNs as a branch. C-FNNs and HADNNs are evaluated on three benchmark indoor localization datasets – TUT2017, TUT2018 and UJIIndoorLoc –, examples of *multi-channel*

data as samples in the datasets are obtained from independent Wi-Fi routers. Results show that C-FNNs increase the floor accuracy by 55%, 30% and 60% in TUT2017, TUT2018 and UJIIndoorLoc, respectively and require more than three times and five times the parameters than baselines. On the other hand, HADNNs achieve as high an accuracy as C-FNNs with much less parameters than C-FNNs. As HADNNs require less parameters and do not require the coarse classes at the test phase, it is a practical solution to any problem where coarse classes should be considered.

As another example of *multi-channel data*, human motion data is analysed to study the significance of the relationship between channels and targets of the task. In human motion data, channels are sensor locations and targets are motion types. The goal is to detect motion types based on the relationship between sensor locations and motion types. In Chapter 5, we divide the detection algorithm into five steps. In the first step, WT is applied to measurements to remove noise, and angular displacement is calculated as it provides meaningful information about motion movements. In the second step, weights are obtained based on the relationship between motion types and measurements from sensor locations. In the third step, we build a hierarchical decision tree and design a set of characteristic functions based on the weights. In the fourth step, the ROC curve is used to find the optimal thresholds for the characteristic functions. Finally, characteristic vectors are found using the characteristic functions and thresholds. Evaluations are done with subjects, who were patients and students of Inha University Hospital, Korea. All subjects were required to perform nine motions as far as their abilities could permit. We use four metrics including recall, precision, accuracy and F-measure to validate the proposed detection algorithm and the results were compared against the conventional method, namely the PCA. From the results, analysing the relationship between sensor locations and motion types can help in designing functions that can extract important features and hierarchically differentiate motion types.

In Chapter 6, we integrate Chapters 3 to 5. We use human motion data as an example of *multi-channel data* with machine learning. Due to difficulties in gathering motion data, we first apply the proposed generative model in Chapter 3. At every iteration, a set of labelled synthetic data is fed to a neural network together with the original training set. The number of the labelled synthetic data is half of the number of the original training set. This allows the neural networks to focus on the training set more than synthetic data and to prevent overfitting by synthetic data, which results in generalization of neural networks. In addition, we apply the proposed hierarchical learning models in Chapter 4 to datasets,

the targets of which have hierarchy. ResNet is selected as a baseline since it performs well with *multi-channel data*. Then, we replace fully-connected layers with C-FNNs and HADNNs to utilize coarse classes. The proposed detection process with machine learning is evaluated by the accuracy of the test set using InhaMotion dataset and nine open source human motion datasets. In InhaMotion dataset, the accuracy of the test set increases up to 17.65% when using both synthetic data and coarse classes with C-FNNs. In seven of the nine open source human motion datasets, the accuracies of the test sets improve when training neural networks for a classification task using a combination of synthetic data and the original training set. Two of the nine datasets are used as examples of coarse classes with two levels and three of the remaining seven datasets are used as examples of coarse classes with one level. In all five datasets with coarse classes, the accuracies of the test sets increase when utilizing coarse classes. Note that coarse classes are not required at the test phase since C-FNNs and HADNNs are attached to ResNet. Therefore, we conclude that the proposed detection process is applicable to *multi-channel data*, which have difficulty in gathering abundant amounts of data and demand the utilization of hierarchy of targets. Future study is discussed in the next section.

7.2 Future Work

Although the proposed models solve several problems mentioned in this thesis, there are still a lot of things to be complemented. In this section, we list future research derived from this thesis.

First, we observed the impact of the standard deviation on the proposed generative model. The results showed that there is a certain range of the standard deviations which guarantees the creation of a set of good synthetic data. Therefore, one can design an algorithm to automatically find the range.

Second, HAL showed great performance on a coarse class and fine class classification tasks. Since it is proposed to utilize the hierarchy among target classes to address scalability issues, the experiments on larger datasets should be carried out in the future. In addition, the accuracy of the test set is affected by the coarse class information. Therefore, we will design an algorithm to find the optimal coarse classes which can improve the performance the most.

Third, the combination of ResNet and C-FNNs improves the accuracies of the test sets while the combination of ResNet and HADNNs decreases the accuracies of the test sets

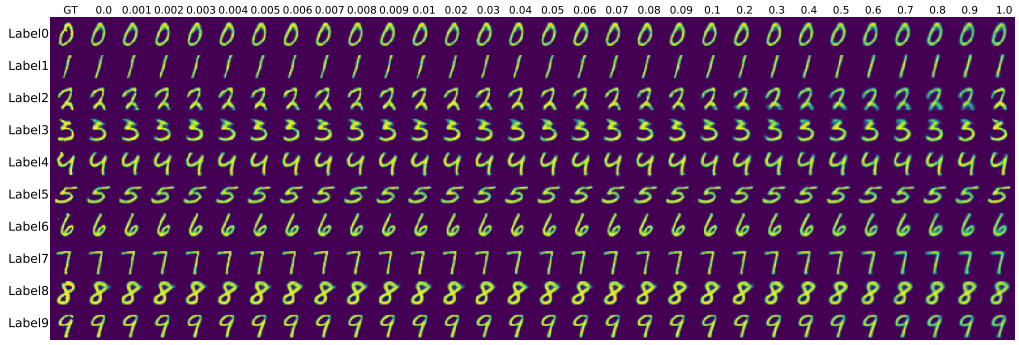
when coarse classes are used. Accordingly, we will focus on figuring out the reason for the dropping performance of the combination of ResNet and HADNNs. This can lead us to design more effective neural networks which can utilize coarse classes and fine classes and do not require coarse classes at the test phase.

Fourth, the results of the generative models show that generating synthetic data, which are convincingly drawn from the training set, can improve the performance of a classification task. In the future study, the model can be used to solve imbalanced class problems, caused when the numbers of samples with respect to each class are different.

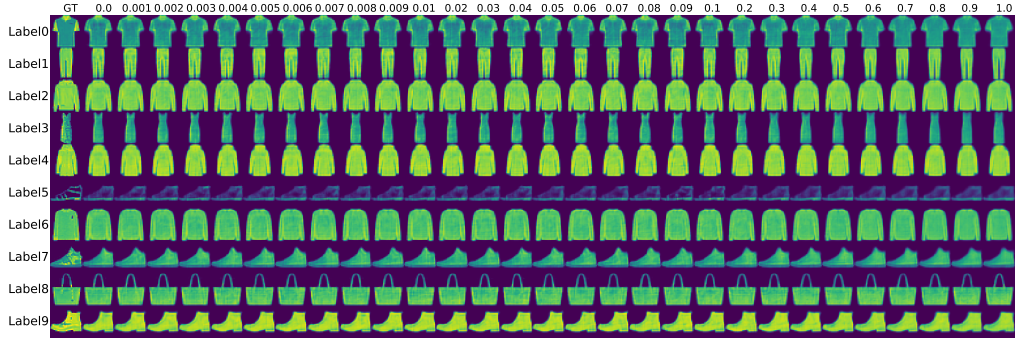
Finally, the results of the proposed hierarchical learning models show that they can be successfully used in fields where the coarse classes are available. Consequently, these hierarchical learning models can be applied to fields where coarse classes based on multiple channels are available. Examples of such fields are text categorization, drug discovery and a recommendation system.

Appendix A

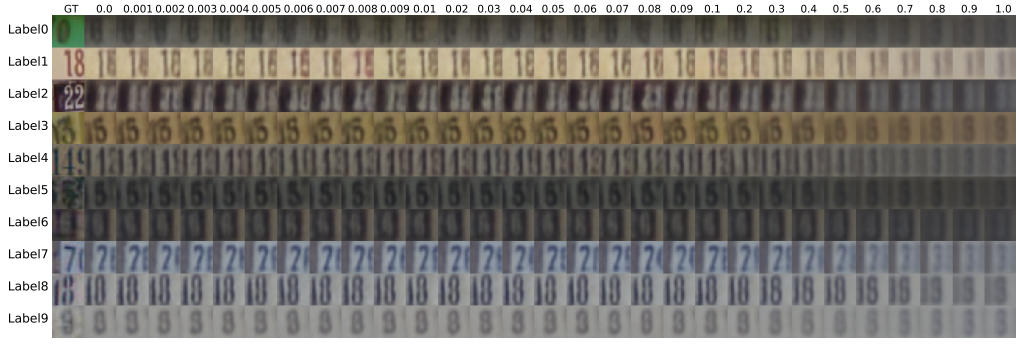
Reconstructed images from the proposed generative models and available methods



(a)

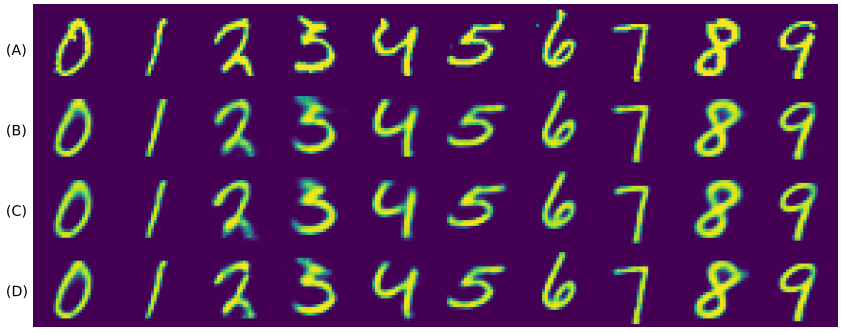


(b)

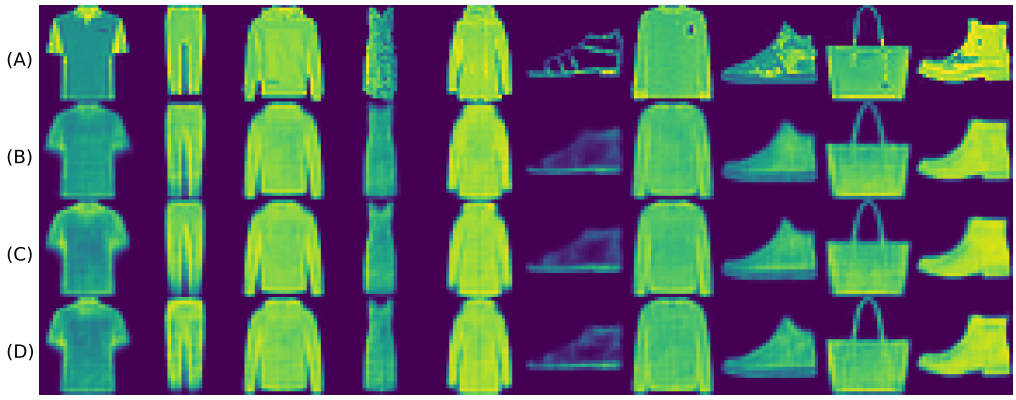


(c)

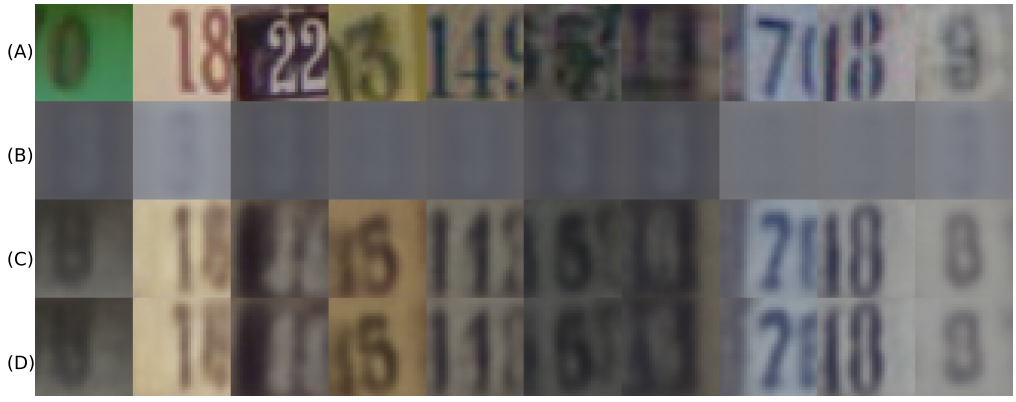
Figure A.1: Visualization of the 10 reconstructed images with respect to the standard deviation. The first column contains the ground truth images. The images in each column are reconstructed from the ground truth based on the proposed model with the standard deviation, which is written at the top of each column. The generative models are trained by all images in the training set.



(a)

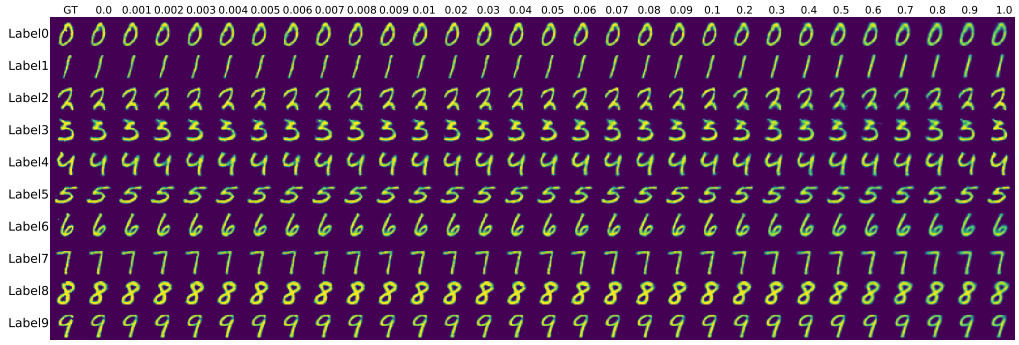


(b)

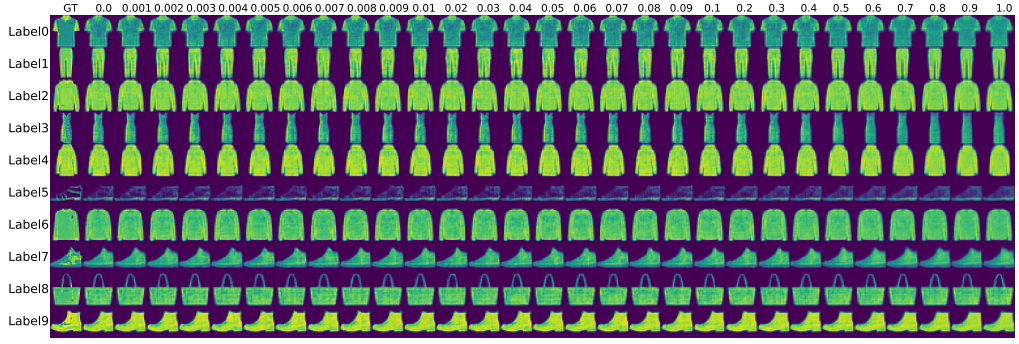


(c)

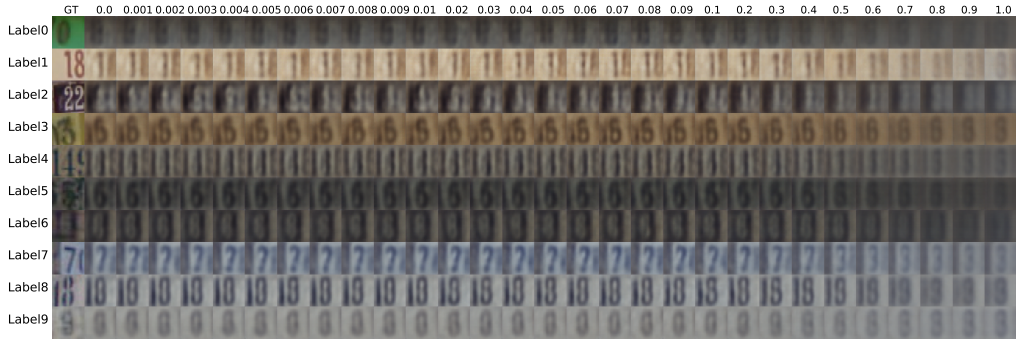
Figure A.2: Visualization of a comparison of 10 reconstructed images. Images in (A) are the ground truth. Images of (B) are reconstructed images by VAE, (C) are by MMD, and (D) are by the proposed model with std of 0.001. The generative models are trained by all images in the training set.



(a)

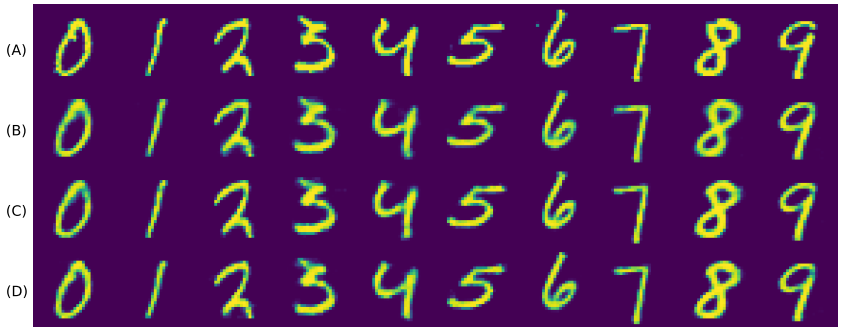


(b)

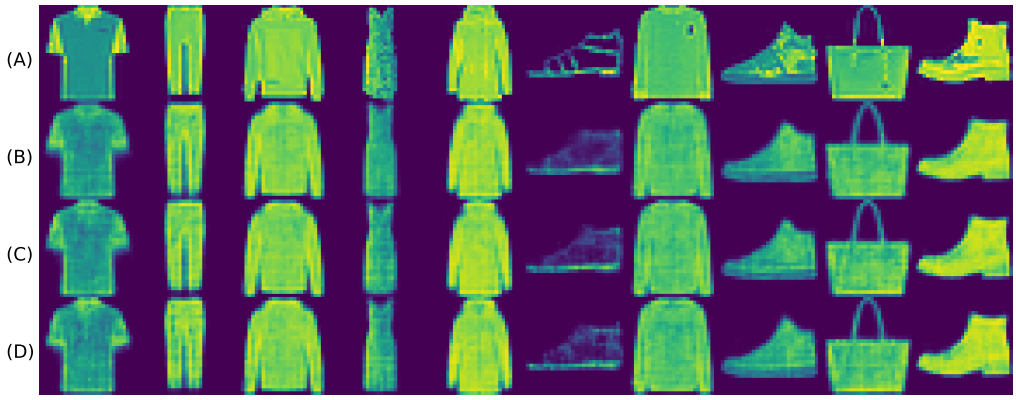


(c)

Figure A.3: Visualization of the 10 reconstructed images with respect to the standard deviation. The first column contains the ground truth images. The images in each column are reconstructed from the ground truth based on the proposed model with the standard deviation, which is written at the top of each column. The generative models are trained by a portion of the training set.



(a)



(b)

(c)

Figure A.4: Visualization of a comparison of 10 reconstructed images. Images in (A) are the ground truth. Images of (B) are reconstructed images by VAE, (C) are by MMD, and (D) are by the proposed model with std of 0.001. The generative models are trained by a portion of the training set.

Appendix B

Losses and auxiliary scores with different coarse class cases on three datasets

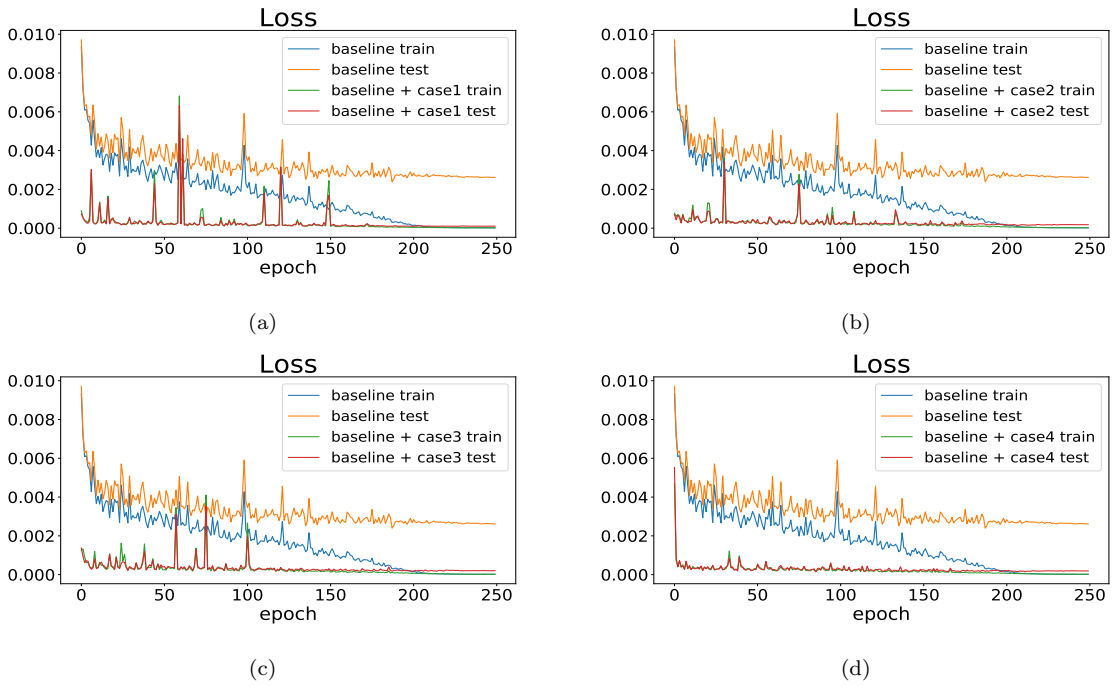


Figure B.1: Loss comparison of the traiingn and test datasets at each epoch during the training phase between the baseline and (a) Case1 (b) Case2 (c) Case3 and (d) Case4 with MNIST dataset.

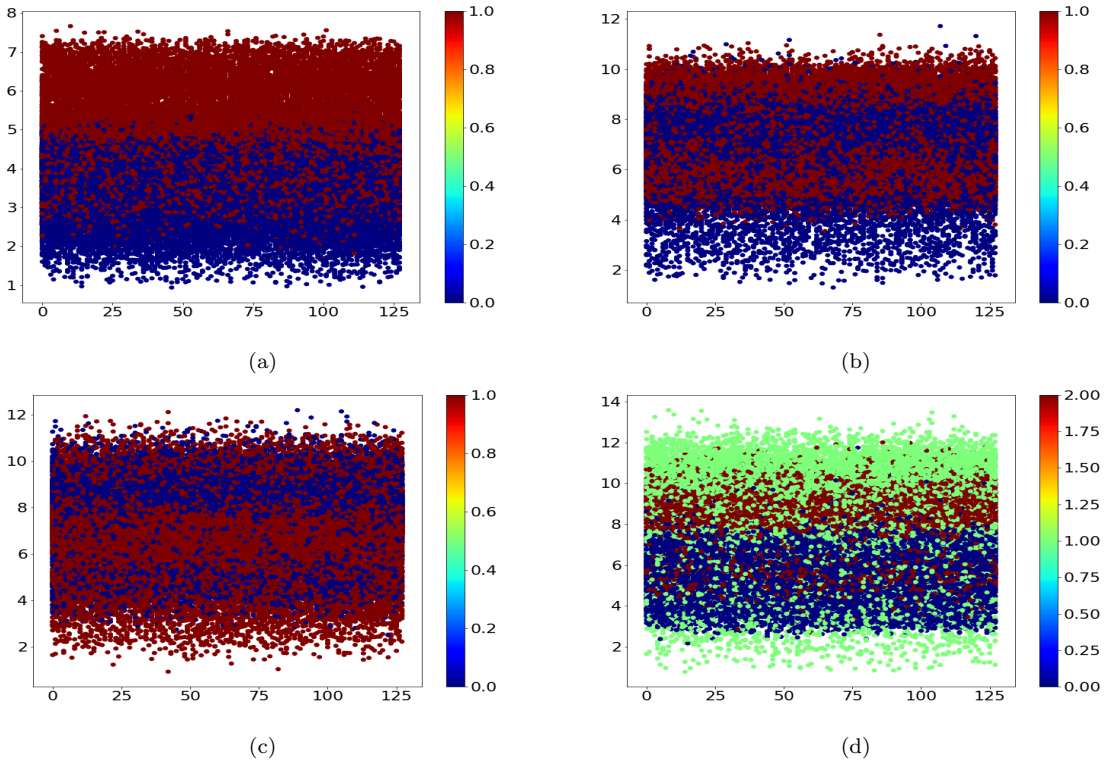


Figure B.2: Auxiliary scores of all training images corresponding to their coarse classes of (a) Case1, (b) Case2, (c) Case3, and (d) Case4 with MNIST dataset.

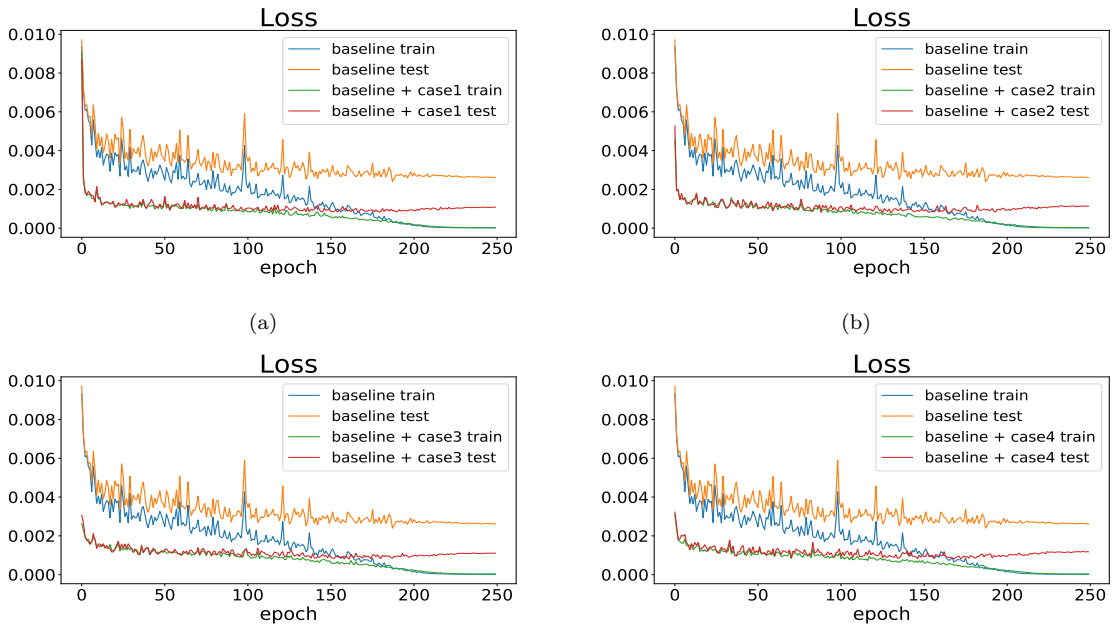


Figure B.3: Loss comparison of training and test datasets at each epoch during the training phase between the baseline and (a) Case1 (b) Case2 (c) Case3 and (d) Case4 with SVHN dataset.

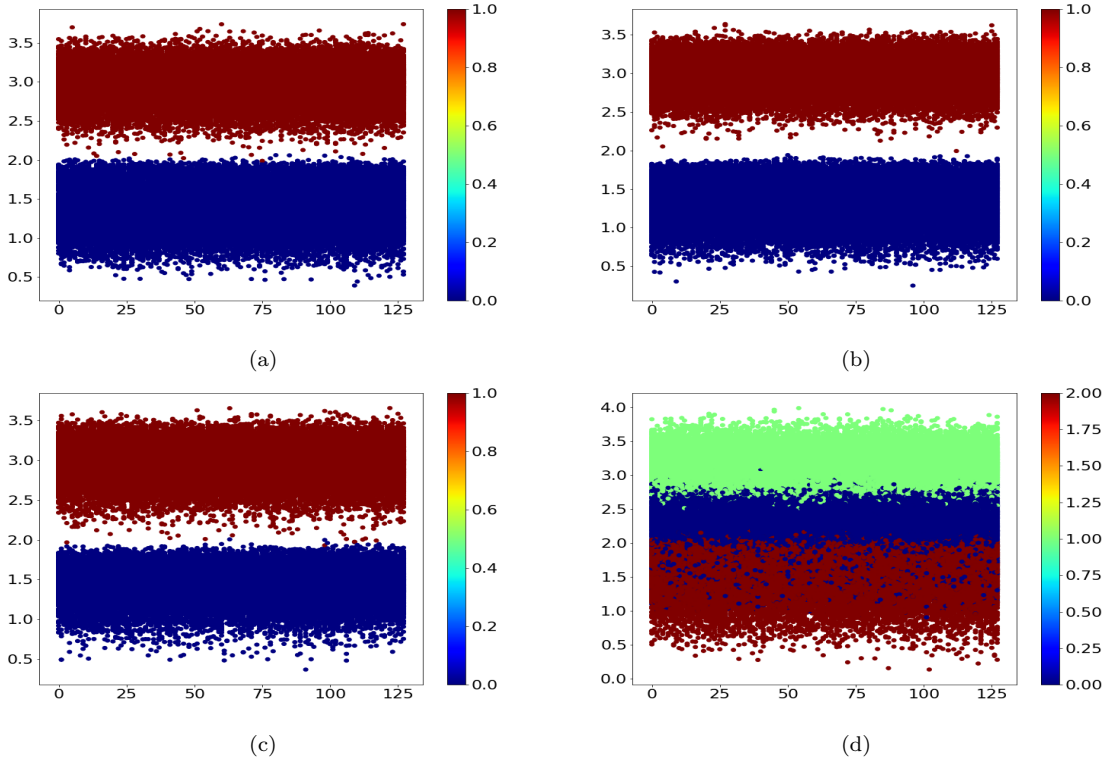


Figure B.4: Auxiliary scores of all training images corresponding to their coarse classes of (a) Case1, (b) Case2, (c) Case3, and (d) Case4 with SVHN dataset.

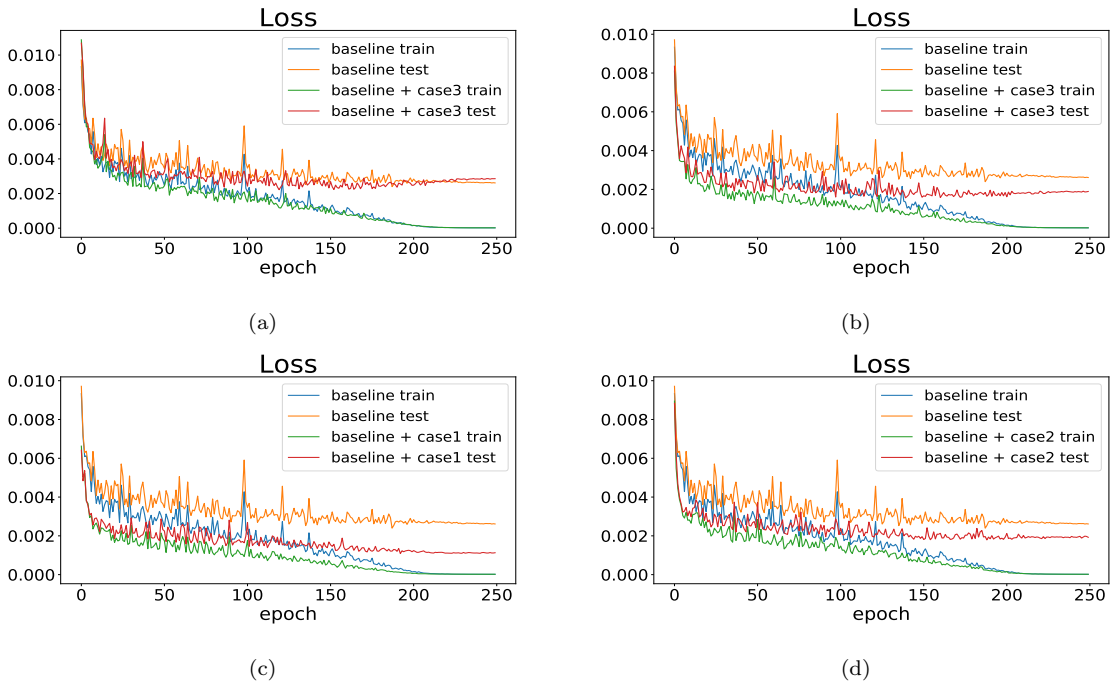


Figure B.5: Loss comparison of training and test datasets at each epoch during the training phase between the baseline and (a) Case1 (b) Case2 (c) Case3 and (d) Case4 with CIFAR-10 dataset.

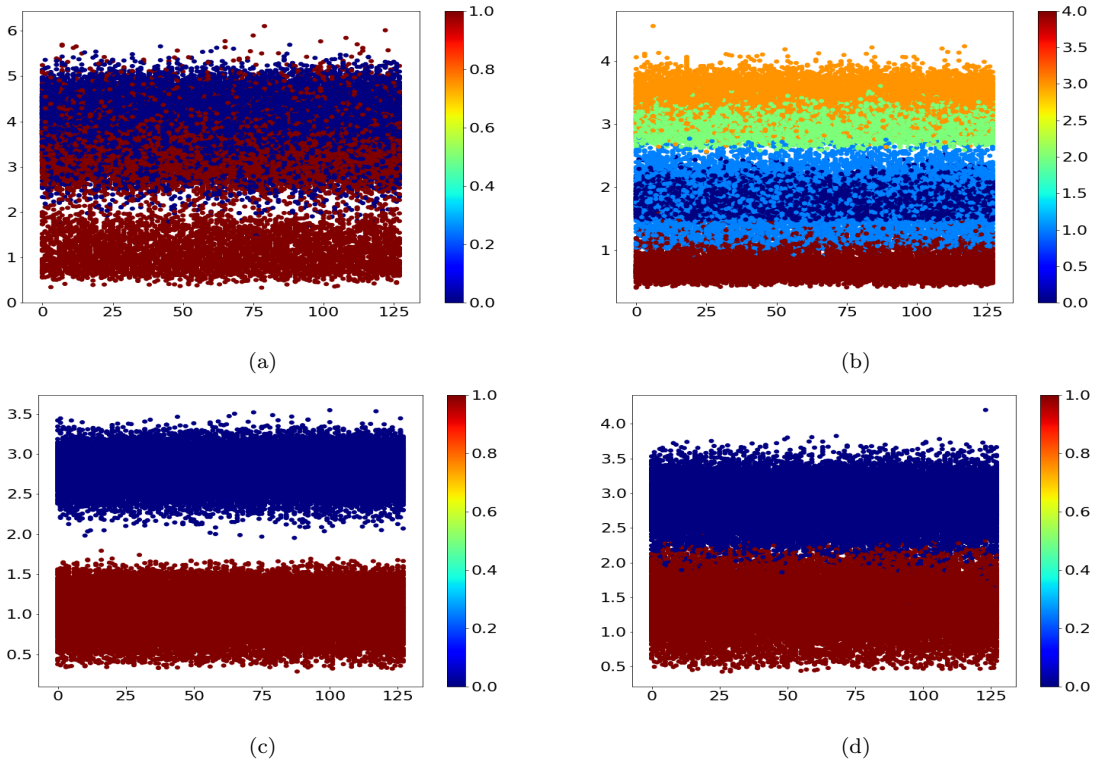


Figure B.6: Auxiliary scores of all training images corresponding to their coarse classes of (a) Case1, (b) Case2, (c) Case3, and (d) Case4 with CIFAR-10 dataset.

Appendix C

Algorithms to find weights based on the global patterns in the InhaMotion dataset

Algorithm 2 Feature weights calculation

Input: Motion number
for r in orientation angles **do**
    for s in Subjects **do**
        σ := a standard deviation of r
        $P := \{p | r$ has a local minimum or local maximum at $p\}$
        for $i = 1 : 1 : (len(P) - 1)$ **do**
            σ_L := a standard deviation between p_i and p_{i+1}
            if $\sigma_L < \sigma$ **then**
                add the mean of p_i and p_{i+1} to P and
                delete p_i and p_{i+1} from P
            end if
        end for
        Convert signals to a binary code.
    end for
    distance_r := sum of code length differences among all subjects
end for
$\text{Weight}_r = \frac{max_{r' \in R}(\text{distance}_{r'}) - \text{distance}_r}{max_{r' \in R}(\text{distance}_{r'})}$ where R is a set of all orientation angles.

Appendix D

Nine open source human motion datsets

Table D.1: data1

1. Articulary Word Recognition					
N of train	N of test	N of classes	N of channels	Lenght	Type
275	300	25	9	144	Motion
Description	Collected from multiple native English speakers producing 25 words				
Channel	each sensor produces X, Y and Z, (LL, T1, UL)				
Frequency	200 Hz				
N of subjects	N/A				
Classes	25 words in Derivation of twenty-five-word PB Lists[citation]				

Table D.2: data2

2. Basic Motions					
N of train	N of test	N of classes	N of channels	Lenght	Type
40	40	4	6	100	Motion
Description	four students performed four activities				
Channel	6, 3d acc and 3d gyroscope				
Frequency	10 per second				
N of subjects	4				
Classes	standing(0) running(1) walking(2) badminton(3)				

Table D.3: data3

3. Cricket					
N of train	N of test	N of classes	N of channels	Lenght	Type
108	72	12	6	1197	Motion
Description	In the game of Cricket umpire's gestures: motions of the hands for different events.				
Channel	Acc (X, Y and Z)on the two wrists				
Frequency	184Hz				
N of subjects	N/A				
Classes	12 events in the Cricket game				

Table D.4: data4

4. Epilepsy					
N of train	N of test	N of classes	N of channels	Lenght	Type
137	138	4	3	206	Motion
Description	Class activities. Each subjects performed each 10 times at least. Truncate data to the length of the shortest series.				
Channel	3, Acc (X, Y and Z)on the dominant wrist				
Frequency	16Hz				
N of subjects	6				
Classes	seizure mimicking(0) walking(1) running(2) sawing (3)				

Table D.5: data5

5. Ering					
N of train	N of test	N of classes	N of channels	Lenght	Type
30	270	6	4	65	Motion
Description	Detect hand and finger gesutres.				
Channel	thumb, index finger, and middle finger, distance between thumb and middle finger				
Frequency	N/A				
N of subjects	1				
Classes	open(0), fist(1) two(2), pointing(3) ring(4), grasp(5)				

Table D.6: data6

6. Libras					
N of train	N of test	N of classes	N of channels	Lenght	Type
180	180	15	2	45	Motion
Description	Brazilian sign language. Obtained from videos				
Channel	2 bidirection				
Frequency	45 frame each measured for 7 seconds				
N of subjects	4				
Classes	15 body language				

Table D.7: data7

7. NATOPS					
N of train	N of test	N of classes	N of channels	Lenght	Type
180	180	6	24	51	Motion
Description	Classify 6 motions				
Channel	24 = 8,3 (hands, elbows, wrists, and thumbs x,y, and z)				
Frequency	N/A				
N of subjects	N/A				
Classes	have command(0), all clear(1), not clear(2), spread wings(3), fold wings(4), lock wings(5),				

Table D.8: data8

8. RacketSports					
N of train	N of test	N of classes	N of channels	Lenght	Type
151	152	4	6	30	Motion
Description	To classify movements in two racket sports				
Channel	6, acc and xyz				
Frequency	10 Hz over 3 seconds				
N of subjects	N/A				
Classes	smash (0)/clear(1) in badminton forehand(2)/backhand(3) in squash				

Table D.9: data9

9. U Wave Gesture Library					
N of train	N of test	N of classes	N of channels	Lenght	Type
120	320	8	3	315	Motion
Description	Classify 8 simple gestures generated by Wii				
Channel	x,y and z by wii				
Frequency	100 Hz				
N of subjects	8				
Classes	8 simple gesture				

Bibliography

[1] Kerem Altun and Billur Barshan. Human activity recognition using inertial/magnetic sensor units. In *International workshop on human behavior understanding*, pages 38–51. Springer, 2010.

[2] Pawel Badura and Ewa Pietka. Automatic Berg Balance Scale assessment system based on accelerometric signals. *Biomedical Signal Processing and Control*, 24:114–119, 2016.

[3] Elnaz Barshan and Paul Fieguth. Stage-wise training: An improved feature learning strategy for deep models. In *Feature Extraction: Modern Questions and Challenges*, pages 49–59, 2015.

[4] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[5] Yoshua Bengio and Samy Bengio. Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems*, pages 400–406, 2000.

[6] Katherine Berg, Sharon Wood-Dauphine, JI Williams, and David Gayton. Measuring balance in the elderly: preliminary development of an instrument. *Physiotherapy Canada*, 41(6):304–311, 1989.

[7] Viv Bewick, Liz Cheek, and Jonathan Ball. Statistics review 13: receiver operating characteristic curves. *Critical care*, 8(6):508, 2004.

[8] Ricardo Cerri, Rodrigo C Barros, and André CPLF De Carvalho. Hierarchical multi-label classification using local neural networks. *Journal of Computer and System Sciences*, 80(1):39–56, 2014.

[9] Tao Chen, Shijian Lu, and Jiayuan Fan. SS-HCNN: Semi-supervised hierarchical convolutional neural network for image classification. *IEEE Transactions on Image Processing*, 28(5):2389–2398, 2019.

[10] Jaegul Choo and Shixia Liu. Visual analytics for explainable deep learning. *IEEE computer graphics and applications*, 38(4):84–92, 2018.

[11] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[12] Alana Elza Fontes Da Gama, Thiago de Menezes Chaves, Pascal Fallavollita, Lucas Silva Figueiredo, and Veronica Teichrieb. Rehabilitation motion recognition based on the international biomechanical standards. *Expert Systems with Applications*, 116:396–409, 2019.

[13] Daniel B Dias, Renata CB Madeo, Thiago Rocha, Helton H Bíscaro, and Sarajane M Peres. Hand movement recognition for brazilian sign language: a study using distance-based neural networks. In *2009 international joint conference on neural networks*, pages 697–704. IEEE, 2009.

[14] Nikita Dvornik, Julien Mairal, and Cordelia Schmid. On the importance of visual context for data augmentation in scene understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[15] Mostafa Elhoushi, Jacques Georgy, Aboelmagd Noureldin, and Michael J Korenberg. A survey on approaches of motion mode recognition using sensors. *IEEE Transactions on intelligent transportation systems*, 18(7):1662–1686, 2016.

[16] Mostafa Elhoushi, Jacques Georgy, Aboelmagd Noureldin, and Michael J Korenberg. A survey on approaches of motion mode recognition using sensors. *IEEE Transactions on Intelligent Transportation Systems*, 18(7):1662–1686, 2017.

[17] Sarah M Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58:121–134, 2016.

[18] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Data augmentation using synthetic data for time series classification with deep residual networks. *arXiv preprint arXiv:1808.02455*, 2018.

[19] Germain Forestier, François Petitjean, Hoang Anh Dau, Geoffrey I Webb, and Eamonn Keogh. Generating synthetic time series to augment sparse datasets. In *2017 IEEE international conference on data mining (ICDM)*, pages 865–870. IEEE, 2017.

[20] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

[21] Yishuang Geng, Jin Chen, Ruijun Fu, Guanqun Bao, and Kaveh Pahlavan. Enlighten wearable physiological monitoring systems: On-body RF characteristics based human motion classification using a support vector machine. *IEEE transactions on mobile computing*, 15(3):656–671, 2016.

[22] Nehla Ghouaiel, Pierre-François Marteau, and Marc Dupont. Continuous pattern detection and recognition in stream-a benchmark for online gesture recognition. *International Journal of Applied Pattern Recognition*, 4(2):146–160, 2017.

[23] Alan Godfrey, AK Bourke, GM Olaighin, P Van De Ven, and J Nelson. Activity classification using a single chest mounted tri-axial accelerometer. *Medical engineering & physics*, 33(9):1127–1135, 2011.

[24] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[25] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *Iclr*, 2(5):6, 2017.

[29] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[30] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[31] Norio Ishigaki, Teiji Kimura, Yuki Usui, Kaoru Aoki, Nobuyo Narita, Masayuki Shimizu, Kazuo Hara, Nobuhide Ogihara, Koichi Nakamura, Hiroyuki Kato, et al. Analysis of pelvic movement in the elderly during walking using a posture monitoring system equipped with a triaxial accelerometer and a gyroscope. *Journal of biomechanics*, 44(9):1788–1792, 2011.

[32] Jin-Woo Jang and Song-Nam Hong. Indoor localization with wifi fingerprinting using convolutional neural network. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 753–758. IEEE, 2018.

[33] Lei Jing and Zixue Cheng. Recognition of daily routines and accidental event with multipoint wearable inertial sensing for seniors home care. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2324–2389. IEEE, 2017.

[34] Iain M Johnstone and Bernard W Silverman. Wavelet threshold estimators for data with correlated noise. *Journal of the royal statistical society: series B (statistical methodology)*, 59(2):319–351, 1997.

[35] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[36] Enric Junqué de Fortuny, David Martens, and Foster Provost. Predictive modeling with big data: Is bigger really better? *Big Data*, 1(4):215–226, 2013.

[37] Dean M Karantonis, Michael R Narayanan, Merryn Mathie, Nigel H Lovell, and Branko G Celler. Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring. *IEEE transactions on information technology in biomedicine*, 10(1):156–167, 2006.

[38] Kyeong Soo Kim, Sanghyuk Lee, and Kaizhu Huang. A scalable deep neural network architecture for multi-building and multi-floor indoor localization based on Wi-Fi fingerprinting. *Big Data Analytics*, 3(4), April 2018.

[39] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[40] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016.

[41] David G Kleinbaum, Lawrence L Kupper, Azhar Nizam, and Eli S Rosenberg. *Applied regression analysis and other multivariable methods*. Nelson Education, 2013.

[42] Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. *arXiv preprint arXiv:1802.03690*, 2018.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[44] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[45] Yinyin Liu, Janusz A Starzyk, and Zhen Zhu. Optimized approximation algorithm in neural networks without overfitting. *IEEE transactions on neural networks*, 19(6):983–995, 2008.

[46] Elena Simona Lohan, Joaquín Torres-Sospedra, Helena Leppäkoski, Philipp Richter, Zhe Peng, and Joaquín Huerta. Wi-fi crowdsourced fingerprinting dataset for indoor positioning. *Data*, 2(4):32, 2017.

[47] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[48] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. Understanding posterior collapse in generative latent variable models. *International Conference on Learning Representations, Workshop Paper*, 2019.

[49] Lee B. Lusted. Roc recollected. *Medical Decision Making*, 4(2):131–135, 1984.

[50] Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11(7):674–693, 1989.

[51] Christopher D Manning, Hinrich Schütze, and Prabhakar Raghavan. *Introduction to information retrieval*. Cambridge university press, 2008.

[52] Louise C Mâsse, Barbara E Ainsworth, Susan Tortolero, Sarah Levin, Janet E Fulton, Karla A Henderson, and Kelly Mayo. Measuring physical activity in midlife, older, and minority women: issues from an expert panel. *Journal of Women's Health*, 7(1):57–67, 1998.

[53] Sina Mehdizadeh. The largest lyapunov exponent of gait in young and elderly individuals: a systematic review. *Gait & posture*, 60:241–250, 2018.

[54] Germán Martín Mendoza-Silva, Philipp Richter, Joaquín Torres-Sospedra, Elena Simona Lohan, and Joaquín Huerta. Long-term wifi fingerprinting dataset for research on robust indoor positioning. *Data*, 3(1):3, 2018.

[55] Charles E Metz. Roc methodology in radiologic imaging. *Investigative radiology*, 21(9):720–733, 1986.

[56] Taylor Mordan, Nicolas Thome, Gilles Henaff, and Matthieu Cord. Revisiting multitask learning with rock: a deep residual auxiliary block for visual detection. In *Advances in Neural Information Processing Systems*, pages 1310–1322, 2018.

[57] Adriano Moreira, Maria J. Nicolau, Filipe Meneses, and António Costa. Wi-Fi fingerprinting in the real world – RTLSUM at the EvAAL competition. In *Proc. International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–10, Banff, Alberta, Canada, October 2015.

[58] Bijan Najafi, Kamiar Aminian, François Loew, Yves Blanc, and Philippe A Robert. Measurement of stand-sit and sit-stand transitions using a miniature gyroscope and

its application in fall risk evaluation in the elderly. *IEEE Transactions on biomedical Engineering*, 49(8):843–851, 2002.

[59] Bijan Najafi, Kamiar Aminian, Anisoara Paraschiv-Ionescu, François Loew, Christophe J Bula, and Philippe Robert. Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly. *IEEE Transactions on biomedical Engineering*, 50(6):711–723, 2003.

[60] Bijan Najafi, David G Armstrong, and Jane Mohler. Novel wearable technology for assessing spontaneous daily physical activity and risk of falling in older adults with diabetes, 2013.

[61] Lakshmanan Nataraj, Tajuddin Manhar Mohammed, BS Manjunath, Shivkumar Chandrasekaran, Arjuna Flenner, Jawadul H Bappy, and Amit K Roy-Chowdhury. Detecting gan generated fake images using co-occurrence matrices. *Electronic Imaging*, 2019(5):532–1, 2019.

[62] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[63] Tamara ME Nijsen, Ronald M Aarts, Pierre JM Cluitmans, and Paul AM Griep. Time-frequency analysis of accelerometry data for detection of myoclonic seizures. *IEEE Transactions on Information Technology in Biomedicine*, 14(5):1197–1203, 2010.

[64] M. Nowicki and J. Wietrzykowski. Low-effort place recognition with WiFi fingerprints using deep learning. In Roman Szewczyk, Cezary Zieliński, and Małgorzata Kaliczyńska, editors, *Automation 2017*, pages 575–584, Cham, 2017. Springer International Publishing.

[65] Martin A O'Reilly, Darragh F Whelan, Tomas E Ward, Eamonn Delahunt, and Brian M Caulfield. Classification of deadlift biomechanics with wearable inertial measurement units. *Journal of biomechanics*, 58:155–161, 2017.

[66] Giulia Pacini Panebianco, Maria Cristina Bisi, Rita Stagni, and Silvia Fantozzi. Analysis of the performance of 17 algorithms from a systematic review: Influence

of sensor position, analysed variable and computational approach in gait timing estimation from imu measurements. *Gait & posture*, 66:76–82, 2018.

[67] Mitesh Patel. Action observation in the modification of postural sway and gait: Theory and use in rehabilitation. *Gait & posture*, 58:115–120, 2017.

[68] Hillary A Plummer, Federico Pozzi, and Lori A Michener. Comparison of two trunk electromagnetic sensor placement methods during shoulder motion analysis. *Journal of biomechanics*, 68:132–135, 2018.

[69] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[70] Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.

[71] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[72] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[73] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 158–167, 2000.

[74] Bernhard Scholkopf, Kah-Kay Sung, Christopher JC Burges, Federico Girosi, Partha Niyogi, Tomaso Poggio, and Vladimir Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE transactions on Signal Processing*, 45(11):2758–2765, 1997.

[75] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[76] Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. Generalizing dtw to the multi-dimensional case requires an adaptive approach. *Data mining and knowledge discovery*, 31(1):1–31, 2017.

[77] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[78] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[79] Xudong Song, Xiaochen Fan, Xiangjian He, Chaocan Xiang, Qianwen Ye, Xiang Huang, Gengfa Fang, Liming Luke Chen, Jing Qin, and Zumin Wang. Cnnloc: Deep-learning based indoor localization with wifi fingerprinting. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 589–595. IEEE, 2019.

[80] Melania Susi, Valérie Renaudin, and Gérard Lachapelle. Motion mode recognition and step detection algorithms for mobile phone users. *Sensors*, 13(2):1539–1562, 2013.

[81] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-ResNet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[82] Mary E Tinetti, Mark Speechley, and Sandra F Ginter. Risk factors for falls among elderly persons living in the community. *New England journal of medicine*, 319(26):1701–1707, 1988.

[83] Joaquín Torres-Sospedra, Raúl Montoliu, Adolfo Martínez-Usó, Joan P. Avariento, Tomás J. Arnau, Mauri Benedito-Bordonau, and Joaquín Huerta. UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems. In *Proc. International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 261–270, Busan, Korea, October 2014.

[84] Loring W.. Tu. *An introduction to manifolds*. Springer., 2011.

[85] Peter H Veltink, HansB J Bussmann, Wiebe De Vries, WimL J Martens, and Rob C Van Lummel. Detection of static and dynamic activities using uniaxial accelerometers. *IEEE Transactions on Rehabilitation Engineering*, 4(4):375–385, 1996.

[86] Jose R Villar, Paula Vergara, Manuel Menéndez, Enrique de la Cal, Víctor M González, and Javier Sedano. Generalized models for the classification of abnormal movements in daily life and its applicability to epilepsy convulsion recognition. *International journal of neural systems*, 26(06):1650037, 2016.

[87] Mark P Wachowiak, Gregory S Rash, Peter M Quesada, and Ahmed H Desoky. Wavelet-based noise removal for biomechanical signals: A comparative study. *IEEE Transactions on biomedical engineering*, 47(3):360–368, 2000.

[88] Fang Wan and Chaoyang Song. A neural network with logical reasoning based on auxiliary inputs. *Frontiers in Robotics and AI*, 5:86, 2018.

[89] Xinggang Wang, Yongluan Yan, Peng Tang, Xiang Bai, and Wenyu Liu. Revisiting multiple instance neural networks. *Pattern Recognition*, 74:15–24, 2018.

[90] Zhelong Wang, Donghui Wu, Jianming Chen, Ahmed Ghoneim, and Mohammad Anwar Hossain. A triaxial accelerometer-based human activity recognition via EEMD-based features and game-theory-based feature selection. *IEEE Sens. J*, 16(9):3198–3207, 2016.

[91] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[92] Qingsong Wen, Liang Sun, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

[93] Maciej Wiatrak and Stefano V Albrecht. Stabilizing generative adversarial network training: A survey. *arXiv preprint arXiv:1910.00927*, 2019.

[94] Mathias Wilhelm, Daniel Krakowczyk, Frank Trollmann, and Sahin Albayrak. ering: multiple finger gesture recognition with one ring using an electric field. In *Proceedings of the 2nd international Workshop on Sensor-based Activity Recognition and Interaction*, pages 1–6, 2015.

[95] Cinna Wu, Mark Tygert, and Yann LeCun. Hierarchical loss for classification. *arXiv preprint arXiv:1709.01062*, 2017.

[96] Wu-Qiang Wu, Hao-Lin Feng, Hong-Yan Chen, Dai-Bin Kuang, and Cheng-Yong Su. Recent advances in hierarchical three-dimensional titanium dioxide nanotree arrays for high-performance solar cells. *Journal of Materials Chemistry A*, 5(25):12699–12717, 2017.

[97] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[98] Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. Hd-cnn: hierarchical deep convolutional neural networks for large scale visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 2740–2748, 2015.

[99] Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 2740–2748, 2015.

[100] Che-Chang Yang and Yeh-Liang Hsu. A review of accelerometry-based wearable motion detectors for physical activity monitoring. *Sensors*, 10(8):7772–7788, 2010.

[101] Faheem Zafari, Athanasios Gkelias, and Kin K Leung. A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials*, 21(3):2568–2599, 2019.

[102] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[103] David Zambrano-Montenegro, FJ Bellido-Outeiriño, Rodolfo García-Bermúdez, JM Flores-Arias, and Alexander Huhn. Advanced monitoring system for daily activity in elderly people. In *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–2. IEEE, 2019.

[104] Yijia Zhang, Wei Zheng, Hongfei Lin, Jian Wang, Zhihao Yang, and Michel Dumontier. Drug–drug interaction extraction via hierarchical rnns on sequence and shortest dependency paths. *Bioinformatics*, 34(5):828–835, 2018.

[105] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Balancing learning and inference in variational autoencoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5885–5892, 2019.

[106] Xinqi Zhu and Michael Bain. B-cnn: branch convolutional neural network for hierarchical classification. *arXiv preprint arXiv:1709.09890*, 2017.

Curriculum Vitae

Jaehoon Cha

- Date of Birth: 1989.09.05
- Gender : M
- Country of Birth : Republic of Korea
- Nationality : Republic of Korea

Personal Details : Jaehoon Cha, BS, MS
E.mail Address: Jaehoon.Cha@xjtlu.edu.cn

Academic Qualifications

PhD, Electrical Engineering, University of Liverpool, United Kingdom, 2020
M.S, Mathematical Science, Chungnam National University, Republic of Korea, 2016
B.S, Mathematical Science, Chungnam National University, Republic of Korea, 2013

Teaching Assistant Experience

2016 - 2019: at the Department of Electrical and Electronic Engineering - (normally 40 - 120 students per modules)
2014 - 2015: at the Department of Mathematics - (normally 40 - 100 students per modules)

Research Participants

2019 - 2020: Visiting Scientist at Science and Technology Facilities Council, United Kingdom
2017 - 2019: Research Assistant at Xi'an Jiaotong – Liverpool University, China

Publication and Accepted papers

International Journal

1. Moon Keun Kim, Jaehoon Cha, Eunmi Lee, Van Huy Pham, Sanghyuk Lee and Nipon Theera-Umpon, Simplified Neural Network Model Design with Sensitivity Analysis and Electricity Consumption Prediction in a Commercial Building, Energies 2019, 12, 1201; doi:10.3390/en12071201 (SCIE)

2. Sanghyuk Lee, Jaehoon Cha, Moon Keun Kim, Kyeong Soo Kim, Van Huy Pham, and Mark Leach, Neural-Network-Based Building Energy Consumption Prediction with Training Data Generation, Processes 2019, 7, 731; doi:10.3390/pr7100731 (SCIE)

3. Kyeong Soo Kim, Ruihao Wang, Zhenghang Zhong, Zikun Tan, Haowei Songy, Jaehoon Cha, and Sanghyuk Lee, Large-Scale Location-Aware Services in Access: Hierarchical Building/Floor Classification and Location Estimation using Wi-Fi Fingerprinting Based on Deep Neural Networks," (Extended version of the FOAN 2017 paper), Fiber and Integrated Optics, vol. 37, no. 5, pp. 277-289, Apr. 27, 2018.(SCIE)

4. Sanghyuk Lee, Jaehoon Cha, Nipon Theera-Umpon and Kyeong Soo Kim, Analysis of a Similarity Measure for Non-Overlapped Data, Symmetry 2017, 9, 68; doi:10.3390/sym9050068 (SCIE)

International Conference

1. Sanghyuk Lee, Jaehoon Cha and Kyeong Soo Kim, "Data gathering and application to building energy optimization with sensitivity analysis for IoT applications," Proc. 2019 International SoC Design Conference (ISOCC), Jeju, Korea, Oct. 6-9, 2019. (EI index) https://ieeexplore.ieee.org/document/9027650

2. Jaehoon Cha, Kyeong Soo Kim, Haolan Zhang, and Sanghyuk Lee, "Analysis on EEG signal with machine learning," Proc. SPIE 11321, 2019 International Conference on Image and Video Processing, and Artificial Intelligence (IVPAI2019), 113212E, Shanghai, China, Nov. 27, 2019. (DOI)

3. Jaehoon Cha, Sanghyuk Lee and Kyeong Soo Kim, Automatic building and floor classification using two consecutive multi-layer perceptron, Proceeding of International Conference on Control, Automation and Systems(ICCAS 2018), pp. 87-91, Pyeongchang, Korea, Oct. 2018 (DBpia) (EI index) https://ieeexplore.ieee.org/document/8571593

4. Kyeong Soo Kim, Ruihao Wang, Zhenghang Zhong, Zikun Tan, Haowei Song, Jaehoon Cha, and Sanghyuk Lee, "Large-scale location-aware services in access: Hierarchical building/floor classification and location estimation using Wi-Fi fingerprinting based on deep neural networks," Proc. FOAN 2017 Workshop, Munich, Germany, Nov. 7, 2017. (DOI) (arXiv) (EI index) https://ieeexplore.ieee.org/document/8215259