Deep Learning Systems with Linguistic Interpretability in Manufacturing Image Classifications

Zhen Xi

A thesis presented for the degree of

Doctor of Philosophy

Department of Automatic Control and Systems Engineering

University of Sheffield

March 2023

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my supervisor, Professor George Panoutsos for giving me the opportunity to do research under his supervision and providing invaluable and patiently guidance throughout this research. His dynamism, widely vision, sincerity and constructive advice give me a deep inspiration. I do appreciate his understanding and support during the lockdown time in 2020.

Secondly, I am extremely grateful to my parents and my fiancee Meng ‘Jiu’ Zhang, they give a strong back to me when I were low, even I were in bad psychological state they also help me and encourage me.

Finally, my thanks go to all the friends and staffs at the department of Automatic Control and System Engineering who have supported me and give me kind help throughout the time of my PhD study.

Zhen Xi

Abstract

Machine learning has received considerable attention in recent decades in data-driven modelling systems and methods. Machine Learning focuses on applied maths and computing algorithms for creating 'computational machines' that can learn to imitate system behaviours automatically. Unlike traditional system modelling methods (physics-based, numerical etc.), machine learning does not require a dynamic process model but sufficient data, including input and output data of a specific system. It thus could get high prediction accuracy but lack interpretability.

A method to add transparency to deep CNNs is adding a fuzzy logic radius basis function to specific CNN structures named RBF-CNN. With the deletion of the defuzzy layer, a more generalised form was introduced, namely ND-RBF-CNN.

Both RBF-CNN and ND-RBF-CNN were benchmarked for linguistic fuzzy rules' prediction accuracy and interpretability. Both structures demonstrated good interpretability at a small cost of prediction accuracy. To improve the prediction accuracy, a general RBF layer initialisation methodology was explored.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence.

CNN Convolutional Neural Network.

DQN Deep Q-Network.

EBM Electron Beam Melting.

FCM Fuzzy C-Means.

FL Fuzzy Logic.

GAN Generative Adversarial Net.

KNN K-Nearest Neighbor.

ML Machine Learning.

ND-RBF-CNN No Defuzzy-Radial Basis Function-Convolutional Neural Network.

RBF Radial Basis Function.

RBF-CNN Radial Basis Function-Convolutional Neural Network.

ReLU	Rectangular Linear Unit.
SGD	Stochastic Gradient Descent.
SIMD	Single Instruction, Multiple Data.
SVM	Support Vector Machine.
XCT	X-Ray Computed Tomography.

1 Introduction

Machine learning, as a research discipline, has been developed in multiple directions for decades, which can be defined as a set of methods that can automatically detect patterns in data and then use uncovered patterns to predict future data, and to study self-improvement methods of computers that to obtain new knowledge and new skills, identify existing knowledge, and continuously improve the performance and achievement [51]. In terms of data-driven modelling systems and methods, machine learning has received considerable attention in recent decades. Machine Learning focuses on applied maths and computing algorithms for creating 'computational machines' that can learn to imitate system behaviours automatically [69]. As a subarea of Artificial Intelligence (AI), using Machine Learning (ML) one could also construct computer systems and algorithms to improve performance based on what has already been experienced (empirical-based, learning from examples) [69, 34].

As a core subarea of AI, ML has much flexibility. After development over two decades, machine learning has emerged as a choice for models used in natural language processing, speech recognition, computer vision, robot control, and other applications [69, 34, 43].

Unlike traditional system modelling methods (physics-based, numerical etc.), machine learning does not require a dynamic process model but sufficient data, including input data and output data of a specific system, hence a class of machine learning algorithms can be considered as data-driven modelling methods that are able to capture static or dynamic process behaviour in areas such as manufacturing and biomedical systems among others. Gong et al. introduced a way to analysis time series signals and to create a human body model using CNNs [25]. Segreto et al. evaluated the correlation between wavelet processed time series signals and the machining conditions using neural networks [72]. Based on the type of modelling structures used, machine learning could be broadly viewed in two parts with — to a certain extent — unclear boundaries, which are statistical modelling and learning, and neural and other hybrid network structures [34].

1.1 Problem statement

Machine Learning has been widely used in computer vision, speech recognition, and natural language processing. In particular, Machine Learning in image recognition has been very successful in certain class of problems. Research developments in Machine Learning have enabled the wider use of neural-network type computational structures.

CNNs are considered as black-boxes, which means hard to understand, so more methods are created. Integrated gradients [82] and LIME [65] could spot out the potential regions of interest. Gradient SHAP [46], KernelSHAP [46]

DeepLIFT [75], and DeepLIFT SHAP [46] could figure out the importance of variety features. Guided Backpropagation [79] with Deconvolution [95] could visualise features learnt by CNNs.

In the manufacturing literature, machine learning in general is widely applied. Machine learning methods are used in advanced manufacturing systems and processes, including in product-process design, quality monitoring and control, job shop scheduling, thermo-mechanical manufacturing process, and other related applications [56, 11].

The first challenge relates to the fact that deep learning requires large datasets to train networks, and in general more complex/advanced networks require even larger databases - these are not always available in manufacturing systems. The second challenge is that in manufacturing, fundamental process understanding and ease of application are very important for adopting any computation method as part of the manufacturing workflow. The high complexity associated with deep learning networks limits their interpretability and implementation (towards adoption of the methodology in a manufacturing setting) [62].

However, there are still challenges in the applications of deep learning, in particular when one considers manufacturing processes. In DL large datasets/samples often is needed to train networks, and more complex networks require larger sample size. The second aspect, is that the additional complexity coming with DL structures, often prevents researchers using such structure, not due to lack of performance, but due to lack of fundamental understanding relevant to the process at hand — this is very prominent in manufacturing applications.

Adding interpretability features in machine learning structures could benefit certain applications of machine learning, where interpretability can be of benefit. For example, in advanced manufacturing systems, where understanding and modelling images and videos of complex processes are critical tasks. A process model (or classifier) based on CNNs could be developed to take advantage of processing data in array forms [43] which has already been proven to be very effective [80, 83] in a number of applications.

1.2 Research aims

For complex manufacturing processes it is very challenging to generate accurate and meaningful process models based on physical models alone. The main difficulties often arise due to lack of fundamental process understanding, high non-linear processes as well as the various process uncertainty sources. Data-driven modelling methods, have therefore received significant attention in manufacturing, and in particular Machine Learning methods. Data/information generated in manufacturing often is scarce and in general is only limited to a few samples/trials for high value manufacturing components. Not many ML methodologies address the low sample size problem with success. In addition, there is a rise in demand in interpretable models and algorithms, that non-experts can easily interact with. Towards this vision, this PhD will be focusing on interpretable Artificial Intelligence, in the form of research in Deep Learning methodologies, augmented by interpretable computational structures such as the ones offered by Fuzzy Logic, Rough Sets, Granular Computing etc.

In general, modelling methods that aimed at industrial/manufacturing use, may contain the following attributes:

- Data-driven;
- Capable of dealing with small data sets;
- Capable of dealing with high uncertainty and complexity;
- Transparent and interpretable.

Therefore, the major aims of this research work can be list as follows:

- The first study aim is to classify images with machine learning based on a large dataset. Hence, a Convolutional Neural Network (CNN) model is suggested to solve this problem.
- Secondly, an Radial Basis Function (RBF) network is suggested to combine with existing CNN models in order to create interpretability and improve transparent.
- Finally, this research work aims to dealing with small datasets. As the difficultness existing in training small datasets, transfer learning is suggested to solve the initilastion of an Radial Basis Function (RBF) model.

1.3 Achievements

The main contribution of this research work is to provide a number of methodologies for interpretation image classifications in linguistic fuzzy rules. As the auxiliary of development such a system, a vision interpretation of fuzzy rules has also been discovered. For the better compatibility to other deep convolutional neural networks, an improved model was invested, and proved that the improved model still has the linguistic fuzzy rule interpretability.

The work in this thesis has contributed in part or full to the following publications:

- Zhen Xi and George Panoutsos. “Interpretable Machine Learning: Convolutional Neural Networks with RBF Fuzzy Logic Classification Rules”. In: 2018 International Conference on Intelligent Systems (IS). 2018 International Conference on Intelligent Systems (IS). Sept. 2018, pp. 448–454. - Zhen Xi and George Panoutsos. “Interpretable Convolutional Neural Networks Using a Rule-Based Framework for Classification”. In: Intelligent Systems: Theory, Research and Innovation in Applications. Ed. by Ricardo Jardim-Goncalves et al. Studies in Computational Intelligence. Cham: Springer International Publishing, 2020, pp. 1–24.

1.4 Outline of the thesis

The structure of this thesis is organised in 8 chapters and one appendix. In this chapter the basic background and contributions was introduced. The next 7 chapters describe the current contributions and the conclusion of this thesis.

Chapter 2, covers the main components and techniques of convolutional neural networks, which may be useful to implement image processing systems.

Chapter 3 includes an first attempt combining fuzzy logic radial basis function network with convolutional neural networks, namely RBF-CNN. In this chapter it was proved the combination of such two systems would not affect prediction performance significantly on a simple but popular benchmark dataset MNIST.

Chapter 4, as the continuous part of Chapter 3, introduced the linguistic fuzzy rule analysis methodology. Besides, this chapter also applied RBF-CNN on a more complex dataset Fashion-MNIST, and acquired good prediction accuracy as expectations.

Chapter 5 aims on removing a component, defuzzy layer, from RBF-CNN. The new model contains no defuzzy layer, hence named as No Defuzzy RBF-CNN (ND-RBF-CNN). With benchmarks, it was proved that ND-RBF-CNN can still acquire acceptable prediction accuracy.

Chapter 6 follows the previous chapter, explored the interpretability of linguistic fuzzy rules applied on ND-RBF-CNN. Furthermore, the benchmark was completed with a small manufacturing dataset EBM-XCT.

Chapter 7 aim on solving the initialisation problem, which companies to all FL RBF system. The accuracy of ND-RBF-CNN could reach a higher level with the actions provided in this chapter.

2 A background to machine learning techniques

The main objective of this chapter is to provide an insight about the existing techniques developed in machine learning.

2.1 Convolutional neural networks

In deep learning in particular [34], CNNs have been widely used [83, 40, 42]. CNNs are a kind of feed-forward neural network using convolutional cores to process data in multiple arrays. Multiple arrays could be in the form of variable data modalities: 1D for time-domain signals, 2D for images, and 3D for videos [43].

Backpropagation is a computational method to calculate the error contribution of every weight in neural networks after processing a batch of data. It can be applied to a variety of modelling structures, including neural networks and fuzzy logic systems [88]. For neural networks, backpropagation procedure could compute gradients for all modules constructed by relatively sooth functions of their internal weights and of their inputs [43].

In following equations, a notation as w^l_{jk} is used, which means the weight from k-th neuron in the $(l-1)$-th layer to the k-th neuron in l-th layer. Specially, upper label L in w^l_{jk} presents for the last layer of a neural network. In [52], Nielson provided four equations for a neural network containing layers as defined as $a^l = \sigma(z^l) \equiv \sigma(w^l a^{l-1} + b^l)$:

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j), \tag{2.1}$$

$$\delta^l = ((w^{l+1})^\intercal \delta^{l+1}) \odot \sigma'(z^l), \tag{2.2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j, \tag{2.3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j, \tag{2.4}$$

where C is cost of the neural network, function $f(x) = \sigma(x)$ is the activation function, and $\odot$ denotes Hadamard product.

2.1.1 Wide-used methods

Neural networks are a kind of mathematics model which could be expressed as a combination of layers. In this research project, the most of data was images, hence only CNNs were focused on.

Both advantages and disadvantages of CNNs are important to review. Unlike statistical models, a CNN can extract features automatically, and this feature extraction could be transferred to any other un-trained CNNs. However, a well trained CNN requires large datasets to support effective training, hence the training process is computational expensive [40].

2.1.2 Components

A typical CNN for image classification would contain several layers, grouped in a way to perform specific tasks. These components could be mainly classified into five types, which are

- Convolution layer;
- pooling layer: max-pooling, average-pooling, etc.;
- Non-linear components: ReRU [24], leaky ReLU [31], tanH [26], etc.;
- Dense layer;
- Utility layer: padding layer, dropout [80], etc.

A more comprehensive list of usual components could be found in [85].

A typical CNN structure would start with multiple pairs of a convolution layer and a max pooling layer, these are used for feature extraction. The size of these convolution windows can be different, which ensure convolution layers can extract features in different scales. Fully connected layers would also be used in typical CNNs, in which neurons are fully connected to all outputs from the previous layer. Usually, a Rectangular Linear Unit (ReLU) would be the activation function of convolution layers and fully connected layers because ReLU could provide non-linear property to those layers and is easy to calculate in backpropagation [24]. In order to avoid overfitting, dropout layers, which would drop out several neurons in each iteration randomly,are added before fully connected layers as a simple way [80]. In case of exploding and vanishing gradients in deep networks, batch normalisation could be applied in every layer [33].

Initialisation

CNNs are considered as non-convex function, hence the initial value of parameters would affect the training result significantly. In the practical side, multiple initialisation methods were raised and benchmarked. Xavier normal distribution and uniform distribution rely on both of the input dimension and the output dimension [23], while Kaiming normal and uniform distribution were created and benchmarked with a better performance, which rely on one of the input dimension or the output dimension [31]. While from theoretical side, orthogonal initialisation was derived [68], and was proved that drawing the initial weights from the orthogonal group speeds up convergence relative to the standard Gaussian initialization for deep linear networks [32].

Optimisation

Because of the non-convexity, the optimisation methods of CNNs' losses were also focused on. Stochastic Gradient Descent (SGD) are generally used in backpropagation [3]. In order to improve stability and convergence of regular gradient descent, Nesterov momentum was applied [6]. SGD with Nesterov momentum introduced learning rate as the first-order momentum of the optimisation point. Adagrad [21] changes learning rate for different parameters with variety update frequency, which could be regard as providing the second-order momentum of the optimisation point. Adadelta [94, 70] improved Adagrad and avoid the un-wanted early-stopping. Adam [39] combines the first-order and the second-order momentum of the optimisation point.

2.2 Applications

Using CNN deep learning structures has been very successful for certain class of applications, for example Szegedy et al. proved a deep enough network can classify ImageNet [17] efficiently [83], and He et al. provided a model structure to build deep neural networks without considerable gradient loss [30]. Simonyan and Zisserman show that CNNs could be designed as even 'deeper' structures, and perform even better in ImageNet classification problems [78].

Deep learning is a subset of machine learning, with special focus on complex (deep) neural structures. As discussed above, machine learning systems are used widely in a variety of applications. The boundary between the concept of 'deep learning' and 'shallow learning' is not clear, while often it is assumed that problems whose depth of Credit Assignment Paths > 10 require deep learning [71].

Same as some ordinary neural networks, deep learning systems usually use backpropagation to adjust parameters throughout whole networks [34]. However, backpropagation for very deep networks may not work efficiently, hence these require adjustments to make them work effectively [83].

There are three types of learning methods that have been addressed by deep learning, depending on the way that datasets are utilised, namely: supervised learning, unsupervised learning, and reinforcement learning.

Firstly, supervised learning is the most common, and most widely used machine learning form [43, 34]. The datasets for supervised learning would be labelled [43, 34]. A famous example of this kind of machine learning is GoogLeNet, which achieved 6.67% top-5 error rate in a 1000-label task [83]. Secondly, unsupervised learning usually utilise unlabelled data under assumptions about structural properties [34]. A Generative Adversarial Net (GAN) would generate a new dataset from a given unlabelled dataset [27]. Thirdly, reinforcement learning may describe a model trained by a dataset, whose available information is intermediate between unsupervised and supervised learning [34]. DeepMind made two models, which are Deep Q-Networks (DQNs) [48] for video game playing and AlphaGo [77, 76] for Go competition.

2.3 Radial basis functions

A radial basis functions (RBF) ϕ is a real-value function whose outputs depend on distance between the input $\vec{x}$ and fix points $\vec{c}$, so that $\phi(\vec{x}) = \phi(\|\vec{x} - \vec{c}\|)$. RBFs can be applied as a kernel function in multiple machine learning methodologies, for example in Support Vector Machines (SVMs) in order to solve non-linear classification problems [49]. Broomhead et al˙ [8] formed an interpolating function with RBFs as

$$s(\vec{x}) = \sum_{j=1}^{m} w_j \phi(\|\vec{x} - \vec{c_j}\|) \quad \vec{x} \in \mathbb{R}^n, \tag{2.5}$$

which can be expressed in network forms equivalently. RBF networks can also be implemented in FL-based systems as SVM [56].

A RBF ϕ can be a symmetric convex function. For FL interpretability, Gaussian functions was chosen in several models [10, 56, 91], and the RBF network is summarised in Fig. 2.1.

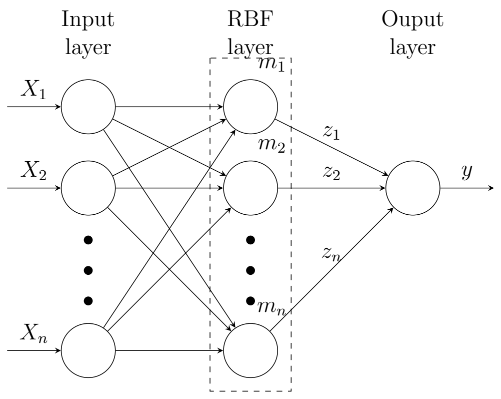


Figure 2.1: RBF network structure

RBF networks were formulated in [8] as a learning network structure. RBF networks can also be used efficiently as as a kernel function for a variety of machine learning methodologies, for example in Support Vector Machines to solve non-linear classification problems [49]. Similar to SVM, RBF networks could be implemented as FL-based systems [56].

In this section, for the benefit of the reader, the RBF-NF network is summarised (Fig. 2.1), and its relevance to the deep learning structure is shown, while full details of the fundamental RBF network as a data-driven model can be found in [10, 56].

Equation (2.6) represents a multiple-input and single-output (MISO) FL system with m system inputs and p number of rules, where $\mu_{ij}(x_j)$ defined in (2.7) is the Gaussian membership function of input x_j belonging to the i-th rule and c_{ij} and σ_{ij} are the centre and width of the Gaussian membership function respectively [56]. The overall function $z(\vec{x})$ could be adjusted to represent one of the following three forms of FL-based systems:

- Singleton;
- Mamdani;
- Takagi-Sugeno.

In the proposed work, the overall system function $z(\vec{x})$ will be considered as a Singleton model. Fig. 2.1 depicts the structure of the RBF network, where X_n are the system's inputs, μ_{ij} is the membership function of each rule-input combination, m_n is the membership function vector of each input, z_n is the Tagaki-Sugeno polynomial function for each rule, and y is the overall output of the system. Hence the output function takes the mathematical form shown in (2.8).

$$\begin{aligned} y &= \sum_{i=1}^{p} z_i \left[\frac{\prod_{j=1}^{m} \mu_{ij}\left(x_j\right)}{\prod_{i=1}^{p} \sum_{j=1}^{m} \mu_{ij}\left(x_j\right)} \right], \\ &= \sum_{i=1}^{p} z_i g_i(x), \end{aligned} \tag{2.6}$$

$$\mu_{ij}(x_j) = \exp\left(-\frac{(x_j - c_{ij})^2}{\sigma_{ij}^2}\right), \tag{2.7}$$

$$z_i = \sum_{i=1}^{p} b_i x_i \tag{2.8}$$

Equation (2.7) could be expressed in vector form, as follows (which is also the expression for a RBF in i dimensions):

$$m_i(\vec{x}) = \exp\left(-\|\vec{x} - \vec{c}_i\|^2 / \vec{\sigma}_i^2\right), \tag{2.9}$$

thus this FL system could be written as:

$$\begin{aligned} y &= \sum_{i=1}^{p} z_i m_i(x) \Big/ \sum_{i=1}^{p} m_i(x), & (2.10)\\ &= \sum_{i=1}^{p} z_i g_i(x), & (2.11)\end{aligned}$$

where

$$\begin{aligned} g_i &= \left[\frac{\prod_{j=1}^{m} \mu_{ij}(x_j)}{\prod_{i=1}^{p} \sum_{j=1}^{m} \mu_{ij}(x_j)}\right],\\ &= m_i(x) \Big/ \sum_{i=1}^{p} m_i(x). & (2.12)\end{aligned}$$

Following from equations (2.9) and (2.12), the activation function becomes:

$$m_j = \exp\left(-\|\vec{x} - \vec{c}_i\|^2 / \vec{\sigma}_i^2\right), \tag{2.13}$$

$$g_j = m_j^j / \sum_{j=1}^{p} m_j, \tag{2.14}$$

therefore, for the convenience of computing, g_j could be denoted as

$$g_j = s\left(-\|\vec{x} - \vec{c}_i\|^2 / \vec{\sigma}_i^2\right), \tag{2.15}$$

where $s(x)$ is a softmax function.

In the defuzzification layer, there would be

$$y = \vec{z} \cdot \vec{g}. \tag{2.16}$$

2.4 Interpretability for deep learning

In the manufacturing literature, machine learning in general is widely applied. Machine learning methods are used in advanced manufacturing systems and processes, including in product-process design, quality monitoring and control, job shop scheduling, thermo-mechanical manufacturing process, and other related applications [56, 11].

Process monitoring is an essential part of manufacturing, while monitoring data would occupy great space. In order to reduce data size without affect monitoring, a system based on deep learning and fuzzy classification were introduced in [50]. Using a deep convolutional autoencoder, an image could be compressed from resolution of 120×120 to 15×15 without affecting the overall performance of the fuzzy classification methodology.

In the steel industry, weight percentage of composite materials of steel and heat treatment regimes are two of main affections of steel quality. A linguistic rule based fuzzy logic optimisation model was introduced to this non-linear processing in [56]. With granular computing (GrC) processed data, the model achieved elongation with RMSE of 1.43.

On the other hand, fuzzy models are still developed into deep with interpretability. Deep Rule Based (DRB) models [4, 28] combine massive fuzzy rules with deep neural networks, in order to achieve a similar accuracy comparing with deep CNNs. SetSVM [44] can classify specific cancer images with high accuracy and good visual interpretation. Incremental Neuro-Fuzzy Gaussian mixture network (INFGMN) [47] focuses on building precise models with a good trade-off between accuracy performance and interpretability.

Fuzzy Logic (FL) is also mixed with CNNs for various purposes. Price et al. inserts fuzzy layers between convolutional layers in a CNN and achieved higher accuracy [60]. Fuzzy pooling layers [74] are also been found and benchmarked. Yeganejou and Dick [93] introduces a fuzzy clustering model training with features extracted with a pre-trained ResNets [30], and gains a good accuracy with visual interpretation.

There are existing attempts in the literature to combine FL with deep learning. Muniategui et al. designed a system in spot welding monitoring [50]. In this approach the authors use the deep learning network only as a method for data pre-processing, followed by the FL classifier as a separate process step. In an attempt to reduce data size without affect monitoring performance, a system based on deep learning and FL classification was introduced. Using a deep convolutional autoencoder, an image could be compressed from resolution of 120×120 to 15×15 without affecting the overall performance of the fuzzy classification methodology. Deng et al. introduced a FL-based deep neural network (FDNN) which extracts information from both neural representation and FL simultaneously [19]. It was shown that the FDNN has higher classification accuracy than networks based on NN or FL separately and then fusions the results from the two kinds of networks. The current gap in the research literature is in that the deep learning methodologies, when combined with FL, are not integrated together as a single system, which means the gap between systems may cause the interpretability broken. For example, a CNN model could extract features from images, and an interpretable system could classify the input image with only the features. Although the interpretable classifier is a white box, the CNN part still remains as a black box.

3 Convolutional Neural Networks with RBF Fuzzy Logic Classification Rules

In [43], LeCun states the usage of convolution layers of CNNs is to extract different scale features. In this research work, it is proposed that a deep learning network, which includes a convolution layered structure, and for the first time in the literature include a FL layer (RBF) to perform the classification task. An extra layer was proposed here, which is an RBF layer to maintain the rule base of the system. To defuzzify the FL statements into crisp classification labels, a normalised exponential function (softmax) is used. Due to the addition of the FL layer one has to consider the credit assignment and error back-propagation for these layers which is not a trivial task.

There are some research studies on the similar topic published, but no one demonstrates linguistic interpretability, which could be described in the form of 'IF . . . THEN . . . '. Su and Cao [81] provides a method using fuzzy logic to detect multiple changes between and but without any CNN part. Sharma et al. [74] invented a fuzzy pooling layer which could be combined with CNNs while the pooling layer would not provide linguistic interpretability and their purpose of the fuzzy pooling layer was to increase classification accuracy not the interpretability. Price et al. [60] introduced a method combines fuzzy logic layers with VGG-16 [96] to improve the classification accuracy.

This chapter provides a new methodology based on FL and CNNs in order to improve linguistic interpretability of deep CNNs applied for image classification tasks. For the interpretability, a FL-based layer (in the form of a hybrid Neural-Fuzzy network) is introduced as an integral part of the overall CNN structure, which acts as the main classification layer (fully connected) of the deep learning structure. Consequently, one could extract directly from the deep learning structure linguistic rules in the form of a FL rule-base. Via simulation results based on a popular benchmark problem/dataset it was shown that the proposed network structure performs as well as state-of-the-art CNN-based structures, hence there is no significant loss of performance by introducing the FL layer as part of the deep learning structure. In addition, the robustness of the learning process is also assessed by consecutively reducing the sample size.

The motivations for developing this new layer structure is to interpret deep CNNs, which lack any significant interpretation, and act as 'black boxes' that predict/classify data well. There is an opportunity therefore, to use the paradigm of FL theory, and attempt to add linguistic interpretability to neural-based structures [56, 57, 53] and achieve the same effect on deep learning structures. Successful implementation would be beneficial to a variety of problems, in particular in cases where there is a need for human-machine interaction, such as in decision support systems for critical applications (healthcare, biomedical, high-value manufacturing etc.). For example, one could use FL theory to provide linguistic interpretation to classification tasks performed by deep learning networks.

3.1 Mathematical model for FL-RBF layers

In Chapter 2, a p-rule RBF model was described as following:

$$\begin{aligned} y &= \sum_{i=1}^{p} z_i \left[\frac{\prod_{j=1}^{m} \mu_{ij}\left(x_j\right)}{\prod_{i=1}^{p} \sum_{j=1}^{m} \mu_{ij}\left(x_j\right)} \right], \\ &= \sum_{i=1}^{p} z_i g_i(\vec{x}), \\ \mu_{ij}(x_j) &= \exp\left(-\frac{\left(x_j - c_{ij}\right)^2}{\sigma_{ij}^2} \right), \end{aligned} \tag{3.1}$$

where z_i is the i-th Tagaki-Sugeno polynomial function, $\mu_{ij}(x_j)$ is the j-th Gaussian membership function corresponding to j-th input, and c_{ij} and σ_{ij} are the centre and width of the Gaussian membership function respectively.

For the convinence, the RBF model could be seperated into two parts, which are named as the RBF layer and the defuzzyfication layer.

3.1.1 RBF layer

As introduced with details in Chapter 2, the equations describing RBF layer are also using p as number of rules. Following from (2.9) and (2.12), the activation function becomes:

$$m_j = \exp\left(-\|\vec{x} - \vec{c}_i\|^2/\vec{\sigma}_i^2\right), \tag{3.2}$$

$$g_j = m_j^j / \sum_{j=1}^{p} m_j, \tag{3.3}$$

therefore,

$$g_j = s\left(-\|\vec{x} - \vec{c}_i\|^2/\vec{\sigma}_i^2\right), \tag{3.4}$$

where $s(x)$ is a softmax function.

For this layer, the vector $\vec{x}$ is the input, and the vector $\vec{g} = (g_1, g_2, \ldots, g_p)$ is the layer output. Both the Gaussian membership function centre matrix $\vec{c}$ and the width matrix $\vec{\sigma}$ could be used as parameters during training.

3.1.2 Defuzzyfication layer

In a defuzzy layer, using $\vec{g}$ to denote the output from RBF layer as a vector in the shape of p-by-1, there would be a singleton defuzzy equation

$$y(\vec{g}) = \vec{z} \cdot \vec{g}, \tag{3.5}$$

where y is the prediction label in the range of $[0, N)$ if there are N labels, z is a defuzzy vector whose shape is 1-by-p.

Noteworthily, the outputs of a RBF layer would be continuous floating numbers rather than discrete integers. Rounding the output of this layer to the nearest integer (based on a predetermined threshold) would give the integer class. Therefore, the final prediction

$$\hat{y} = \mathrm{Round}(y), y \in \mathbb{R}. \tag{3.6}$$

3.2 Proposed CNN-FL modelling structure

Adding interpretability in a CNN deep learning structure could be achieved by performing the final classification task using a FL-based structure. With an FL-RBF layer and a defuzzy layer at the end of CNN rather than one fully connected layer, it is possible to interpret how the features of one image trigger a set 'IF ... THEN ... ' form linguistic rules.

In this section, the integration of a Radial-Basis-Function Neural-Fuzzy layer was described into the deep learning structure, that provides the mechanism to extract a linguistic rule base from the CNN.

3.2.1 A respective CNN structure

A representative CNN structure for image classification would contain several layers, grouped in a way to perform specific tasks. Fig. 3.1 demonstrates a typical CNN architecture. The first few layers would be multiple pairs of convolution layers and pooling layers. The size of these convolution windows can be different, which ensure convolution layers can extract features in different scales. The pooling layers are proposed to subsample features into a smaller size, where a max pooling method is generally used. Then, fully connected layers

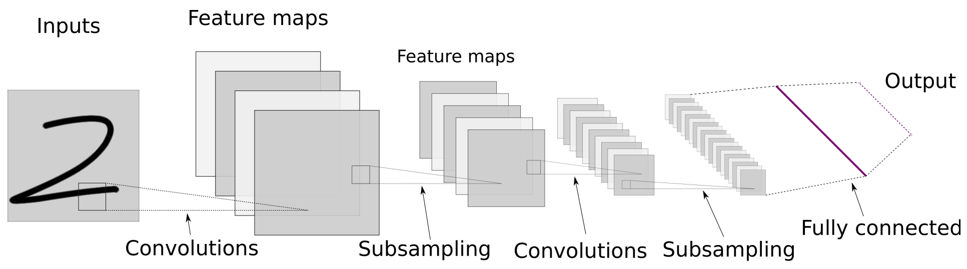


Figure 3.1: Representative CNN structure

would also be used, in which neurons are fully connected to all outputs from the previous layer. These layers also convert the data structure from a multiple-layer structure to a vector form. Rectangular linear units (ReLUs) would normally be the activation function of the convolution layers as well as in the fully connected layers as these can provide non-linear properties to those layers and are also convenient for the calculation of the error backpropagation [24]. To avoid exploding and vanishing gradients in deep networks, batch normalisation can also be applied in every layer [33]. CNNs are not considered as convex functions, which means parametric optimisation for CNNs is challenging, hence numerous optimisation strategies have been developed [70], such as SGD, Nesterov momentum [6], and adaptive subgradient (Adagrad) methods [21].

This model was designed to use 28×28 pixel grey-scale images as input. After two convolutional layers which have 32 3x3 filters and 64 3x3 filters, a max pooling layer whose pooling size is 2x2 was added. Afterwards, the dropout layers were applied to avoid overfitting. Finally, the Flatten layer was added to convert data structure into vectors, and two Dense layers are a fully connected layer with x ($x \in \{16, 32, 64\}$) hidden units, and a fully connected output layer with 10 neurons. For the convenient, this model could be described

Table 3.1: Basement CNN architecture

type	**patch size/stride**	**output size**	**parameters**
convolution	$3 \times 3/0$	$26 \times 26 \times 32$	320
convolution	$3 \times 3/0$	$24 \times 24 \times 64$	18496
maxpooling	$2 \times 2/0$	$12 \times 12 \times 64$	0
dropout (25%)		$12 \times 12 \times 64$	0
flatten		9216	0
linear		64	589888
dropout (50%)		64	0
linear		10	1290
softmax		10	0

as 1x28x28-32C3-64C3-MP2-xN-10N. Fig. 3.2 depicts the overall structure of the CNN deep network.

All activation functions in this model were ReLUs. The loss function of this model was cross entropy loss function, which is widely used in CNNs [40, 83]. In the proposed research work, the SGD optimisation method was applied to perform the learning task, to take advantage of its fast convergence properties. The loss function was calculated with mean square error (MSE). In order to achieve a good balance between training speed and avoidance of overfitting the batch size was chosen as 128. Table 3.1 shows the architecture of the designed CNN.

3.2.2 Convolutional neural network with an RBF fuzzy logic rule-base classification layer

In this section, the main CNN structure is summarised, and it is shown how the RBF-NF layer is integrated into the overall network structure and learning

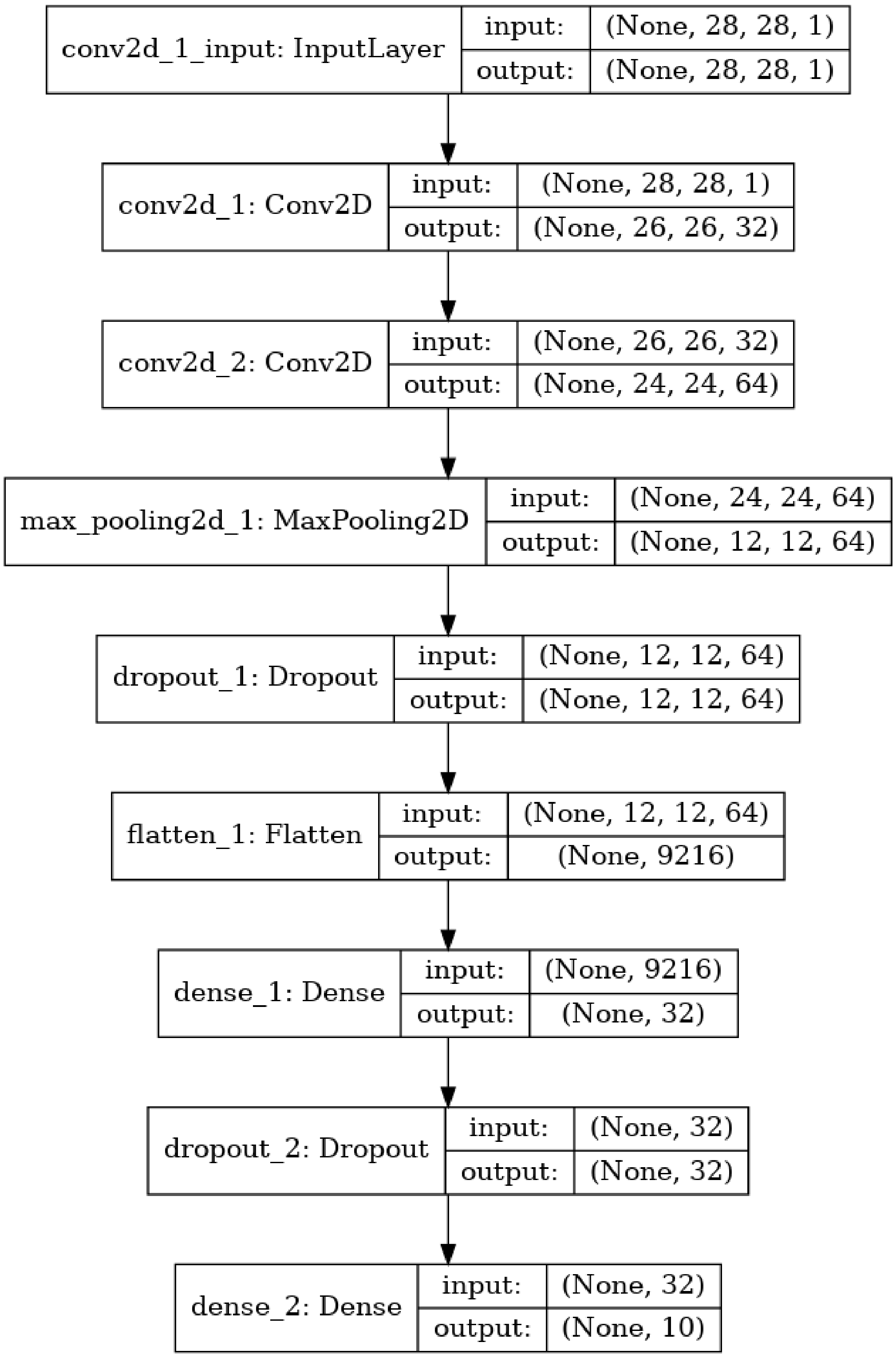


Figure 3.2: basic CNN structure

Table 3.2: FL RBF-CNN architecture

type	patch size/stride	output size	parameters
convolution	$3 \times 3/0$	$26 \times 26 \times 32$	320
convolution	$3 \times 3/0$	$24 \times 24 \times 64$	18496
maxpooling	$2 \times 2/0$	$12 \times 12 \times 64$	0
dropout (25%)	—	$12 \times 12 \times 64$	0
flatten	—	9216	0
linear	—	`feature size`	9216×`feature size`
dropout (50%)	—	`feature size`	0
RBF	—	`rule size`	2×`rule size`
defuzzy	—	1	rule numbers

methodology.

Fig. 3.3 depicts the architecture of the FL Radial Basis Function-Convolutional Neural Network (RBF-CNN), and Table 3.2 shows parameter setting of the FL RBF-CNN. Similarly, the RBF-CNN model structure could be described as 1x28x28-32C3-64C3-MP2-xN-yF-1DF, where $x \in \{16, 32, 64\}, y \in \{3, 5, \ldots, 15\}$. The difference between the RBF-CNN model and the reference model is the 'yF' layer and '1DF' layer at the very end of the model, which means a RBF layer with y rules and a De-Fuzzy layer with 1 single output. In the following section, a set of trials using variety rule size would demonstrate the affect of the rule size in RBF-CNN models.

Similar to FL RBF networks, FL RBF-CNNs will also be sensitive to initial conditions (initial model structure and parameters) of the RBF and defuzzification layers, and the initial parameters could not be determined since the features from CNN layers have not been extracted before training. Therefore, one has to establish some initial conditions for the FL rule base for successful model training.

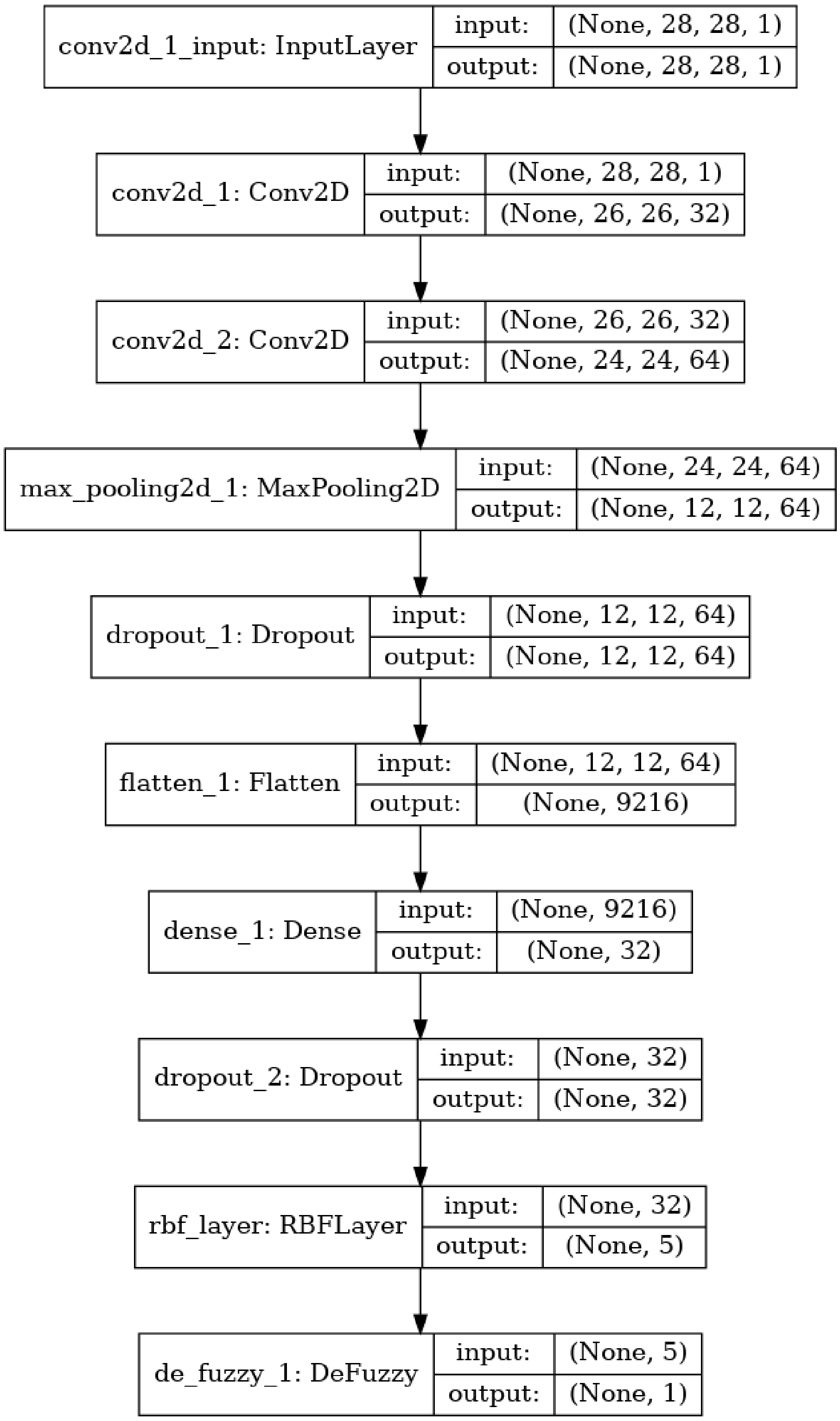


Figure 3.3: FL RBF-CNN structure

The overall training would rely on a square error loss function as the loss function and it would be performed as in the following section.

3.3 Simulation results

Simulation results were created to assess the performance of the developed deep learning structure. This is done in two parts, first the learning performance on a popular benchmark data set is assessed. This is achieved by comparing the proposed learning structure against a classical and state-of-the art CNN structure. On the second part, the robustness of the learning ability of the proposed system is assessed by reducing consecutively the sample size and evaluating the learning and recall performance.

3.3.1 MNIST dataset

The modified National Institute of Standards and Technology (MNIST) database, which was introduced in [42], was chosen as a case study; the MNIST database is a labelled handwriting digits dataset containing 60000 training images and 10000 testing images. This dataset was chosen as a benchmark dataset for variable classification methods, such as linear classifier [42], K-Nearest Neighbors (KNNs) [42, 37, 5], boosted stumps [36], SVMs [42, 16], neural networks [42, 67, 14, 18], and CNNs [42, 41, 12, 13, 15]. The highest accuracy achieved on MNIST based on linear classifiers as 92.4% [42], KNNs as 99.37% [5], boosted stumps as 99.13% [36], non-linear classifiers as 96.4% [42], SVMs as 99.44% [16], neural networks as 99.17% [18], and CNNs as 99.77% [15].

The MNIST data set has 60000-sample of training images and 10000-sample of testing images as shown in Fig. 3.4. The training images were further split into two parts randomly, as 50000-samples for training set and 10000-samples for validation (to avoid overfitting).

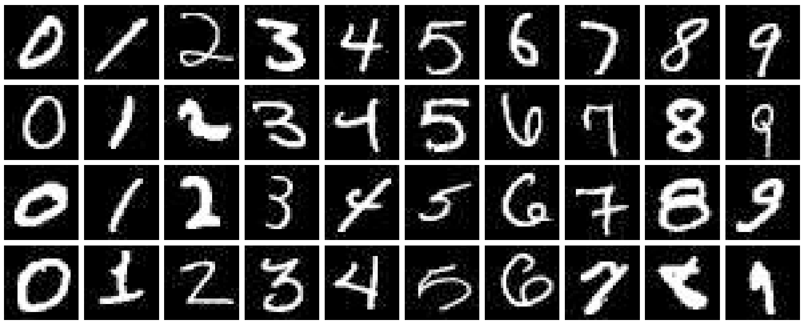


Figure 3.4: Several examples from MNIST dataset

The results were generated on a computer whose CPU was Intel i7–6700k and GPU was Nvidia GTX 1080. The computer was setted up as a `TensorFlow r1.10` and `Keras 2.1.2` platform based on `Python 3.6`.

3.3.2 MNIST training and testing simulation results: baseline CNN

The presented results include the mean classification accuracy as well as the standard deviation in each case. Each set of simulation results shows the loss function during training and validation as well as the classification accuracy for training and validation. This is presented for a number of different rules, for the rule base of the Fuzzy-Logic-based classification layer (varying from 3 — simpler — to 15 rules — more complex). The learning model makes use of Adadelta, an adaptive learning rate optimise method to optimise the model to optimise the model weights. The model is trained for 50 epochs, but also includes an early stopping criterion, to stop earlier if the validation performance is not improved, with an improvement window of 10 epochs. As shown in Fig. 3.5, the training of this network with 64 features converges within the first 30 epochs. The mean training accuracy (for 10 repeats) of this model was 99.80%, and both the validation and test accuracy of this model are at around 99% which is comparable with other state-of-the-art CNN classification structures. As an example comparison, LeNet-5 [42], which has a similar structure (1x32x32-6C5-MP2-16C5-MP5-120C5-84N-10N, containing 84 features), achieves an accuracy of 99%. A higher test classification accuracy (99.77%) is achieved in [15], however this is achieved with a more complex structure (1x29x29-20C4-MP2-40C5-MP3-150N-10N, containing 150 features). One can therefore conclude that the proposed structure does not sacrifice significant performance in this case study, despite the much simpler overall structure that aims at enhancing the interpretability of the model rather than its accuracy showing in the following section.

3.3.3 Fuzzy logic RBF model results: with variable rules

While in some cases, the interpretability of models would be the key part to understand the processing. For example in real industrial/manufacturing processes, the conditions causing faults and defects are eager for understanding.

In this section, the initial values of RBF weights were generated randomly by uniform distribution in the range of $[0, 1]$. The initial values of RBF variance were selected as 0.3 and would be clipped in the range of $[0.2^2, 0.6^2] = [0.04, 0.36]$

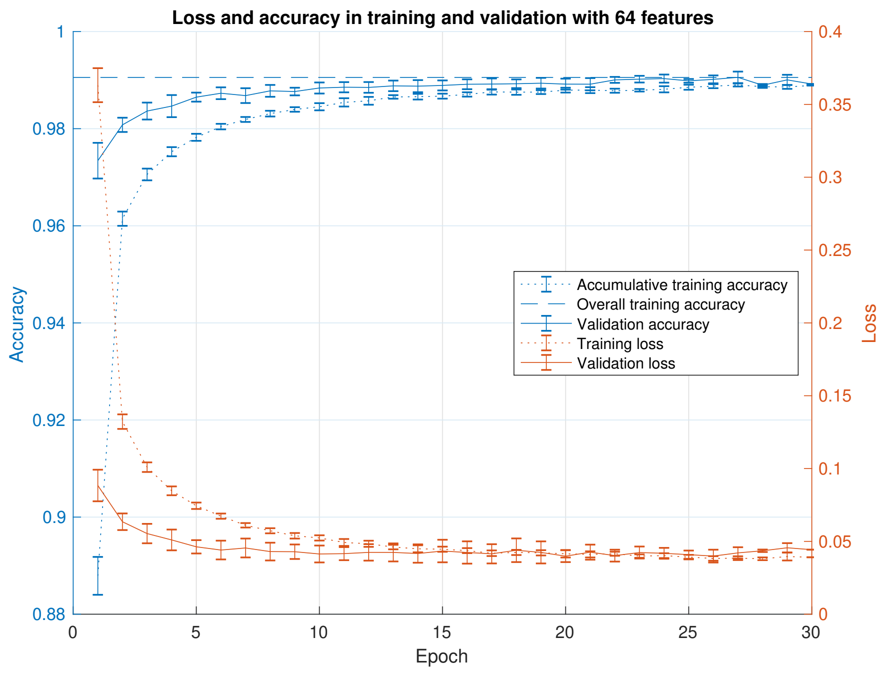


Figure 3.5: Origin CNN model with 64 features result using 10 MNIST data sets

during training.

The performance of this FL RBF-CNN is further assessed via reducing the number of classification features from 64, to 32 and to 16. The same algorithmic approach was followed, as presented in the previous section. Tables 3.3, 3.4, and 3.5 were generated with using the raw simulation results (10 repeats per training case). In each of these three tables, there are two columns whose values are average accuracy and standard variance for training, validation, and test case respectively, and every feature case were trained from 3 to 15 rules as listed in with a reference CNN network result (labelled as REF). As shown in Table 3.3, the mean accuracy has a trend that would reach the best

Table 3.3: Accuracy mean and 95% confidence intervals of the FL RBF-CNN model using 64 features

Rule	Training	Validation	Test
3	$96.64 \pm 0.033\%$	$94.79 \pm 0.035\%$	$94.67 \pm 0.032\%$
5	$98.41 \pm 0.020\%$	$96.79 \pm 0.021\%$	$96.69 \pm 0.020\%$
7	$98.48 \pm 0.030\%$	$96.89 \pm 0.030\%$	$96.92 \pm 0.031\%$
9	$97.78 \pm 0.065\%$	$96.16 \pm 0.062\%$	$96.28 \pm 0.062\%$
11	$94.14 \pm 0.179\%$	$92.55 \pm 0.176\%$	$92.63 \pm 0.174\%$
13	$95.48 \pm 0.089\%$	$94.03 \pm 0.088\%$	$93.97 \pm 0.090\%$
15	$94.90 \pm 0.102\%$	$93.43 \pm 0.101\%$	$93.44 \pm 0.099\%$
REF	$99.75 \pm 0.001\%$	$98.97 \pm 0.003\%$	$99.06 \pm 0.001\%$

Table 3.4: Accuracy means and 95% confidence intervals of the FL RBF-CNN model using 32 features

Rule	Training	Validation	Test
3	$87.17 \pm 0.220\%$	$85.54 \pm 0.217\%$	$85.76 \pm 0.217\%$
5	$94.50 \pm 0.083\%$	$92.91 \pm 0.080\%$	$93.11 \pm 0.080\%$
7	$92.19 \pm 0.103\%$	$90.81 \pm 0.098\%$	$91.02 \pm 0.099\%$
9	$91.48 \pm 0.152\%$	$90.12 \pm 0.149\%$	$90.33 \pm 0.148\%$
11	$90.90 \pm 0.106\%$	$89.58 \pm 0.106\%$	$89.67 \pm 0.102\%$
13	$86.38 \pm 0.147\%$	$85.00 \pm 0.146\%$	$85.32 \pm 0.146\%$
15	$73.84 \pm 0.676\%$	$72.72 \pm 0.665\%$	$73.01 \pm 0.665\%$
REF	$99.59 \pm 0.002\%$	$98.84 \pm 0.003\%$	$98.98 \pm 0.002\%$

performance when fuzzy rules equals to 5 or 7, and the standard deviation also shows a similar trend. However, to a certain extent, in spite of the good performance, a model having 64 features may not be very interpretable, hence models with 32 and 16 features were also simulated to 'stress-test' the performance of the proposed structure. When the size of the classification features decreases, the neurons of the last fully connected layers also gets reduced. It is expected to observe a reduced classification power due to the fewer model parameters

Table 3.5: Accuracy means and 95% confidence intervals of the FL RBF-CNN model using 16 features

Rule	Training	Validation	Test
3	$66.61 \pm 0.175\%$	$65.35 \pm 0.170\%$	$65.56 \pm 0.174\%$
5	$62.93 \pm 0.261\%$	$62.03 \pm 0.258\%$	$62.32 \pm 0.257\%$
7	$68.74 \pm 0.186\%$	$67.53 \pm 0.178\%$	$67.90 \pm 0.179\%$
9	$65.48 \pm 0.196\%$	$64.61 \pm 0.194\%$	$65.04 \pm 0.187\%$
11	$60.60 \pm 0.121\%$	$59.88 \pm 0.120\%$	$59.88 \pm 0.116\%$
13	$59.98 \pm 0.382\%$	$59.06 \pm 0.375\%$	$59.51 \pm 0.377\%$
15	$56.49 \pm 0.496\%$	$55.86 \pm 0.489\%$	$56.08 \pm 0.491\%$
REF	$99.27 \pm 0.002\%$	$98.56 \pm 0.003\%$	$98.66 \pm 0.002\%$

available to capture the classification problem. In general, the classification accuracy is reduced, as demonstrated in Table 3.4 and Table 3.5. Similarly, as in the case with 64 features, the best performance is observed between 5 and 7 rules. In the case of 32 features, the test accuracy of 93.11% could be considered as acceptable, however the test accuracy of 67.90% in the case with 16 features demonstrates that there is a significant performance loss when the number of features is very low.

It can be observed that the FL RBF-CNN is more robust to less training data, as in the training performance for up to 200 samples is outperforms the classic CNN structure. It is however observed that both versions of the network deteriorate significantly with less training samples, only achieving 70% to 80% accuracy which is far from the nearly perfect accuracy of $> 99\%$ of the networks trained with the full dataset. Similar observation can be concluded from the training loss of each network. In terms of each model's testing performance (recall) similar conclusion can be reached, however the differences between the two structures are less prominent. In the testing simulation (on unseen data), the drop-off in performance is observed at an earlier stage, when the equivalent data samples for training become less than 500. In other words, the networks cannot generalise well on unseen data when the training samples are 500 or less, where the classification accuracy drops significantly.

3.3.4 Model interpretability

With the fully connected layer of the CNN structure being a Fuzzy Logic based layer, one can enhance the interpretability of the classification task, by extracting Fuzzy Logic rules directly from the classification layer. Such information can be, for example, further used to aid decision making, or to create human-machine interfaces. Fig. 3.6, as an example, depicts two different rules from the rule base of the 32-feature FL RBF-CNN model; just four inputs (features) and one output (classification weight) are shown for simplicity. Rule 1 for example, translates into the following Singleton-based Fuzzy rule:

'IF Feature 1 is A1, and Feature 2 is B1, and Feature 3 is C1, and . . . etc, THEN the Output class is O1.'

There is no standard measure to assess how good interpretability is [22]. In this situation, the interpretability is measured by the number of features and the number of fuzzy rules utilised. Considering the accuracy, a 32-feature 5-rule RBF model was chosen.

In the example of Fig. 3.6, there were 32 features for each rule. For an image processed by the CNN part, which could be called as feature extractor, a 32-by-1 vector would be calculated as the feature vector $\vec{x}$. This feature vector $\vec{x}$ would be used as the input of the RBF layer. The output of fuzzy rule

$O1 = \mathrm{A1}(x_1) + \mathrm{B1}(x_2) + \ldots$. Because there were five rules, we would have a fuzzy output vector $\vec{O} \in \mathbb{R}^5$ where $O_1 = O1$.

The defuzzy layer would calculate the class label with input as the fuzzy output vector $\vec{O}$. Using the output vector $\vec{O}$ as the input of Eq. (3.5) and Eq. (3.6), we would get the prediction $\hat{y}$ of the given image.

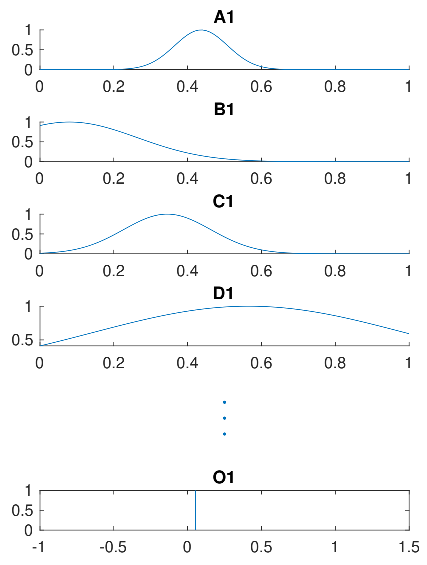
As the feature vector $\vec{x}$ is the output of CNN, the sequence of these features would be sorted randomly after every training process, and hence there is no fix linguistic meaning of each feature. In order to illustrate the meanings, a feature trackback methodology would be introduced in the next chapter.

3.4 Conclusion

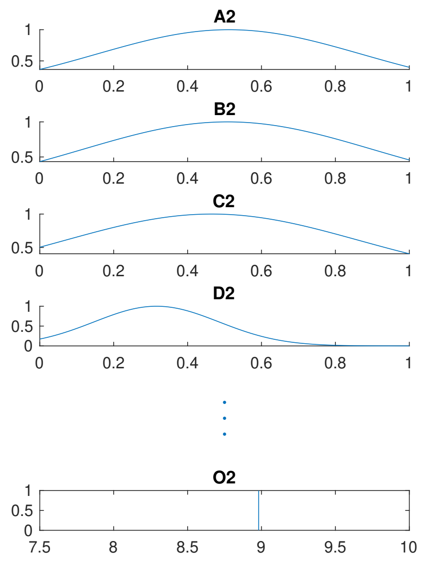
In this research work, an interpretability-oriented deep learning network is presented, based on a CNN structure combined with a Fuzzy Logic structure to perform the classification task and also provide the capability to linguistically interpret the structure's rule base. By combining the good feature extraction property of CNNs and the classification and generalisation ability of FL based systems, a FL RBF-CNNs was developed. The proposed structure relies on a Radial Basis Function realisation of the Neural-Fuzzy network, which is integrated into the CNN structure via an adaptive subgradient method for the credit assignment and error backpropagation.

In simulation results (training, validation and testing/recall) using a popular dataset often used for benchmarking (MNIST 70000 handwriting digit samples) it is shown that the proposed network performs equally well when compared to state-of-the-art CNN-based networks of similar complexity and size. However, the advantage of the proposed structure, is that due to the added classification layer in the form of a FL rule base, one could extract linguistic FL statements for the overall model, which would enhance the interpretability of the system. For example, in decision making applications, one could extract autonomously linguistic rules to assist a human user/operator. To further extend this research work, it would be interesting to capture and visualise the connection between features and predictions via FL RBF-CNN layers.

The work in this chapter has already be published as [91].



(a) Rule 1



(b) Rule 2

Figure 3.6: Two of five rule bases of a FL RBF-CNN model with 32 features

4 Interpretable Convolutional Neural Networks using a rule-based framework for classification

In previous chapter, a neural network model combined CNN and RBF was proposed, and it was claimed that the proposed model has the ability to extract linguistic rules for a specific classification action. In this chapter, via simulation results based on another popular benchmark problem/dataset, it was shown again that the proposed network structure performs as well as CNN-based structures, hence there is a measurable but not significant loss of performance by introducing the FL layer as part of the deep learning structure. Besides, more analyses of linguistic interpretability based on fuzzy entropy are also contained in this chapter.

4.1 Model structure

4.1.1 RBF-CNN structure

The model applied in this chapter was the same as the one described in Section 3.2 except the hyper-parameter settings. In Chapter 3, the Gaussian membership width was clipped in the range $[0.2^2, 0.6^2]$, while the width is clipped in the range $[0.2, 0.6]$ in this chapter. The reason of this mismatch is explained in Appendix 8.1.

This model could be mainly separated into two parts, which are CNN layers as a feature extractor and FL layers as a classifier. As mentioned in Table 4.1, the first 7 layers of this FL RBF-CNN model are identical to the reference CNN model, while the last two layers were changed to RBF layer and defuzzification layer.

4.1.2 A framework for linking FL rules to CNN features

The idea for this interpretable methodology is trying to find out which features would affect the predictions mostly and trying to demonstrate how these features affect the predictions. Because of the multiple fuzzy rules existing in a RBF-CNN model, the importanace of each rule could be estimated using a Fuzzy Logic entropy measure. After establishing the link between rules and the input features, which should be a high-dimention vector and would be hard to read out directly, a set of heatmaps indicating the importance of input features corresponding to one set of rules.

For any RBF-CNN model whose feature extraction part has a output form as

Table 4.1: FL RBF-CNN architecture

role	**type**	**patch size/stride**	**output size**	**parameters**
Extractor	convolution	$3 \times 3/0$	$26 \times 26 \times 32$	320
	convolution	$3 \times 3/0$	$24 \times 24 \times 64$	18496
	maxpooling	$2 \times 2/0$	$12 \times 12 \times 64$	0
	dropout (25%)	—	$12 \times 12 \times 64$	0
	flatten	—	9216	0
	linear	—	`feature size`	9216×`feature size`
	dropout (50%)	—	`feature size`	0
Classifier	RBF	—	`rule size`	2×`rule size`
	defuzzy	—	1	`rule size`

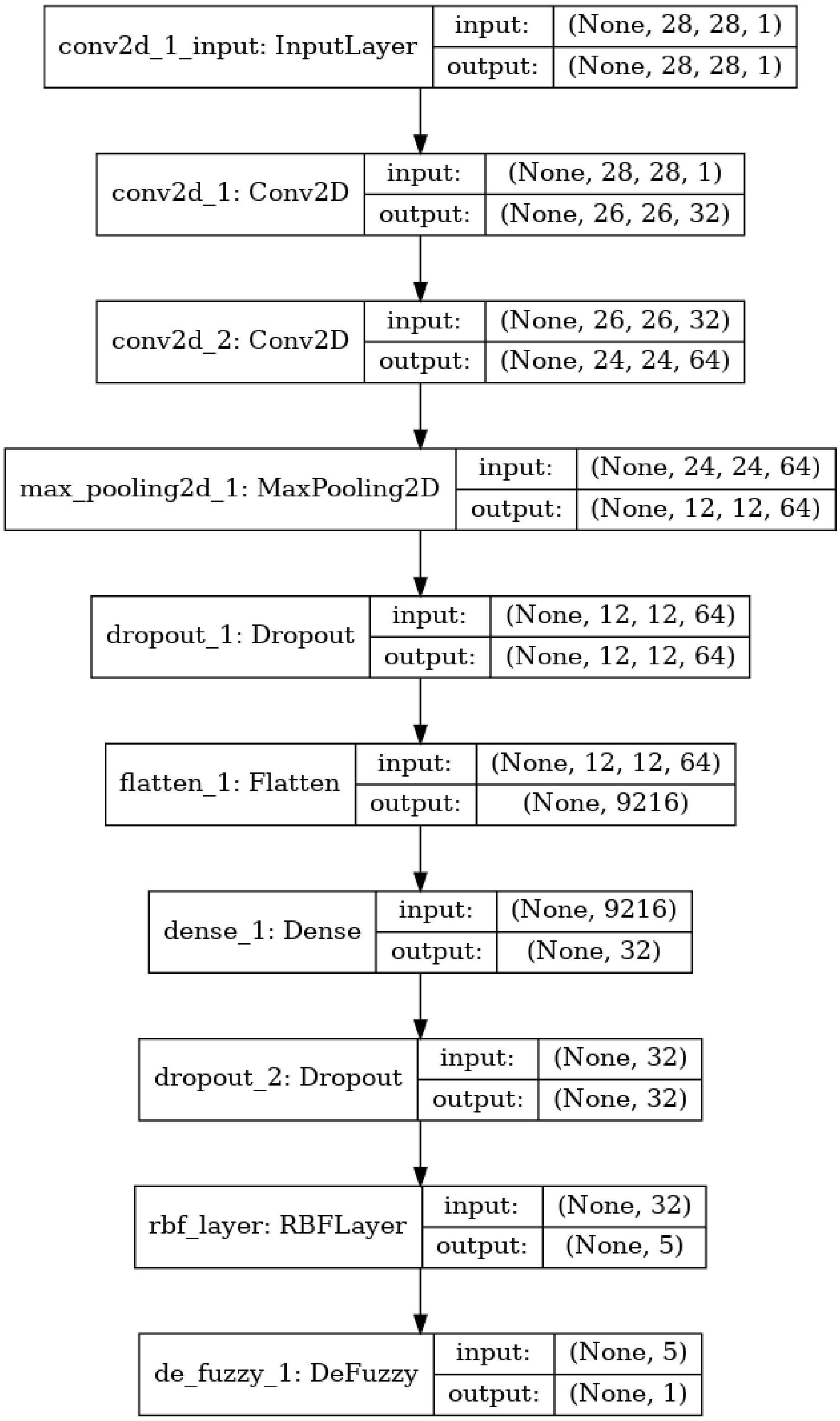


Figure 4.1: FL RBF-CNN structure

a vector, we could have the following pseudo code to describe this methodology.

```
Input: image, model
Output: trackback image
Get trained model feature layer weight $W$;
Get trained model last tensor layer dimension $D$;
for each input image do
    Calculate prediction;
    Sort rules;
    for each rule do
        Calculate estimated feature $x'$ with least square estimation using
          $W$ as the input;
        Reshape $x'$ into the shape $D$;
        Calculate a trackback maxpooling layer result $M_{trackback} = M \circ a$;
        Plot $M_{trackback}$ heatmap for the rule as the output;
    end
end
```

Algorithm 1: Trackback algorithm for RBF-CNN models

The input space for the rule-based structure, in our case the fully connected RBF layer, is a flat vector of weights, as shown in the CNN structure depicted in Fig. 4.1. To enhance the interpretability of the antecedent part of the 'IF ... THEN ... ' Fuzzy Logic rules we propose to 'track back' the weights of the flat input vector of the fully connected layer towards revealing the relevant features of the input image. Effectively, we propose to associate via this mechanism relevant rules to features in the image's feature space, so that the user can appreciate which rules are responsible for each classification decision, and what is the relevant input space for each rule in the feature space. This is achieved as follows:

For any CNN model, as defined in the structure shown in Table 4.1, the final layer acts linearly [80]. Therefore, to track back the weights of the input space of the fully connected layer, one can follow the process:

- Calculate a mask layer using a least mean square solution. The input is a vector of selected features, and the output is a vector of 9216 points.
- Reshape the 9216-point vector as a tensor a with dimensions $12-12-64$.
- Use tensor a as the mask. Calculate a track-back maxpooling layer result using $M_{trackback} = M \circ a$.
- Reveal feature 'heat maps' using mask $M_{trackback}$.

This procedures could be applied on any CNN model of the proposed structure, regardless of having a FL-based RBF layer or not. The advantage of using the FL-based RBF layer is that one can now link linguistic rules to the input feature space using the above described process.

As the FL-RBF layer consists of multiple fuzzy rules, the relative importance of each rule could be estimated using a Fuzzy Logic entropy measure. In the proposed framework, a non probabilistic entropy function could be used [73], this is shown below.

$$H = -K \sum_{i=1}^{n} \left(\mu_i \log\left(\mu_i\right) + \left(1-\mu_i\right)\log\left(1-\mu_i\right)\right), \tag{4.1}$$

where K is a positive constant (usually equal to $1/N$ for normalisation), and μ_i is i-th membership degree.

Using Fuzzy Entropy to identify the most 'active' rules for a given prediction, and by identifying the Membership Functions of each rule with the highest relevance to the input vector (membership degree), then a framework can be established to directly link rules, to input space features; this is demonstrated in the Simulation Results is Section 4.

4.2 Simulation Results

In this section, the proposed RBF-CNN modelling framework is tested against two popular benchmark datasets to assess its learning and recall performance, as well as demonstrate the developed linguistic interpretability. In both benchmark case studies (MNIST character recognition, and MNIST Fashion), Adadelta [94] was used for optimisation, with an adaptive learning rate during training. Early stopping was used to avoid overfitting.

The results were generated on a computer whose CPU was Intel i7–6700k and GPU was Nvidia GTX 1080. The computer was setted up as a `TensorFlow r1.13.1` and `Keras 2.2.4` platform based on `Python 3.7`.

4.2.1 Case study: MNIST

The dataset applied in this section is introduced in previous chapter. Although the same dataset utilised, the results were improved slightly because of the change of hyper-parameters in the defuzzification layer.

The MNIST data set has 60000-sample of training images and 10000-sample of testing images. The training images were further split into two parts randomly, as 50000-samples for training set and 10000-samples for validation (to avoid overfitting).

Modelling performance

Simulation results were created to assess the performance of the developed deep learning structure. This is done in two parts. First the learning performance on a popular benchmark data set is assessed. This is achieved by comparing the proposed learning structure against a classical and state-of-the art CNN structure. On the second part, the robustness of the learning ability of the proposed system is assessed by reducing consecutively the number of features and evaluating the learning and recall performance.

The presented results include the mean classification accuracy as well as the standard deviation in each case. Each set of simulation results shows the loss function during training and validation as well as the classification accuracy for training and validation. This is presented for a number of rules, for the rule base of the Fuzzy-Logic-based classification layer (varying from 3 rules — simpler — to 15 rules — more complex). The learning model makes use of an adaptive learning rate method to optimise the model weights. The model is trained for 50 epochs, but also includes an early stopping criterion, to stop earlier if the validation performance is not improved, with an improvement window of 15 epochs. After the stop, the model weights which resulted the smallest validation loss would be stored for the following process. As shown in Fig. 4.2, the training of this network with 64 features converges within the first 30 epochs. The mean training accuracy (for 10 repeats) of this model was 99.16%, and both the validation and test accuracy of this model are at around 97.5% which is comparable with other state-of-the-art CNN classification structures. As an example comparison, LeNet-5 [42], which has a similar structure, achieves an accuracy of 99%. A higher test classification accuracy (99.77%) is achieved in [15], however this is achieved with a significantly more complex structure. One can therefore conclude that the proposed structure does not sacrifice significant performance in this case study, despite the much simpler overall structure that aims at enhancing the interpretability of the model rather than its accuracy.

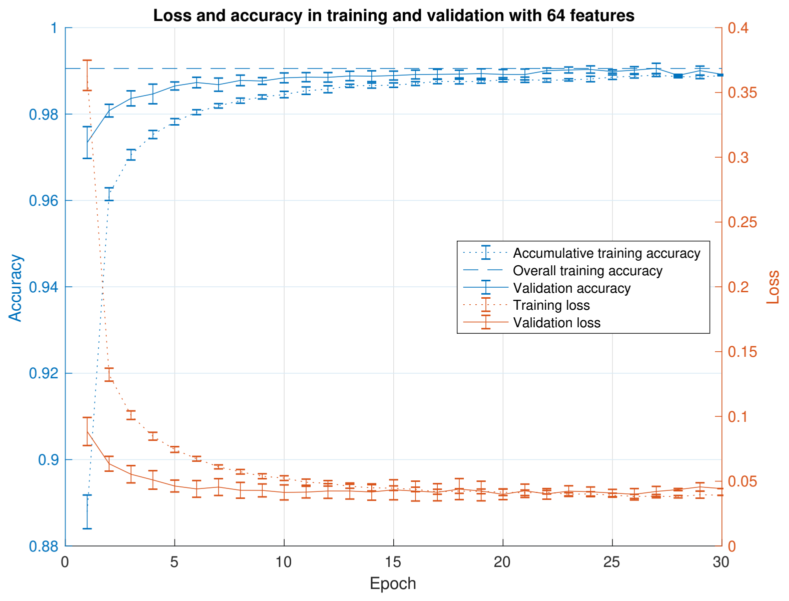


Figure 4.2: CNN model with 64 features, training and validation performance (average of 10 simulations)

The performance of this FL RBF-CNN is further assessed via reducing the number of classification features from 64, to 32 and finally to 16. The same algorithmic approach was followed, as presented in the training of the basement model. Tables 4.2, 4.3, and 4.4 were generated with using the raw simulation results (10 repeats per training case). In each of these three tables, there are two columns whose values are average accuracy and standard variance for training, validation, and test case respectively, and every feature case were trained from 3 to 15 rules as listed in with a reference CNN network result (labelled as REF). As shown in Table 4.2, the mean accuracy has a trend that would reach the best performance when the number of Fuzzy Logic rules equals to 7. However, to a certain extent, despite of the good performance, a model having 64 features may not be very interpretable, hence models with 32 and 16 features were also simulated to 'stress-test' the performance of the proposed structure. When the size of the classification features decreases, the neurons of the last fully connected layers also gets reduced. It is expected to observe a reduced classification power due to the fewer model parameters available to capture the classification problem. In general, the classification accuracy is reduced, as demonstrated in Table 4.3 and Table 4.4. In the case of 32 features, the test accuracy that reached with 13 rules of 97.12% could be considered as acceptable, however the test accuracy of 78.57% in the case with 16 features using 7 rules demonstrates that there is a significant performance loss when the number of features is significantly lower.

Table 4.2: Accuracy mean and 95% confidence intervals of the model using 64 features and MNIST characters data

Rule	Training	Validation	Test
3	$98.80 \pm 0.179\%$	$97.23 \pm 0.193\%$	$97.01 \pm 0.122\%$
5	$98.83 \pm 0.429\%$	$97.32 \pm 0.401\%$	$97.09 \pm 0.379\%$
7	$99.16 \pm 0.129\%$	$97.80 \pm 0.193\%$	$97.52 \pm 0.143\%$
9	$99.05 \pm 0.193\%$	$97.79 \pm 0.243\%$	$97.52 \pm 0.193\%$
11	$98.93 \pm 0.329\%$	$97.63 \pm 0.286\%$	$97.30 \pm 0.279\%$
13	$98.07 \pm 2.075\%$	$96.82 \pm 2.125\%$	$96.56 \pm 2.046\%$
15	$97.89 \pm 1.731\%$	$96.81 \pm 1.745\%$	$96.37 \pm 1.717\%$
REF	$99.74 \pm 0.043\%$	$99.03 \pm 0.043\%$	$99.05 \pm 0.050\%$

Table 4.3: Accuracy mean and 95% confidence intervals of the model using 32 features and MNIST characters datasets

Rule	Training	Validation	Test
3	$91.09 \pm 0.080\%$	$89.91 \pm 0.073\%$	$89.65 \pm 0.077\%$
5	$95.60 \pm 0.065\%$	$94.38 \pm 0.059\%$	$94.25 \pm 0.061\%$
7	$95.12 \pm 0.083\%$	$94.11 \pm 0.074\%$	$93.89 \pm 0.075\%$
9	$94.40 \pm 0.078\%$	$93.46 \pm 0.074\%$	$93.19 \pm 0.075\%$
11	$96.00 \pm 0.068\%$	$94.73 \pm 0.067\%$	$94.54 \pm 0.066\%$
13	$97.12 \pm 0.051\%$	$95.88 \pm 0.045\%$	$95.77 \pm 0.046\%$
15	$95.19 \pm 0.072\%$	$94.31 \pm 0.066\%$	$93.92 \pm 0.067\%$
REF	$99.54 \pm 0.001\%$	$98.84 \pm 0.002\%$	$98.88 \pm 0.001\%$

Table 4.4: Accuracy mean and 95% confidence intervals of the model using 16 features and MNIST characters datasets

Rule	Training	Validation	Test
3	$72.58 \pm 0.176\%$	$72.27 \pm 0.165\%$	$71.97 \pm 0.175\%$
5	$74.22 \pm 0.229\%$	$73.93 \pm 0.206\%$	$73.58 \pm 0.227\%$
7	$79.21 \pm 0.167\%$	$78.78 \pm 0.148\%$	$78.57 \pm 0.157\%$
9	$77.22 \pm 0.176\%$	$76.90 \pm 0.161\%$	$76.70 \pm 0.171\%$
11	$72.56 \pm 0.145\%$	$72.64 \pm 0.123\%$	$71.96 \pm 0.144\%$
13	$76.74 \pm 0.160\%$	$76.38 \pm 0.146\%$	$76.17 \pm 0.150\%$
15	$75.85 \pm 0.134\%$	$75.68 \pm 0.115\%$	$75.27 \pm 0.131\%$
REF	$98.94 \pm 0.002\%$	$98.40 \pm 0.003\%$	$98.37 \pm 0.002\%$

FL-based interpretability: MNIST characters

In some application areas, the interpretability of models could be key to understanding the underlying processes. For example in complex manufacturing processes, when trying to understand the conditions causing faults and defects.

With the fully connected layer of the proposed CNN structure being a Fuzzy Logic based layer, one can enhance the interpretability of the classification task, by extracting Fuzzy Logic linguistic rules directly from the classification layer. Such information can be, for example, further used to aid decision making, or to assist the creation of human-machine interfaces. Fig. 4.3, as an example, depicts two different rules from the rule base of the 32-feature FL RBF-CNN model; just four inputs (features) and one output (classification weight) are shown for simplicity. Rule 1 for example, translates into the following Singleton-based Fuzzy rule:

'IF Feature 1 is A1, and Feature 2 is B1, and Feature 3 is C1, and..etc. THEN the Output class is O1.' (4.2)

During feature extraction and classification, a trained CNN structure would entail a set of image-like matrices, which could be visualised using the method described in Section 1. Fig. 4.4 demonstrates such images for an input digit '0' as in Fig. 4.4a. Fig. 4.4b is extracted from the first CNN layer, which outlines the outer round feature of '0'. Furthermore, Fig. 4.4c depicts more abstract features, including the outer edges and inner edges, and Fig. 4.4d includes

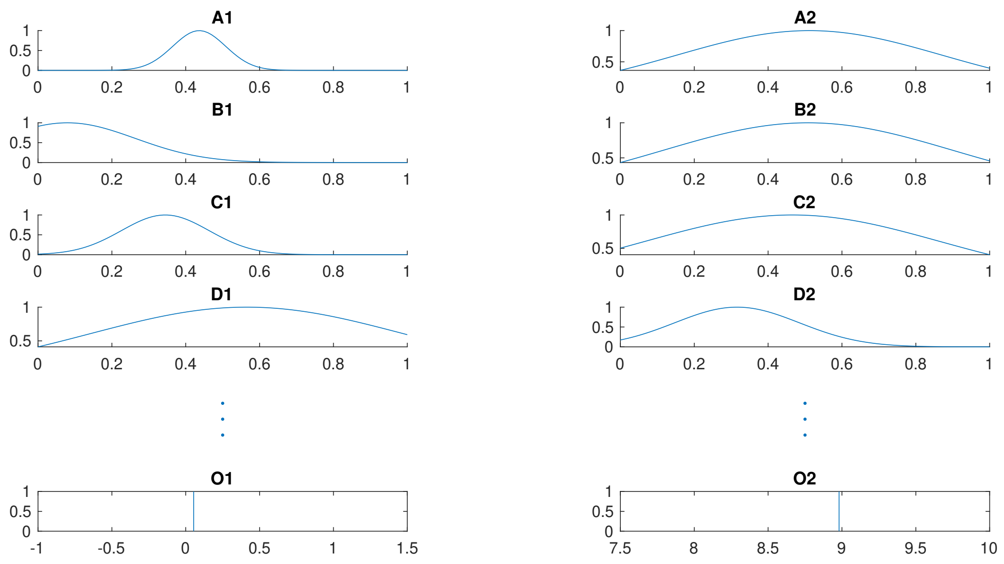
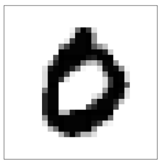


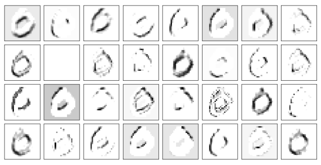
Figure 4.3: Example of two FL rules, of the FL RBF-CNN model with 32 features

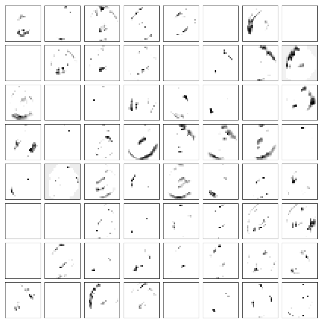
similar features as Fig. 4.4c albeit with a lower definition.

Using the methodology outlined in Section 3.2, feature maps linked to specific Fuzzy Logic rules can be obtained.

Fig. 4.5 demonstrated three fuzzy rules corresponding to cases in Fig. 4.6 in the same sequence. Fig. 4.6 depicts three feature maps corresponding to relevant FL rules which were identified by fuzzy entropy. For a given input vector, using the linguistic FL rules, and the corresponding image-based features it is possible to appreciate why a particular class has been predicted. This may be trivial for the character recognition case study, however it can be extremely important when investigating problems in manufacturing, biomedical systems etc. when trying to understand the feature space for a particular prediction.


(a) The input image for the prediction


(b) The output of the first CNN layer


(c) The output of the second CNN layer

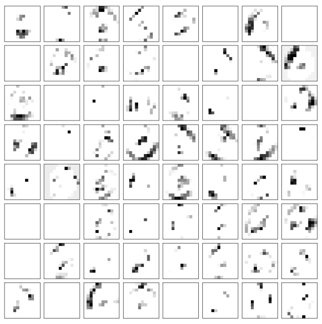

(d) The output of the maxpooling layer

Figure 4.4: Features extracted by FL RBF-CNN during prediction for a sample in MNIST characters dataset

4.2.2 Case study: Fashion MNIST

The Fashion-MNIST dataset was introduced to replace MNIST as a new benchmark dataset [92] for machine learning. The Fashion-MNIST dataset contains 60,000 training images and 10,000 testing images, which includes 28-by-28 greyscale images labelled into 10 classes.

For comparative analysis purposes, the training regime in this case study is set to be the same as the one applied in the MNIST character recognition case, i.e. a 10,000-sample validation set is selected randomly from the training dataset and used to avoid overfitting.

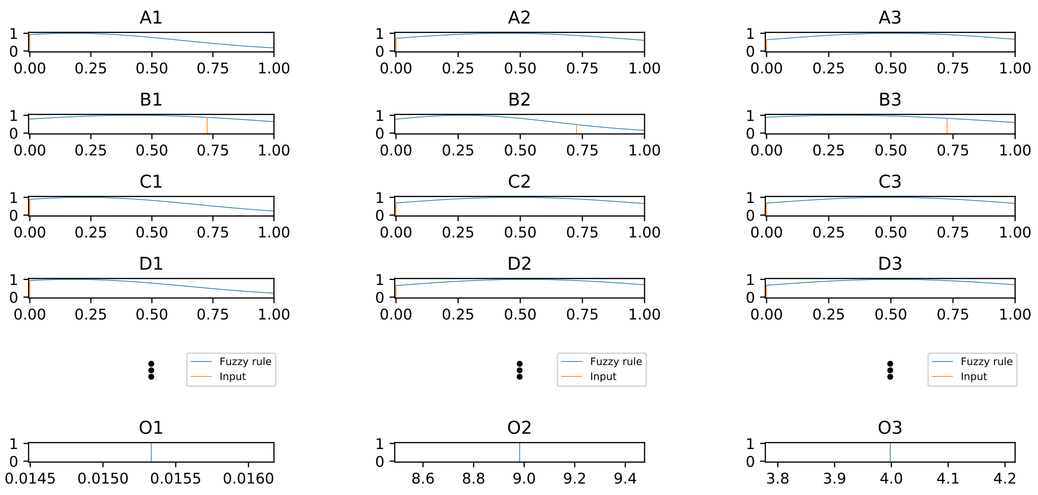
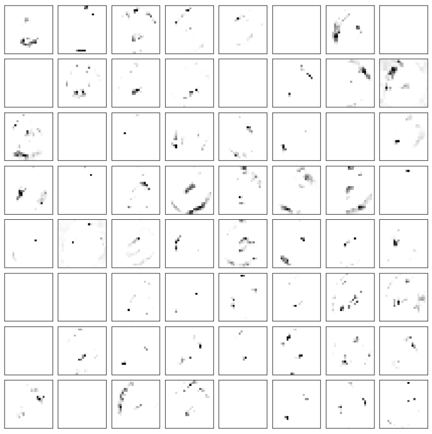
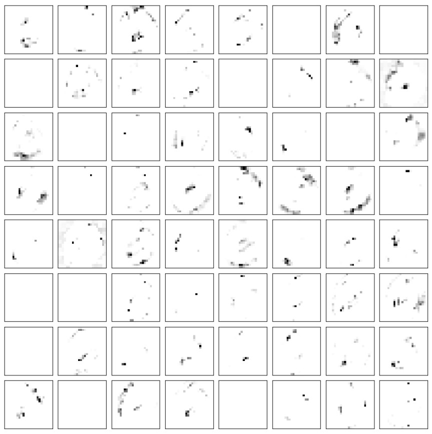


Figure 4.5: Three fuzzy rules, of the FL RBF-CNN model with 32 features for the MNIST digit

Modelling performance

An identical performance assessment is used, as in Section 4.1. Tables 4.5, 4.6, and 4.7 were also generated with 10 times simulations. Similar to the MNIST case, the FL RBF-CNN model would achieve best performance when 5 to 7 rules are used. As shown in Table 4.5, the mean test accuracy fluctuated around 84.0% since 3 rules to 13 rules. In the case of 32 features (Table 4.6) and 16 features (Table 4.7), the test accuracy that reached with 5 rules of 80.03% could be considered as acceptable for a model without specific tuning, however the test accuracy of 62.69% in the case with 16 features using 13 rules demonstrates that there is a significant performance loss when the number of features is significantly lower.


(a) Features extracted by rule one


(b) Features extracted by rule two

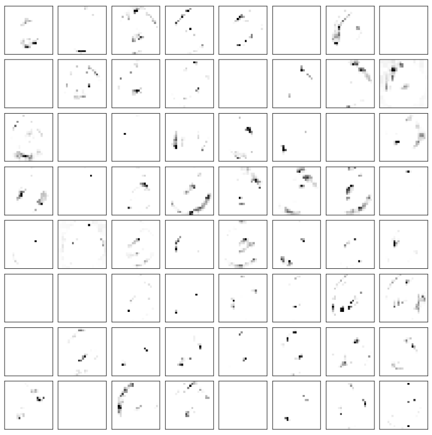

(c) Features extracted by rule three

Figure 4.6: Example of three 'heat maps' sorted by fuzzy entropy of the MNIST digit

Table 4.5: Accuracy mean and 95% confidence intervals of the model using 64 features and fashion MNIST Fashion datasets

Rule	Training	Validation	Test
3	$90.71 \pm 0.033\%$	$84.84 \pm 0.022\%$	$83.52 \pm 0.023\%$
5	$89.04 \pm 0.042\%$	$85.58 \pm 0.022\%$	$83.41 \pm 0.029\%$
7	$90.55 \pm 0.034\%$	$85.88 \pm 0.029\%$	$83.64 \pm 0.029\%$
9	$91.14 \pm 0.031\%$	$86.31 \pm 0.012\%$	$84.47 \pm 0.020\%$
11	$90.55 \pm 0.089\%$	$86.02 \pm 0.050\%$	$83.77 \pm 0.062\%$
13	$92.05 \pm 0.038\%$	$86.60 \pm 0.014\%$	$84.70 \pm 0.022\%$
15	$85.84 \pm 0.129\%$	$83.02 \pm 0.090\%$	$80.67 \pm 0.097\%$
REF	$97.55 \pm 0.006\%$	$92.88 \pm 0.005\%$	$92.34 \pm 0.003\%$

Table 4.6: Accuracy mean and 95% confidence intervals of the model using 32 features and fashion MNIST Fashion datasets

Rule	Training	Validation	Test
3	$82.39 \pm 0.168\%$	$79.57 \pm 0.159\%$	$77.18 \pm 0.155\%$
5	$85.76 \pm 0.077\%$	$81.55 \pm 0.057\%$	$80.03 \pm 0.057\%$
7	$84.78 \pm 0.084\%$	$81.98 \pm 0.049\%$	$79.76 \pm 0.066\%$
9	$83.59 \pm 0.104\%$	$80.75 \pm 0.080\%$	$78.83 \pm 0.081\%$
11	$82.56 \pm 0.101\%$	$79.75 \pm 0.079\%$	$77.87 \pm 0.075\%$
13	$79.73 \pm 0.116\%$	$77.98 \pm 0.085\%$	$75.62 \pm 0.096\%$
15	$81.99 \pm 0.107\%$	$79.20 \pm 0.078\%$	$77.49 \pm 0.082\%$
REF	$95.83 \pm 0.004\%$	$92.44 \pm 0.007\%$	$91.77 \pm 0.002\%$

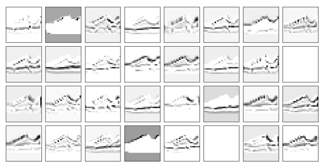
Table 4.7: Accuracy mean and 95% confidence intervals of the model using 16 features and fashion MNIST Fashion datasets

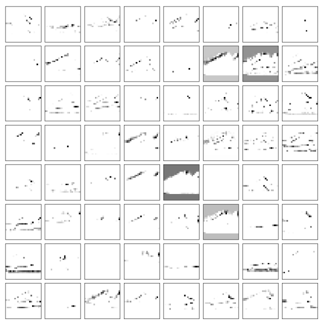
Rule	Training	Validation	Test
3	$61.42 \pm 0.084\%$	$59.81 \pm 0.085\%$	$58.65 \pm 0.086\%$
5	$61.95 \pm 0.092\%$	$60.57 \pm 0.081\%$	$59.08 \pm 0.083\%$
7	$63.01 \pm 0.111\%$	$61.58 \pm 0.105\%$	$60.64 \pm 0.100\%$
9	$64.30 \pm 0.096\%$	$62.63 \pm 0.084\%$	$61.84 \pm 0.084\%$
11	$63.12 \pm 0.095\%$	$62.27 \pm 0.089\%$	$60.82 \pm 0.089\%$
13	$65.19 \pm 0.059\%$	$63.99 \pm 0.048\%$	$62.69 \pm 0.054\%$
15	$60.68 \pm 0.126\%$	$59.63 \pm 0.113\%$	$58.86 \pm 0.118\%$
REF	$93.96 \pm 0.007\%$	$91.26 \pm 0.007\%$	$90.73 \pm 0.005\%$

FL-based interpretability: Fashion MNIST

A sneaker shoe is used here as an example. Fig. 4.7 demonstrates the relevant features obtained within this particular prediction. Fig. 4.7b shows the outline shape of the shoe, and Fig. 4.7c and Fig. 4.7d demonstrates further abstract features, linked to the relevant FL rules.

(a) The input image for the prediction


(b) The output of the first CNN layer


(c) The output of the second CNN layer

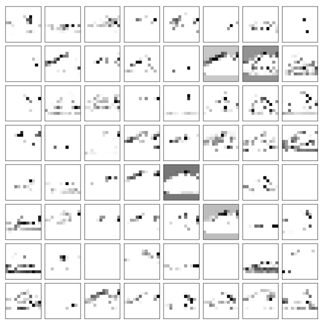

(d) The output of the maxpooling layer

Figure 4.7: Features extracted by FL RBF-CNN during prediction for a sample in Fashion MNIST

Similar to the case in the MNIST benchmark, Fig. 4.8 shows three fuzzy rules could be used to track back to feature maps, and Fig. 4.9 contains three feature maps corresponding to tracked feature maps with three fuzzy rules respectively.

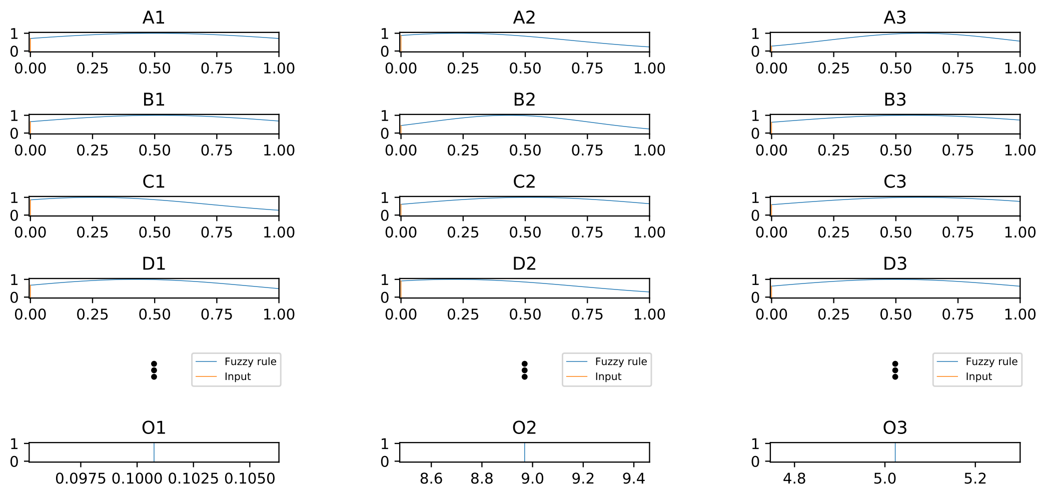
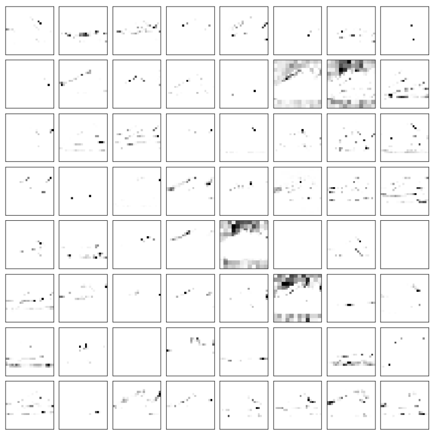
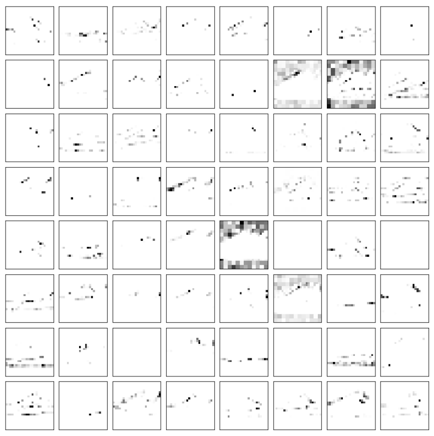


Figure 4.8: Three fuzzy rules, of the FL RBF-CNN model with 32 features for the Fashion MNIST image

4.3 Conclusion

In this research work, an interpretability-oriented deep learning network is presented, based on a CNN structure combined with a Fuzzy Logic structure to perform the classification task and also provide the capability to linguistically interpret the structure's rule base. By combining the feature extraction property of CNNs and the classification and interpretability ability of FL based systems, an FL RBF-CNNs was developed. The proposed structure relies on a Radial Basis Function realisation of the Neural-Fuzzy network, which is integrated into the CNN structure via an adaptive subgradient method for the credit assignment and error backpropagation. A systematic algorithmic process is also developed to assign features to specific FL linguistic rules, and identify such rules using an entropy function. The combination of the new modelling structure, with the rule identification and linking to the input feature space, yields a


(a) Features extracted by rule one


(b) Features extracted by rule two

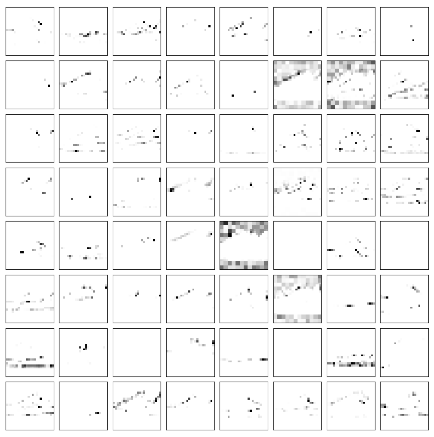

(c) Features extracted by rule three

Figure 4.9: Example of three 'heat maps' sorted by fuzzy entropy of the Fashion MNIST image

methodology that can be used to provide linguistic interpretability to a deep learning structure. We demonstrate, via two case studies (MNIST characters, and MNIST fashion) that there is no significant predictive performance loss, given enough features are used, and the rules to features maps can be used to provide interpretability to a given classification.

This chapter was already published as the extension book chapter [90].

5 A mathematical structuring legally implemented RBF-CNN model

This chapter introduces a new kind of model structure combining CNNs and RBFs networks. The model structure providing in this chapter is an improvement form of the model structure in Chapter 3 and Chapter 4. The structure proposed in this chapter is a multiple-input-multiple-output (MIMO) system rather than multiple-input-single-output (MISO) system in previous chapters, which may benefit classification tasks by providing the probabilities for multiple classes. The MNIST dataset and F-MNIST dataset were selected as the benchmark dataset because of their popularity and simplicity.

5.1 Introduction

In Chapter 3 and 4, a methodology concluding image classification and rule-base linguistic interpretable analysis was explained and tested on two popular datasets. However, the provided method was based on a mathematical compromise which will limit the application cases of this model. In this chapter, an improved mathematical model is provided.

5.2 The models weaknesses in the previous chapters

In Chapter 3 and 4, a method combining CNNs and RBFs is provided. Based on this method, several models were trained and achieved an accuracy over 95% on dataset MNIST. However, this method has two limitations caused by its mathematical model.

The first limitation is a model based on RBF-CNN method can only output a single value as its final output value. For classification neural network models, the output results are usually formed in vector forms. With vector outputs, the application of cross-entropy loss is possible. Cross-entropy loss is a common loss function because of its better performance comparing with other loss functions such as squared error [52], and it is possible to output multiple prediction probability of labels with cross-entropy. With vector outputs, cross-entropy loss among several loss functions provided by packages, says `PyTorch` or `TensorFlow`, could be applied on directly. These open-source packages could reduce possibility of implementations containing mistakes.

The second limitation is the output value cannot match to dataset labels directly. A common classification neural network would have a vector output indicating the predication probability for each class, while an RBF network can only output a single value. In order to label the output class using the method described in Chapter 3 and 4, a round up function is necessary to convert a float number into integer, which is the labels' form. However, a round up function is not a first-order smooth function which would break the backpropagation of losses during training neural networks. As a compromise, the round up function was only applied in the test phase of models.

5.3 Methodology

5.3.1 Proposed model's structure

Because of the weaknesses discussed in Section 5.2, an updated model structure is necessary. This new structure should resolve these two disadvantages. It could be found that the two disadvantages listed in previous section are all introduced by the Defuzzy layer. Therefore, one solution would be removing the Defuzzy layer from the model developed in Chapter 3 and 4.

In order to benchmark the performance of the new proposed model, a reference model was first created. Fig. 5.1 depicts the overall structure of a CNN network. This model was designed to use 28×28 pixel grey-scale images as input. After two convolutional layers, a max pooling layer was added. The dropout layers were applied to avoid overfitting. The Flatten layer was added to convert data structure into vectors, and two Dense layers are fully connected layers. All activation functions in this model were ReLUs. The loss function of this model was cross entropy loss function, which is widely used in CNNs [40, 83]. In the proposed research work, the Adam [39] optimiser was applied to perform the learning task, to take advantage of its automatically tuning variable learning rate. The loss function was set as cross-entropy function. Table 5.1 shows the architecture of the designed CNN.

5.3.2 Convolutional neural network with an RBF fuzzy logic rule-base classification layer

In the previous section, a reference CNN was set. In this section, the No Defuzzy-Radial Basis Function-Convolutional Neural Network (ND-RBF-CNN) would be detailed.

The reference model could be treated as a combination of two parts, which are first seven layers as a feature extractor and last three layers as a classifier. The ND-RBF-CNN has the similar structure. As described in Fig. 5.2, the first six layers of this ND-RBF-CNN model are identical to the reference CNN model, while the layers after `dense_1` layer was modified. The activation function of `dense_1` layer was changed from ReLU to a hard tanh function in order to clip the output value in the range of $[0.0, 1.0]$. The reason to clip the output of `dense_1` layer, i.e., the input of `rbf_layer`, is to guarantee the input of Gaussian membership function is meaningful. Therefore, the output of `dense_1` layer should be always in the range of $[0, 1]$. The output of `rbf_layer` was fixed as 10 because there are 10 classes in both MNIST and Fashion-MNIST datasets.

Table 5.2 shows parameter setting of the FL RBF-CNN. Similar to FL RBF networks and FL RBF-CNNs, ND-RBF-CNNs will also be sensitive to initial conditions (initial model structure and parameters) of the RBF and

Table 5.1: Basic CNN architecture

role	**type**	**patch size/stride**	**output size**	**parameters**
	convolution	$3 \times 3/0$	$26 \times 26 \times 32$	320
	convolution	$3 \times 3/0$	$24 \times 24 \times 64$	18496
	maxpooling	$2 \times 2/0$	$12 \times 12 \times 64$	0
Extractor	dropout (25%)		$12 \times 12 \times 64$	0
	flatten		9216	0
	linear(HardTanh(0.0,1.0))		64	589888
	dropout (50%)		64	0
Classifier	linear		10	1290

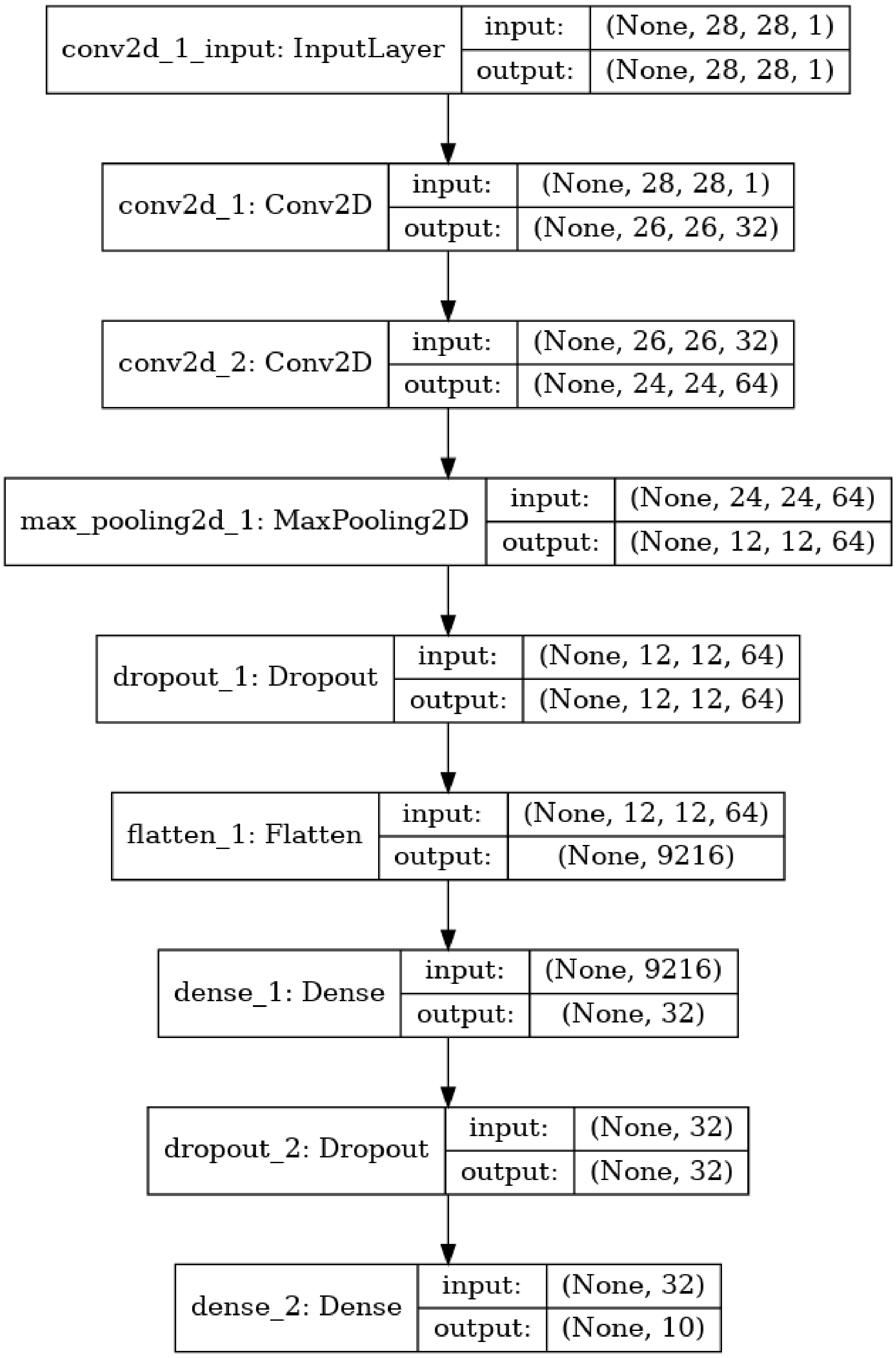


Figure 5.1: basic CNN structure

defuzzification layers, and the initial parameters could not be determined since the features from CNN layers have not been extracted before training. Therefore, one has to establish some initial conditions for the FL rule base for successful model training.

5.4 Results

5.4.1 Training results

As a further improvement adding Multiple-Output ability to RBF-CNN, the model described in this chapter would be compared with results recorded in previous chapters. In Chapter 3 and 4, there were two datasets used for benchmark, which were MNIST and Fashion-MNIST. For the convenience of comparison, these two datasets were also been applied in this chapter.

For each dataset, there were six groups tests set. These groups were trained with different feature size, says 16, 32, and 64 features respectively. For each feature size, there was a group used the new RBF layer as the experiment group and a group applied ordinary linear layer as the control group.

The results were generated on a computer whose CPU was Intel i7–6700k and GPU was Nvidia GTX 1080. Noteworthily, the computer was set up as a `PyTorch v1.5.0`, rather than `TensorFlow`. The platform based on `Python 3.8`.

In all benchmarks, the batch size was chosen as 256 in order to achieve a good balance between training speed and avoidance of overfitting. The models was set with early stopping if the loss does not reduce in 15 epochs, and

Table 5.2: ND-RBF-CNN architecture

role	**type**	**patch size/stride**	**output size**	**parameters**
	convolution	$3 \times 3/0$	$26 \times 26 \times 32$	320
	convolution	$3 \times 3/0$	$24 \times 24 \times 64$	18496
	maxpooling	$2 \times 2/0$	$12 \times 12 \times 64$	0
Extractor	dropout (25%)	—	$12 \times 12 \times 64$	0
	flatten	—	9216	0
	linear(HardTanh(0.0,1.0))	—	`feature size`	9216×`feature size`
	dropout (50%)	—	`feature size`	0
Classifier	RBF	—	`rule size`	2×`rule size`

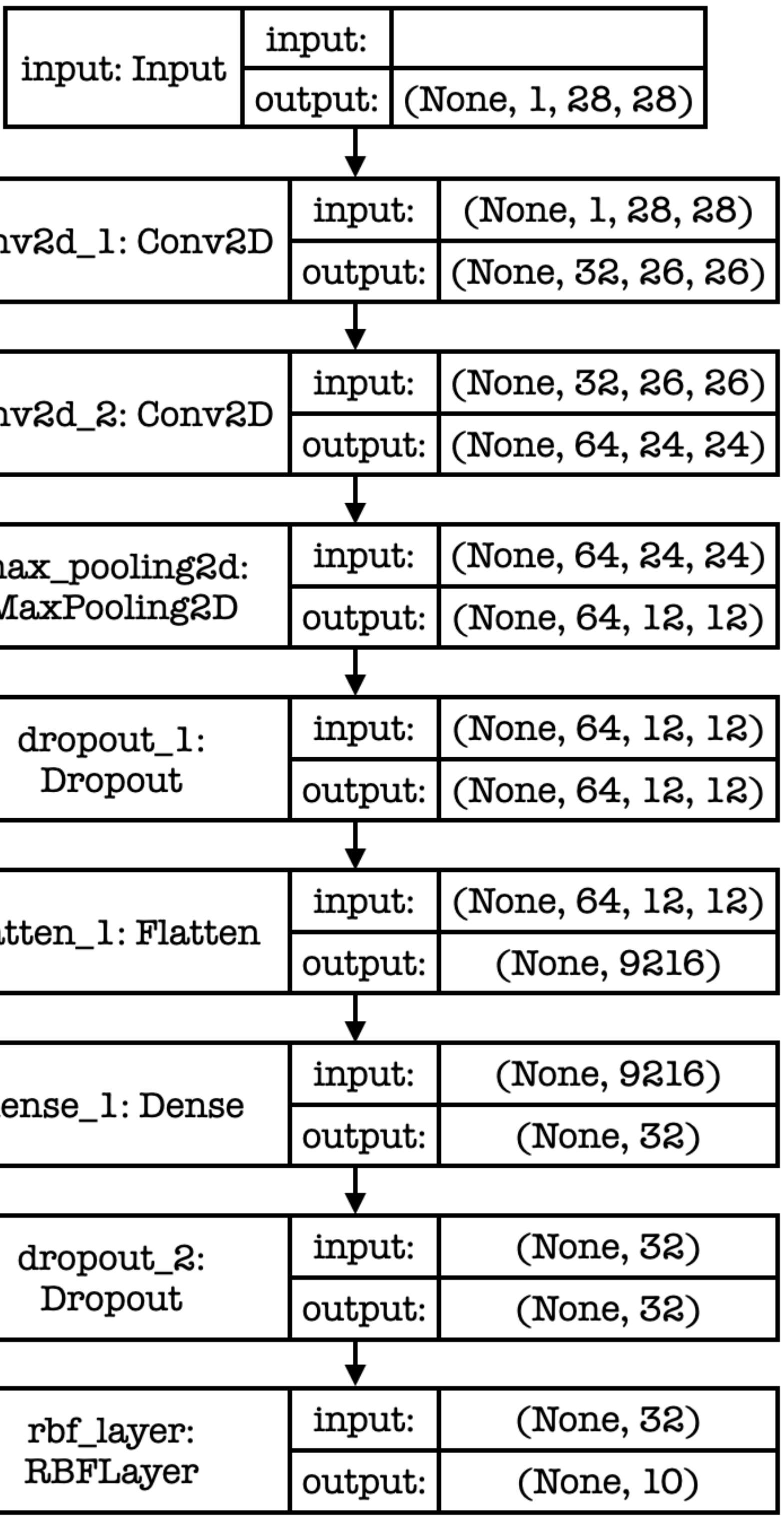


Figure 5.2: FL RBF-CNN layered structure

would stop training at 50 epochs at most. For the linear reference model, the initialisation method was Kaiming normal distribution for its performance and generalise. While for the ND-RBF-CNN model, variety initialisation method was utilised. The parameters of linear parts would be generate by Kaiming normal distribution; the RBF layer weights c was chosen as constant 0 because it was found uniform distribution in the range of $[0, 1]$ would cause the model hard to train, while variety constant values would not affect training results significantly; the standard deviations σ of Gaussian membership functions was fixed at 0.3 because models cannot be trained if the weight and the standard deviations are all trainable together.

MNIST

MNIST [42] is a popular image dataset provided as a 50000-sample training set and an 10000-sample test set. In this benchmark, the provided training set would be separated into two parts, which are 50000-sample training set and 10000-sample validation set in every training. Every situation was tested 30 times.

Table 5.3 contains three sets of results grouped by the feature size. It is clear that the average accuracy would increase while feature size increased from 16 to 64 for both linear model and ND-RBF model. Comparing with the highest MNIST accuracy 99.77% [15], it could be found the accuracies of both linear model and ND-RBF model using 64 features are acceptable. Noteworthily, the average test losses of linear models dropped significantly and end around 0, while the ones of ND-RBF models ended around 1.5. It indicates the rules of

ND-RBF models would be activated simultaneously even with 64 features.

Table 5.3: Test loss and accuracy for MNIST dataset

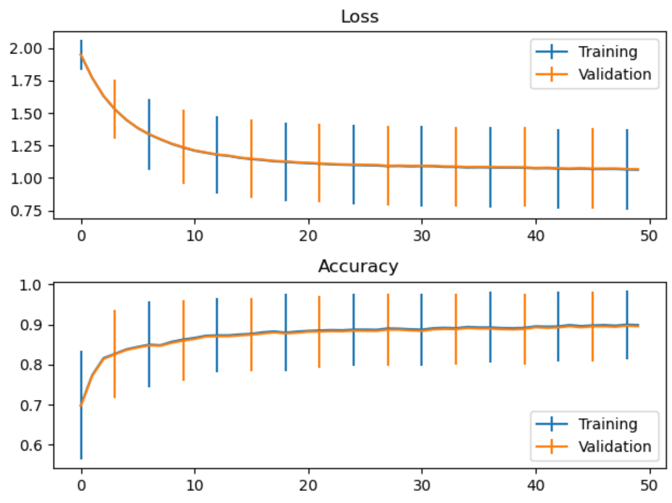
Feature size	Model type	Test loss	Test accuracy/%
16	linear	(1.0572 ± 0.3068)	(90.23 ± 8.74)
	RBF	(1.9301 ± 0.1063)	(61.26 ± 12.44)
32	linear	(0.3316 ± 0.1288)	(97.25 ± 0.54)
	RBF	(1.6443 ± 0.0918)	(81.14 ± 10.48)
64	linear	(0.0643 ± 0.0176)	(98.34 ± 0.22)
	RBF	(1.5042 ± 0.0227)	(95.72 ± 2.29)

Fig 5.3, 5.4, and 5.5 demonstrates the average loss and accuracy with error bars for linear models and ND-RBF models in the cases of 16, 32, and 64 features respectively. It could be found in all cases of different features, linear models could converge, while ND-RBF model only converge in the case of 64 features. In Fig 5.3b, the plot ends at 26 epochs, which means the lost of most ND-RBF models would not decreases after $26 - 15 = 11$ epochs.

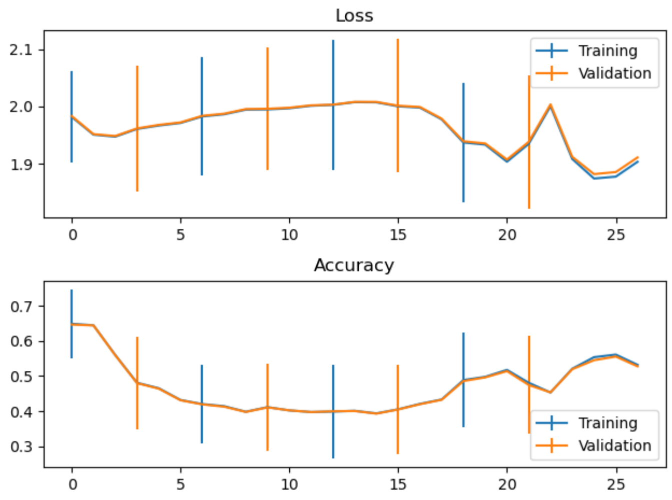
Fashion-MNIST

Fashion MNIST [92] is an image dataset in the same dimension of MNIST, and provides a 50000-sample training set and an 10000-sample test set. The purpose of creation of Fashion MNIST is to increase benchmark difficulty. In this benchmark, the provided training set would be separated into two parts, which are 50000-sample training set and 10000-sample validation set in every training. Every situation was tested 30 times.

Table 5.4 contains three sets of results grouped by the feature size. The results in Table 5.4 could draw a similar conclusion as the previous section

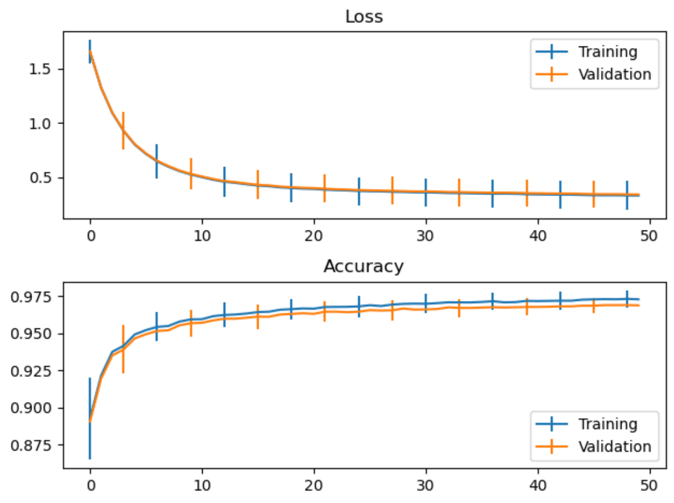


(a) Linear models

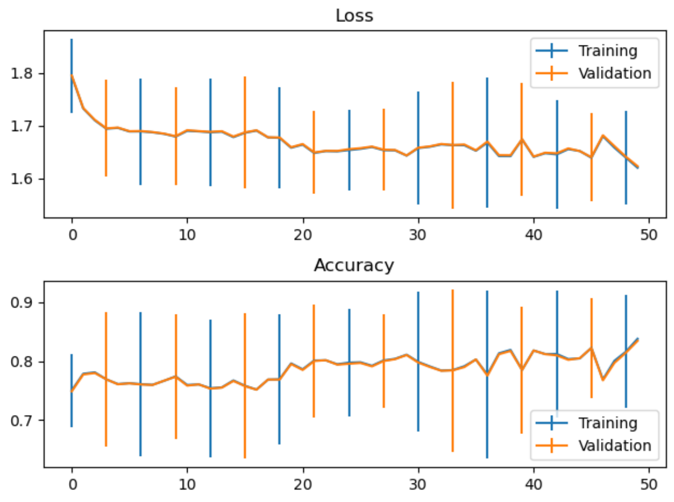


(b) ND-RBF models

Figure 5.3: Training records of 16 features for MNIST dataset

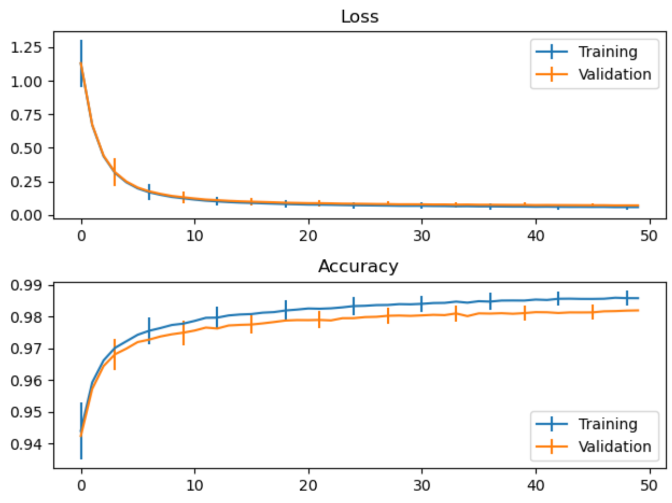


(a) Linear models

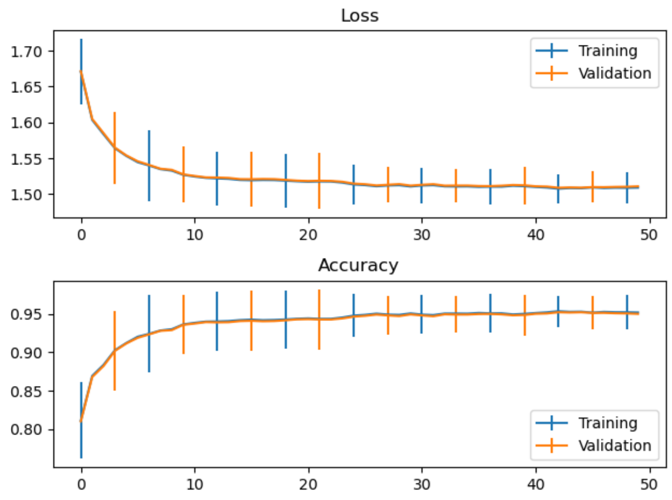


(b) ND-RBF models

Figure 5.4: Training records of 32 features for MNIST dataset



(a) Linear models



(b) ND-RBF models

Figure 5.5: Training records of 64 features for MNIST dataset

using MNIST dataset. It is clear that the average accuracy would increase while feature size increased from 16 to 64 for both linear model and ND-RBF model. The average test losses of linear models dropped significantly and end around 0.34, while the ones of ND-RBF models ended around 1.67.

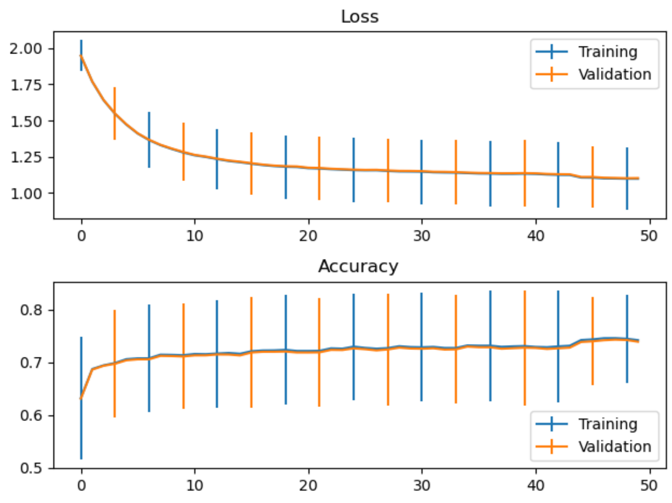
Table 5.4: Test loss and accuracy for Fashion-MNIST dataset

Feature size	Model type	Test loss	Test accuracy/%
16	linear	(1.1166 ± 0.2267)	(72.82 ± 10.25)
	RBF	(1.8990 ± 0.0984)	(58.73 ± 10.18)
32	linear	(0.5591 ± 0.0638)	(82.83 ± 2.14)
	RBF	(1.7259 ± 0.0380)	(73.39 ± 4.27)
64	linear	(0.3397 ± 0.0264)	(88.71 ± 0.83)
	RBF	(1.6713 ± 0.0270)	(78.93 ± 2.67)

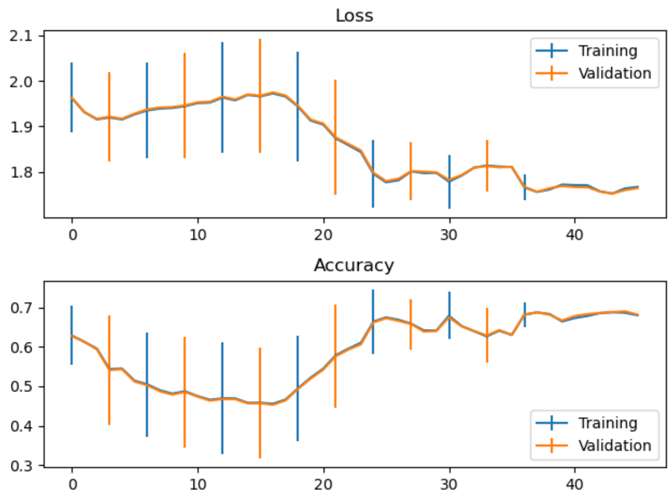
Fig 5.6, 5.7, and 5.8 demonstrates the average loss and accuracy with error bars for linear models and ND-RBF models in the cases of 16, 32, and 64 features respectively. Similar, it could be found in all cases of different features, linear models could converge, while ND-RBF model only converge in the case of 64 features. In Fig 5.6b, the average loss remains around 1.95 steadily to 17-th epoch, then dropped to around 1.8 at 24-th epoch. It indecates that in several tests ND-RBF models could be trained, while in other tests the models cannot be trained effectively.

5.4.2 Test result comparison

The benchmark results in previous section could be compared with the benchmark results of FL RBF-CNN. In Chapter 4, both MNIST and Fashion-MNIST dataset were applied, and hence a comparison table could be created. Table 5.5

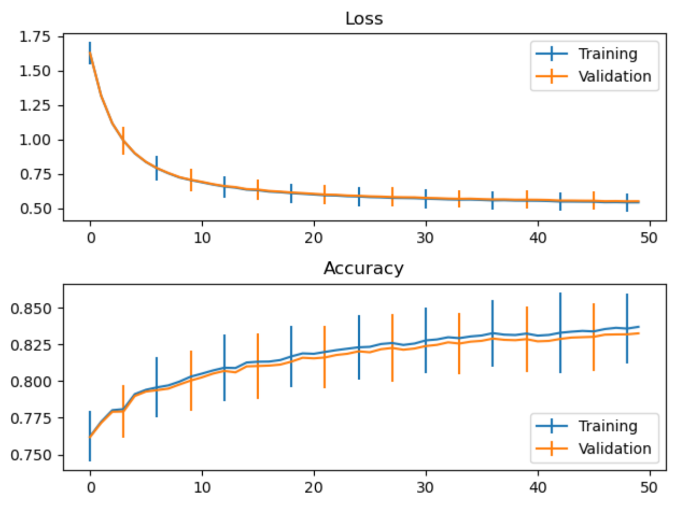


(a) Linear models

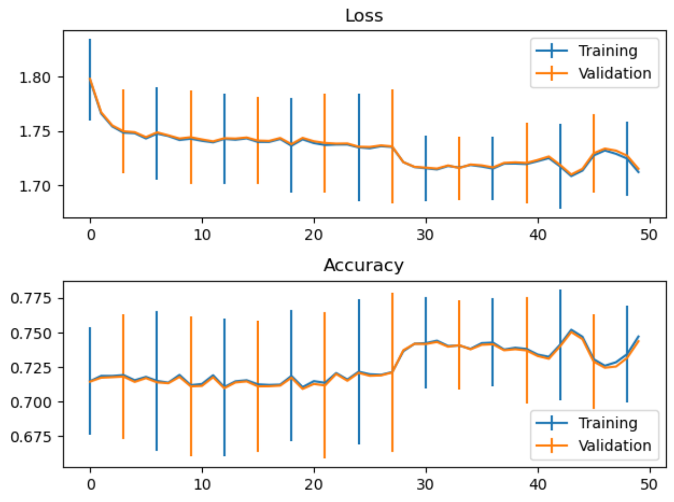


(b) ND-RBF models

Figure 5.6: Training records of 16 features for Fashion-MNIST dataset

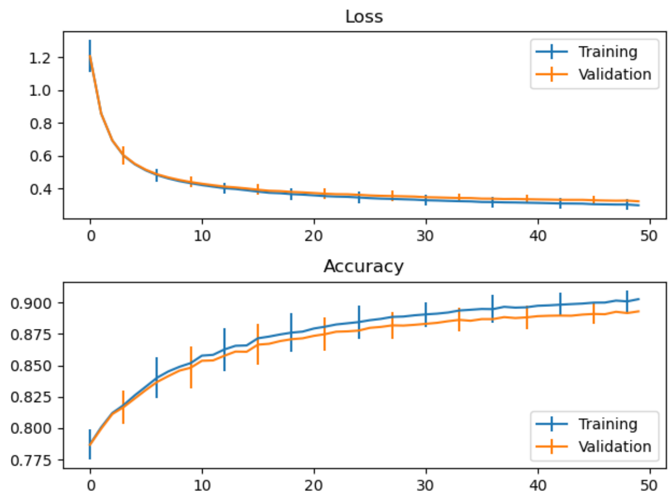


(a) Linear models

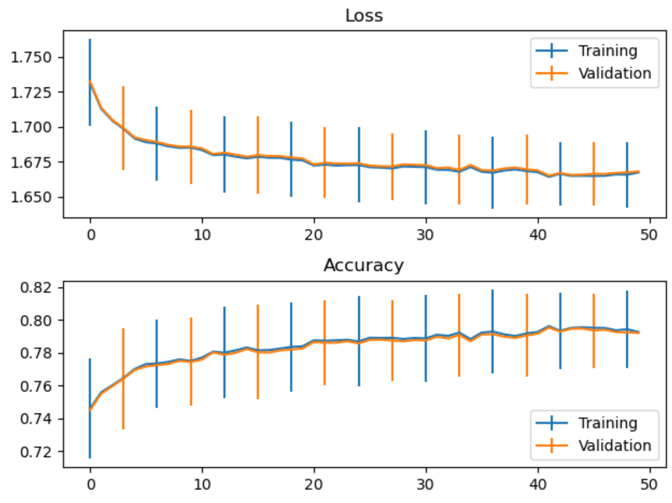


(b) ND-RBF models

Figure 5.7: Training records of 32 features for Fashion-MNIST dataset



(a) Linear models



(b) ND-RBF models

Figure 5.8: Training records of 64 features for Fashion-MNIST dataset

and 5.6 were created from data in Table from 4.2 to 4.7, Table 5.3, and Table 5.4. The test loss cells of FL RBF-CNN model in Table 5.5 and 5.6 are empty because of the missing of data. All results of FL RBF-CNN models were selected as the best results in different cases of rule size.

From the results listed in Table 5.5 and 5.6, it could be found the average and the standard deviation of test accuracy of ND-RBF-CNN models in every case was worse than FL RBF-CNN. However, with 64 features, the prediction performance of ND-RBF-CNN dropped in an acceptable range (from 97.52% to 95.72% with MNIST dataset and from 84.70% to 78.93% with Fashion-MNIST dataset).

Table 5.5: Test loss and accuracy for MNIST dataset

Model type	Feature size	Test loss	Test accuracy/%
FL RBF-CNN	16	—	(78.57 ± 7.72)
	32	—	(95.77 ± 2.26)
	64	—	(97.52 ± 0.20)
ND-RBF-CNN	16	(1.9301 ± 0.1063)	(61.26 ± 12.44)
	32	(1.6443 ± 0.0918)	(81.14 ± 10.48)
	64	(1.5042 ± 0.0227)	(95.72 ± 2.29)

Table 5.6: Test loss and accuracy for Fashion-MNIST dataset

Model type	Feature size	Test loss	Test accuracy/%
FL RBF-CNN	16	—	(62.69 ± 2.66)
	32	—	(80.03 ± 2.80)
	64	—	(84.70 ± 1.07)
ND-RBF-CNN	16	(1.8990 ± 0.0984)	(58.73 ± 10.18)
	32	(1.7259 ± 0.0380)	(73.39 ± 4.27)
	64	(1.6713 ± 0.0270)	(78.93 ± 2.67)

5.5 Conclusion

In this chapter, a model, which combines RBF and CNN but has no compromise in mathematics was defined and benchmarked. As the sacrifice of mathematical legalisation, the test accuracy dropped at an adequate cost. The interpretability of this new proposed model would be explore in future.

6 Applications of the model without defuzzification in manufacturing

In this chapter, a new dataset XCT should be introduced and the model should be trained based on this dataset. Because of the low accuracy results, a 2-stage model was developed and achieved good performance.

In Chapter 3, a methodology with FL and CNN for image classification was introduced as RBF-CNN, and the methodology to track the rule bases was introduced in Chapter 4. In Chapter 5, a RBF-CNN model get rid of mathematical flaws was introduced as ND-RBF-CNN.

6.1 Introduction

Machine learning helps people in manufacturing [61, 63, 2, 66, 9, 20, 11, 72]. As an non-destructive flaw detection method, XCT was applied in high-value components detections [84, 87]. On the other hand, XCT would generate plenty of data and experts spent lot of time in classification.

Specificity, CNNs made it possible to analysis images with a high accuracy. For steady images, ImageNet [17] contains over 5247 classes, and VGG16 [96] achieved 7.0% of top-5 error rate. Furthermore, ResNet [30] acquired 6.7% of top-5 error rate.

However, CNNs are deep neural networks, which were black-boxes. People could not understand what features of an image cause a CNN classify or predict the class that the image should be in. Some efforts were paid on the interpretability of CNNs.

In this chapter, a new method will be introduced and tested with XCT dataset. This method combines FL with CNNs for the goal aiming on good classification performance provided by CNNs and interpretability provided by FL. However, as a compromise, the proposed method does not defuzzification parts, and uses Gaussian membership function with fixed variance. Comparing with other CNN with RBF models, the ND-RBF-CNN remains the ability to link rules and features, and provides the agility to convert CNN to ND-RBF-CNN models.

6.2 Background

6.2.1 A convolutional neural network

Neural networks are a large group among machine learning models, in which convolutional layers are generally used for feature extraction. Fig. 6.1 shows a typical CNN model for single image classification. In this typical CNN model, the first few layers combines convolutional layers and sub-sampling layers, which are stacks of several two-dimension arrays. These convolutional layers could use variety window sizes to extract features in different scales, and sub-sampling layers, where a max pooling method is generally used, are applied to sub-sample features into a smaller size. After the last stack of two-dimension layers, a fully connected layer would be used, in which neurons are connected to every outputs from the previous layer.

A CNN structure for image classification would contain several layers, grouped in a way to perform specific tasks. Fig. 6.1 demonstrates a typical CNN architecture. The first few layers would be multiple pairs of convolution layers and pooling layers. The size of these convolution windows can be different, which ensure convolution layers can extract features in different scales. The pooling layers are proposed to sub-sample features into a smaller size, where a max pooling method is generally used. Then, fully connected layers would also be used, in which neurons are fully connected to all outputs from the previous layer. These layers also convert the data structure from a multiple-layer structure to a vector form. Rectangular linear units (ReLUs) would normally be the activation function of the convolution layers as well as in the fully connected layers as these can provide non-linear properties to those layers and are also convenient for the calculation of the error backpropagation [24]. To avoid exploding and vanishing gradients in deep networks, batch normalisation can also be applied in every layer [33]. CNNs are not considered as convex functions, which means parametric optimisation for CNNs is challenging, hence numerous optimisation strategies have been developed [70], such as stochastic

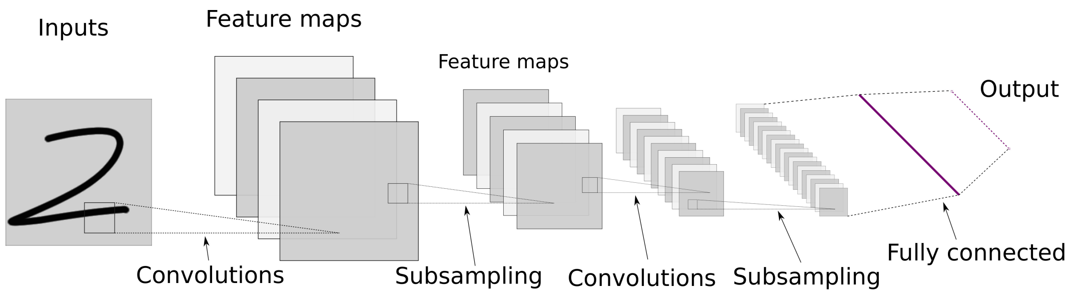


Figure 6.1: Representative CNN structure

gradient descent (SGD), Nesterov momentum [6], and adaptive subgradient (Adagrad) methods [21].

The performance of a CNN is improved by error backpropagation. After all layers are initialised, a CNN could output a value in a reasonable range. When an input passed into a CNN, which typically is an image, a loss value would be calculated with the output and the label of this image. This loss value would be backpropagated via derivative.

Besides convolutional layers and max-polling layers, some other structures are also generally used for different purposes. Rectangular linear units (ReLUs) would normally be the activation function of the convolution layers as well as in the fully connected layers as these can provide non-linear properties to those layers and are also convenient for the calculation of the error backpropagation [24]. To avoid exploding and vanishing gradients in deep networks, batch normalisation can also be applied in every layer [33].

6.2.2 EBM-XCT dataset

In this chapter, a new image dataset was created to test whether a model applied the RBF layer could be used in manufacturing. In manufacturing, products may only be produced in a small number, and defects can only be detected with non-destructive testing. Therefore, this new dataset was created based on a series of X-Ray Computed Tomography (XCT) scans of Electron Beam Melting (EBM) samples.

Experimental trials

There were three EBM 3D printed CM247 [29] alloy cubes manufactured and XCT scanned. In Fig. 6.2, there were 12 cubes printed together, while only labeled three ones, says No. 1, 9, and 12, were XCT scanned. From XCT scan samples shown in Fig 6.3, there were three kinds of areas could be distinguished as perfect, void, and crack parts.

Processed data

In order to train CNNs, a labeled dataset, containing three classes, named as EBM-XCT was created. The three classes of this dataset were named and selected according to defections generated during EBM printing procedures, which are 'perfect', 'void', and 'crack', as showing in Fig. 6.4. Every sampled image was chopped into 80-by-80 pixels in grey-scale. Fig 6.4a indicated the sample has no defects, while Fig 6.4b and Fig 6.4c were two major kinds of defections respectively, which were void and crack.

For testing the model performance on unseen data, the EBM-XCT dataset were separated into training set and test set. Table 6.1 shows the sample size of each class. The training set is not balanced as the lack of XCT images containing high-quality void or cracking area. Because of the small size of

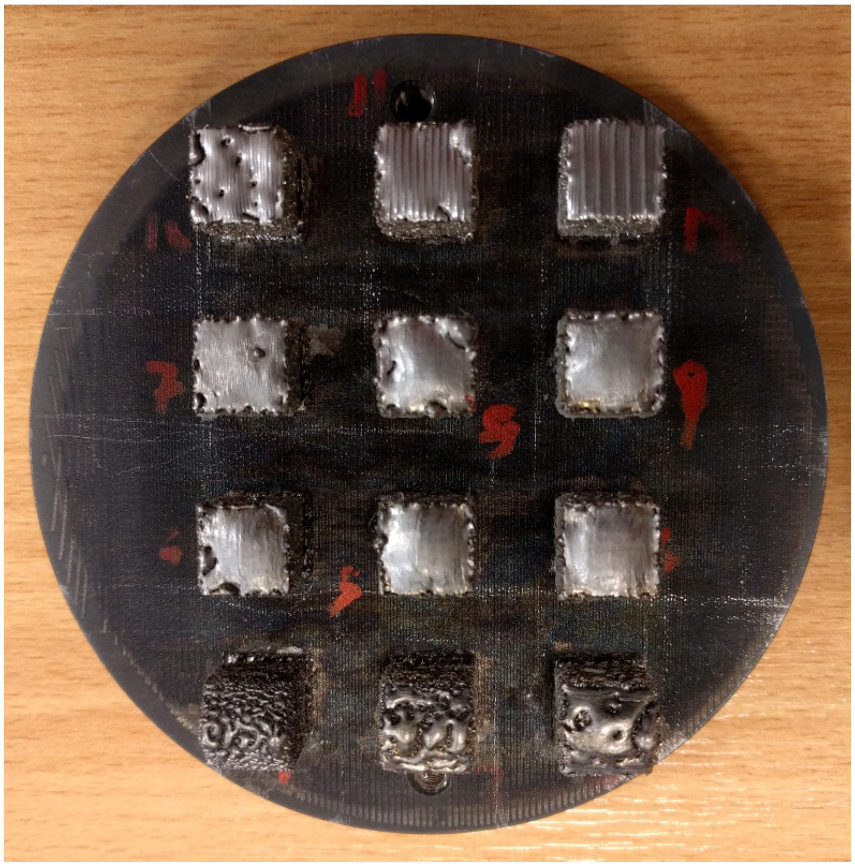
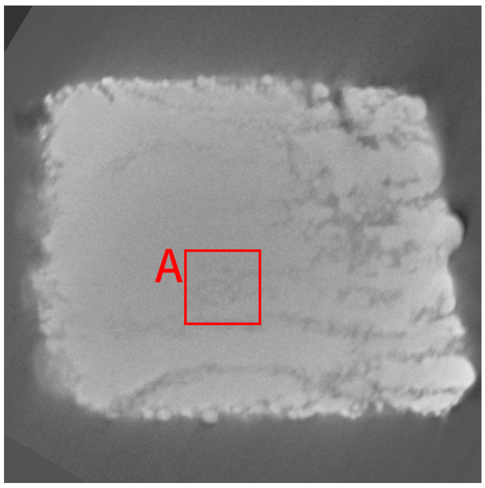
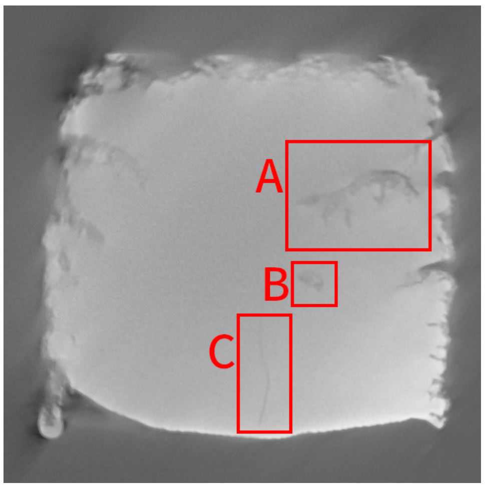


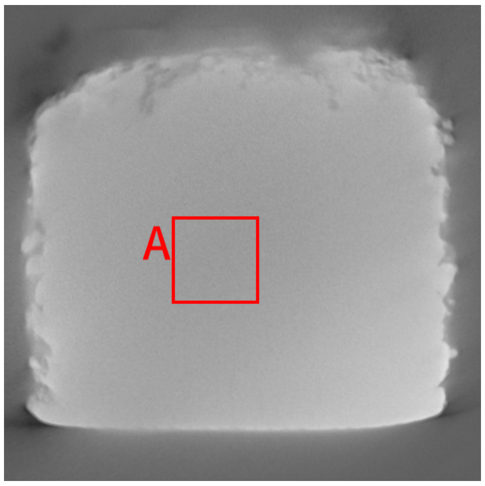
Figure 6.2: EBM 3D printed CM247 alloy cubes



(a) A sample image of No. 1 cube: A. void part



(b) A sample image of No. 9 cube: A. B. void part; C. crack part



(c) A sample image of No. 12 cube: A. perfect part

Figure 6.3: Original XCT images with notes

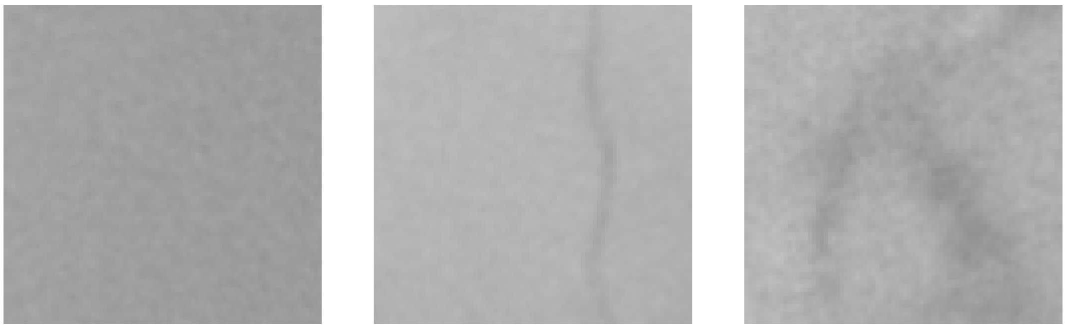

(a) Original perfect (b) Original crack (c) Original void

Figure 6.4: EBM-XCT examples

EBM-XCT dataset, every image in the dataset would be rotated 90, 180, and 270 degrees clockwise and added into the dataset with corresponding labels. Therefore, the size of the datasets applied in this paper is four times larger than the raw datasets respectively.

Table 6.1: Sample size of each class in the training set and the test set

	Set	Perfect	Void	Crack
Before process	Training set	55	46	30
	Test set	5	5	5
After process	Training set	220	184	120
	Test set	20	20	20

6.3 Methodology

Same as all previous chapters, there are a set of models would be formed. The first model would be a regular CNN model, and the second one would be a CNN model based on the former and other FL part. In this section, the main CNN structure is detailed, and it is shown how the RBF-NF layer is integrated into the overall network structure and learning methodology.

6.3.1 Reference CNN

Fig. 6.5 depicts the overall structure of a CNN network. This model was designed to use 80×80 pixel grey-scale images as input. After two convolutional layers, a max pooling layer was added. The dropout layers were applied to avoid overfitting. The Flatten layer was added to convert data structure into vectors, and two Dense layers are fully connected layers. All activation functions in this model were ReLUs. The loss function of this model was cross entropy loss function, which is widely used in CNNs [40, 83]. In the proposed research work, the Adam [39] optimiser was applied to take the advantage of its self-tuning learning rate for potential difficulties with a small training dataset, and cross-entropy loss was applied. Table 6.2 shows the architecture of the designed CNN.

6.3.2 Proposed CNN-FL modelling structure

As reviewed in Chapter 2, FL has already been mixed with CNNs for various purposes. Price et al. inserts fuzzy layers between convolutional layers in a CNN and achieved higher accuracy [60]. Fuzzy pooling layers [74] are also been found and benchmarked. Yeganejou and Dick [93] introduces a fuzzy clustering model training with features extracted with a pre-trained ResNets [30], and gains a good accuracy with visual interpretation. However, no linguistic fuzzy interpretable models combined with CNN layers which demonstrate what features in an image cause the model classified it into a catalogue.

Table 6.2: Basic CNN architecture

role	type	patch size/stride	output size	parameters
	convolution	$3 \times 3/0$	$77 \times 77 \times 32$	544
	convolution	$3 \times 3/0$	$74 \times 74 \times 64$	32832
	maxpooling	$2 \times 2/0$	$37 \times 37 \times 64$	0
Extractor	dropout (25%)		$37 \times 37 \times 64$	0
	flatten		87616	0
	linear(HardTanh(0.0,1.0))		32	5607488
	dropout (50%)		32	0
Classifier	linear		10	1290

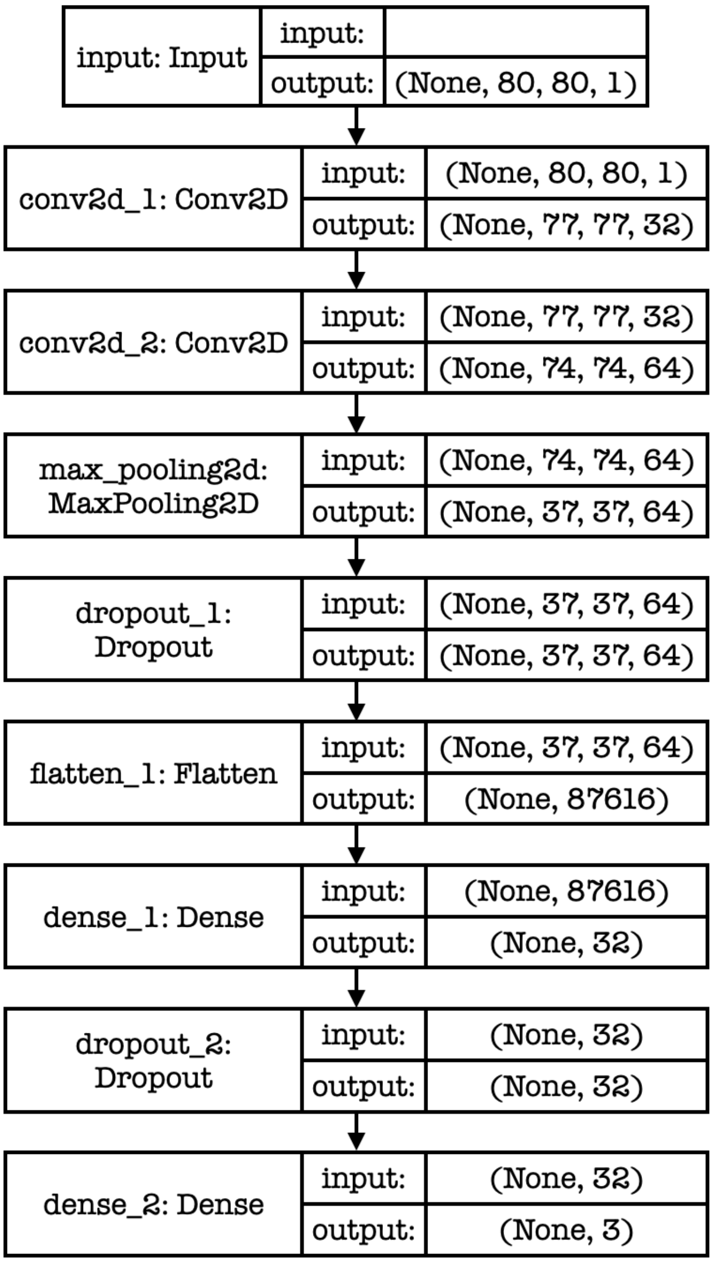


Figure 6.5: Basic CNN layered structure

Linguistic fuzzy models could help people understand what features leading to the results. Targeting to such a model, it is proposed here that a model combining CNNs and RBF fuzzy model.

In Chapter 5, a method named ND-RBF-CNN was introduced. Because of its mathematical legalisation structure and the compatible ability to existing optimisers, a model based on ND-RBF-CNN would be created referencing the linear model designed in previous section and benchmarked.

Fig. 6.6 depicts the architecture of the FL RBF-CNN, and Table 6.3 shows parameter setting of the FL RBF-CNN. The model has the same typography of the layers as the model defined in Table 6.3, while the parameter sizes of layers were modified adapting to EBM-XCT dataset. The patch sizes of the convolution layers were increased to 4 because of the image size was increased. The RBF layer would have an output of 3 as EBM-XCT dataset has three classes.

Because of the part of FL RBF networks, ND-RBF-CNNs will also be sensitive to initial conditions (initial model structure and parameters) of the RBF layer, and the initial parameters could not be determined since the features from CNN layers have not been extracted before training. Therefore, one has to establish some initial conditions for the FL rule base for successful model training. The overall training would rely on a cross-entropy loss function and optimised by Adam optimiser.

Table 6.3: ND-RBF-CNN architecture

role	**type**	**patch size/stride**	**output size**	**parameters**
	convolution	$4 \times 4/0$	$77 \times 77 \times 32$	544
	convolution	$4 \times 4/0$	$74 \times 74 \times 64$	32832
	maxpooling	$2 \times 2/0$	$37 \times 37 \times 64$	0
Extractor	dropout (25%)		$37 \times 37 \times 64$	0
	flatten		87616	0
	linear(HardTanh(0.0,1.0))		32	5607488
	dropout (50%)		32	0
Classifier	RBF		3	96

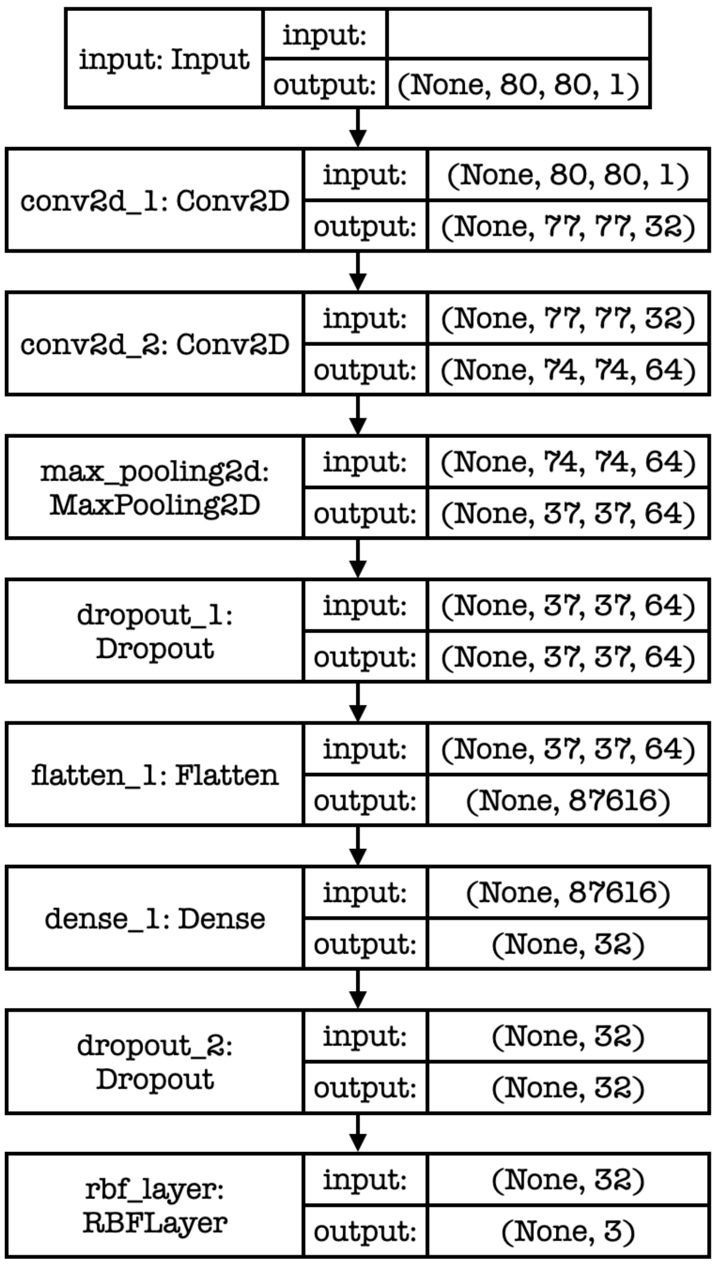


Figure 6.6: FL RBF-CNN layered structure

6.3.3 Identification of interpretable features

The input space for the rule-based structure, in our case the fully connected RBF layer, is a flat vector of weights, as shown in the CNN structure depicted in Fig. 6.6. To enhance the interpretability of the antecedent part of the 'IF... THEN...' Fuzzy Logic rules we propose to 'track back' the weights of the flat input vector of the fully connected layer towards revealing the relevant features of the input image. Effectively, we propose to associate via this mechanism relevant rules to features in the image's feature space, so that the user can appreciate which rules are responsible for each classification decision, and what is the relevant input space for each rule in the feature space. This is achieved as follows:

For any CNN model, as defined in the structure shown in Table 6.3, the final layer acts linearly [80]. Therefore, to track back the weights of the input space of the fully connected layer, one can follow the process:

- Calculate a mask layer using a least mean square solution. The input is a vector of selected features, and the output is a vector of 5607488 points.
- Reshape the 5607488-point vector as a tensor a with dimensions 37–37–64.
- Use tensor a as the mask. Calculate a track-back maxpooling layer result using $M_{trackback} = M \circ a$.
- Reveal feature 'heat maps' using mask $M_{trackback}$.

This procedures could be applied on any CNN model of the proposed structure, regardless of having a FL-based RBF layer or not. The advantage of using the

FL-based RBF layer is that one can now link linguistic rules to the input feature space using the above described process. After converting FL-RBF-CNN to ND-RBF-CNN, there is no more defuzzification layer, and hence there is no link between rules to linguistic formats. However, the link between rules and features still remains.

As the FL-RBF layer consists of multiple fuzzy rules, the relative importance of each rule could be estimated using a Fuzzy Logic entropy measure. In the proposed framework, a non probabilistic entropy function could be used [73], this is shown below.

$$H = -K \sum_{i=1}^{n} \left(\mu_i \log\left(\mu_i\right) + \left(1 - \mu_i\right) \log\left(1 - \mu_i\right) \right), \tag{6.1}$$

where K is a positive constant (usually equal to $1/N$ for normalisation), and μ_i is i-th membership degree.

Using Fuzzy Entropy to identify the most ‘active’ rules for a given prediction, and by identifying the Membership Functions of each rule with the highest relevance to the input vector (membership degree), then a framework can be established to directly link rules, to input space features; this is demonstrated in the Simulation Results is Section 4.

6.3.4 Simulation results

The benchmarks could be separated into two parts. In the first part, the models would be trained targeting on higher performance. In the second part, the linguistic interpretation method provided in Chapter 4 would be tested.

General models' performance

In this section, all models were trained and tested with EBM-XCT dataset introduced in previous section. Because of different sub-datasets applied in variety situation, the size of training, validation, test set are not common. All models were setted with an early-stopper if the traini ng loss stops decreasing for 15 epochs, and all training would stop after 50 epochs.

The results in this section were introduced in a chronical sequence. At the vary beginning, a model was developed for three-class dataset. Regarding to the unsatisfying prediction accuracy, a two-stage cascade model was developed, which could predict unseen data at an accuracy above 92%.

Three-class dataset case

As a intuitive result, a set of models with three prediction classes was created and tested first. The applied training dataset and validation dataset were separated from EBM-XCT training dataset at a ratio of 0.8 : 0.2 randomly, which were $524 * 0.8 \approx 419$ samples for test and $524 * 0.2 \approx 105$ samples for:s validation. Afterwards, the trained models were tested with the test set described in table 6.1, says 20 samples per class. All these data were tested in three groups, which were separated based on feature size varied in 16, 32, and 64. After 30 trials, results were record as in table 6.4.

In table 6.4, we could find some conclusions. Firstly, we can find performances of test loss and overall accuracy would be better when feature size increased, while the improvement of performance was not significant from 32 features to 64 features. Secondly, both linear and RBF models can classify 'Perfect'

Table 6.4: Test results for three-class cases with a variety of feature size

Feature size	Model type	Test loss	Accuracy/%			
			Overall	Perfect	Void	Crack
16	Linear	(0.9278 ± 0.8883)	(52.41 ± 11.90)	(76.75 ± 15.43)	(49.00 ± 49.08)	(31.50 ± 30.46)
	RBF	(0.9738 ± 0.0762)	(57.33 ± 13.10)	(85.00 ± 9.49)	(58.75 ± 44.24)	(28.25 ± 31.20)
32	Linear	(0.8138 ± 0.0972)	(64.67 ± 11.98)	(82.25 ± 51.17)	(74.50 ± 41.14)	(37.25 ± 29.30)
	RBF	(0.9025 ± 0.0747)	(65.00 ± 12.37)	(79.00 ± 7.18)	(72.50 ± 39.67)	(43.50 ± 32.83)
64	Linear	(0.7642 ± 0.0622)	(65.08 ± 10.29)	(83.00 ± 6.00)	(69.75 ± 40.57)	(42.50 ± 29.77)
	RBF	(0.8232 ± 0.0808)	(74.25 ± 8.68)	(79.00 ± 7.52)	(84.25 ± 29.76)	(59.50 ± 25.88)

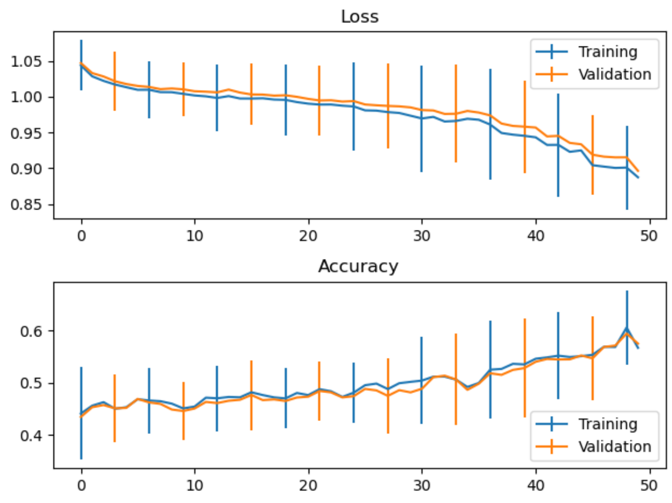
class more accurately comparing with the other two defect classes. With 32 features, both models got an 'Perfect' class accuracy around 80%. But at the same time, the 'Crack' class accuracy was close to 33.3%, which is the random guessing accuracy. Therefore, the three-class models cannot classify this dataset properly. However, the good 'Perfect' class accuracy indicated these models maybe work well on classify samples with 'Perfect' labels.

Fig. 6.7 demonstrates the training records for both reference models and ND-RBF-CNN models. It could be found both type of models were still training when 50-epoch limit reached, which means the models can still be trained further. However, the training losses reduced slowly means the models can only learn from the training slowly.

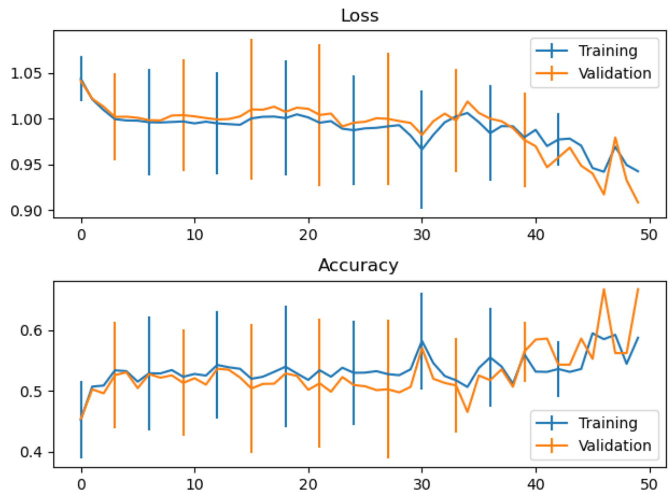
In order to verify if the RBF-CNN model could be trained on the EBM-XCT dataset, a binary case was tested.

Two-class dataset case

In this case, the EBM-XCT dataset was re-allocated into two classes: perfect class and defect class. The perfect class is identical to the one in EBM-XCT dataset, while the defect class is a combination of crack class and void class. Because of the performance comparison varying feature size in Table 6.4, models with 32 features, which may around the knee point of a accuracy versus feature size curve, were trained and tested. The training dataset was not balanced: 220 samples labeled as 'Perfect', while 304 samples labeled as 'Defect'. As the same as the previous section, the original training dataset was split into training part and validation part at a ratio of 80 : 20, says 419 samples in the training



(a) Linear models



(b) ND-RBF models

Figure 6.7: Training records of the three-class case trained with 32 features

set and 105 samples in the validation set. Afterwards, the models were tested with the test set created in Section 6.2.2. The benchmark ended after 30 trials.

Table 6.5 concludes linear and ND-RBF-CNN models test results. As there were two classes which can be described as positive and negative in the trials, the table contains sensitivity and specificity, where the detection of defects was marked as positive. Although both kind of models cannot achieve an prediction accuracy over 90%, it could be found the performance of ND-RBF-CNN models was slightly better than reference models.

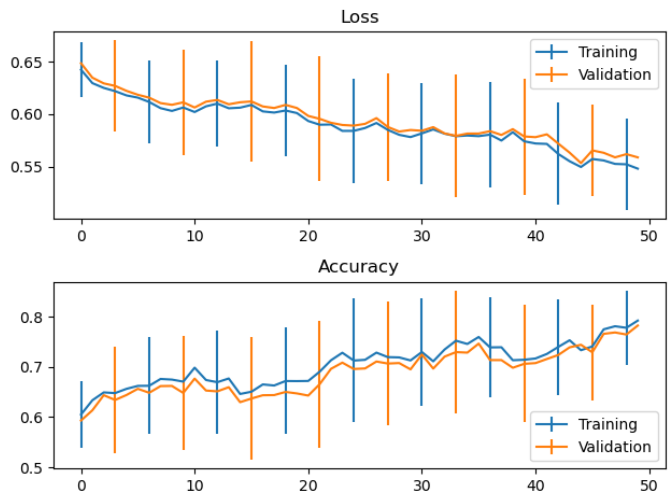
Table 6.5: Test results for 'Perfect-Defect' two-class dataset with 32 features

	Loss	Accuracy/%	Sensitivity/%	Specificity/%
linear	(0.5517 ± 0.0470)	(77.44 ± 8.83)	(50.67 ± 32.40)	(90.83 ± 10.88)
rbf	(0.5201 ± 0.0496)	(80.44 ± 7.44)	(64.83 ± 21.89)	(88.25 ± 11.22)

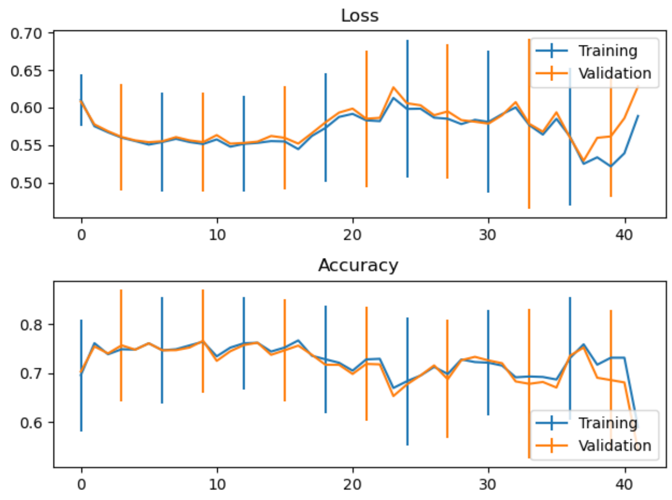
Fig. 6.8 demonstrates the training records for both reference models and ND-RBF-CNN models. As a supplement to data in Table 6.5, reference models performed more robust than ND-RBF-CNN models. The above two clues indicates that ND-RBF-CNN models could be trained very well in some trials.

Void and Crack case

In this case, the EBM-XCT dataset was only used 'Void' and 'Crack' classes, which are the components of 'Defect' mentioned in previous case. The training dataset was created with 184 'Void' samples and 120 'Crack' samples. Afterwards, the dataset was separated randomly into a set of $(184 + 120) * 0.8 \approx 243$ samples for training and a set of $304 * 0.2 \approx 61$ for validation. The benchmark



(a) Linear models



(b) ND-RBF models

Figure 6.8: Training records of ‘Perfect-Defect’ two-class case trained with 32 features

stopped after 30 trials for each kind of model.

Table 6.6 concludes linear and ND-RBF-CNN models test results. It could be found again that the average performance of ND-RBF-CNN models was better than reference models.

Table 6.6: Test results for 'Void-Crack' two-class dataset with 32 feature

	Loss	Overall accuracy/%	Void accuracy/%	Crack accuracy/%
linear	(0.5158 ± 0.1554)	(74.17 ± 23.31)	(50.67 ± 48.59)	(97.67 ± 5.44)
rbf	(0.4255 ± 0.0980)	(92.17 ± 12.79)	(93.17 ± 24.92)	(91.17 ± 12.43)

Fig. 6.9 demonstrates the training records for both reference models and ND-RBF-CNN models. Similar, as a supplement to data in Table 6.6, reference models performed more robust than ND-RBF-CNN models.

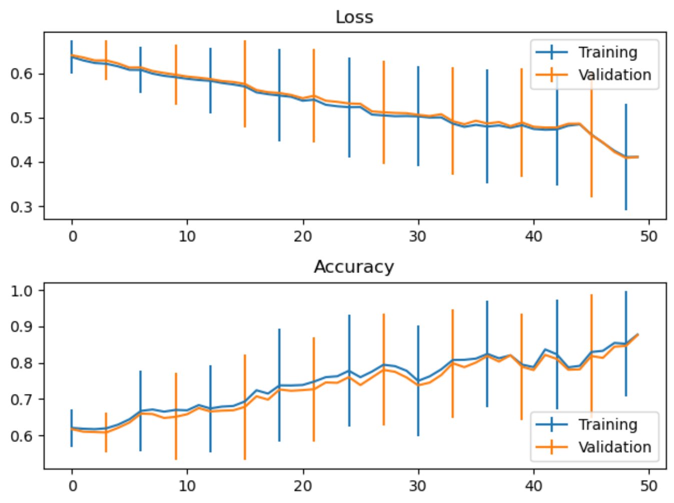
6.3.5 Trackback based on 'Void-Crack' two class model

Among all three cases above, the average accuracy of ND-RBF-CNN models was the highest one, hence a model was chosen from 'Void-Crack' case, whose test accuracy was 100%.

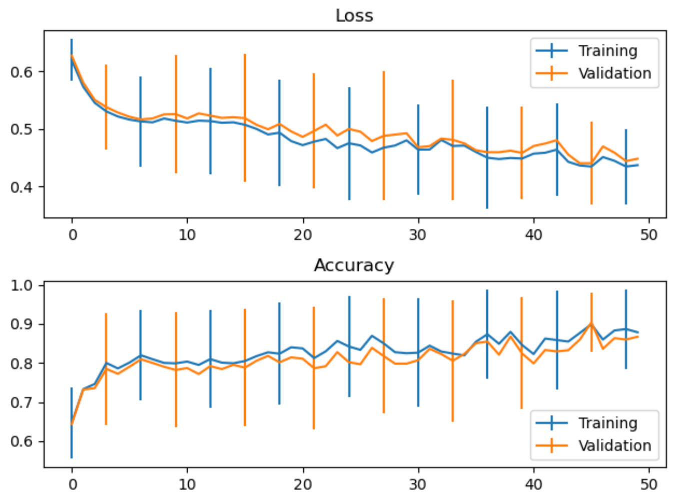
Fig 6.10 demonstrates the features extracted by the CNN part.

Fig 6.11 and 6.12 shows all membership function in each rules separately. The blue curve indicates membership function, and the red lines indicate the input of membership functions, i.e. features.

After re-order the membership functions in decreasing order of outputs, there would be a linguistic rule plot as Fig. 6.13. Although there is no Singleton defuzzy layer after RBF layer, the linguistic rules still could be formed as:

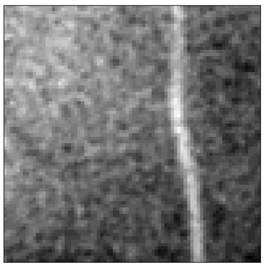


(a) Linear models

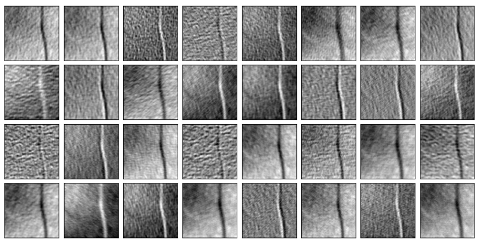


(b) ND-RBF models

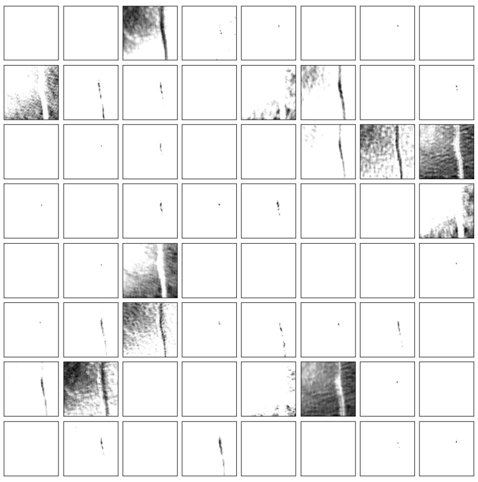
Figure 6.9: Training records of ‘Void-Crack’ two-class case trained with 32 features



(a) Input image



(b) First CNN layer output



(c) Second CNN layer output

Figure 6.10: Outputs of CNN layers

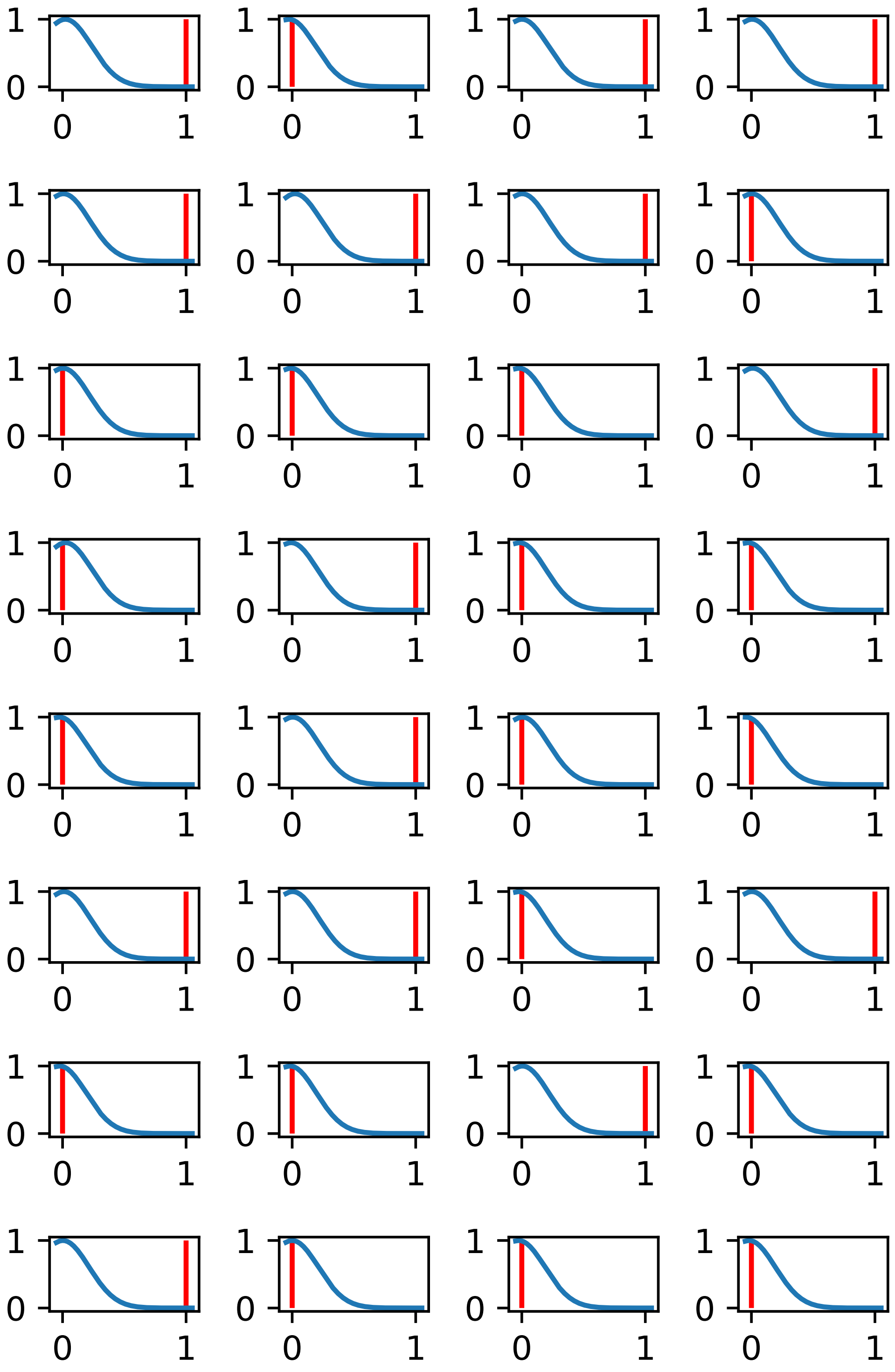


Figure 6.11: Rule base 1: corresponding to crack

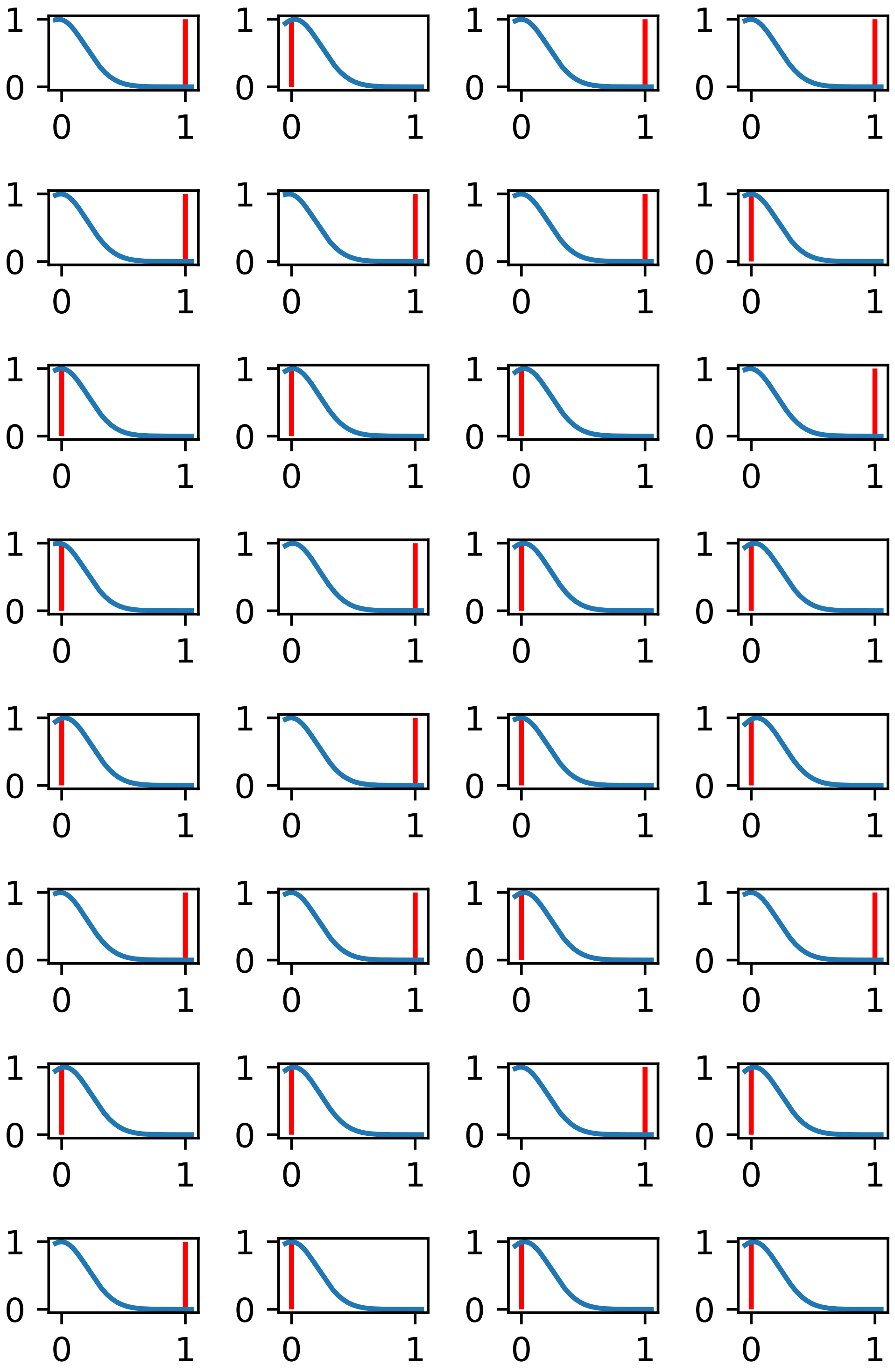
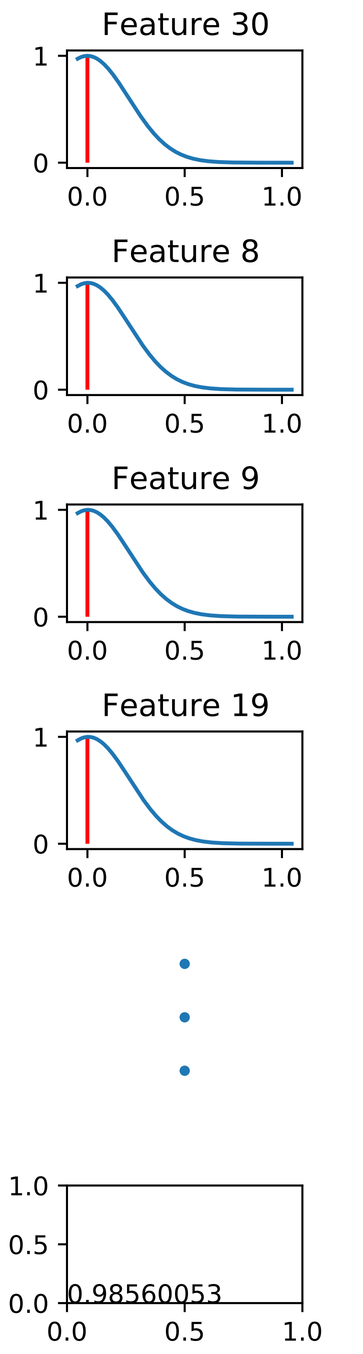
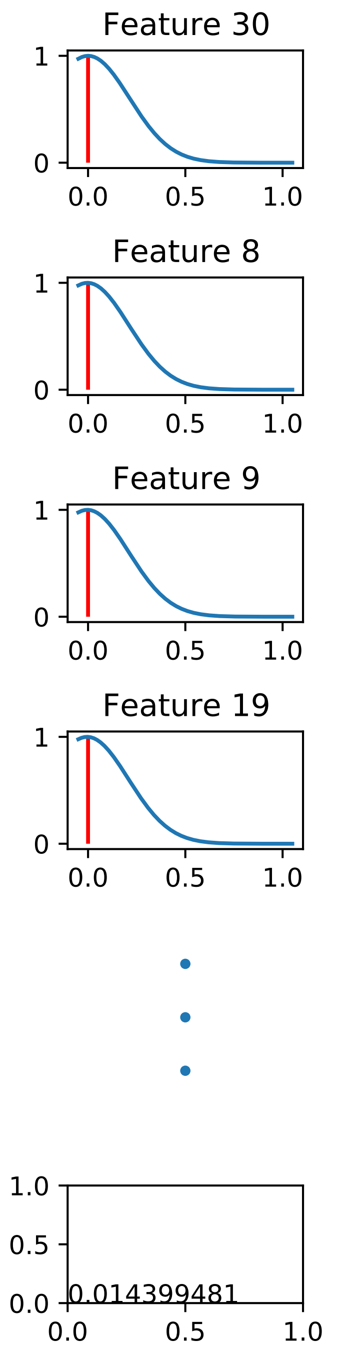


Figure 6.12: Rule base 2: corresponding to crack



(a) Linguistic rule base 1: corresponding to crack



(b) Linguistic rule base 2: corresponding to void

Figure 6.13: Effective features corresponding to 2 most significant features

$$\text{`}\textit{IF Feature 30 is A1, and Feature 8 is B1, and Feature 9 is C1, and..etc.}\quad \textit{THEN the Output class is O1.'} \tag{6.2}$$

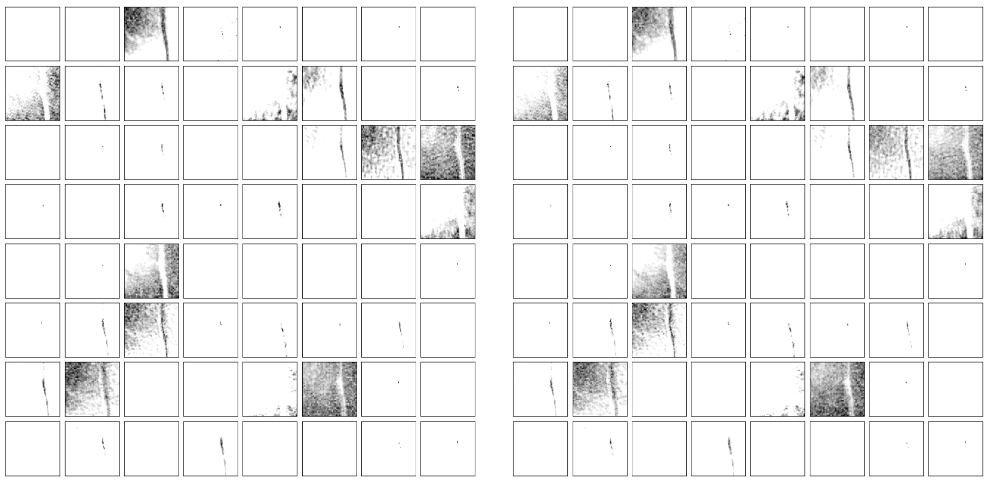

(a) Rule 1: corresponding to crack (b) Rule 2: corresponding to void

Figure 6.14: Effective features corresponding to 2 most significant features

With the trackback technique introduced in Chapter 4, visual interpretation can still be achieved. Fig. 6.14 demonstrates the trackback feature maps according most significant 10 features in each rule.

6.3.6 Comparison with FL-RBF-CNN model

As a comparison, the FL-RBF-CNN introduced in Chapter 4 is also tested with the EBM-XCT dataset. In order to control variables, the benchmark FL-RBF-CNN has 32 features and 3 rules, as the same as ND-RBF-CNN models demonstrated in previous sections. The benchmark has three test cases, which are ‘three class’ case, ‘perfect and defect’ case, and ‘void and crack’ case.

Table 6.7 contains the benchmark results. Generally, it could be found FL-RBF-CNN models cannot predict any sensible results after training. In all three cases, the FL-RBF-CNN models convergent because of the small test loss 95% confidence interval, while the predictions all bias to one specific class. It could be say that a FL-RBF-CNN with 32 features and 3 rules cannot work in this situation.

6.4 Conclusion

In this chapter, a small manufacturing image dataset was created. Because of the small size, both reference linear model, who has the same typography in previous chapters, and ND-RBF-CNN cannot be trained effectively. In order to verify the performance of ND-RBF-CNN, the EBM-XCT dataset was separated into two-class datasets. It was proved that ND-RBF-CNN could be trained in most times and still has the ability of linguistic interpretation. However, during training of ND-RBF-CNNs, the training losses and accuracies has a large variance, which indicates it would be possible to improve robustness by optimisubg the initialisation method of ND-RBF-CNNs.

Table 6.7: Test results of FL-RBF-CNN with 32 features and 3 rules on EBM-XCT dataset

		Accuracy/%			
Test case	Test loss	Overall	Perfect	Void	Crack
Three class	0.6691	33.33	0.00	100.00	0.00
Perfect and defect	0.2267 ± 0.0010	66.67	0.00	100.00	
Void and crack	0.2731 ± 0.0020	50.00	—	100.00	0.00

7 Enhancing network interpretability via constrained transfer learning

Although the model got a high accuracy in previous chapter, the training robustness was not in an ideal situation. A transfer learning model was trained to solve this problem.

In this chapter, a transfer learning model training procedure was tested and measured. Transfer learning has been proved to be a rigid method for image classification with a well-pre-trained model.s

7.1 Introduction

Transfer learning is a method using knowledge gained while solving one problem and applying it to a different but related problem [59]. With transfer learning, computational expensive models, such as ResNets [30] and VGG19, could be adopted on another problem with less training time [64, 45, 38, 35].

Transfer learning cannot only be applied in ordinary CNNs, but also other form neural networks. Yeganejou and Dick [93] introduced a FL model combining ResNets and used transfer learning to initialise the FL layer parameters.

7.2 Methodology

Transfer learning may be a method to solve the initialisation problems with RBF-CNN and ND-RBF-CNN. In Chapter 3, the RBF layer was raised first time. However, the initialisation of RBF layer was not fixed. In Chapter 3 and Chapter 4, the centres of Gaussian membership functions were generated by uniform distribution, and the variances of Gaussian membership functions were initialised as a number. In Chapter 5, 6, and 7, the centres of Gaussian membership functions were initialised as a number, while the variances of Gaussian membership functions were fixed to a constant. There was no methodology for using initialisation methods above except trying-error. All of these values were selected after multiple times of trying-error.

Algorithm 2 introduces the procedures to complete the transfer learning with parameter initialisation for ND-RBF-CNN models. At the beginning, an ordinary CNN model would be trained as usual till the prediction accuracy reach expectations. Afterwards, An ND-RBF-CNN sharing the same feature extractor structure with the previous ordinary CNN would be constructed. The parameters of the feature extractor part in the ND-RBF-CNN should be transferred from the ordinary CNN model, and the parameters of the RBF layer in the ND-RBF-CNN could be initialised with Fuzzy C-Means (FCM) algorithm using the target training dataset. Finally, the ND-RBF-CNN model should be trained again with the target training dataset in order to improve the prediction accuracy by reducing the loss.

```
Input: training dataset, validation dataset, ND-RBF-CNN model,
       corresponding CNN model
Output: trained ND-RBF-CNN model
for each training dataset do
    Train the corresponding CNN model with the input training dataset;
    Validate the prediction accuracy of the corresponding CNN model
      with the validation dataset;
    if the prediction accuracy reach expectations then
        Initialise the parameters of RBF layer in the ND-RBF-CNN
          model using FCM algorithm and the input training dataset;
          Transfer the parameters of CNN part in the CNN model into
          ND-RBF-CNN model;
        Train the initialised ND-RBF-CNN model with the training
          dataset as the output;
        Break;
    else
        Continue;
    end
end
```

Algorithm 2: Transfer learning with parameter initialisation algorithm for ND-RBF-CNN models

7.2.1 Fuzzy C-means algorithm

The Fuzzy C-Means (FCM) [7] algorithm is a form of clustering in which a data point could belong to multiple clusters. After years of development, FCM has multiple variants [54, 55, 58]. However, FCM algorithm was found problem with high dimensionality datasets, where the majority of cluster centres are pulled into the overall centre of gravity [89].

The FCM algorithm could be described as follows. Given a finite collection of n elements $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$, the FCM algorithm returns a set of c cluster centres $C = \{\vec{c}_1, \ldots, \vec{c}_n\}$ and a partition matrix $\vec{w} = w_{i,j} \in [0, 1], i = 1, \ldots, n, j = 1, \ldots, c$, where each element $w_{i,j}$ indicates the degree of elements $\vec{x}_i$ belongs to cluster $\vec{c}_j$. The algorithm aims to minimise an objective function:

$$\arg\min_{x} \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^{m} \|\vec{x}_i - \vec{c}_j\|^2, \tag{7.1}$$

where

$$w_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|\vec{x}_i - \vec{c}_j\|}{\|\vec{x}_i - \vec{c}_k\|} \right)^{\frac{2}{m-1}}}$$

7.3 Benchmarks

In this section, there are three datasets applied in benchmarks, which are MNIST dataset, Fashion MNIST dataset, and EBM-XCT dataset. The MNIST and Fashion MNIST dataset are chosen because of their large sample size and general usage. The EBM-XCT dataset introduced in Chapter 6 is applied because of the small sample size and the industrial case.

The model structure is also slightly adjusted as Table 7.1. The models in Chapter 5 and 6 were using hard tanh function to limit the output range of Extractor part. In this Chapter, the hard tanh function is changed to sigmoid function because of its high-order continuity.

Transfer learning method only requires a model with stored knowledge close to target dataset, but not restricts the model type. In the case of initialing RBF layer, the requirement of pre-trained model is that the pre-trained model should have an output whose shape is the same as the input shape of RBF layer. After the training of the 'pre-trained' model, the training dataset would be used as the pre-trained model's input again, while the output of CNN part would be recorded, i.e., the output of the first 7 layer of models described in Table 7.1. With the aid of function `fuzz.cluster.cmeans()` provided in package `skfuzzy` [86], the centres of Gaussian memberships were calculated by FCM algorithm. Afterwards, the cluster centres would be applied as the initial centre values of RBF layer in the transfer-learning ND-RBF-CNN. Finally, the transfer-learning model would be trained again with the training dataset as usual.

7.3.1 MNIST dataset case

In this section, the MNIST dataset is used as the training and test dataset. The MNIST dataset has 60000-sample of training images and 10000-sample of testing images. The training images were further split into two parts randomly, as 50000-samples for training set and 10000-samples for validation (to avoid overfitting). There are two benchmarks tested with MNIST dataset, which

Table 7.1: ND-RBF-CNN architecture

role	**type**	**patch size/stride**	**output size**	**parameters**
	convolution	*depends on the dataset*	—	—
	convolution	*depends on the dataset*	—	—
	maxpooling	*depends on the dataset*	—	—
Extractor	dropout (25%)		—	0
	flatten		—	0
	linear(Sigmoid())		—	—
	dropout (50%)		—	0
Classifier	RBF		*depends on the dataset*	—

are transferring from ND-RBF-CNN models to ND-RBF-CNN models, and transferring from linear CNN (or ordinary CNN) models to ND-RBF-CNN models.

ND-RBF-CNN to ND-RBF-CNN

In this case, an ND-RBF-CNN model is trained first as the 'pre-trained' model, and then the Algorithm 2 is applied to another ND-RBF-CNN for 20 times, and each time both the pre-training model and the transfer-learning model would be terminate after 50 epochs. Table 7.2 shows the benchmark results. Generally, a transferred model would have a lower accuracy except the case of 32 features. Besides, he 95% confidence interval of transferred test accuracy in the case of 16 and 64 features is greater than the 95% confidence interval of transferred test accuracy in the case of 32 features. As a conclusion, in the 32 feature case, transfer learning form an ND-RBF-CNN could reach a better accuracy than the original model ($98.790 \pm 0.073\%$ versus $98.710 \pm 0.097\%$), and a good performance may achieved in the case of 16 features and 64 features ($92.210 \pm 5.031\%$ and $97.610 \pm 1.670\%$ respectively).

Linear CNN to ND-RBF-CNN

In this case, a linear CNN model is trained first as the 'pre-trained' model, and then the Algorithm 2 is applied to one ND-RBF-CNN for 20 times, and each time both the pre-training model and the transfer-learning model would be terminate after 50 epochs. Table 7.3 shows the benchmark results. It could be found that in all three feature size cases, the transferred models could reach a

Table 7.2: Test results of transfer learning from ND-RBF-CNN to ND-RBF-CNN based on MNIST dataset

Feature size	Model type	Test loss	Test accuracy/%
16	original	1.5063 ± 0.0200	96.280 ± 1.492
	transferred	1.5413 ± 0.0500	92.210 ± 5.031
32	original	1.4742 ± 0.0010	98.710 ± 0.097
	transferred	1.4735 ± 0.0010	98.790 ± 0.073
64	original	1.4712	99.010 ± 0.047
	transferred	1.4845 ± 0.0160	97.610 ± 1.670

Table 7.3: Test results of transfer learning from linear CNN to ND-RBF-CNN based on MNIST dataset

Feature size	Model type	Test loss	Test accuracy/%
16	original	0.0544 ± 0.0030	98.620 ± 0.064
	transferred	1.4792 ± 0.0010	98.260 ± 0.100
32	original	0.0352 ± 0.0010	98.980 ± 0.022
	transferred	1.4741	98.730 ± 0.047
64	original	0.0297 ± 0.0010	99.100 ± 0.041
	transferred	1.4723	98.910 ± 0.047

similar test accuracy as the original models.

For MNIST dataset, using the transfer learning methodology could produce a well-trained ND-RBF-CNN, no matter the 'pre-trained' model, i.e., the original model is an ND-RBF-CNN or a linear CNN.

7.3.2 Fashion MNIST dataset case

In this section, the fashion MNIST dataset is used as the training and test dataset. The Fashion MNIST dataset has 60000-sample of training images and 10000-sample of testing images. The training images were further split into

Table 7.4: Test results of transfer learning from linear CNN to ND-RBF-CNN based on Fashion MNIST dataset

Feature size	Model type	Test loss	Test accuracy/%
16	original	0.2607 ± 0.0050	91.780 ± 0.148
	transferred	1.6663 ± 0.0670	79.240 ± 6.974
32	original	0.2326 ± 0.0020	92.150 ± 0.080
	transferred	1.5681 ± 0.0170	89.350 ± 1.782
64	original	0.2179 ± 0.0020	92.440 ± 0.107
	transferred	1.5441 ± 0.0080	91.790 ± 0.794

two parts randomly, as 50000-samples for training set and 10000-samples for validation (to avoid overfitting). There is one benchmark tested with Fashion MNIST dataset, which is transferring from linear CNN models to ND-RBF-CNN models. The reason to deleting ND-RBF-CNN to ND-RBF-CNN tests is that ND-RBF-CNNs may not extract features properly as the low test accuracy on Fashion MNIST shown in Table 5.6.

Linear CNN to ND-RBF-CNN

In this case, a linear CNN model is trained first as the 'pre-trained' model, and then the Algorithm 2 is applied to one ND-RBF-CNN for 20 times, and each time both the pre-training model and the transfer-learning model would be terminate after 50 epochs. Table 7.4 shows the benchmark results. It could be found that the test accuracy of both original and transferred model would increase respectively as the feature size increase. In the case of 64 features, the test accuracy of transferred models would only drop slightly ($91.790 \pm 0.794\%$ versus $92.440 \pm 0.107\%$).

7.3.3 EBM-XCT dataset case

In this section, the EBM-XCT dataset introduced in Chapter 6 was applied. The models would be test in three feature size, which are 16, 32, and 64.

The benchmarks were designed in two groups. The first group would verify if a not-well-trained model would generate a good initial set of fuzzy trigger functions, and another group would use well-trained models. In the not-well-trained group, the dataset was designated as the same as ‘three-class’ case in Chapter 6. While in the well-trained group, the dataset was decided as the same as ‘Void-Crack’ case in Chapter 6.

All benchmarks contained 20 trials. In each trial, both the pre-training model and the transfer-learning model would be terminate after 50 epochs. An early stopper was also set as stopping if training loss stopping decrease for 15 epochs for every model, including ‘pre-training’ models and transfer-learning models. The ‘pre-training’ models are all linear CNNs.

From the training logs in Chapter 6, it could be found that either reference model or ND-RBF-CNN model would not be trained effectively, and thus the prediction label for test dataset would always be the same one. In order to filter these untrained models, a filter was added in this section. This filter would ignore models whose accuracy below 33.33% in three-class case or below 50.00% in two-class case.

Not-well-trained models

Table 7.5 shows the transfer-training results of the ‘three-class’ case. It could be found that the test accuracy of both pre-trained models and the transferred models increases when more features applied. However, the test accuracy of transferred model only increase slightly (from$0.5615 \pm 0.0440\%$ to $0.5881 \pm 0.0390\%$).

Well-trained models

Table 7.6 shows the transfer-training results of the 'Void-Crack' case. The pre-trained model could reach 100% test accuracy in all three feature size cases. Regarding to transferred models, the best test accuracy achieved in the 16 feature case, which was $0.9750 \pm 0.0590\%$. However, the test accuracy of transferred models would drop when feature size increases.

From the above two groups of results with EBM-XCT dataset, it could be found that the proposed transfer learning methodology works in every situation tested, while the performance varies depends on the performance of the pre-trained model. In the situation of not-well-trained models, the transferred model could only achieve a test accuracy of 0.5881 ± 0.0390, which is similar to the performance of direct trained model (0.5733 ± 0.1310). Moreover, in the well-trained model case, the transferred model has a better performance than direct training model (0.9750 ± 0.0590 versus 0.9217 ± 0.1219).

7.3.4 Comparison with results in the previous Chapters

In Chapter 5, the ND-RBF-CNN models were also benchmarked on MNIST dataset and Fashion MNIST dataset. Table 7.7 and Table 7.8 shows the comparison between direct training results obtained in Chapter 5 and transfer learning results got in this Chapter on MNIST dataset and Fashion MNIST

Table 7.5: Training results of the 'three-class' case

Pre-train model feature size	Training rate		Accuracy/%	
	Pre-trained model	Transferred model	Pre-trained model	Transferred model
16	18 / 20 = 90.00%	16 / 18 = 88.89%	0.6250 ± 0.0490	0.5615 ± 0.0440
32	20 / 20 = 100.00%	16 / 20 = 80.00%	0.7858 ± 0.0550	0.5813 ± 0.0480
64	20 / 20 = 100.00%	14 / 20 = 70.00%	0.8183 ± 0.0630	0.5881 ± 0.0390

Table 7.6: Training results of the 'Void-Crack' case

Pre-train model feature size	Training rate		Accuracy/%	
	Pre-trained model	Transferred model	Pre-trained model	Transferred model
16	16 / 20 = 80.00%	8 / 16 = 50.00%	1.0000	0.9750 ± 0.0590
32	20 / 20 = 100.00%	10 / 20 = 50.00%	1.0000	0.9175 ± 0.1080
64	20 / 20 = 100.00%	10 / 20 = 50.00%	1.0000	0.8700 ± 0.1160

Table 7.7: Comparison between direct training and transfer learning on MNIST dataset

Feature size	Model type	Test loss	Test accuracy/%
16	direct	1.9301 ± 0.1060	61.26 ± 12.44
	transferred	1.4792 ± 0.0010	98.260 ± 0.100
32	direct	1.6443 ± 0.0920	81.14 ± 10.48
	transferred	1.4741	98.730 ± 0.047
64	direct	1.5042 ± 0.0230	95.72 ± 2.29
	transferred	1.4723	98.910 ± 0.047

Table 7.8: Comparison between direct training and transfer learning on Fashion MNIST dataset

Feature size	Model type	Test loss	Test accuracy/%
16	direct	1.8990 ± 0.0980	58.73 ± 10.18
	transferred	1.6663 ± 0.0670	79.240 ± 6.974
32	direct	1.7259 ± 0.0380	73.39 ± 4.27
	transferred	1.5681 ± 0.0170	89.350 ± 1.782
64	direct	1.6713 ± 0.0270	78.93 ± 2.67
	transferred	1.5441 ± 0.0080	91.790 ± 0.794

dataset respectively. It could be found that the test accuracy of the transfer learning models is better than the one of direct training models in every case.

7.4 Conclusion

In this chapter, an initialisation method for RBF layers combined with CNN was proposed and tested. The transfer learning method inspired the initialisation procedures. The initialisation was implemented with FCM algorithm. Besides, the feature extractor part is also transferred from a pre-trained model.

Generally, the results of transfer learning is better. For large datasets, says MNIST and Fashion MNIST dataset, the test accuracy of transfer learning models is better than the one of direct training models. Similarly, a transfer learning model could also reach higher accuracy than a direct training model in small dataset cases.

8 Conclusion

In this research work, we have elaborated a number of methodologies to interpret a deep CNN features into linguistic fuzzy rules. Generally, a CNN combines with FL RBF networks could be regard as models containing a feature extractor, i.e. CNN part, and the classifier, says FL RBF part. With different FL RBF part, we have RBF-CNN models and ND-RBF-CNN models.

Although ND-RBF-CNN as the successor of RBF-CNN could be applied more generally, the prediction performance was decreased. In order to remains the prediction performance, some work were paid. In Chapter 6, the input dataset was separated into different groups, which decreased the difficulty of prediction. In Chapter 7, a general initialisation methodology was raised, and the prediction performance was increased by applying transfer learning.

However, for both RBF-CNN and ND-RBF-CNN, the interpretability of linguistic fuzzy rule are the same. In Chapter 4, the method to interpret rule bases was introduced for RBF-CNN. In Chapter 6, the same method was tested on ND-RBF-CNN.

8.1 Future work

Theoretically, RBF-CNN, and ND-RBF-CNN especially, could be combined with other deep CNNs. However, there was only one CNN structure was tested. If the ND-RBF-CNN were tested and proved that the combination with other CNN only at a tiny cost, the prediction process of deep CNNs would be more transparent.

ND-RBF-CNN could be combined with other defuzzification method rather than Singleton. In Chapter 5, the defuzzy layer was removed because of its single output and the round function inside. If possible, a defuzzy matrix rather than polynomial could be applied.

In Chapter 6, for the prediction accuracy, the EBM-XCT dataset was seperated into two cascades. The model could be extended with the same method in [1], i.e. reconstruction two cascades into one whole model.

Bibliography

[1] Davis Bonsu Agyemang and Mohamed Bader. "Surface Crack Detection Using Hierarchal Convolutional Neural Network". In: *Advances in Computational Intelligence Systems*. Ed. by Zhaojie Ju et al. Vol. 1043. Cham: Springer International Publishing, 2020, pp. 173–186. URL: http://link.springer.com/10.1007/978-3-030-29933-0_15 (visited on 10/11/2019).

[2] Derya Eren Akyol and G. Mirac Bayhan. "A Review on Evolution of Production Scheduling with Neural Networks". In: *Computers & Industrial Engineering* 53.1 (Aug. 2007), pp. 95–122. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360835207000666 (visited on 03/16/2019).

[3] Shun-ichi Amari. "Backpropagation and Stochastic Gradient Descent Method". In: *Neurocomputing* 5.4 (June 1, 1993), pp. 185–196. URL: http://www.sciencedirect.com/science/article/pii/092523129390006O (visited on 12/24/2020).

[4] Plamen P. Angelov and Xiaowei Gu. "Deep Rule-Based Classifier with Human-Level Performance and Characteristics". In: *Information Sciences*

463–464 (Oct. 2018), pp. 196–213. URL: https://linkinghub.elsevier.com/retrieve/pii/S0020025518304894 (visited on 10/18/2018).

[5] Serge Belongie, Jitendra Malik, and Jan Puzicha. "Shape Matching and Object Recognition Using Shape Contexts". In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* 24.24 (2002), p. 14.

[6] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. "Advances in Optimizing Recurrent Networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing.* 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. May 2013, pp. 8624–8628.

[7] James C. Bezdek, Robert Ehrlich, and William Full. "FCM: The Fuzzy c-Means Clustering Algorithm". In: *Computers & Geosciences* 10.2 (Jan. 1, 1984), pp. 191–203. URL: http://www.sciencedirect.com/science/article/pii/0098300484900207 (visited on 08/15/2019).

[8] David S. Broomhead and David Lowe. *Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks.* Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[9] M. Chandrasekaran et al. "Application of Soft Computing Techniques in Machining Performance Prediction and Optimization: A Literature Review". In: *International Journal of Advanced Manufacturing Technology* 46.5-8 (Jan. 2010), pp. 445–464. URL: http://link.springer.com/10.1007/s00170-009-2104-x.

[10] Min-You Chen and D.A. Linkens. "A Systematic Neuro-Fuzzy Modeling Framework with Application to Material Property Prediction". In: *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)* 31.5 (Oct. 2001), pp. 781–790. URL: `http://ieeexplore.ieee.org/document/956039/` (visited on 06/26/2019).

[11] A. K. Choudhary, J. A. Harding, and M. K. Tiwari. "Data Mining in Manufacturing: A Review Based on the Kind of Knowledge". In: *Journal of Intelligent Manufacturing* 20.5 (Oct. 2009), pp. 501–521. URL: `http://link.springer.com/10.1007/s10845-008-0145-x` (visited on 06/26/2019).

[12] Dan C Ciresan et al. "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: (2011), p. 6.

[13] Dan Claudiu Ciresan et al. "Convolutional Neural Network Committees for Handwritten Character Classification". In: *2011 International Conference on Document Analysis and Recognition*. 2011 International Conference on Document Analysis and Recognition (ICDAR). Beijing, China: IEEE, Sept. 2011, pp. 1135–1139. URL: `http://ieeexplore.ieee.org/document/6065487/` (visited on 10/21/2020).

[14] Dan Claudiu Ciresan et al. "Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition". In: *Neural Computation* 22.12 (Dec. 2010), pp. 3207–3220. arXiv: `1003.0358`. URL: `http://arxiv.org/abs/1003.0358` (visited on 10/21/2020).

[15] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. "Multi-Column Deep Neural Networks for Image Classification". Feb. 13, 2012. arXiv: `1202.2745 [cs]`. URL: `http://arxiv.org/abs/1202.2745` (visited on 06/26/2019).

[16] Dennis Decoste. "Training Invariant Support Vector Machines". In: (2002), p. 30.

[17] Jia Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition.* 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops). Miami, FL: IEEE, June 2009, pp. 248–255. URL: `https://ieeexplore.ieee.org/document/5206848/` (visited on 06/26/2019).

[18] Li Deng and Dong Yu. "Deep Convex Net: A Scalable Architecture for Speech Pattern Classification". In: (2011), p. 4.

[19] Yue Deng et al. "A Hierarchical Fused Fuzzy Deep Neural Network for Data Classification". In: *IEEE Transactions on Fuzzy Systems* 25.4 (Aug. 2017), pp. 1006–1012. URL: `10/gbrskx` (visited on 06/26/2019).

[20] Wangzhe Du et al. "Automated Detection of Manufacturing Defects Based on Deep Learning".

[21] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (Jul 2011), pp. 2121–2159. URL: `http://www.jmlr.org/papers/v12/duchi11a.html` (visited on 02/18/2019).

[22] M.J. Gacto, R. Alcalá, and F. Herrera. “Interpretability of Linguistic Fuzzy Rule-Based Systems: An Overview of Interpretability Measures”. In: *Information Sciences* 181.20 (Oct. 2011), pp. 4340–4360. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0020025511001034` (visited on 04/14/2022).

[23] Xavier Glorot and Yoshua Bengio. “Understanding the Difficulty of Training Deep Feedforward Neural Networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 249–256.

[24] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. 2011, pp. 315–323.

[25] Gong, Goldman, and Lach. “Deepmotion: A Deep Convolutional Neural Network on Inertial Body Sensors for Gait Assessment in Multiple Sclerosis*”. In: *2016 IEEE Wireless Health (WH)*. 2016 IEEE Wireless Health (WH). Bethesda, MD, USA: IEEE, Oct. 2016, pp. 1–8. URL: `http://ieeexplore.ieee.org/document/7764572/` (visited on 06/26/2019).

[26] Ian Goodfellow et al. *Deep Learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[27] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[28] Xiaowei Gu et al. "A Massively Parallel Deep Rule-Based Ensemble Classifier for Remote Sensing Scenes". In: *IEEE Geoscience and Remote Sensing Letters* 15.3 (Mar. 2018), pp. 345–349. URL: `http://ieeexplore.ieee.org/document/8281516/` (visited on 06/26/2019).

[29] K Harris, G. L. Erickson, and R. E. Schwer. "MAR M 247 DERIVATIONS - CM 247 LC DS ALLOY CMSX SINGLE CRYSTAL ALLOYS PROPERTIES & PERFORMANCE". In: (1984), p. 10.

[30] Kaiming He et al. "Deep Residual Learning for Image Recognition". Dec. 10, 2015. arXiv: `1512.03385 [cs]`. URL: `http://arxiv.org/abs/1512.03385` (visited on 02/18/2019).

[31] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV).* 2015 IEEE International Conference on Computer Vision (ICCV). Santiago, Chile: IEEE, Dec. 2015, pp. 1026–1034. URL: `http://ieeexplore.ieee.org/document/7410480/` (visited on 06/26/2019).

[32] Wei Hu, Lechao Xiao, and Jeffrey Pennington. "Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks". Jan. 16, 2020. arXiv: `2001.05992 [cs, math, stat]`. URL: `http://arxiv.org/abs/2001.05992` (visited on 12/24/2020).

[33] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". Feb. 10,

2015. arXiv: 1502.03167 [cs]. URL: http://arxiv.org/abs/1502.03167 (visited on 06/26/2019).

[34] M. I. Jordan and T. M. Mitchell. "Machine Learning: Trends, Perspectives, and Prospects". In: *Science* 349.6245 (July 17, 2015), pp. 255–260. URL: http://www.sciencemag.org/cgi/doi/10.1126/science.aaa8415 (visited on 06/26/2019).

[35] Thommen George Karimpanal and Roland Bouffanais. "Self-Organizing Maps for Storage and Transfer of Knowledge in Reinforcement Learning". In: *Adaptive Behavior* 27.2 (Apr. 2019), pp. 111–126. arXiv: 1811.08318. URL: http://arxiv.org/abs/1811.08318 (visited on 12/24/2020).

[36] Balázs Kégl and Róbert Busa-Fekete. "Boosting Products of Base Classifiers". In: *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*. The 26th Annual International Conference. Montreal, Quebec, Canada: ACM Press, 2009, pp. 1–8. URL: http://portal.acm.org/citation.cfm?doid=1553374.1553439 (visited on 10/21/2020).

[37] D. Keysers et al. "Deformation Models for Image Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.8 (Aug. 2007), pp. 1422–1435.

[38] SanaUllah Khan et al. "A Novel Deep Learning Based Framework for the Detection and Classification of Breast Cancer Using Transfer Learning". In: *Pattern Recognition Letters* 125 (July 1, 2019), pp. 1–6. URL: http://

www.sciencedirect.com/science/article/pii/S0167865519301059 (visited on 12/24/2020).

[39] Diederik P Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. 2014. arXiv: 1412.6980.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 03/14/2019).

[41] Fabien Lauer, Ching Y. Suen, and Gérard Bloch. “A Trainable Feature Extractor for Handwritten Digit Recognition”. In: *Pattern Recognition* 40.6 (June 2007), pp. 1816–1824. URL: https://linkinghub.elsevier.com/retrieve/pii/S0031320306004250 (visited on 10/21/2020).

[42] Y. LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324.

[43] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. URL: http://www.nature.com/articles/nature14539 (visited on 06/26/2019).

[44] Chi Liu et al. “SetSVM: An Approach to Set Classification in Nuclei-Based Cancer Detection”. In: *IEEE Journal of Biomedical and Health Informatics* 23.1 (Jan. 2019), pp. 351–361. URL: https://ieeexplore.ieee.org/document/8286915/ (visited on 06/26/2019).

[45] Shaopeng Liu, Guohui Tian, and Yuan Xu. “A Novel Scene Classification Model Combining ResNet Based Transfer Learning and Data Augmentation with a Filter”. In: *Neurocomputing* 338 (Apr. 21, 2019), pp. 191–206. URL: `http://www.sciencedirect.com/science/article/pii/S0925231219301833` (visited on 12/24/2020).

[46] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774. URL: `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf` (visited on 10/15/2019).

[47] Tiago Mazzutti, Mauro Roisenberg, and Paulo José de Freitas Filho. “INFGMN – Incremental Neuro-Fuzzy Gaussian Mixture Network”. In: *Expert Systems with Applications* 89 (Dec. 2017), pp. 160–178. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0957417417305092` (visited on 06/26/2019).

[48] Volodymyr Mnih et al. “Human-Level Control through Deep Reinforcement Learning”. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. URL: `http://www.nature.com/articles/nature14236` (visited on 06/26/2019).

[49] K.-R. Muller et al. “An Introduction to Kernel-Based Learning Algorithms”. In: *IEEE Transactions on Neural Networks* 12.2 (Mar. 2001), pp. 181–201. URL: `http://ieeexplore.ieee.org/document/914517/` (visited on 06/26/2019).

[50] Ander Muniategui et al. “Spot Welding Monitoring System Based on Fuzzy Classification and Deep Learning”. In: *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). Naples, Italy: IEEE, July 2017, pp. 1–6. URL: `http://ieeexplore.ieee.org/document/8015618/` (visited on 06/26/2019).

[51] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[52] Michael A. Nielsen. *Neural Networks and Deep Learning*. 2015. URL: `http://neuralnetworksanddeeplearning.com` (visited on 03/20/2019).

[53] Rui Pedro Paiva and António Dourado. “Interpretability and Learning in Neuro-Fuzzy Systems”. In: *Fuzzy Sets and Systems*. Hybrid Methods for Adaptive Systems 147.1 (Oct. 1, 2004), pp. 17–38. URL: `http://www.sciencedirect.com/science/article/pii/S0165011403005128` (visited on 03/31/2020).

[54] N.R. Pal, K. Pal, and J.C. Bezdek. “A Mixed C-Means Clustering Model”. In: *Proceedings of 6th International Fuzzy Systems Conference*. 6th International Fuzzy Systems Conference. Vol. 1. Barcelona, Spain: IEEE, 1997, pp. 11–21. URL: `http://ieeexplore.ieee.org/document/616338/` (visited on 07/28/2020).

[55] N.R. Pal et al. “A Possibilistic Fuzzy C-Means Clustering Algorithm”. In: *IEEE Transactions on Fuzzy Systems* 13.4 (Aug. 2005), pp. 517–530.

[56] George Panoutsos and Mahdi Mahfouf. "A Neural-Fuzzy Modelling Framework Based on Granular Computing: Concepts and Applications". In: *Fuzzy Sets and Systems* 161.21 (Nov. 2010), pp. 2808–2830. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0165011410002484` (visited on 06/26/2019).

[57] George Panoutsos et al. "A Generic Framework for Enhancing the Interpretability Of Granular Computing-Based Information". In: *2010 5th IEEE International Conference Intelligent Systems*. 2010 5th IEEE International Conference Intelligent Systems. IEEE. July 2010, pp. 19–24. URL: `10/bs27pp`.

[58] Bruno A. Pimentel and Renata M. C. R. de Souza. "A Multivariate Fuzzy C-Means Method". In: *Applied Soft Computing* 13.4 (Apr. 1, 2013), pp. 1592–1607. URL: `http://www.sciencedirect.com/science/article/pii/S1568494612005686` (visited on 12/24/2020).

[59] Lorien Y Pratt. "Discriminability-Based Transfer between Neural Networks". In: *Advances in Neural Information Processing Systems*. 1993, pp. 204–211.

[60] Stanton R. Price, Steven R. Price, and Derek T. Anderson. "Introducing Fuzzy Layers for Deep Learning". In: *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). June 2019, pp. 1–6.

[61] Paolo Priore et al. "A Review of Machine Learning in Dynamic Scheduling of Flexible Manufacturing Systems". In: *Artificial Intelligence for Engi-*

neering Design, Analysis and Manufacturing 15.3 (June 2001), pp. 251–263. URL: https://www.cambridge.org/core/product/identifier/S0890060401153059/type/journal_article (visited on 06/26/2019).

[62] S. Joe Qin. “Survey on Data-Driven Industrial Process Monitoring and Diagnosis”. In: *Annual Reviews in Control* 36.2 (Dec. 2012), pp. 220–234. URL: https://linkinghub.elsevier.com/retrieve/pii/S1367578812000399 (visited on 06/26/2019).

[63] Mohd Fadzil Faisae Rashid, Windo Hutabarat, and Ashutosh Tiwari. “A Review on Assembly Sequence Planning and Assembly Line Balancing Optimisation Using Soft Computing Approaches”. In: *The International Journal of Advanced Manufacturing Technology* 59.1-4 (Mar. 2012), pp. 335–349. URL: http://link.springer.com/10.1007/s00170-011-3499-8 (visited on 06/26/2019).

[64] E. Rezende et al. “Malicious Software Classification Using Transfer Learning of ResNet-50 Deep Neural Network”. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA).* 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). Dec. 2017, pp. 1011–1014.

[65] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *CoRR* abs/1602.04938 (2016). arXiv: 1602.04938. URL: http://arxiv.org/abs/1602.04938.

[66] F. A. Saiz et al. “A Robust and Fast Deep Learning-Based Method for Defect Classification in Steel Surfaces”. In: *2018 International Conference on Intelligent Systems (IS)*. 2018 International Conference on Intelligent Systems (IS). Sept. 2018, pp. 455–460.

[67] Ruslan Salakhutdinov and Geoffrey Hinton. “Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure”. In: (2007), p. 8.

[68] Andrew M Saxe, James L McClelland, and Surya Ganguli. “Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks”. 2013. arXiv: `1312.6120`.

[69] Rob Schapire. *Computer Science 511 Theoretical Machine Learning*. Computer Science Department, Princeton University, Feb. 2008. URL: `https://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf`.

[70] Tom Schaul, Ioannis Antonoglou, and David Silver. “Unit Tests for Stochastic Optimization”. Dec. 20, 2013. arXiv: `1312.6055 [cs]`. URL: `http://arxiv.org/abs/1312.6055` (visited on 06/26/2019).

[71] Jürgen Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0893608014002135` (visited on 06/26/2019).

[72] Tiziana Segreto et al. “Vibration Sensor Monitoring of Nickel-Titanium Alloy Turning for Machinability Evaluation”. In: *Sensors* 17.12 (Dec. 12, 2017), p. 2885. URL: `http://www.mdpi.com/1424-8220/17/12/2885` (visited on 06/26/2019).

[73] S. Al-sharhan et al. “Fuzzy Entropy: A Brief Survey”. In: *10th IEEE International Conference on Fuzzy Systems. (Cat. No.01CH37297)*. 10th Annual IEEE Conference on Fuzzy Systems. Vol. 2. Melbourne, Vic., Australia: IEEE, Dec. 2001, pp. 1135–1139. URL: `http://ieeexplore.ieee.org/document/1008855/` (visited on 06/26/2019).

[74] Teena Sharma et al. “Fuzzy Based Pooling in Convolutional Neural Network for Image Classification”. In: *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). June 2019, pp. 1–6.

[75] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Learning Important Features Through Propagating Activation Differences”. Apr. 9, 2017. arXiv: `1704.02685 [cs]`. URL: `http://arxiv.org/abs/1704.02685` (visited on 10/15/2019).

[76] David Silver et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. URL: `http://www.nature.com/articles/nature16961` (visited on 06/26/2019).

[77] David Silver et al. “Mastering the Game of Go without Human Knowledge”. In: *Nature* 550.7676 (Oct. 2017), pp. 354–359. URL: `http://www.nature.com/articles/nature24270` (visited on 06/26/2019).

[78] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. Sept. 4, 2014. arXiv:

1409.1556 [cs]. URL: http://arxiv.org/abs/1409.1556 (visited on 06/26/2019).

[79] Jost Tobias Springenberg et al. "Striving for Simplicity: The All Convolutional Net". 2014. arXiv: 1412.6806.

[80] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[81] Linzhi Su and Xin Cao. "Fuzzy Autoencoder for Multiple Change Detection in Remote Sensing Images". In: *Journal of Applied Remote Sensing* 12.03 (Aug. 31, 2018), p. 1. URL: https://www.spiedigitallibrary.org/journals/journal-of-applied-remote-sensing/volume-12/issue-03/035014/Fuzzy-autoencoder-for-multiple-change-detection-in-remote-sensing-images/10.1117/1.JRS.12.035014.full (visited on 06/26/2019).

[82] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic Attribution for Deep Networks". Mar. 3, 2017. arXiv: 1703.01365 [cs]. URL: http://arxiv.org/abs/1703.01365 (visited on 10/15/2019).

[83] Christian Szegedy et al. "Going Deeper with Convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Boston, MA, USA: IEEE, June 2015, pp. 1–9. URL: http://ieeexplore.ieee.org/document/7298594/ (visited on 06/26/2019).

[84] S. Tammas-Williams et al. “XCT Analysis of the Influence of Melt Strategies on Defect Population in Ti–6Al–4V Components Manufactured by Selective Electron Beam Melting”. In: *Materials Characterization* 102 (Apr. 1, 2015), pp. 47–61. URL: `http://www.sciencedirect.com/science/article/pii/S104458031500039X` (visited on 12/24/2020).

[85] *Torch.Nn — PyTorch 1.7.0 Documentation.* URL: `https://pytorch.org/docs/stable/nn.html#` (visited on 12/24/2020).

[86] Josh Warner et al. *JDWarner/Scikit-Fuzzy: Scikit-Fuzzy Version 0.4.2.* Version v0.4.2. Zenodo, Nov. 2019. URL: `https://doi.org/10.5281/zenodo.3541386`.

[87] Joseph M Wells. “Quantitative XCT Evaluation of Porosity in an Aluminum Alloy Casting”. In: *Shape Casting: 2nd International Symposium.* 2007, pp. 978–.

[88] P.J. Werbos. “Backpropagation through Time: What It Does and How to Do It”. In: *Proceedings of the IEEE* 78.10 (Oct. 1990), pp. 1550–1560. URL: `http://ieeexplore.ieee.org/document/58337/` (visited on 06/26/2019).

[89] Roland Winkler, Frank Klawonn, and Rudolf Kruse. “Fuzzy C-Means in High Dimensional Spaces”. In: *International Journal of Fuzzy System Applications* 11 (June 2010).

[90] Zhen Xi and George Panoutsos. “Interpretable Convolutional Neural Networks Using a Rule-Based Framework for Classification”. In: *Intelligent Systems: Theory, Research and Innovation in Applications.* Ed. by Ricardo

Jardim-Goncalves et al. Studies in Computational Intelligence. Cham: Springer International Publishing, 2020, pp. 1–24. URL: `https://doi.org/10.1007/978-3-030-38704-4_1` (visited on 03/10/2020).

[91] Zhen Xi and George Panoutsos. “Interpretable Machine Learning: Convolutional Neural Networks with RBF Fuzzy Logic Classification Rules”. In: *2018 International Conference on Intelligent Systems (IS)*. 2018 International Conference on Intelligent Systems (IS). Sept. 2018, pp. 448–454.

[92] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms”. Aug. 25, 2017. arXiv: `1708.07747 [cs, stat]`. URL: `http://arxiv.org/abs/1708.07747` (visited on 06/26/2019).

[93] Mojtaba Yeganejou and Scott Dick. “Improved Deep Fuzzy Clustering for Accurate and Interpretable Classifiers”. In: *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). June 2019, pp. 1–7.

[94] Matthew D. Zeiler. “ADADELTA: An Adaptive Learning Rate Method”. Dec. 22, 2012. arXiv: `1212.5701 [cs]`. URL: `http://arxiv.org/abs/1212.5701` (visited on 06/26/2019).

[95] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *CoRR* abs/1311.2901 (2013). arXiv: `1311.2901`. URL: `http://arxiv.org/abs/1311.2901`.

[96] Xiangyu Zhang et al. “Accelerating Very Deep Convolutional Networks for Classification and Detection”. Nov. 18, 2015. arXiv: `1505.06798 [cs]`. URL: `http://arxiv.org/abs/1505.06798` (visited on 12/23/2020).

Fuzzy layer implementations in Chapter 3 and Chapter 4

As described in Chapter 4, there is a small correction used in Chapter 4 comparing with the model used in chapter 3. The relevant code snippets are listed in List 1 and List 2 respectively. In these two snippets, a `Python` class named *RBFLayer* was defined to act as mathematical model defined from Eq. (2.7) to (2.9). For the calculation performance, these equations were implemented in vector form in order to accept multiple inputs one time, i.e. Single Instruction, Multiple Data (SIMD).

```python
import keras
from keras import backend as K
from keras.engine.topology import InputSpec, Layer


class RBFLayer(Layer):
    def __init__(self, ruleNumber, **kwargs):
        self.ruleNumber = ruleNumber
        super(RBFLayer, self).__init__(**kwargs)
        self.input_spec = InputSpec(min_ndim=2)

    def build(self, input_shape):
        assert len(input_shape) >= 2
        input_dim = input_shape[-1]
        # Fuzzy center matrix
        self.c = self.add_weight(
            name='fuzzyCenter',
            shape=(self.ruleNumber, input_dim),
            initializer=keras.initializers.RandomUniform(minval=0.0,
                                                          maxval=1.0,
                                                          seed=None),
            constraint=keras.constraints.MinMaxNorm(min_value=0, max_value=1),
            trainable=True)
        # Fuzzy variance matrix
        self.sigma = self.add_weight(
            name='fuzzyVariance',
            shape=(self.ruleNumber, input_dim),
            initializer=keras.initializers.Constant(value=0.3),
            constraint=keras.constraints.MinMaxNorm(min_value=0.2,
                                                     max_value=0.6),
            trainable=True)
        self.input_spec = InputSpec(min_ndim=2, axes={-1: input_dim})
        super(RBFLayer, self).build(input_shape)

    def call(self, x):
        normalized_x = K.batch_normalization(x,
                                             mean=0,
                                             var=1,
                                             beta=0.5,
                                             gamma=0.5,
                                             epsilon=1e-3)
        extended_x = K.repeat(normalized_x, self.ruleNumber)
        normedX = -K.sum(K.square((extended_x - self.c) / self.sigma), axis=2)
        return K.softmax(normedX)

    def compute_output_shape(self, input_shape):
        assert input_shape and len(input_shape) >= 2
        assert input_shape[-1]
        output_shape = list(input_shape)
        output_shape[-1] = self.ruleNumber
        return tuple(output_shape)
```

Listing 1: Fuzzy layer implementations in Chapter 3

```
import keras
import tensorflow as tf
from keras import backend as K
from keras.engine.topology import InputSpec, Layer


class RBFLayer(Layer):
    def __init__(self, rule_number, **kwargs):
        self.rule_number = rule_number
        super(RBFLayer, self).__init__(**kwargs)
        self.input_spec = InputSpec(min_ndim=2)

    def build(self, input_shape):
        assert len(input_shape) >= 2
        input_dim = input_shape[-1]
        # Fuzzy center matrix
        self.center = self.add_weight(
            name='fuzzyCenter',
            shape=(self.rule_number, input_dim),
            initializer=keras.initializers.RandomUniform(minval=0.0,
                                                         maxval=1.0,
                                                         seed=None),
            constraint=keras.constraints.MinMaxNorm(min_value=0, max_value=1),
            trainable=True)
        # Fuzzy variance matrix
        self.sigma2 = self.add_weight(
            name='fuzzyVariance',
            shape=(self.rule_number, input_dim),
            initializer=keras.initializers.Constant(value=0.3),
            constraint=keras.constraints.MinMaxNorm(min_value=0.2,
                                                    max_value=0.6),
            trainable=True)
        self.input_spec = InputSpec(min_ndim=2, axes={-1: input_dim})
        super(RBFLayer, self).build(input_shape)

    def call(self, x):
        normalized_x = K.batch_normalization(x,
                                             mean=0,
                                             var=1,
                                             beta=0.5,
                                             gamma=0.5,
                                             epsilon=1e-3)
        extended_x = K.repeat(normalized_x, self.rule_number)
        norm_x = K.sum(tf.subtract(
            0.0,
            K.square(extended_x - self.center) / self.sigma2),
                       axis=2)
        return K.softmax(norm_x)

    def compute_output_shape(self, input_shape):
        assert input_shape and len(input_shape) >= 2
        assert input_shape[-1]
        output_shape = list(input_shape)
        output_shape[-1] = self.rule_number
        return tuple(output_shape)
```

Listing 2: Fuzzy layer implementations in Chapter 4

The code listed in the above two lists are most the same except the definition of $\vec{\sigma}$, corresponding to line 25–31 in List 1 and line 26–31 in List 2; and the method `RBFLayer.call(x)`, corresponding to line 35–44 in List 1 and line 36–48 in List 2. In order to simplify the equation, the following equations would only use scalar form.

In the both versions of equation definitions, the corresponding key equation would be

$$\mu_{ij}(x_j) = \exp\left(-\frac{(x_j - c_{ij})^2}{\sigma_{ij}^2}\right), \tag{1}$$

where $\vec{c}$ corresponds to `RBFLayer.center` correctly in both version, and $\vec{\sigma}$ corresponds to `RBFLayer.sigma` ideally. However, because of the typo-mistake happened in the first version, the model in Chapter 3 was not the same as descriptions.

In Chapter 3, there was a claim that the value range of elements of $\vec{\sigma^2}$ was clipped in the range of $[0.2, 0.6]$. However, in List 1, the actual clipped value range was $[0.2^2, 0.6^2] = [0.04, 0.36]$. This happened at Line 43 in List 1, where `self.sigma` was also included in function `K.square()`. At line 46 in List 2, `self.sigma2`, denoting σ^2, was moved outside of function `K.square()`.