DISSERTATION

On

Developing Fuzzy Dilation and Pooling Based Deep Convolutional Neural Network

Thesis Submitted in the partial fulfillment of the requirements for the degree of

Master of Engineering

In

Computer Science & Engineering

Submitted By

Rangan Das

Examination Roll number: M4CSE19030

Registration number: 140768 of 2017-2018

Under Guidance of

Prof. (Dr.) Ujjwal Maulik

Jadavpur University

Dept. of Computer Science & Engineering

Faculty Council of Engineering and Technology

JADAVPUR UNIVERSITY

KOLKATA – 700032

2018 – 2019

Department of Computer Science & Engineering

Faculty Council of Engineering and Technology

JADAVPUR UNIVERSITY, KOLKATA – 700032

Certificate of Recommendation

This is to certify that Rangan Das (Examination Roll number: M4CSE19030) has completed his dissertation entitled "Developing Fuzzy Dilation and Pooling Based Deep Convolutional Neural Network", under the supervision and guidance of Prof. (Dr.) Ujjwal Maulik, Jadavpur University, Kolkata. We are satisfied with his work, which is being presented for the partial fulfillment of the degree of Master of Engineering in Computer Science & Engineering, Jadavpur University, Kolkata - 700032.

Prof. (Dr.) Ujjwal Maulik
Teacher in Charge of Thesis
Professor, Dept. of Computer Science & Engineering
Jadavpur University, Kolkata – 700 032

Prof. (Dr.) Mahantapas Kundu
HOD, Dept. of Computer Science & Engineering
Jadavpur University, Kolkata – 700 032

Prof. (Dr.) Chiranjib Bhattacharjee
Dean, Faculty Council of Engineering and Technology
Jadavpur University, Kolkata – 700 032

Faculty Council of Engineering and Technology

JADAVPUR UNIVERSITY, KOLKATA – 700032

Certificate of Approval[*]

The foregoing thesis entitled "Developing Fuzzy Dilation and Pooling Based Deep Convolutional Neural Network", is hereby approved as a creditable study of Master of Engineering in Computer Science & Engineering and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion therein but approve this thesis only for the purpose for which it is submitted.

Final Examination for Evaluation of the Thesis

_____

_____

Signature of Examiners

[*] *Only in case the thesis is approved.*

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of her Master of Engineering in Computer Science & Engineering.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name:	Rangan Das
Exam Roll Number:	M4CSE19030
Registration number:	140768 of 2017-2018
Thesis Title:	Developing Fuzzy Dilation and Pooling Based Deep Convolutional Neural Network

Signature with date:

Acknowledgements

This dissertation would not have been possible without the support of many people. First and foremost, I would like to thank my advisor, Dr. Ujjwal Maulik, Professor, Department of Computer Science and Engineering, Jadavpur University, Kolkata. He has provided with provided me with the perfect balance of guidance and freedom. He is the first person who introduced me to the world of deep learning and provided me with the guidance that was essential in making this dissertation. I also want to thank him for his perspective and helping me pursue and define projects with more impact.

I would also like to thank Mr. Sagnik Sen, Research Scholar, Department of Computer Science and Engineering, Jadavpur University for his immense contribution to my knowledge of deep learning and bioinformatics. A lot of this dissertation would not have happened without the long, brainstorming sessions and the technical discussions we had during the coffee breaks at the university. He has questioned me at every step of the way, allowing me to make numerous improvements on my project. In many ways, he has shown me the path for how to apply, develop and understand deep learning.

This dissertation would be incomplete without the support of Ms. Ashmita Dey, Research Scholar, Department of Computer Science and Engineering, Jadavpur University. I would not be able to complete this dissertation without her support and encouragement. She helped me a lot by proof-reading my papers and suggesting necessary changes. I'm also thankful to Mrs. Kakuli Mishra, Research Scholar, Department of Computer Science and Engineering, Jadavpur University for her invaluable suggestions regarding this dissertation.

Finally, I would like to thank Jadavpur University for providing me with the opportunity to work in such a productive environment. I would also like to thank all the other professors and research scholars of the department who have extended their helping hands whenever I needed help.

Date:

Place:

Rangan Das, M.E in CSE,

Exam Roll: M4CSE19030

Contents

List of Figures

List of Tables

ABSTRACT

Convolutional neural networks are a class of deep learning algorithms that are commonly used for processing data that has a grid like topology. It is commonly used for processing image data. Convolutional neural networks are computationally intensive, but certain techniques such as dilation and pooling help reduce the computational requirements. However, they do come with their drawbacks that may reduce the expressive power of the network.

The goal of this work is to improve dilation and pooling by incorporating fuzzy systems so that it convolutional neural networks provide better performance with the same amount of training.

Chapter 1:

Introduction

Introduction

Background

Deep Learning is a family of machine learning algorithms that progressively extracts complex features from the data, as opposed to task specific algorithms. [1] The idea of deep learning has been inspired from the organization of the mammalian brain. The classic experiments by Hubel and Wiesel [2] are fundamental to our understanding of how neurons along the visual pathway extract increasingly complex information from the pattern of light cast on the retina to construct an image. This is what deep neural networks try to replicate. Modern day deep learning architectures are extensively used in areas of image analysis and computer vision, natural language processing, time series data analysis, biomedical and computational biology along with many other areas [3].

Most deep learning techniques use the same fundamental architecture which is based on layers of perceptrons or artificial neurons. Perceptrons are non-linear processing units that are organized in the form of layers. These layers are stacked one after the other and each layer successively extracts more complex features. The learning is based on backpropagation algorithm, where the error in the final layer is propagated backwards through the preceding layers while tweaking the parameters of these layers [4]. An artificial neural network is called "deep", when it has more than 2 hidden layers. However, with increasing number of hidden layers, the conventional learning algorithms often fail. Furthermore, deep learning is also computationally intensive, and does not guarantee optimality [5].

Modern day deep learning architectures have been hybridized with other machine learning paradigms for improved performance on certain tasks. In this context a seldom explored area is the use of fuzzy systems. Previously, neural networks were extensively combined with fuzzy logic to create neuro-fuzzy models [6]. This, often, provided better performance with little or no extra performance cost, and in certain cases, it has also reduced the computational requirements. Fuzzy systems have also been used in other machine learning domains, most prominently, in data clustering [7] since the idea of fuzzy sets align well with the concept of partitioning.

Fuzzy systems are often employed to denote partial truth [8]. It is a mathematical tool that helps represent vagueness and imprecise information. In daily life, we often use terminologies such as big/small, high/low, near/far, and other without expressing quantities precisely using numeric values. This is exactly what fuzzy systems deals with. Fuzzy sets, as opposed to classical sets, allow an item to belong to a set with a certain degree of membership. For instance, in a classical set, which is also known as a 'crisp' set, an item can either be in a set or not. The boundary between what is in a set and what is not is distinct or 'crisp'. In case of a fuzzy set, that is not the case. So, "integers less than 10" is a crisp set, whereas "integers near 10" is a fuzzy set. Similarly, fuzzy logic has been used to extend Boolean logic by allowing truth values to be any real number between 0 and 1.

Different areas of fuzzy systems have also been implemented in a few deep learning architectures. This includes fuzzy sets, fuzzy logic and fuzzy numbers. Exploring the hybridization between contemporary deep learning architectures and fuzzy systems can help us uncover newer architectures that have better performance in certain applications. In this thesis, certain areas of convolutional neural networks are enhanced by implementing fuzzy systems. This has shown performance gains over traditional non-fuzzy systems.

Motivation

Convolutional neural network is a deep learning model that is extensively used in the many domain including image analysis [9]. The process of convolution helps extract features from the image data that is used in the classification of data. Convolutional neural networks heavily rely on two operations: convolution and pooling. Improving the performance of these operations by tweaking the algorithm can help in creating neural networks that can be trained with fewer parameters.

Implementation Framework

In this thesis, two different deep convolutional neural networks are developed. In CNNs, two CNNs, the convolution operation is performed on the input data, along with pooling, a technique that is used to subsample the data and reduce computation.

The convolution operation uses a "kernel" matrix to extract relevant features from the input data matrix [10]. The kernel matrix slides over the input matrix, extracting the features that

are local to the overlapping regions. The larger the size of the kernel, the more global features the kernel can extract. However, a larger kernel also increases computational complexity. Dilation is an inexpensive way for a smaller kernel to extract features from a larger area by excluding parts of the input. In fuzzy dilation, no part of the input is excluded when the smaller kernel is dilated. This provides marginal gains in terms of classification accuracy.

To speed up computation, CNNs also use pooling. Commonly used pooling methods include max pooling and average pooling. These methods naively discard a large fraction of the data for subsampling. A fuzzy pooling operation combines different pooling operations and performs the subsampling to reduce data loss.

The proposed models were implemented using the Python programming language and using an open source machine learning library for Python, called PyTorch.

Python allows wiring clear, concise and readable code and is extensively used for developing machine learning algorithms. PyTorch is a machine learning library that offers strong GPU acceleration and extensive interoperability with other popular Python libraries such as NumPy. One of the best features of PyTorch is how easy it is to construct the custom operations that are performed during the training and testing of the neural networks. This was an essential requirement since implementing fuzzy systems in convolutional neural networks meant rewriting the code of the whole operations. PyTorch allows programmers to easily declare tensors (n-dimensional vectors) and perform basic operations using them on GPUs.

To test the performance of any deep learning architecture, it is crucial to have a proper dataset. The performance often depends on how well the data is curated in the first place. Both the models were tested on two standard image datasets that are used for verifying the performance of deep learning models:

The first dataset is the MNIST dataset, which is a set of 10,000 grayscale images of handwritten digits. The digits have been size-normalized and centered in a fixed-size image, and hence is a great starting point for experimenting with CNNs.

The second dataset is the CIFAR-10 (Canadian Institute for Advanced Research) image dataset, which is one of the most commonly used image datasets for machine learning experiments. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes.

We have also tested both the models on another dataset containing digital pathology data. These are histopathology images containing Invasive Ductal Carcinoma (IDC), the most common type of breast cancer. The dataset contains over 200,000 image patches belonging to two classes: IDC and non-IDC.

Organization of thesis work

Chapter 1 introduces the thesis. This section provides an overview of deep neural networks and fuzzy systems, and how they have been used. This section also further discusses how the experiments has been implemented.

Chapter 2 provides a detailed literature survey. The extensive literature survey introduces the concept of fuzzy systems in deep neural networks and discusses in details previously studied models.

Chapter 3 introduces the methodologies that have been proposed. Two different methods have been proposed in the thesis – fuzzy dilated convolution and fuzzy pooling.

Chapter 4 discusses the experimental results. A total of nine experiments are performed using three different neural network models and three datasets.

Chapter 5 concludes the thesis and discusses future scope of the project.

Chapter 2:

Literature Survey

Literature Survey

Deep neural networks (DNNs) learn representations of the data in a hierarchical manner, building complex features out of simple features. DNNs are an extension of multi-layer perceptrons (MLPs). Such networks typically organize artificial neurons in a fixed topology connected by predefined links. Neurons are non-linear processing elements are organized into columns, called layers. The output of one layer is the input of the succeeding layer. The first layer which takes the input data is called the input layer. The intermediate layers are called hidden layers since they are hidden from the output. The last layer is called the output layer. The learning algorithm propagates the input forward through the layers of the processing elements and finds the error or the loss in the output layer. An increased number of layers allow the network to learn complex representations of the data, but new challenges arise when it comes to the training of such networks.

The target of training any deep neural network is to minimize the loss or error given an input vector. For every neural network, a loss function is defined that is used to calculate the loss of error for the input. The loss or the error is minimized by changing the weights of the network. The most common method for weight updating is by using gradient descent, which is a calculus-based method that iteratively computes the local minima of the error surface. The objective is to get to these minima by moving opposite to the direction of the slope. The obvious drawback of the calculus-based method is that it may not find the global minima of the error surface [11]. Furthermore, this process can be computationally intensive as well.

Gradient descent has three primary variations that is determined by what fraction of the input data we use to calculate the loss function. The parameters of the network are updated based on the time needed for the update as well as the volume of data used for the training process. The standard gradient descent computes the gradient of the loss function with respect to the parameters θ for the entire dataset, stochastic gradient descent or SGD performs the update for each training example. In between these two is mini-batch gradient descent that makes the update for every batch of n tuple. However, there are some common difficulties faced by all the variants of gradient descent.

These primarily include:

- Choosing a proper learning rate: Learning rate is a hyperparameter that the user has to supply. Finding the optimum value of the learning rate is done by trial and error. If it is too small, the network takes a lot of time to converge, whereas a value too big may even prevent the network from reaching the minima. Learning rate schedules which change the learning rate during the training process alleviate the problems a little, but these have to be defined in advance. Therefore, it becomes difficult to determine the threshold and schedules in advance without looking at the dataset. Depending on the architecture, they may also be other hyperparameters that are to be tuned manually [12].
- Minimizing highly non-convex error functions: Another key challenge is to avoid getting trapped in multiple local optima. The problem arises in the saddle points, where the value of the gradient is extremely small across all dimensions [13].

These challenges are somewhat tackled using gradient descent optimization methods such as Momentum [14], Nesterov accelerated gradient (NAG) [15], Adagrad [16], Adadelta [17], RMSprop [18], Adam [19], AdaMax and Nadam (Nesterov-accelerated Adaptive Momentum Estimation) [20]. Most of these strategies aim to provide improved convergence for MLPs with a larger number of layers. Some other techniques such as early stopping, gradient noise, batch normalization, shuffling and curriculum learning are used in conjunction. Furthermore, a lot of problems suffer from low sample size, have noisy samples or heterogeneous samples, or have severe class imbalance issues. These are some of the major drawbacks of deep learning techniques at present.

Besides, deep learning models needs a lot of data for training. Since it requires such copious amounts of data, the training time is also significant. Once trained, the model can be used for performing a very specific task. To deal with a slightly different problem, the entire system requires retraining. Another problem of deep learning is the computation power that is required for the training procedure, even though it is being addressed now through GPU-based parallel computing frameworks. Deep learning models work well with homogeneous features. If the features are of the same type (pixels, words count, etc.), then the deep learning model can be easily developed. But when the features are heterogeneous, the weight updates in the network happens on different scales. Hence, the input data should be normalized using some method.

An alternative method that is seldom explored in optimizing the performance of deep neural networks is by forming a hybrid model with fuzzy systems. Fuzzy systems allow dealing with the uncertainties and the ambiguities of real-world data. Fuzzy learning has been previously used in solving multiple real-world problems including image processing, computer vision, computational biology and bioinformatics, brain and cognition modelling, portfolio management and motor control. Fuzzy systems have also been extensively used in combination with neural networks to give rise to neuro-fuzzy systems. However, it has not been extensively used in conjunction with modern day deep learning architectures.

To specifically address the drawbacks of deep learning, multiple approaches have been proposed. Some models, such as Fuzzy Restricted Boltzmann Machines use the concept of fuzzy numbers to denote the network weights. An implementation of this was proposed for airline passenger profiling and for an early warning system for industrial accidents. Another way in which fuzzy systems are integrated into deep learning is by using fuzzy logic units instead of perceptrons in the network, which is similar to the neuro-fuzzy approach. Fuzzy systems have also been implemented to train some of the network parameters of a deep neural network.

Table 1: A Chronological Overview of Deep Learning Architectures using Fuzzy Logic.

Year	Model	Key Feature
2014	Deep Belief Networks with Fuzzy Granulated Inputs [Zhang et al. 2014][21]	• First deep neural network architecture to use fuzzified time-series input data • Effectively reduces noise and redundant information
2014	Deep Belief Network with fuzzified outputs [Zhou et al. 2014][22]	• Extends a deep binary classification model by implementing fuzzy memberships.
2015	Fuzzy Restricted Boltzmann Machine [Chen et al. 2015] [23]	• Extends the concept of RBM by implementing fuzzy weights • Fuzzy weights search for optima over a larger space • Handles noisy data well
2016	Fuzzy Deep Learning for tumor variation prediction [Park et al. 2016] [24]	• Uses fuzzy logic operator instead of ANNs. • Reduced number of parameters drastically improves training times.
2016	Deep Learning with Fuzzy Feature Points [Wang et al. 2016] [25]	• Uses radial basis function to fuzzify input data • Makes use of traditional deep CNN
2017	Restricted Boltzmann Machine with Fuzzy Inputs [Chopade and Narvekar 2017] [26]	• Uses fuzzy logic to understand semantic similarity between input vectors and assign importance value to input vectors - RBM is used for feature extraction

2017	Pythagorean Fuzzy Deep Boltzmann machine [Zheng et al. 2017] [27]	• Extension of Fuzzy Restricted Boltzmann Machine • Uses Pythagorean fuzzy numbers as weights • Can effectively handle noisy or incomplete data • Implements a Biogeography based learning
2017	Hierarchical Fused Fuzzy Deep Neural Network [Deng et al. 2017] [28]	• Extracts deep representation and fuzzy membership values of data in parallel • Implements multi-modal learning
2018	Stacked Auto Encoder trained using Fuzzy Logic [El Hatri and Boumhidi 2018] [29]	• Fuzzy logic system is used to adjust multiple neural network parameters • Fuzzy logic is also used to adjust learning rate
2018	Deep Convolutional Neural Networks using Weighted Fuzzy Active Shape Model [Tabrizi et al. 2018] [30]	• Uses Gabor filter to fuzzify input data and reduce noise. • Makes use of traditional deep CNN

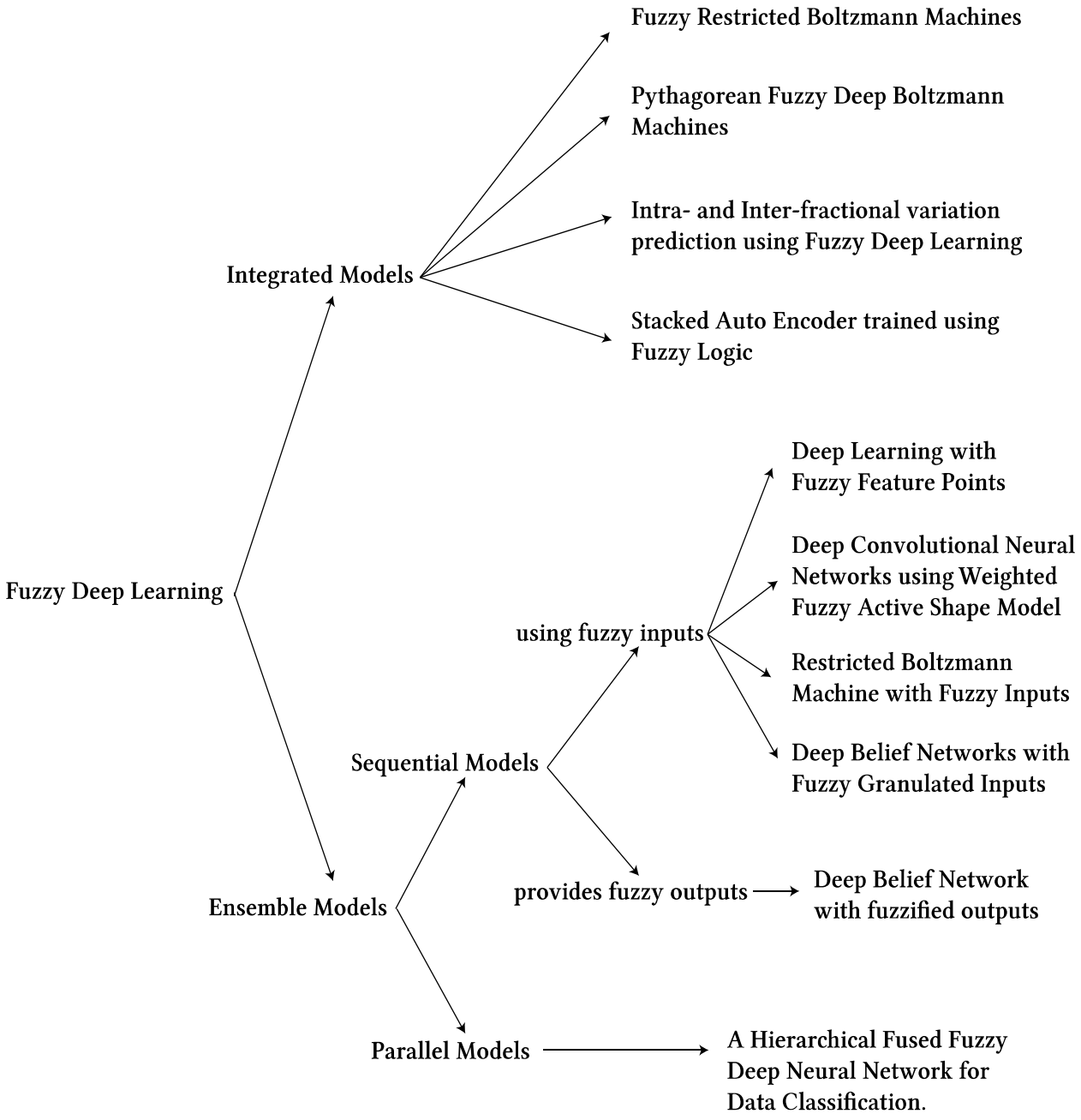

Figure 1: Overview of Deep Learning Architectures using Fuzzy Logic.

Integrated Models

This section describes in details the models that make use of fuzzy systems as a part of the learning mechanism.

Fuzzy Restricted Boltzmann Machines

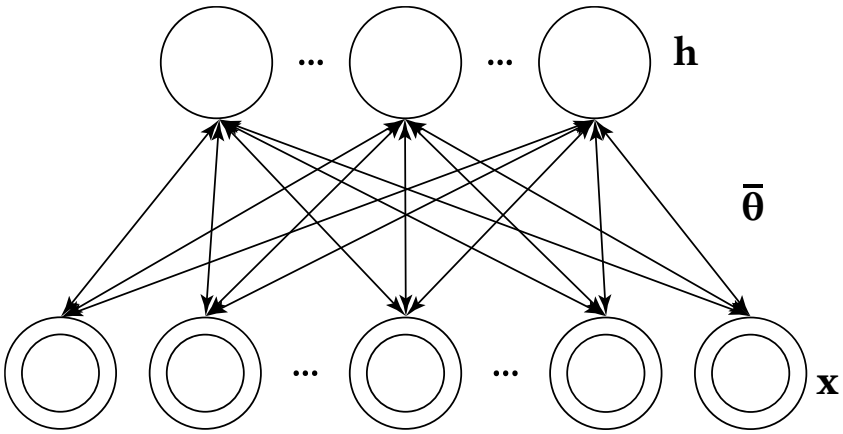

Figure 2. A fuzzy restricted Boltzmann Machine with fuzzy parameters set $\bar{\theta}$, visible vector x and hidden units h.

The Fuzzy Restricted Boltzmann Machine [21] is one of the first works in integrating fuzzy systems with a deep learning architecture. The Restricted Boltzmann Machine (RBM) plays a significant role in current deep neural network architecture and learning. Architectures such as the deep auto-encoder, deep belief networks, and the deep Boltzmann machines have a wide range of applications. Prior to this, most research focused on improving the learning algorithms for RBMs. Gaussian RBMs, temporal RBMs and recurrent RBMs were also proposed. However, regular RBMs or their variations have some drawbacks. RBMs have a set of visible units and a set of hidden units organized in form of a bipartite graph. A set of parameters θ are the weights of the edges between the hidden and the visible units and the biases associated with each unit. This architecture has a few drawbacks. The first drawback is the constant parameters that determine how the visible and the hidden units

are related. This reduces the representation capability of the RBMs. Secondly, the performance of the RBM heavily degrades with the presence of noise. Finally, during the learning procedure, a very small space is searched for the optimal parameters. Hence Fuzzy Restricted Boltzmann Machines (FRBMs) are used to overcome the inaccuracy and the linear relationships between the cross-layer units.

A Boltzmann machine is defined in terms of the combined state of the visible and the hidden units. The state is commonly known as energy, the value of which is determined by using a probability function. The relationship between the energy and the probability is given as p (v,h) $\propto e^{-E(v,h)}$

where p(v,h) is the joint probability and E(v,h) is the joint energy. For a given training set, the objective is to maximize the product of the probabilities of a training sample. This is the same as trying to maximize the expected log probability. Hence, an RBM can be said to be an energy-based probabilistic model. It is defined as:

$$P(v,h,\theta) = \frac{e^{-E(v,h,\theta)}}{Z}$$

where Z is called the partition function. It is used for normalization. Z is defined as:

$$Z = \sum_{\tilde{v}} \sum_{\tilde{h}} e^{-E(\tilde{v},\tilde{h},\theta)}$$

The partition function is all the possible number of configurations that can be taken up by the hidden and the visible unit. The energy function is given as:

$$E(v,h,\theta) = -b^T v - c^T h - h^T W v$$

where b_j and c_i are the offsets and W_{ij} is the weight of the connection from the j^{th} visible unit to the i^{th} hidden unit and $\theta = \{b, c, W\}$. The crisp energy function is given as follows:

$$\mathcal{F}(v,\theta) = -\log \sum_{\tilde{h}} e^{-E(v,\tilde{h},\theta)}$$

For the visible vector v, the free energy is the configuration of the nodes that is required to have the same probability as that of all the configurations with v.

In case of a fuzzy restricted Boltzmann machine (FRBM), the connection weights and biases are determined by $\overline{\theta}$, which is a fuzzy parameter. For this, the fuzzy energy function is derived from the crisp energy function and it is defined as

$$\overline{E}(v, h, \overline{\theta}) = -\overline{b}^T x - \overline{c}^T h - h^T \overline{W} v$$

where $\overline{E}(v, h, \overline{\theta})$ is the fuzzified energy function and $\overline{\theta}$ is a set containing $\overline{b}, \overline{c}, \overline{W}$ is the set of fuzzy parameters. Correspondingly, the free energy function implementing fuzzy numbers, which is given as $\overline{\mathcal{F}}$ is as follows.

$$\overline{\mathscr{F}}(v, \overline{\theta}) = -\log \sum_{\tilde{h}} e^{-\overline{E}(v, \tilde{h}, \overline{\theta})}$$

The probability defined using a fuzzy free energy function gives us fuzzy probability [34] which is an NP-hard problem and hence is intractable [32]. Next, using the centroid method, it is defuzzified. This converts the fuzzy optimization problem into a real valued one. The centroid of a fuzzy number [33] is determined using the following formula

$$\mathscr{F}_c(v, \overline{\theta}) = \frac{\int \theta \mathscr{F}(v, \theta) d\theta}{\int (v, \theta) d\theta}, \ \theta \in \overline{\theta}$$

Now, converting the fuzzy energy function into a crisp one using the above method, the probability can be defined as

$$P_c(v, \overline{\theta}) = \frac{e^{-\mathscr{F}_c(v, \overline{\theta})}}{Z}, \ Z = \sum_{\tilde{v}} e^{-\mathscr{F}_c(\tilde{v}, \overline{\theta})}$$

Here, the negative log-likelihood objective function is defined as

$$\mathcal{L}(\overline{\theta}, D) = -\sum_{v \in D} \log P_c(v, \overline{\theta})$$

Where D represents the training dataset.

Here, the optimization problem is finding the value of $\overline{\theta}$ that is, the parameters, so that the objective function $\mathcal{L}(\overline{\theta}, D)$. is minimized. For the learning algorithm, α-cuts of the fuzzy function are used to approximate the integral that required to determine the centroid of the fuzzy number. The α -cut of $\overline{\mathcal{F}_c}(v, \overline{\theta})$ can be given as follows:

$$\overline{\mathscr{F}_c}(v,\overline{\theta})[\alpha] = \mathscr{F}_c\left(v,\overline{\theta}[\alpha]\right) = [\mathscr{F}_c(v,\theta_R),\mathscr{F}_c(v,\theta_L)]$$

Here, θ_L and θ_R are the lower and upper bounds of the triangular fuzzy membership function. Therefore, the approximate value of the centroid is as defined as follows:

$$\overline{\mathscr{F}_c}(v,\overline{\theta}) \approx \frac{\sum_{i=1}^{M}\alpha_i[\mathscr{F}(v,\theta_{iL}) + \mathscr{F}(v,\theta_{iR})]}{2\sum_{i=1}^{M}\alpha_i}$$

where $\alpha = (\alpha_1,\dots,\alpha_N), \alpha \in [0,1]^N$ and $\overline{\theta}[\alpha_i] = [\theta_{iL},\theta_{iR}]$. Since all the α - cuts are fuzzy intervals, only the special case where $\alpha = 1$ is considered. This allows us to define the free energy function as:

$$\overline{\mathscr{F}_c}(v,\overline{\theta}) \approx \frac{1}{2}[\mathscr{F}(v,\theta_{iL}) + \mathscr{F}(v,\theta_{iR})]$$

To perform gradient descent-based optimization, the derivative of the negative log-likelihood is done with respect to the parameters, θ. This is done as follows:

$$\begin{aligned} -\frac{\partial \log P(v,\theta)}{\partial\theta} &= \frac{\mathscr{F}_c(v,\theta)}{\partial\theta} - \frac{1}{Z}\sum_{\tilde{v}} e^{-\mathscr{F}_c(\tilde{v},\theta)}\frac{\partial\mathscr{F}_c(\tilde{V},\theta)}{\partial\theta} \\ &= \frac{\mathscr{F}_c(v,\theta)}{\partial\theta} - \sum_{\tilde{v}} P(\tilde{v})\frac{\partial\mathscr{F}_c(\tilde{v},\theta)}{\partial\theta} \\ &= \frac{\mathscr{F}_c(v,\theta)}{\partial\theta} - E_p[\frac{\partial\mathscr{F}_c(v,\theta)}{\partial\theta}] \end{aligned}$$

Here, E_p is the expectation of the target probability distribution P. In the case of the defuzzied free energy function, the derivative is performed with respect to θ_L and θ_R. Therefore, we will have the following equations:

$$-\frac{\partial \log P_c(v,\overline{\theta})}{\partial\theta_L} = \frac{\mathscr{F}_c(v,\theta_L)}{\partial\theta_L} - E_p\left[\frac{\partial\mathscr{F}_c(v,\theta_L)}{\partial\theta_L}\right]$$

$$-\frac{\partial \log P_c(v,\overline{\theta})}{\partial\theta_R} = \frac{\mathcal{F}_c(v,\theta_R)}{\partial\theta_R} - E_p\left[\frac{\partial\mathcal{F}_c(v,\theta_R)}{\partial\theta_R}\right]$$

For these equations, it can be challenging to compute the gradients. To find the value of $E_p\left[\frac{\partial \mathcal{F}_c(v,\theta_L)}{\partial \theta_L}\right]$ or $E_p\left[\frac{\partial \mathcal{F}_c(v,\theta_R)}{\partial \theta_R}\right]$, we simply find the expectation over all possible configurations for the input x. Assuming there are $\mathcal{N}$ samples, then the gradient can be approximated as

$$-\frac{\partial \log P_c(v,\overline{\theta})}{\partial \theta} \approx -\frac{\partial \mathscr{F}_c(v,\theta)}{\partial \theta} - \frac{1}{|\mathcal{N}|}\sum_{\tilde{v}\in\mathcal{N}} \frac{\partial \mathscr{F}_c(\tilde{v},\theta)}{\partial \theta}$$

For an RBM, the conditional probability is defined as

$$P(h|v) = \frac{e^{-E(v,h)}}{\sum_{\overline{h}} e^{-E(v,\overline{h})}}$$

$$P(v|h) = \frac{e^{-E(v,h)}}{\sum_{\tilde{v}} e^{-E(\tilde{v},h)}}$$

In case of the RBMs that use binary units, v_i and $h_j \in 0,1$. Hence the above equations can be written as

$$P(h_i = 1|v) = \frac{e^{c_i + W_i v}}{1 + e^{c_i + W_i v}} = \sigma(W_i v + c_i)$$

$$P(v_j = 1|h) = \frac{e^{b_j + {W_j}^T h}}{1 + e^{b_j + {W_j}^T h}} = \sigma\left({W_j}^T h + b_j\right)$$

σ is the logistic expression, W_i and W_j determine the i^{th} and the j^{th} column of W. For fuzzy RBMs, the equations can be written as

$$P(h_i = 1|v) = \sigma\left(\overline{c_i} + \overline{W}_i v\right)$$

$$P\left(v_j = 1|h\right) = \sigma\left(\overline{b_j} + \overline{W_j}^T h\right)$$

Next, the Monte Carlo Markov Chain (MCMC) method is used to sample from the conditional distributions. This approximation alleviates the challenges of calculating the expression itself. Now, the α - cut of the probabilites are found using a certain value of α.:

$$P(h_i = 1|v)[\alpha] = [P_L(h_i = 1|v), P_R(h_i = 1|v)]$$

$$P\left(v_j = 1|h\right)[\alpha] = \left[P_L\left(v_j = 1|h\right), P_R\left(v_j = 1|h\right)\right]$$

$P_L(h_i|\ v)$, $P_R(h_i|\ v)$, $P_L(v_j|\ h)$ and $P_R(v_j|\ h)$ are all conditional probabilities in terms of the lower bounds and upper bounds of the parameters that define the model. These have the following forms:

$$P_L(h_i|\ v)\ =\ P(h_i|\ v;\ \theta_L)\ =\ \sigma(c_i{}^L\ +\ W_i{}^L v)$$

$$P_R(h_i|\ v)\ = P(h_i|\ v;\ \theta_R)\ =\ \sigma(c_i{}^R\ +\ W_i{}^R v)$$

and

$$P_L(v_j|\ h)\ =\ P(v_j|\ h;\ \theta_L)\ =\ \sigma(b_j{}^L\ +\ W_j{}^L h)$$

$$P_R(v_j|\ h)\ =\ P(v_j|\ h;\ \theta_R)\ =\ \sigma(b_j{}^R\ +\ W_j{}^R h)$$

Therefore, we have six parameters for the hidden and the visible units that we have to tune energy function and consequently the negative log-likelihood. The six parameters are lower bound of the connection weight W_{ij}^L, the visible bias b_j^L , the hidden bias c_i^L, and the respective upper bounds W_{ij}^R, b_j^R and c_i^R. For simplicity, the energy function can be represented using the terms that are associated with one hidden unit.

$$E(v,h) = -\mu(v) - \sum_{i=1}^{m} \phi_i(v,h_i)$$

where

$$\mu(v) = b^T V,\ \ \phi_i(v,h_i) = -h_i(c_i + W_i v)$$

A simplified expression of free energy of an RBM with binary units can be given as

$$\mathcal{F}(x) = -b^T v - \sum_{i=1}^{m} log(1 + e^{(c_i + W_i v)})$$

Now, the gradients can be easily calculated when the energy function has the form of $E(v,h)$.

$$
\begin{aligned}
-\frac{\partial \log P_c(v)}{\partial W_{ij}^L} &= E_P\left[P_L(h_i|\, v).\, v_j^L\right] - P_L(h_i|v).\, v_j^L \\
-\frac{\partial \log P_c(v)}{\partial b_j^L} &= E_P[P_L(h_i|\, v)] - v_j^L \\
-\frac{\partial \log P_c(v)}{\partial c_i^L} &= E_P[P_L(h_i|\, v)] - P_L(h_i|v) \\
-\frac{\partial \log P_c(v)}{\partial W_{ij}^R} &= E_P\left[P_R(h_i|\, v).\, v_j^R\right] - P_R(h_i|v).\, v_j^R \\
-\frac{\partial \log P_c(v)}{\partial b_j^R} &= E_P[P_R(h_i|\, v)] - v_j^R \\
-\frac{\partial \log P_c(v)}{\partial c_i^R} &= E_P[P_R(h_i|\, v)] - P_R(h_i|v)
\end{aligned}
$$

Where $P_c(v)$ is the centroid probability.

The expectations are then approximated. The samples of $P_L(v)$ and $P_R(v)$ can be obtained by running two separate Markov chains, using Gibbs sampling as the transition operator. Since Gibbs sampling is computationally intensive, we run it for a smaller number of steps. This is known as Contrastive Divergence. CD-k uses of Gibbs sampling for k-steps.

Pythagorean Fuzzy Deep Boltzmann Machines

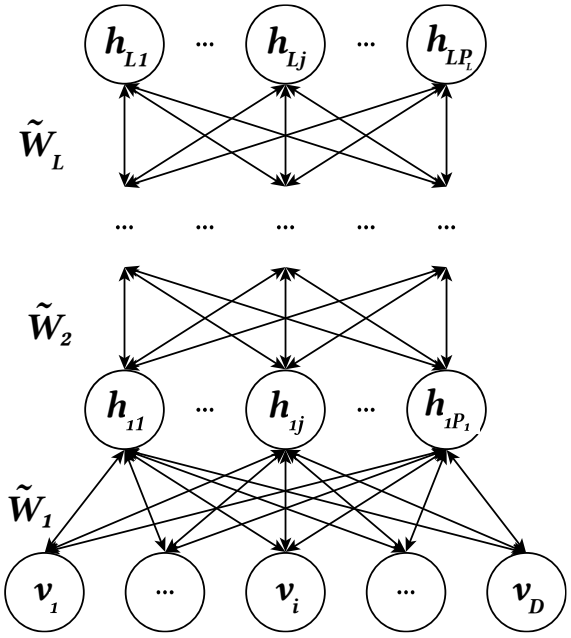

Figure 3. Pythagorean Fuzzy Deep Belief Network (PFDBM)

Zheung et al. [27] extend the Fuzzy Restricted Boltzmann Machine by proposing a Fuzzy Deep neural network that uses a Pythagorean Type Fuzzy Deep Boltzmann Machine. This model is used for airline passenger profiling where data is often incomplete and vague data. Fuzzified neural networks are known to handle both labelled and unlabeled data as well as incomplete features. Moreover, fuzzy deep Boltzmann machine performs parameter learning over a larger area. The fuzzy DBM proposed by Zheung et al. incorporates the concepts of Deep Boltzmann Machines, Pythagorean Fuzzy Sets and Pythagorean Fuzzy Numbers A standard Deep Boltzmann machine [35] is an extension of the restricted Boltzmann machine where the number of hidden layers is more than one. The increased number of hidden layers allow capturing more complex correlations of the activities of the preceding layers [36]. For a DBM with two hidden layers, the set of layers is $\{x, h_1, h_2\}$. This makes the free energy function as follows:

$$E(x, h_1, h_2, \theta) = -x^T W_1 h_1 - h_1^T W_2 h_2$$

$\theta = [W_1, W_2]$ is the vector containing the set of parameters. Now, the probability of the model assigning a certain parameter to the input vector x is simply

$$P(v,\theta) = \frac{1}{Z(\theta)} \sum_{h1} \sum_{h2} e^{-E(x,h_1,h_2,\theta)}$$

The basic RBM works with binary data. To work with real-valued data, the real value is transformed into a binary vector using Gaussian-Bernoulli RBM (GRBM). GRBM, as the name implies, makes use of Gaussian units. The energy function is defined as

$$E(x,h,\theta) = \sum_{i=1}^{D} \frac{(x_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{D} \sum_{j=1}^{P} W_{ij} h_j \frac{x_i}{\sigma_i^2} - \sum_{j=1}^{P} c_j h_j$$

Where the Gaussian visible neuron $x_i (1 < i < D)$ has standard deviation σ_i.

Now, the concept of the basic fuzzy sets is extended by Intuitionistic fuzzy set (IFS) [37] that introduces a non-membership degree besides the standard membership degree. Pythagorean fuzzy sets [38] is a further extension of IFS where the sum of the squares of the membership degree lies from 0 to 1.

A Pythagorean Fuzzy Set (PFS), P, is defined as the following mathematical object:

$$P = \{< x, P(\mu_p(x), v_p(x)) > | x \in S\}$$

Here, $\mu_p(x): S \to [0,1]$ and $v_p(x): S \to [0,1]$ are respectively the membership degree of the element x to S in P, satisfying the expression $\mu_p(x)^2 + v_p(x)^2 \leq 1$. The hesitant degree of $x \in X$ is written as

$$\pi_p(x) = \sqrt{1 - \mu_p^2(x) - v_p^2(x)}$$

If $\beta = P(\mu_\beta, v_\beta)$ is a Pythagorean Fuzzy Number (PFN) which satisfied $\mu_\beta, v_\beta \in [0,1]$ and $\mu_\beta^2 + v_\beta^2 \leq 1$, then the following operations are defined on the PFN:

$$
\begin{aligned}
\beta^{C} &= P(v_{\beta}, \mu_{\beta}) \\
\beta_1 + \beta_2 &= P\left(\sqrt{\mu_{\beta_1}^2 + \mu_{\beta_2}^2 - \mu_{\beta_1}^2 \mu_{\beta_2}^2}, v_{\beta_1}, v_{\beta_2}\right) \\
\beta_1 + \beta_2 &= P\left(\sqrt{\mu_{\beta_1}^2 + \mu_{\beta_2}^2 - \mu_{\beta_1}^2 \mu_{\beta_2}^2}, v_{\beta_1}, v_{\beta_2}\right) \\
\lambda\beta &= P\left(\sqrt{1 - \left(1 - \mu_{\beta}^2\right)^{\lambda}}, v_{\beta}^{\lambda}\right) \\
\beta^{\lambda} &= P\left(\mu_{\beta}^{\lambda}, \sqrt{1 - \left(1 - v_{\beta}^2\right)^{\lambda}}\right)
\end{aligned}
$$

Using these operations, two more functions are defined that are used to rank a PFN. These are as follows:

$$
\begin{aligned}
s(\beta) &= \mu_{\beta}^2 - v_{\beta}^2 \\
h(\beta) &= \mu_{\beta}^2 + v_{\beta}^2
\end{aligned}
$$

$s(\beta)$ is known as the score function and $h(\beta)$ is the accuracy function. Based on these the ranking of two PFN $\beta = P(\mu_{\beta_1}, v_{\beta_1})$ and $\beta = P(\mu_{\beta_2}, v_{\beta_2})$ is performed as follows:

1. If $s(\beta_1) < s(\beta_2)$ then $\beta_1 < \beta_2$.
2. If $s(\beta_1) = s(\beta_2)$, then
    a. If $h(\beta_1) < h(\beta_2)$, then $\beta_1 < \beta_2$
    b. If $h(\beta_1) = h(\beta_2)$, then $\beta_1 = \beta_2$

The Pythagorean Fuzzy Deep Belief Network (PFDBN) is built upon the DBM by using PFNs in place of the standard real-valued paramters. This fuzzy neural network can work with fuzzy and/or incomplete data [39][40]. Furthermore, fuzzy parameters provide a better representation of the data using fuzzy probability. Moreover, the parameter learning space of the DBM increases with the introduction of the PFS.

The associated PFN parameters can learn new features, and also sees how much a certain feature influences the output

Given the fuzzy parameters are $\tilde{\theta} = [\widetilde{W}_1, \dots, \widetilde{W}_L]$ of a PFDBM with layers , $h_1, \dots, h_L$, then the energy function can be written as:

$$\tilde{E}\left(v, h_1, \ldots, h_L, \tilde{\theta}\right) = -v^T \widetilde{W_1} h_1 - \sum_{l=2}^{L} h_{l-1}^T \widetilde{W_l}\, h_l$$

And the corresponding probability is

$$\tilde{P}\left(v, \tilde{\theta}\right) = \frac{1}{\tilde{Z}\left(\tilde{\theta}\right)} \sum_{h_1} \ldots \sum_{h_L} e^{-\tilde{E}\left(v, h_1, \ldots, h_L, \tilde{\theta}\right)}$$

Now, the objective of the learning algorithms is to learn a set of optimal parameters such that the log likelihood is maximized (or the negative log likelihood is minimized).

$$\max_{\tilde{\theta}} \tilde{\mathcal{L}}\left(\tilde{\theta}, D\right) = \sum_{v \in D} \log\left(\tilde{P}\left(v, \tilde{\theta}\right)\right)$$

However, in general, fuzzy optimization problems are quite intractable. So, using the scoring functions, it is converted to a crisp optimization problem. Now, the PFDBM is trained using a combination of gradient descent and metaheuristic techniques.
The PFDBM is trained by iteratively using greedy layer-wise training and Biogeography based learning algorithm since traditional methods are suboptimal. Evolutionary algorithms have been used previously for ANNs too, but not widely DNNs [41][42][43]. The biogeography-based learning algorithm is a metaheuristic optimization method that is inspired by biogeography.

The training algorithm randomly initiates a population of n solutions and a local topology. Next, it uses gradient-descent to train the network weights. Then immigration and emigration operation, along with mutation operation is performed. Then again, gradient descent is used to optimize the network. The old solution is replaced with a new one if the new one provides better values. Until the termination condition is satisfied, this process is continued.

Fuzzy Deep Learning for Lung Tumor Tracking

The synergy of Deep learning and fuzzy logic has also been demonstrated by Park et al. [24] in a machine that is used to predict the intra- and inter-fractional variation of lung tumors. In image-guided radiation therapy, it is important to monitor the target of the radiation very

precisely to prevent damage to the surrounding healthy tissues. Features are extracted from respiratory signals, from which multiple metrics are first computed as done in previous studies too [44][45][46][47], such as "Standard deviation of time series data", "maximum likelihood estimates", "breathing frequencies" and more. Based on these values, the patients are clustered according to the similarity of the breathing features. For each patient group, the FDNN in trained to predict the two different types of variation.

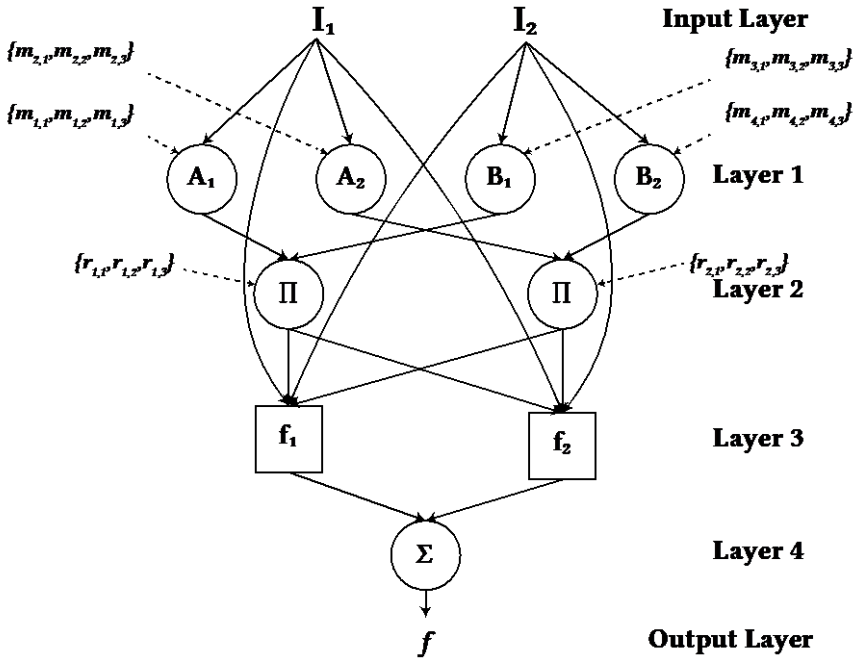

Figure 4. Fuzzy Deep Learning architecture for tracking lung tumors

The fuzzy deep learning (FDL) model is a fuzzy logic system that is organized in form of a neural network. Since the neural network is more than two layers deep, it is called a deep network. In the network, the functional units are fuzzy rules instead of traditional artificial neurons with activation functions.

The neural network architecture gives the fuzzy logic system a self-learning feature that allows it to set its parameters automatically by using a variation of the gradient descent algorithm. In FDL, a small number of fuzzy parameters determine the weight values between the nodes. These are the prediction parameters. Furthermore, since there are fewer parameters, the performance is optimal for real-time prediction.

The network has four layers. These are summarized as follows: The layer 1 assigns membership values based on certain membership functions to the input. The functions take in a set of parameters $M = \{m_{i,1}, m_{i,2}, m_{i,3}\}$. Then T-norm operation in performed in the next layer, that is layer 2. In layer 3 based in the set of input parameters $R = \{r_{i,1}, r_{i,2}, r_{i,3}\}$, linear regression is done and finally, layer 4 provides the output of the entire network by providing an additive outcome depending on the fuzzy if-then rules. According to Fig. 5, the fuzzy if-then rules, which is the same as the number of nodes in layers 2 and 3, are given as follows:

1: If I_1 is A_1 and I_2 is B_1, then $f_1 = r_{1,1}I_1 + r_{1,2}I_2 + r_{1,3}$
2: If I_1 is A_2 and I_2 is B_2, then $f_2 = r_{2,1}I_1 + r_{2,2}I_2 + r_{2,3}$ Here, I_1 and I_2 are the inputs of the network and A_i and B_i are the fuzzy sets.

Layer 1: The output of the first layer, $O_{1,i}$, is given as:

$$O_{1,i} = \begin{cases} \mu_{A_i}(I_1) = \dfrac{1}{[1 + |(1 - m_{i,3})/m_{i,1}|]^{2m_{i,2}}} & 1 \le i \le 2 \\ \mu_{B_{i-2}}(I_2) = \dfrac{1}{[1 + |1 - m_{i,3}/m_{i,1}|]^{2mi,2}} & 3 \le i \le 4 \end{cases}$$

Where for the fuzzy sets A_i and B_i, $\mu_{A_i}(I_1)$ and $\mu_{B_{i-2}}(I_2)$ are the membership functions. The training procedure chooses the membership parameters $m_{i,1}, m_{i,2}, m_{i,3}$.

Layer 2: Layer 2 performs a product of all the inputs coming from the preceding layer.

$$O_{2,i} = w_i = \mu_{A_i}(I_1).\mu_{B_i}(I_2) \quad 1 \le i \le 2$$

Multiplication is analogous to the T-norm operator and determines the output determines the degree of influence of the rule.
Layer 3: In the third layer, the ratio of the ith rule's degree of influence and the total degree of influence of all the rules is used to perform linear regression.:

$$O_{3,i} = \frac{w_i}{\sum_j w_j}\left(r_{i,1}I_1 + r_{i,2}I_2 + r_{i,3}\right)\ 1 \le i \le 2$$

Where $\{r_{i,1}, r_{i,2}, r_{i,3}\}$ is the set of learning rate parameters determined from the training procedure.

Layer 4: This layer performs the mean as follows:

$$O_{4,1} = f = \sum \frac{w_i}{\sum_j w_i} f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

The FDL is trained using a Hybrid Training Algorithm for Feed-forward Neural Networks as in [31]. The training procedure finds the parameters set M and R which provides the membership values and the regression parameters. These parameters are then used to train the network. Intra- and inter-fractional variation data is acquired from the CyberKnife dataset. For this data, variable number of nodes are used. The dataset has 3D coordinates and for each channel, a seperate network is employed. This also changes the number of nodes in each layer. For instance, both layer 2 and layer 3, in this case, will have 27 nodes. The rules are modified as follows:

Rule 1: If I_1 is A_1 and I_2 is B_1 and I_3 is C_1, then $f_1 = r_{1,1}I_1 + r_{1,2}I_2 + r_{1,3}I_3 + r_{1,4}$

Rule 27: If I_1 is A_3 and I_2 is B_3 and I_3 is C_3, then $f_27 = r_{27,1}I_1 + r_{27,2}I_2 + r_{27,3}I_3 + r_{27,4}$ Here, the input vector $\{I_1, I_2, I_3\}$ corresponds to the three channels of the CyberKnife machine. A_i, B_i and C_i are the fuzzy sets. The computation of the output of the initial layer is now given as:

$$O_{1,i} = \begin{cases} \mu_{A_i}(I_1) = \dfrac{1}{[1 + |(1 - m_{i,3})/m_{i,1}|]^{2m_{i,2}}} & 1 \le i \le 3 \\ \mu_{B_{i-3}}(I_2) = \dfrac{1}{[1 + |1 - m_{i,3}/m_{i,1}|]^{2mi,2}} & 4 \le i \le 6 \\ \mu_{C_{i-6}}(I_3) = \dfrac{1}{[1 + |1 - m_{i,3}/m_{i,1}|]^{2mi,2}} & 7 \le i \le 9 \end{cases}$$

Where the different membership functions, μ_{A_i}, $\mu_{B_{i-3}}$ and $\mu_{C_{i-6}}$, are determined by the $M = \{m_{i,1}, m_{i,2}, m_{i,3}\}$. Similarly, the output of the succeeding layer is given as

$$O_{2,i} = w_i = \mu_{A_k}(I_1).\mu_{B_l}(I_2).\mu_{C_m}(I_3) \quad 1 \le i \le 27$$

For the third layer, it is

$$O_{3,i} = \frac{w_i}{\sum_j w_j}\left(r_{i,1}I_1 + r_{i,2}I_2 + r_{i,3}I_3 + r_{i,4}\right) 1 \le i \le 2$$

Where $\{r_{i,1}, r_{i,2}, r_{i,3}, r_{i,4}\}$ is the set of LR parameters. Finally, the output is calculated as

$$O_{4,1} = f = \sum \frac{w_i}{\sum_j w_i} f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

The final layer produces a single coordinate, that is, x, y or z estimate. The IIFDL uses 3 FDL for each of the coordinates, allowing the estimation of 3D coordinates.

The model was tested using CyberKnife patient databases and was compared with two ANN-based models - a CNN and a hybrid implementation of the extended Kalman filter (HEKF) . The IIFDL model outperformed the CNN both in terms of prediction accuracy and training speed. Since the IIFDL has fewer parameters, the training process is much faster. Prediction accuracy was measured using root means square error for different time intervals

Stacked Auto Encoder trained using Fuzzy Logic

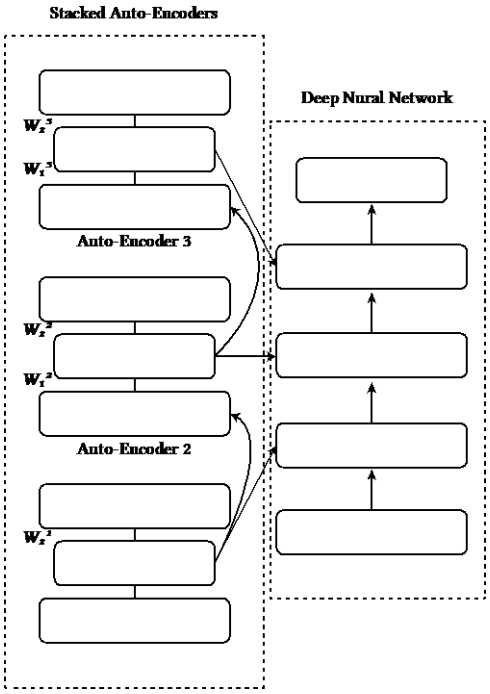

Figure 5. Stacked auto-encoders

Hatri et al. proposed a deep learning model that uses fuzzy logic to train some of the network parameters that improve the learning speed and has a higher probability to reach a better global optimum [29]. The deep learning network is based on stacked-auto encoders (SAE) that are trained using backpropagation but the learning rate and the momentum are determined using fuzzy logic systems.

A stacked auto-encoder is constructed using a series of auto-encoders (AE), one on top of the other where the output of one auto-encoder is the input of the auto-encoder that is above it. In the simplest form, an auto-encoder is a neural network with a single hidden layer. The input layer nodes and output layer nodes are same in number. The number of nodes in the hidden layer is fewer.

The auto-encoder tries to replicate the input at the output by passing it through a fewer number of nodes. As the data is passed through the hidden layers, which are fewer in number, the network compresses the data and learns only the important features. For a set of input vectors $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots\}$ where each input $x^{(i)} \in \mathbb{R}^d$, the autoencoder first encodes every input $x^{(i)}$ to a compressed form $y\left(x^{(i)}\right)$ which extracts the dominant features and then reconstructs or decodes that compressed representation into $z\left(x^{(i)}\right)$. This is given as:

$$y(x) = f(W_1.x + b_1)z(x) = g(w_2.y(x) + b_2)$$

Where W_1 and b_1 are the encoding of weights and bias, and W_2 and b_2 are the encoding of weights and bias. The training process can be done using traditional backpropagation. The error or the loss is calculated as

$$E(X\ ,\ Z) = \frac{1}{2}\sum_{i-1}^{N}\left|x^i - z^i\right|^2$$

Where the number is samples is N, x is the sample set and Z is the output that is constructed by the AE. When training an SAE, each autoencoder is trained individually, layer by layer. This means for a three-layer SAE, the first auto-encoder is trained first and we obtain the weights W_1^1 and W_1^2. Then, for each AE i ($i = 1,2,3$), the current autoencoder is trained using the output of the preceding one, that is, y^{i-1}, in such a way that W_1^i and W_2^i are obtained. In this method, the SAE is trained.

The fine-tuning strategy that is used in this case is the simple back-propagation algorithm. During fine-tuning, from a high level, all the SAEs are treated as a single model. Therefore, the tuning process improves all the weights of the auto-encoders in a single iteration. Hence, the error of the entire network is defined as

$$E(t) = \frac{1}{2}|e(t)|^2$$

Where e(t) is the error value. This difference between the input and the output determines the changes in the network weights, which is given as

$$\triangle w_j(t+1) = \eta\left(\frac{\delta E(t)}{\delta w_j}\right) + \alpha \triangle w_j(t)$$

Here, η is the learning rate parameter and α is a positive value called the momentum. The learning rate and the momentum is dynamically adjusted using a fuzzy logic control system. A fuzzy logic control is used to adaptively adjust the neural network parameters such that the mean squared error (MSE) is reduced. To create the fuzzy logic system, four parameters are used. These include:

$$\begin{aligned} Relative\ Error: RE(t) &= E(t) - E(t-1) \\ Change\ in\ Relative\ error: CRE &= RE(t) - RE(t-1) \\ Sign\ change\ of\ error: SC(t) &= 1 - ||\frac{1}{2}(sign(RE(t-1)) + sign(RE(t))|| \\ Cumulative\ sum\ of\ sign\ change: CSC &= \sum_{m=t-4}^{t} SC(m) \end{aligned}$$

Using the fuzzy logic system, the change of learning rate and the change of momentum are done.

The system was used to predict traffic incidents while using an urban traffic simulator to synthesize training and testing data. When compared to deep neural network, the FDNN had much better training times and marginally better accuracy when detecting traffic incidents.

Ensemble models

The following models use fuzzy logic and deep learning in a sequential or parallel fashion. The input vector to the deep neural network is either pre-processed using a fuzzy logic system, or the input is transformed by the deep neural network and fuzzy systems in parallel.

Sequential Models

This section lists all the sequential models. Sequential models either make use of fuzzified inputs to existing deep learning architectures, or provides a fuzzy output. Based on the nature

of the input and output, this section is further divided into two parts: Models with fuzzy inputs and models with fuzzy outputs.

Models with Fuzzy Inputs

Deep Learning using Fuzzy Feature Points:

Wang et al. proposed a model that uses Deep Learning with Fuzzy Feature Points for damaged fingerprint classification [25]. The model essentially involves fuzzyfying the input data and then providing the input to a simple convolutional neural network. Since it provides a very generic model that uses image data, the model can be leveraged for other visual recognition tasks too.

Before the data it fed to a CNN, it undergoes preprocessing, feature extraction and matching. The fuzzy features that are fed to the model is created from the preprocessed fingerprint image data. This involves the core and delta extraction, as well as endpoint and branch point extraction. To extract the core and delta points, the Poincare formula is used.

Once the feature points are extracted, they are fuzzified using the radial basis function. Here, the radial basis function is used for interpolation of the data points.

A radial basis function is given as $\phi(x) = \phi(||\ x\ ||)$. There are various types of radial basis functions. The Gaussian type is given as $\phi(r) = e^{-(\epsilon r)^2}$ where $r = ||x - x_i||$. In this case, the "Gauss distribution function of Kriging method" is used, which is given as $\phi(r) = e^{-(r/\sigma)^2}$. The goal here is to reduce the influence certain defective fingerprint lines so that the relationship between the different feature points are highlighted. Weakening the specific positions of the feature points provide a higher degree of rotational invariance. The fuzzy graph of the feature points is then provided to a CNN which predicts the class of the fingerprint. A generic deep CNN using sigmoidal activation function is used in the classification of the fuzzified data.

This provided a marginally improved recognition rate over the pre-processed original fingerprint image and is massively outperforms the CNN trained with the original fingerprint image.

Deep Convolutional Neural Networks using Weighted Fuzzy Active Shape Model:

A similar model where deep learning and fuzzy logic is used for image segmentation is proposed in [30]. In this case, 3D ultrasound images of kidneys are performed. The idea is quite simple: fuzzy clustering is performed on the image data for better characterization of the different parts of the images. Furthermore, to reduce the noise of the image and to enhance the edges, an omni-directional Gabor-filter is used.

The model is compared with manual segmentation of 3D ultrasound images of kidneys, and the performance of the DNN is at par with the manual process. This process can be used for segmentation of other medical images and even in other areas of computer vision.

The DNN has two convolution layers. They are succeeded by two fully connected layers. Each layer uses a rectified linear unit. The last layer uses the SoftMax probability function that is used to evaluate the position, orientation and scale. To reduce overfitting, a dropout rate of 0.4 was set in the last fully connected layer.

Deep Belief Networks with Fuzzy Granulated Inputs:

Zhang et al. propose a model utilizing fuzzy granulation and DBN for predicting time series data [25]. The concept of information granules comes from granular computing. Information granules are groups of entities that are numerically similar. The similarity can be because of their similar functionality, similar physical properties, or because of any other attributes that are similar numerically. The aim is to decompose a problem into simpler problems while removing some irrelevant information. In this case, fuzzy information granulation is done on time series data before it is used to train and test deep belief networks for continuous data (CDBN).

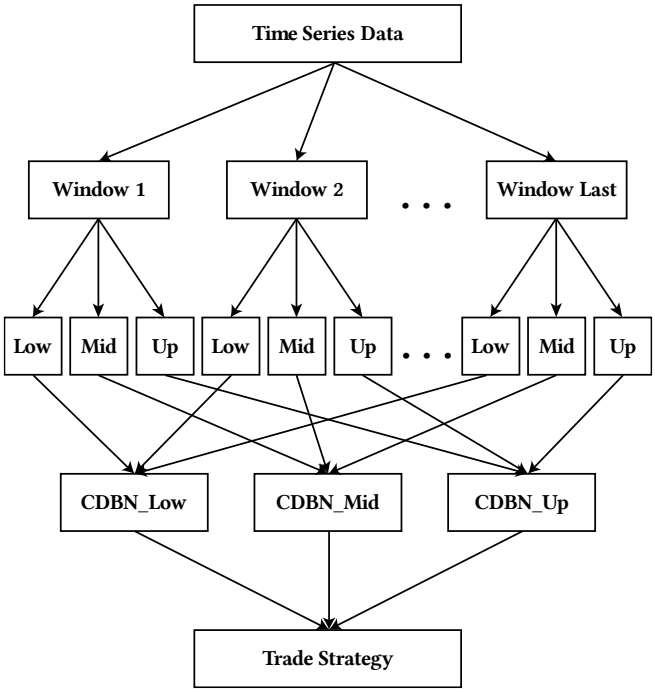

Figure 6. Framework of the CDBN-FG that combines fuzzy granulation and deep belief networks for continuous data

In fuzzy information granulation, the data is granulated into three sets, or in this context, granules, given as Low, Mid, High, using multiple different granulation methods. Ruan et al. proposed five fuzzy granulation methods in - triangular membership function-based granulation, trapezoidal membership function-based granulation, gaussian membership function-based granulation, fuzzy c-means based granulation and min-max based granulation. The time series data is first split into multiple windows. Given that $T = t_1, t_2, \dots, t_n$ is a time series data and w is the length of the granulation windows where $1 \leq w \leq n$, the time series data can be split as $\{t_1, t_2, \dots, t_w\}, \{t_{w+1}, t_{w+2}, \dots, t_{w+w}\}, \dots, \{t_{([n/w]-1)w+1}, t_{([n/w]-1)w+2}, \dots, t_{([n/w]-1)w+w}\}$. $[n/w]$. The granulation methods are described as follows: Triangular MF based granulation:

$$
\begin{aligned}
low_i &= \frac{2\sum_{j=1}^{w/2} t_{j'}^{i}}{w/2} - median\left(t_{1'}^{i}, t_{2'}^{i}, \dots, t_{w'}^{i}\right) \\
mid_i &= median\left(t_{1'}^{i}, t_{2'}^{i}, \dots, t_{w'}^{i}\right) \\
up_i &= \frac{2\sum_{j=w/2+1}^{w} t_{j'}^{i}}{w/2} - median\left(t_{1'}^{i}, t_{2'}^{i}, \dots, t_{w'}^{i}\right)
\end{aligned}
$$

Trapezoidal MF based granulation:

$$\begin{aligned} low_i &= \frac{2\sum_{j=1}^{w/2} t_{j'}^{i}}{w/2} - t_{(w/2)'}^{i} \\ mid_i &= median\left(t_{1'}^{i}, t_{2'}^{i}, \ldots, t_{w'}^{i}\right) \\ up_i &= \frac{2\sum_{j=w/2+1}^{w} t_{j'}^{i}}{w/2} - t_{(w/2+1)'}^{i} \end{aligned}$$

Minmax-based granulation function:

$$\begin{aligned} low_i &= t_{1'}^{i} \\ mid_i &= median\left(t_{1'}^{i}, t_{2'}^{i}, \ldots, t_{w'}^{i}\right) \\ \mathrm{up}_i &= t_{w'}^{i} \end{aligned}$$

The time series data is first split into multiple windows and each window is granulated using fuzzy granulation techniques. There are three granules that are derived from each window. The Low, Mid and Up granules are trained using three separate CDBNs. CDBNs employ continuous valued restricted Boltzmann machines (CRBMs) which use continuous valued stochastic units. Let s_j be the output of the neuron j, with units from neurons with state $\{s_i\}$.

$$s_j = \varphi_j\left(\sum_i w_{ij}s_i + \gamma.N_j(0,1)\right)$$

With $\varphi_j(x_j) = \theta_L + (\theta_H - \theta_L).\frac{1}{1+e^{-a_j x_j}}$. The random gaussian number $N_j(0,1)$ has a mean value 0 and a variance of 1. The value γ is a constant. The sigmoid function $\varphi_j(x_j)$ has θ_L and θ_H as asymptotes and te value a_j is a parameter that is introduced to control the noise. The update rules for w_{ij} and a_j are given as

$$\begin{aligned} \triangle w_i j &= \eta_w(< s_i s_j > - < s_i' s_j' >) \\ \triangle a_j &= \frac{\eta_a}{a_j^2}(< s_j^2 > - < s_j'^2 >) \end{aligned}$$

The learning rates are η_w and η_a. s_j' denotes a single step of sampling of the state on unit j and <.> denotes the mean. The CDBN is trained in a layer-by-layer fashion where the hidden

layers or each CRBM is trained sequentially. For training a network with k input nodes, the initial training sample for $CDBN_{Low}$ is composed of $\{Low_1, Low_2, \dots, Low_k\}$. Given this input, the CDBN is trained to predictLow_{k+1}. In the next phase, the input vector $\{Low_2, Low_3, \dots, Low_{k+1}\}$ is used to predict the value of Low_{k+2} and so on. The same is done with $CDBN_{Mid}$ and $CDBN_{Up}$.

When tested on stock market data, the CDBN, while using fuzzy granulation outperforms the standard CDBN. The granulation window size, as well as the membership function parameters, can be tweaked for improved performance on various datasets.

Models with fuzzy outputs

Restricted Boltzmann Machine with Fuzzy Outputs:

Chopade et al. implemented a combination of fuzzy logic and deep learning to perform document summarization [26]. Here, the summarization is done in multiple phases. First, features are extracted from the sentences and a sentence matrix is formed based on the features. Each sentence is also manually assigned an importance value. This dataset is used to train a restricted Boltzmann machine to get a more refined set of sentences. Next, the sentences are processed and a table is created to rank the sentences using a priority value. From there, a fuzzy membership function is used to determine the sentences that will be used for summarization.

In the feature extraction phase, the features that are extracted includes title similarity, term weight, count of named entities and the amount of numeric data. Using these features, a sentence matrix is formed.

The implementation of a restricted Boltzmann machine, in this case, is quite simple. The sentence matrix is the input of the restricted Boltzmann machine with three layers. The output of the machine is a much more refined set of sentences.

In the next stages, the sentences are processed more. Stemming, stop word removal and part of speech tagging is performed. Other than the sentence table, two more tables are created, a seed table containing word priority values and a word table containing the word frequency values. From these three tables, the rank of each sentence is calculated. This is essentially a fuzzy membership function that determines the priority of the sentence in the document summary.

$$Rank(S_i) = \sum_{i=1}^{n} f\,requency(w_{ij}) + \sum_{i=1}^{n} membership\_with\_seed(w_ij)$$

Where the total count of sentences in the sentence table is denoted by **n**, the ith sentence is S_i and the jth word is W_{ij}. The fuzzy membership is determined by the function Membership_with_seed ().

Deep Belief Network with fuzzified outputs:

A "fuzzy deep belief network for semi-supervised sentiment classification" [22] is a simple example of a sequential model that uses a deep belief network and fuzzy sets for sentiment classification is proposed in. A Deep belief network (DBN) are trained one layer after the other, in a layer-wise fashion, using a greedy layer-wise training algorithm and then are fine-tuned by standard backpropagation algorithms. In the context of sentiment classification performed in a semi-supervised fashion, fuzzy sets were used to describe the membership value with which a pattern belongs to a positive or a negative class. The model can be used for other binary class problems too.

First, the DBN is trained with both unlabeled and labelled data. Training with unlabeled data is performed in a greedy layer-wise manner, by considering two adjacent layers as a restricted Boltzmann machine (RBM). Next, labelled data is used to further fine-tune the network using backpropagation and gradient descent. In this sentiment classification problem, is a positive class and a negative class. Hence, a single hyperplane is needed for separating the two classes. Two membership functions are formulated that describes the membership of the input in a class and the distance of the input from the hyperplane.

The DBN is trained using gradient descent. Now, since there are just two classes, only one hyperplane is needed. If h^N is the last layer and z^i is the ith input vector, then $d(z^i) = \left(h_1^N(z^i) - h_2^N(z^i)\right)/\sqrt{(2)}$ is the distance between the separating line and the point $h_1^N(z^i)$. z^i is positive when If $d(z^i) > 0$, and otherwise it is negative.

Now, the membership functions $\mu_A(z)$ and $\mu_B(z)$ are gives as

$$\mu_A(z;\beta,\gamma) = \begin{cases} S(d(z);\gamma-\beta,\gamma-\beta/2,\gamma), d(z) \le \gamma \\ 1, d(z) \ge \gamma \end{cases}$$

$$\mu_B(z;\beta,-\gamma)=\begin{cases}1, d(z)\leq\ \gamma\\ 1-S(d(z);\gamma,\gamma-\beta,\gamma-\beta/2), d(z)\geq\gamma\end{cases}$$

Where $S(d;\alpha,\beta,\gamma)$ is

$$S(d;\alpha,\beta,\gamma)=\begin{cases}0,\ d\ \leq\alpha\\ 2\left(\frac{d-\alpha}{\gamma-\alpha}\right)^2, \alpha\leq d\leq\beta\\ 1-2\left(\frac{d-\alpha}{\gamma-\alpha}\right)^2, \beta\leq d\leq\gamma 1,\quad d\ \leq\ \gamma\end{cases}$$

Now, there are two parameters to be determined, β and γ. The distance metric can be used in the following manner to get the values:

$$\begin{aligned}\gamma\ &=max\left|d\left(z^i\right)\right|, i=1,\dots,R+T\\ \beta\ &=\ \xi\times\gamma,\ \xi\geq 2\end{aligned}$$

Parallel Models

A Hierarchical Fused Fuzzy Deep Neural Network

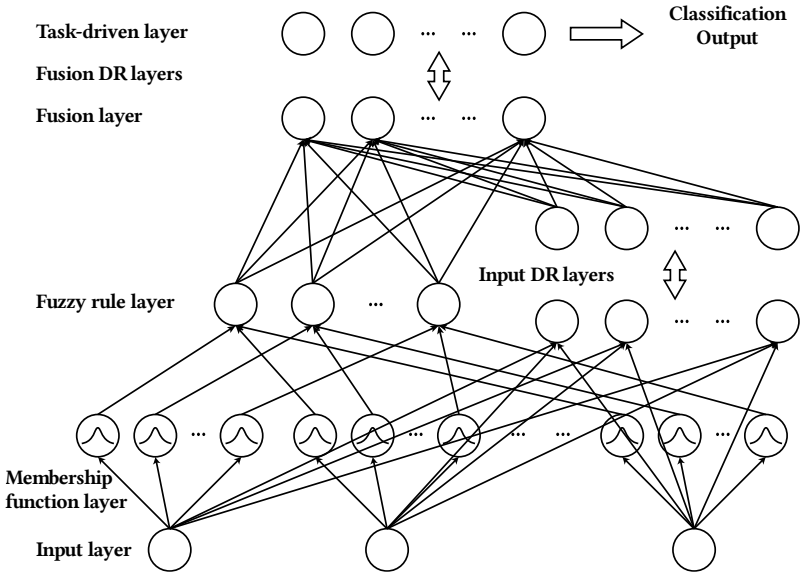

Figure 7. Structure of the hierarchical Fused Fuzzy Deep Neural Network

Deng et al. proposed a deep learning architecture has deep learning layers and fuzzy membership functions running in parallel [28]. The deep representation and the fuzzy representation are then fused using multimodal learning techniques [48].

The model architecture can be divided into four sections.

Fuzzy Logic Representation:

Each node of the input layer is connected to fuzzy logic nodes that computes fuzzy membership of each of the inputs. The input is an uni-dimensional vector. This layer determines how much a given input vector belongs to a certain set. It uses a standard Gaussian membership function to do so. This layer maps the input to a real value between 0 and 1, which is basically the fuzzy membership value of the input variable. The fuzzy membership function is $u_i(.)$ and the output of the function is defined as follows:

$$o_i^{(l)} = u_i(a_k^{(l)}) = e^{-\frac{a_k^{(l)} - \mu_i}{\sigma_i^2}^2}$$

where l denotes the position of the layer from the input layer, the input of node i is given as $a_i^{(l)}$ and the corresponding output is given as $o_i^{(l)}$. The succeeding layer performs another fuzzy operation. In this case, it is an AND operation.

$$o_i^{(l)} = \prod_j o_j^{(l-1)} \ \forall j \in \Omega_i$$

where Ω_i is the set of nodes in the preceding layer that connect to the node i. The output is a real-value in the range (0,1).

Neural Representation or Deep Representation:

This is a simple fully connected layer that is implemented in multi-layer perceptrons. The feed-forward layers pass the input from the preceding layer through a sigmoid function. The input of the preceding layer is the output of the layer before that times the weight and the bias. This is given as $o_i^{(l)} = \frac{1}{1+e^{-a_i^{(l)}}}$ where $a_i^{(l)} = w_i^{(l)} o^{(l-1)} + b_i^{(l)}$. Here $w_i^{(l)}$ is the weight of the node i on layer l and $b_i^{(l)}$ is the corresponding bias.

Fusion layers:

The fusion layers bring together the output of the fuzzy layers and the neural representation layers. The idea of fusion layers is inspired by multimodal learning. In this set of layers, the model can learn parameters based on the classification task. Multi-modal learning is a good way to represent heterogeneous data or data that has been acquired from different areas. This allows one data source to fill in the gaps that are present in the other. So, in cases where extracting feature from a single source is not possible, such learning techniques are used. In the case of this model, output of the neural and the fuzzy layers are combined in the fusion layers. However, this is quite a trivial task. As the output of the sigmoidal nodes as well as the fuzzy nodes both lie in the range (0,1), combining the results is quite simple.

$$a_i^{(l)} = (w_d)_i^{(l)}(o_d)^{(l-1)} + (w_f)_i^{(l)}(o_f)^{(l-1)} + b_i^{(l)}$$

Here, the output a_i takes in the output and connection weight of the deep representation part, given by o_d and w_d respectively. Similarly, for the fuzzy part, the output and the weights are o_f and w_f.

Task Driven Layer:

The final layer takes input from the fusion layers and provides corresponding class probabilities. Class probabilities are computed using the soft-max function. Given that f_i is the ith input, y_i is the label of the input and $\pi_\theta(fi)$ is the transformation of the input after passing throught the network, then, the output of the task driven layer is given as:

$$\hat{y}_{ic} = p(y_i|f_i) = \frac{e^{w_c\pi_\theta(f_i)+b_c}}{\sum_c e^{w_c\pi_\theta(f_i)+b_c}}$$

For a given class c, w_c is the regression coefficient and b_c is the bias. $\hat{y}_i = [\hat{y}_{i1}, \ldots, \hat{y}_{ik}]$ is the list of labels for a k-class problem.

The training is done in two distinct phases - learning and fine-tuning. Initially, all the biases are set to 0 which the weights between the nodes are randomly sampled from an uniform distribution $U\left[-\frac{1}{\sqrt{n_k}}, \frac{1}{\sqrt{n_k}}\right]$ [49] where n_k is the number of nodes in the preceding layer. All the weights between the fuzzy nodes are set as 1.

To set the parameters, that is, the mean and the standard deviation, of the fuzzy membership nodes, a fast clustering algorithm is run on the input data set. A k-means clustering is done where k is the number of classes. This clustering gives the required parameters after which the model is trained. The network is trained using back-propagation. For fine-tuning, stochastic gradient descent is used.

The model is also compared with a standard DNN as well as a sequential fuzzy DNN (SFDNN). The DNN only uses the neural representation part while the SFDNN fuzzifies data first and then uses it for prediction. Two benchmarks were used to compare the models.

The first is a natural scene image classification problem. The dataset has 4,500 images and 15 classes. Two-thirds of the data was used for training while the rest was used for testing. The FDNN had approximately 0.8% improvement over the other models.

The second benchmark is Stock Trend Prediction. This was a three-class problem, where the model would predict if the stock prices would drop, stay the same or rise. In this case too, the FDNN showed marginal improvement over the other competing models.

Applications

Deep learning models with fuzzy counterparts have been implemented in various studies and real-life applications.

Fuzzy Restricted Boltzmann Machines have been the inspiration behind Pythagorean fuzzy deep Boltzmann machine that is used for Airline passenger profiling and also developing an industrial accident early warning system [22]. Fuzzy restricted Boltzmann machines have also been used for designing models for network traffic prediction [50]. It has also been used for traffic flow prediction [51] and radar target recognition [52]. Fuzzy Deep Learning architecture proposed by Park et al. has been further used for gestational diabetes data analytics [53], real-time tumor tracking, mortality prediction in ICUs [54], as well as license plate image segmentation [55]. Stacked auto-encoders trained used for urban traffic incident detection and traffic congestion detection [56].

Deep learning with fuzzy granulated inputs has been used in multiple time series predictions such as drought prediction [57], prediction of algal blooms [58], and even road safety analysis [59]. Models with fuzzified outputs also have extensive applications. It has been used in

gearbox fault analysis [60], sarcasm detection in tweets [61], recognizing human activity in smart homes [62], and for assessing the performance of online health question-answering services [63]. The model in has been used for driver drowsiness detection [64] using wearable devices, brain tissue image analysis [65] and in surveillance scene representation [66].

Chapter 3:

Methodology

Methodology

This section describes the implementation details of fuzzy dilated convolutions and fuzzy pooling in details. This chapter is divided into two sections. The first section talks about fuzzy dilated convolutions and the second part talks about pooling.

Fuzzy Dilated Convolutions

Fuzzy dilated convolution is a variant of the convolution operation aims to improve the performance of convolutional neural networks.

Overview of Convolution

In deep learning architectures, Convolutional Neural Networks [67], or CNNs or ConvNets are a family of deep neural networks that are most commonly applied in the field of computer vision, or in the case of data that has a grid like topology. CNNs are based on the convolution operation. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

In image processing, convolution is a common technique that changes the intensities of pixels to reflect the intensities of surrounding pixels. A common use of convolution is to create image filters. Using convolution, one can get popular image effects like blur, sharpen, and edge detection. This process is used by deep neural networks to extract features such as edges, color patches and more. The filter, or the kernel matrix is determined by the training algorithm so that the network extracts the most expressive features from the data [68].

In its most general form, convolution is an operation on two functions of a real-valued argument [10]. This can be exemplified in the following manner.

Suppose, a distance to a moving object is being tracked using some kind of a sensor. The sensor provides x(t), a single output, that is the distance of the object at time t. Both x and t are real values. Suppose, the measurement is affected by noise and to reduce the effect of the noise, the measurement has to be averaged over time. More recent measurements are more accurate than previous ones. So, this can be done by giving more weight to the recent measurements, and less weight to the older ones. Suppose, *w(a)* is a weighting function, and ***a*** denotes the age of the measurement. If the averaging occurs at every moment, then a smoothed measurement will take the form

$$s(t) = \int x(a)w(t-a)da$$

This operation is known as convolution and is conventionally denoted using the asterisk (∗).

$$s(t) = (x * w)(t)$$

In this example, w should be a valid probability function and 0 for all negative arguments. In the case of convolution, x is referred to as the input, the weight is known as the kernel and the output is often called a feature map. If x and w are defined only on integer t, then the discrete convolution operation takes the form:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. These multidimensional arrays are referred to as tensors.

Finally, convolutions are often performed over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Since, convolution is commutative, we can also write:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

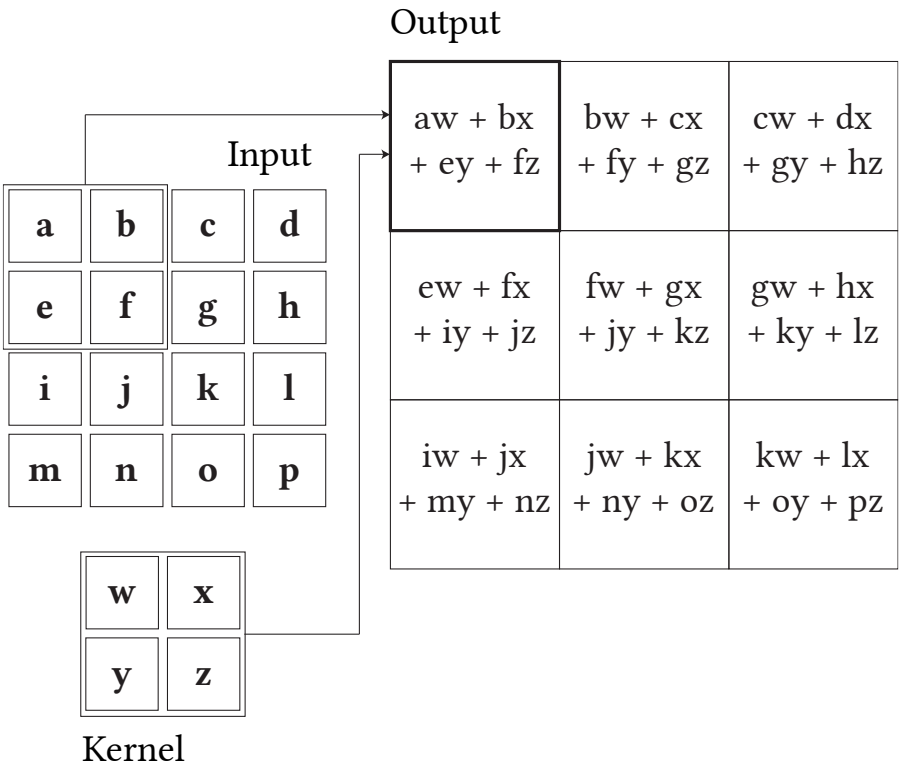

Figure 8. Overview of the Convolution operation.

Convolutions are implemented as a sliding window operation in convolutional neural networks. The kernel slides over the input and computes every element of the output or activation map or feature map. Obviously, size of the feature map is smaller than the size of the input. To prevent the reduction is the size of the input as convolutions happen one after the other, the input is often padded with zero on all sides. This is called zero padding, or just padding. The number of cells the kernel moves on the input matrix is called the stride of the convolution operation.

Now, assuming that the input is a $i \times i$ 2-D matrix, with a square kernel of size $k \times k$ and with stride s and padding p along both axes, the size of the output or the feature map can be given as:

Relationship 1:

$$o = floor\left(\frac{i + 2p - k}{s}\right) + 1$$

Conventionally, at a single stage of convolution, multiple kernels are used and give rise to multiple feature maps. These feature maps are stacked together and are passed on to the next stage where they may be subsampled or further convolved. As more convolutions happen, the feature map increases in depth.

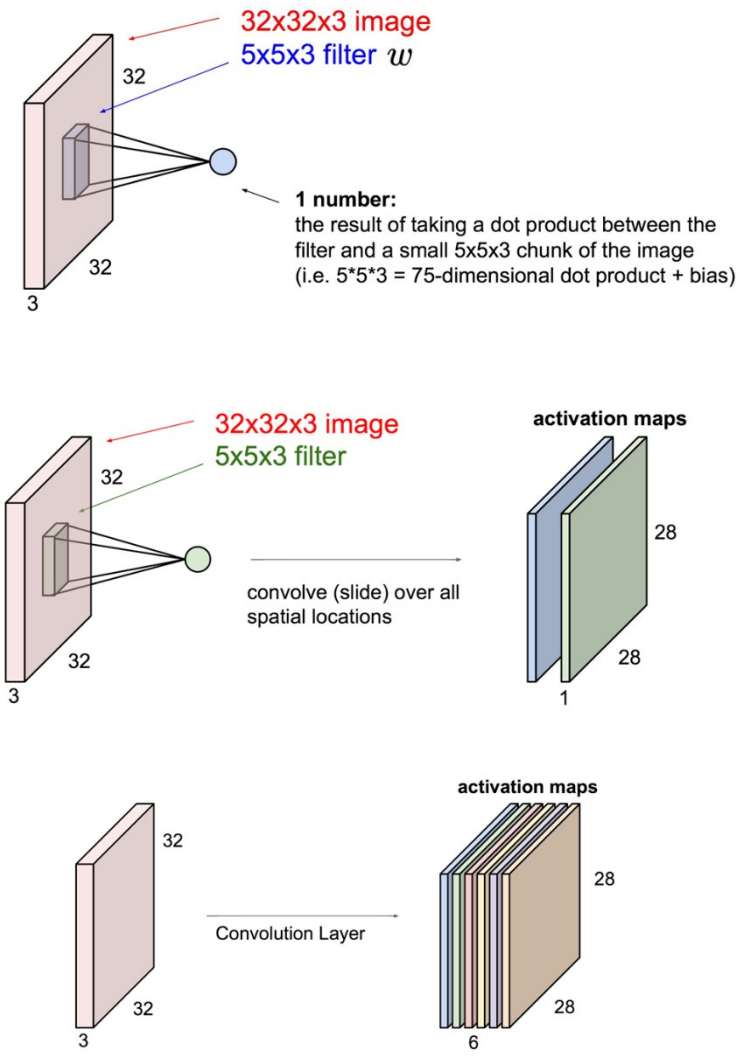

Figure 9. Stacking activation maps after convolving with different kernels.

Therefore, the previous equation can be summarized as follows:

Suppose, a convolutional layer in a CNN accepts a volume of size W_1 x H_1 x D_1. It will require four manually supplied parameters or hyperparameters:

1. Number of filters: N
2. their spatial extent or their size: K
3. the stride: S
4. The amount of zero padding: P

Produces a volume of size W_2 x H_2 x D_2 where

- $W_2 = (W_1 - K + 2P)/S+1$
- $H_2 = (H_1 - K + 2P)/S+1$
- $D_2 = N$

Dilation in Convolution

Dilated convolutions [69] "inflate" the kernel by inserting spaces between the kernel elements. The dilation "rate" is controlled by an additional hyperparameter d. Implementations may vary, but there are usually d−1 spaces inserted between kernel elements such that d = 1 corresponds to a regular convolution. Dilated convolutions are used to cheaply increase the receptive field of output units without increasing the kernel size, which is especially effective when multiple dilated convolutions are stacked one after another.

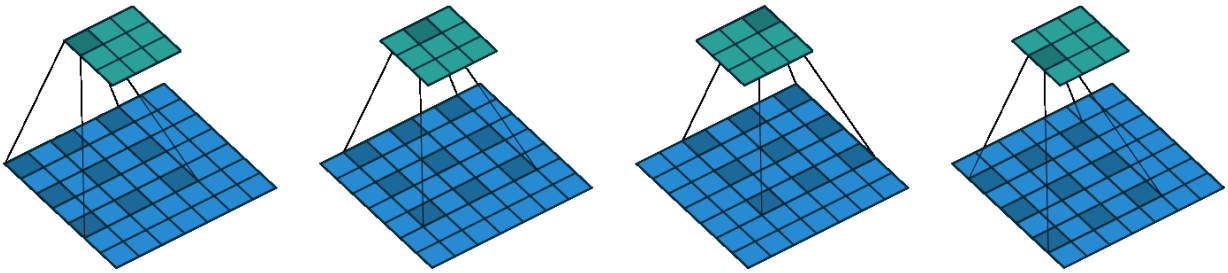

Figure 10. Dilated convolution: Convolving a 3×3 kernel over a 7×7 input with a dilation factor of 2, stride 1 and padding 0.

To understand the relationship between the dilation rate *d* and the output size *o*, it is useful to think of the impact of d on the effective kernel size. A kernel of size *k* dilated by a factor *d* has an effective size:

$$\hat{k} = k + (k-1)(d-1)$$

This can be used to generalize relationship 1 as follows:

Relationship 2:

$$o = floor\left(\frac{i + 2p - k - (k-1)(d-1)}{s}\right) + 1$$

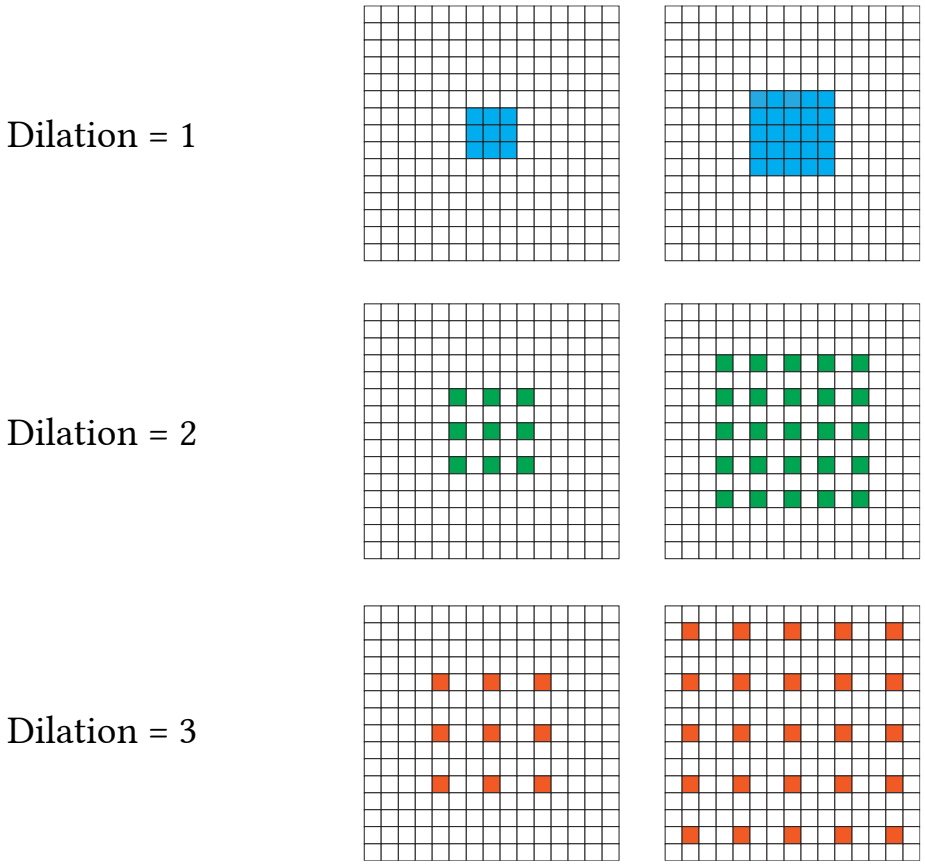

Figure 11. Receptive fields of kernels having dilation 1, 2 and 3

Fuzzy Dilation

Like standard dilation, fuzzy dilation also uses a kernel where the elements are separated by zeros. However, it also takes into account part of the pixel values of the input that are overlapped by the dilating zeros of the kernel. Convolution implemented using fuzzy dilation takes in another hyperparameter, that is, the membership value, μ.

In dilated convolutions, because of the spaces that are inserted in the kernels, some parts of the input matrix that are overlapped by these spaces are ignored. This is truer in the case where the stride has a value more than one. For instance, a 3×3 kernel, with dilation 2 and stride 2 will completely ignore alternating rows and columns of the matrix. This makes dilation disadvantageous to use in these cases. Again, using a larger kernel is computationally intensive, since every single element of the kernel will require training. So, to prevent such instances, fuzzy dilation is proposed.

In fuzzy dilation, every element of the kernel not only affects the underlying element of the input, but also its neighbors. The proportion is determined by the membership value, μ. Convolution is performed in two parts, as described before: a product followed by summation. μ parts of the underlying pixel is taken, and $(1 - \mu)/(d^2-1)$ parts of the neighboring pixels are taken during calculation of the product, where d is the dilation value.

This is exemplified below:

Suppose the input matrix is:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} \end{bmatrix}$$

And the kernel matrix is a 3×3 matrix with dilation 2, given as:

$$K = \begin{bmatrix} k_{1,1} & 0 & k_{1,3} & 0 & k_{1,5} \\ 0 & 0 & 0 & 0 & 0 \\ k_{3,1} & 0 & k_{3,3} & 0 & k_{3,5} \\ 0 & 0 & 0 & 0 & 0 \\ k_{5,1} & 0 & k_{5,3} & 0 & k_{5,5} \end{bmatrix}$$

Then, the value of the output is given as:

$$A *_f k = \mu(k_{1,1}.a_{1,1}) + (1-\mu)/3.a_{1,2} + \mu(k_{1,3}.a_{1,3}) + (1-\mu)/3.a_{1,4} + \mu(k_{1,5}.a_{1,5}) + (1-\mu)/3.(a_{2,1} + a_{2,2} + a_{2,3} + a_{2,4} + a_{2,5}) + \mu(k_{3,1}.a_{3,1}) + (1-\mu)/3.a_{3,2} + \mu(k_{3,3}.a_{3,3}) + (1-\mu)/3.a_{3,4} + \mu(k_{3,5}.a_{3,5}) + (1-\mu)/3.(a_{4,1} + a_{4,2} + a_{4,3} + a_{4,4} + a_{4,5}) + \mu(k_{5,1}.a_{5,1}) + (1-\mu)/3.a_{5,2} + \mu(k_{5,3}.a_{5,3}) + (1-\mu)/3.a_{5,4} + \mu(k_{5,5}.a_{5,5})$$

Where $*_f$ is the fuzzy dilated convolution operation, d = 2 and hence, $(d^2-1) = 3$.

Hence, the convolution operation can be rewritten as:

$$S(i,j) = (K * I)(i,j)$$
$$= \mu \sum_{m;m(modd)\equiv 0} \sum_{n;n(modd)\equiv 0} I(i-m, j-n)K(m,n)$$
$$+ (1-\mu)/(d^2-1) \sum_{m;m(modd)\not\equiv 0} \sum_{n;n(modd)\not\equiv 0} I(i-m, j-n)K(m,n)$$

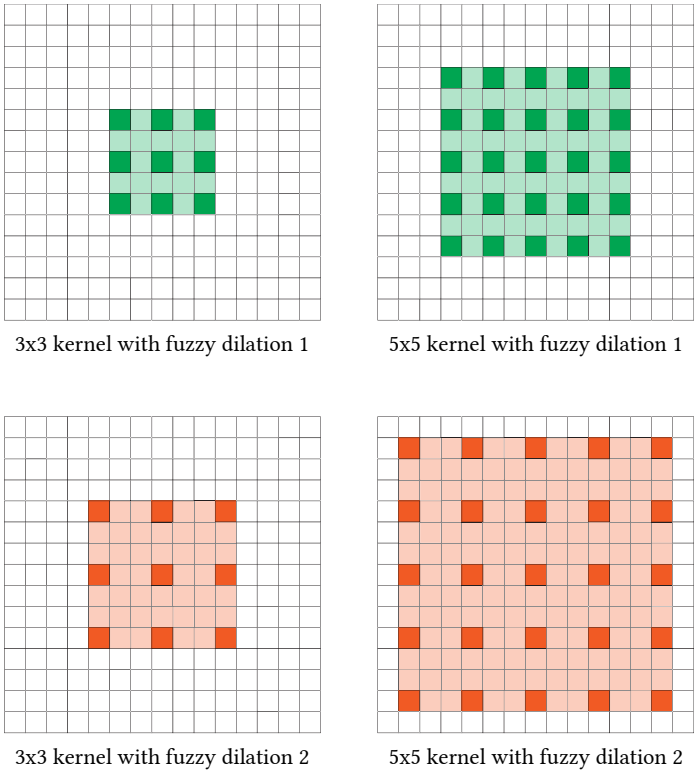

Figure 12. Receptive fields of fuzzy dilated kernels.

Comparison between Dilation and Fuzzy Dilation

The following section shows the effect of a kernel without dilation and along with kernel with normal dilation and fuzzy dilation. Figure 13a (left) is the original image which is processed using a 3x3 edge detect kernel in Figure 13a (right). The effect of using a dilated kernel and a fuzzy dilated kernel is demonstrated in figure 13b.

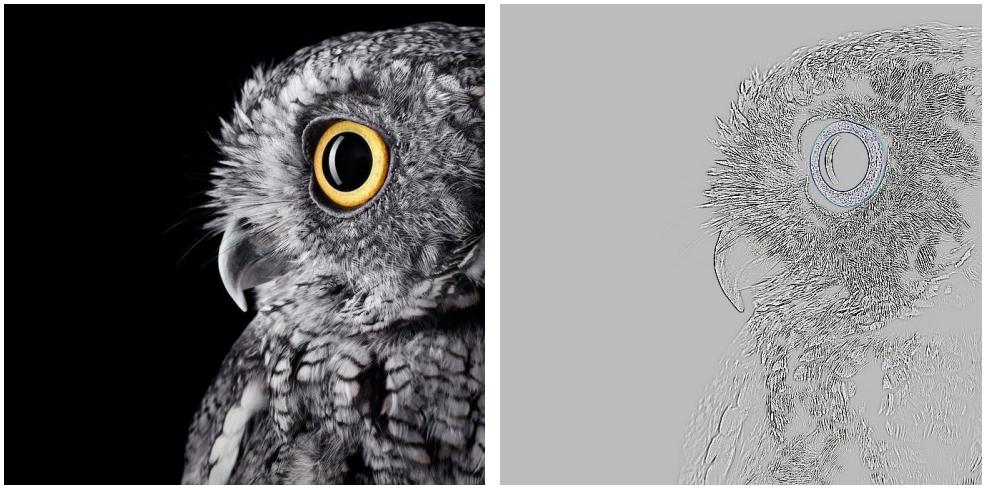

Figure 13 a. Original image (left) and image convolved with edge detect kernel (right) with dilation set to 1, i.e., without any dilation.

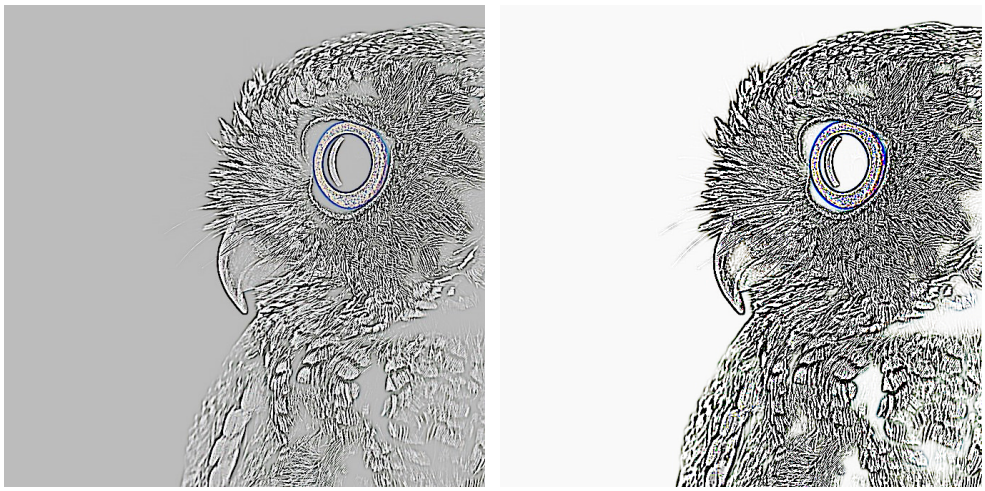

Figure 13 b. image convolved with edge detect kernel with dilation 2 (left) and image convolved with edge detect kernel (right) with dilation set to 2 and $\mu = 0.2$.

Fuzzy Pooling

The proposed Fuzzy Pooling scheme uses the ordered weighted average operator to combine different types of pooling techniques.

Overview of Pooling

It is common to periodically insert a Pooling layer in-between successive Convolution layers in a ConvNet architecture [70][10]. Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX or AVG operation. The purpose of pooling is to transform the joint feature representation into a more usable one that preserves important information while discarding irrelevant details. The employment of pooling layer in CNNs aims to achieve invariance to changes in position or lighting conditions, robustness to clutter, and compactness of representation. In general, the pooling layer summarizes the outputs of neighboring groups of neurons in the same kernel map. In the pooling layer, the resolution of the feature maps is reduced by pooling over local neighborhood on the feature maps of the previous layer, thereby enhancing the invariance to distortions on the inputs. In CNNs, there are two conventional pooling methods, including max pooling and average pooling. The max pooling method selects the largest element in each pooling region as:

$$y_{k,i,j} = \max_{(p,q)\in R_{i,j}} x_{k,p,q}$$

Where $y_{k,i,j}$ is the output of the pooling operation related to the kth feature map. $x_{k,p,q}$ is the element at (p, q) within the pooling region $R_{i,j}$ which represents a local neighborhood around the position (i,j). For the average pooling method, it takes the arithmetic mean of the elements in each pooling region as:

$$y_{k,i,j} = \frac{1}{|R_{i,j}|} \sum_{(p,q)\in R_{i,j}} x_{k,p,q}$$

Where $|R_{i,j}|$ is the size of the pooling region.

These are the two most commonly used pooling operators since they are quite easy to compute and hence, they are so widely used. Furthermore, they have provided decent results in a lot of use cases.

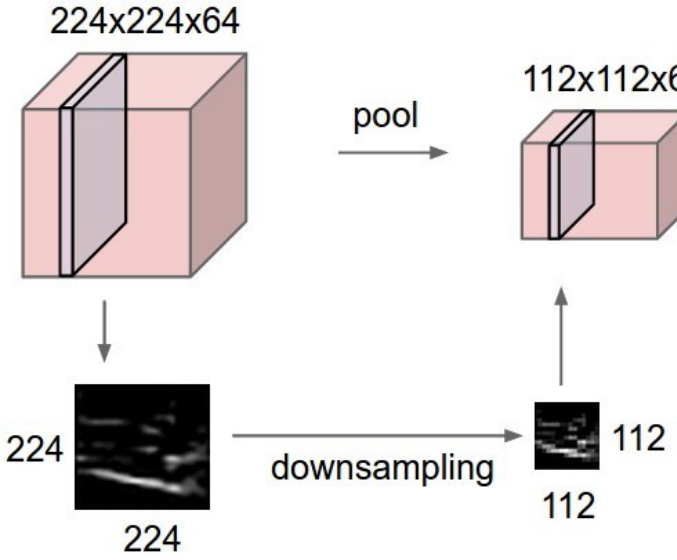

Figure 14 a. How pooling is used for layer by layer subsampling of activation maps [71].

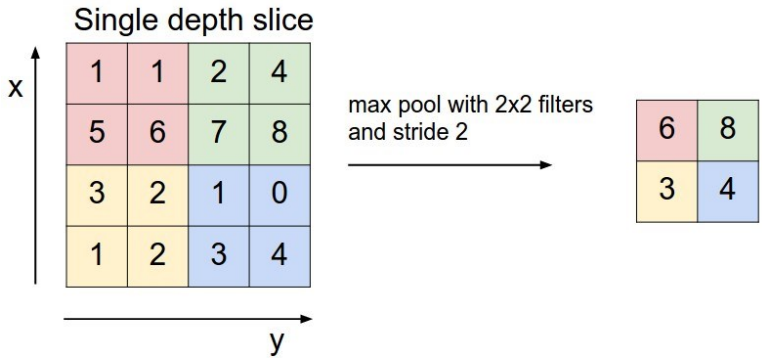

Figure 14 b. Illustration of max pooling with 2x2 filter or window and with stride 2 [71].

Using Ordered Weighted averaging aggregation operator to combine pooling schemes

Overview of Ordered Weighted averaging aggregation operator

In mathematics and fuzzy logic, the ordered weighted averaging (OWA) operators provide a parameterized class of mean type aggregation operators [72]. Formally, an OWA operator of order **n** is a mapping $F: R_n \to R$ that has an associated collection of weights $W = [w_1, w_2, \dots, w_n]$ lying in the unit interval and summing to one with:

$$F(a_1, \dots, a_n) = \sum_{j=1}^{n} w_j b_j$$

Where b_j is the j^{th} largest in the collection a_i.

Motivation

As mentioned before, the max pooling and average pooling methods are two popular choices employed by CNNs due to their computational efficiency. Although these two kinds of pooling operators can work very well on some datasets, it is still unknown which will work better for addressing a new problem. In another word, it is a kind of empiricism to choose the pooling operator. On the other hand, both the max pooling and average pooling operators have their own drawbacks. About max pooling, it only considers the maximum element and ignores the others in the pooling region. Sometimes, this will lead to an unacceptable result. For example, if most of the elements in the pooling region are of high magnitudes, the distinguishing feature vanishes after max pooling as shown in Fig. 15. In the case of average pooling, it calculates the mean of all the elements within the pooling region. This operator will take all the low magnitudes into consideration and the contrast of the new feature map after pooling will be reduced. Even worse, if there are many near-zero elements, the characteristic of the feature map will be reduced largely, as illustrated in Fig. 15.

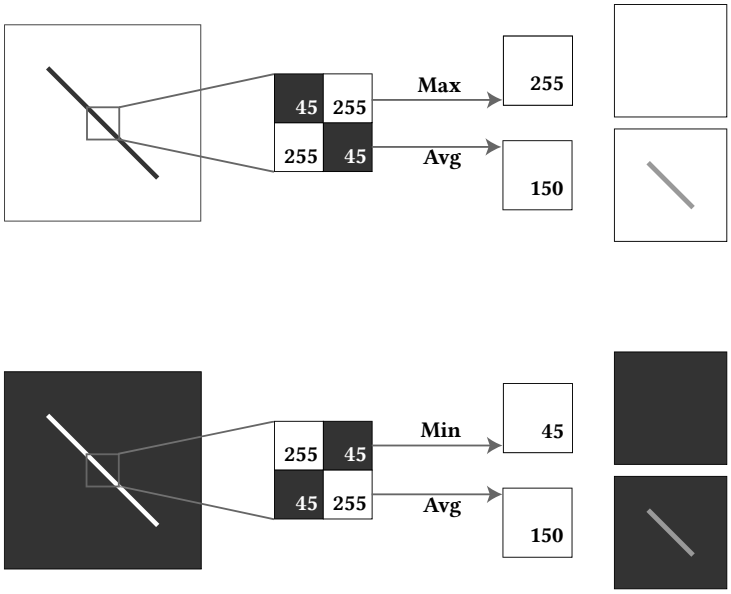

Figure 15. Toy example illustrating the drawbacks of max pooling (top) and average pooling (bottom).

Pooling Scheme

The proposed pooling scheme is inspired by the concept of weighted averaging operator. During the pooling operation, the maximum element in the pooling region as well as the minimum element in the pooling region may be equally important. Therefore, using the OWA operator, the maximum, the minimum and the average values are combined based on their proportion in the pooling region. The pooling scheme checks how important each of the max, min and average values are and then subsamples the region accordingly. This is done as a part of the forward pass itself and hence, the pooling layer, separately does not contain any trainable parameters. This ensures that the performance hit is kept to a minimum.

For using the OWA operator, the calculation of the weights is important. The weights are calculated using the softmax scores of the sum of the maximum, average and the minimum of the input. For every depth slice, for every window in each depth slice, the max, average and the min values are calculated. The maximum, average and the minimum values are aggregated by performing a sum. The list of the sums is scaled using the softmax function, and the softmax scores are used to perform the weighted average. The softmax function encourages the maximum values, but if the average and minimum values are high, then it

adjusts the proportion of the values so that the minimum and the average values are considered too. The effect of this pooling scheme is shown in Figure 16 using toy examples.

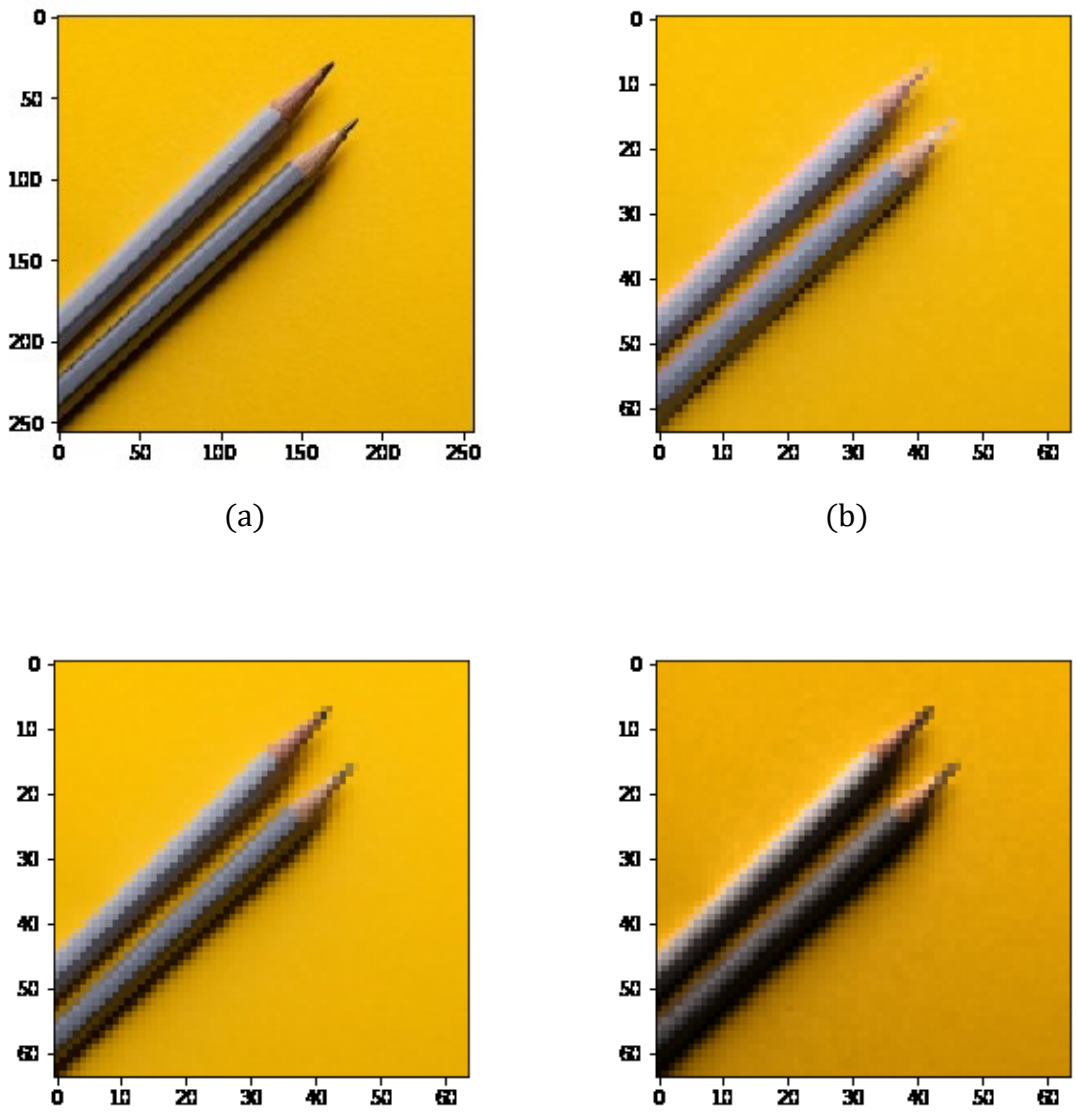

Figure 16. a. Original image; b. image processed using max pooling; c. image processed using average pooling, d. image processed using fuzzy pooling.

Chapter 4:

Data Description and Results

Data description

For the purpose of benchmarking the models, three different datasets were used: The MNIST digits dataset, the CIFAR-10 images dataset and a digital pathology images dataset.

The first dataset is the commonly used MNIST dataset. The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

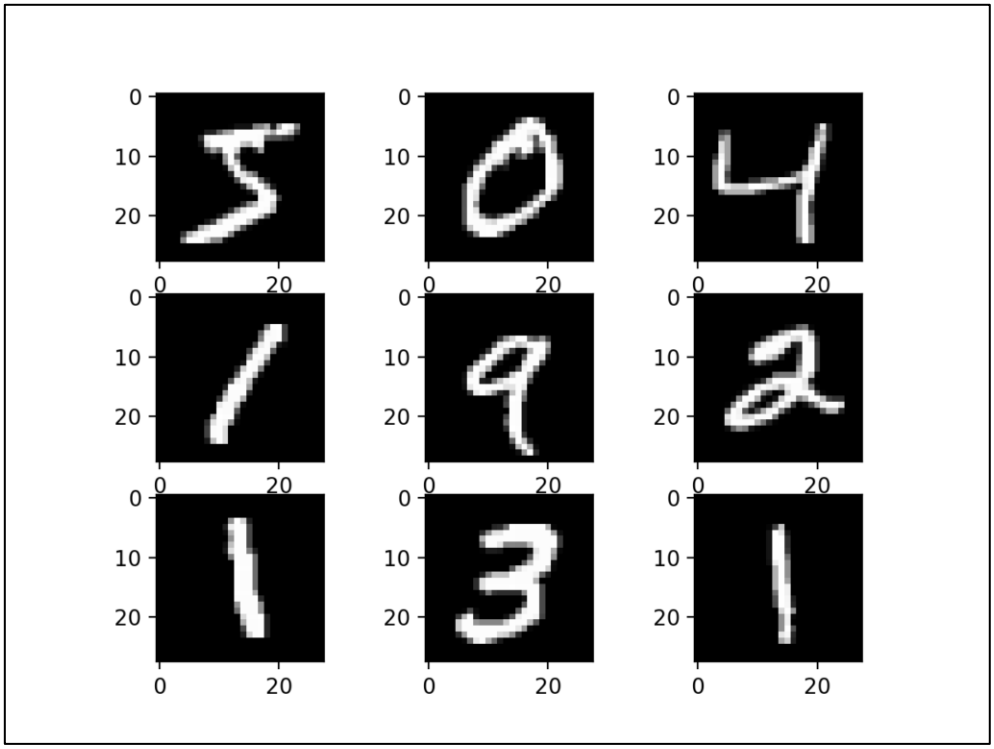

Figure 17. MNIST handwritten digits dataset

The second dataset is the CIFAR-10 dataset. This dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000

images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. The ten classes in the CIFAR-10 dataset is airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.

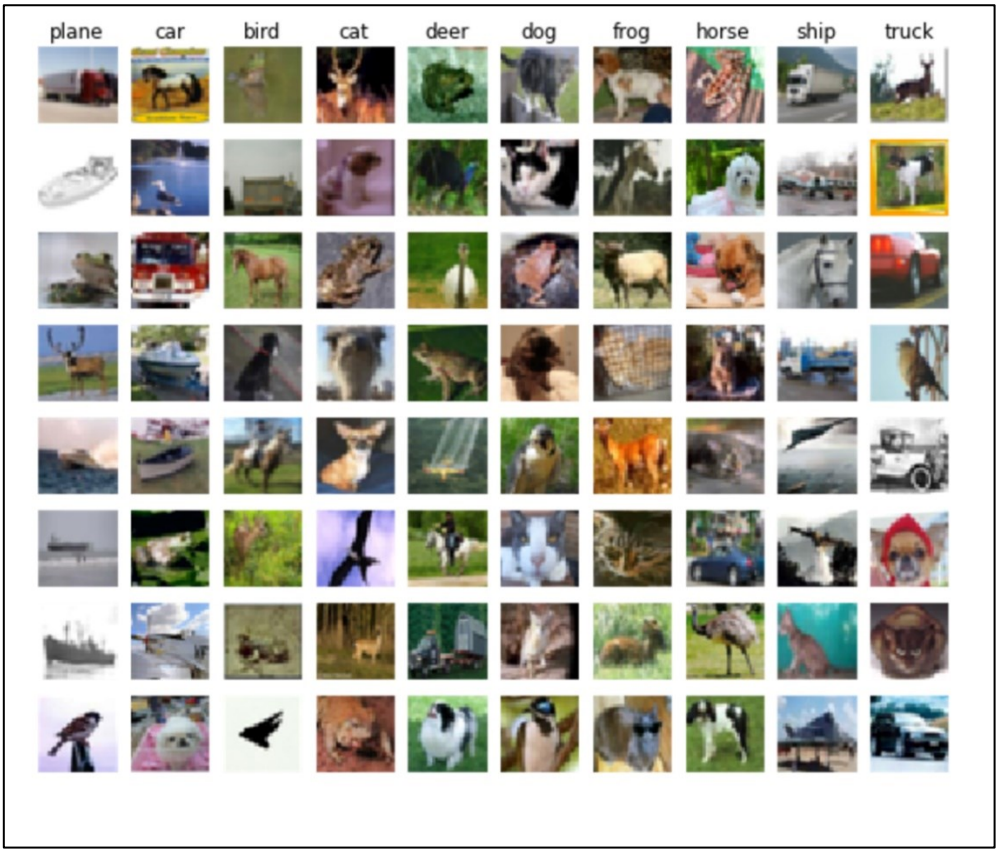

Figure 18. CIFAR-10 images dataset

The third dataset contains digital pathology images of breast cancer specimens [73]. Invasive Ductile Carcinoma or IDC is the most common type of breast cancer. While studying whole mount samples, pathologists generally focus on regions which contain the carcinoma. In this dataset, there are 162 whole mount slide images of breast cancer samples which are scanned at 40x. From these samples, patches of dimensions 50 pixels x 50 pixels were extracted, totaling to 277,524 patch images. Out of these, 198,738 patches are negative while the remaining 78,786 are positive.

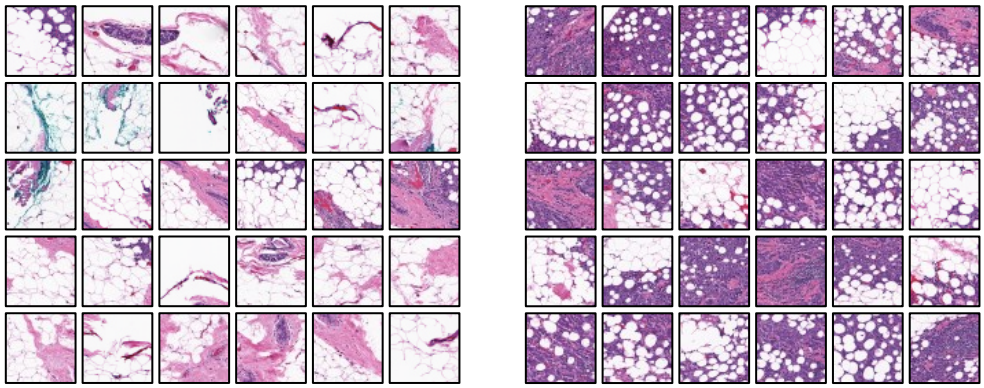

Figure 19. Breast cancer histopathology image patches showing non-IDC (left) and IDC (right) images

Results

The following section discusses the performance of the different models on the dataset. A different deep learning model was designed for each dataset to ensure that the model is appropriate to the dataset being used. This baseline model is further modified with the proposed convolution and pooling layer and was trained separately. The following three sections are divided based on the datasets. Each section contains a figure illustrating the baseline model.

For each model, precision, recall, macro-F1 score and accuracy was calculated. This was done using the following formula:

Precision (P) is defined as the number of true positives (tp) over the number of true positives plus the number of false positives (fp).

$$P = \frac{tp}{tp + fp}$$

Recall (R) is defined as the number of true positives (tp) over the number of true positives plus the number of false negatives (fn).

$$\mathrm{R} = \frac{tp}{tp + fn}$$

These quantities are also related to the (F1) score, which is defined as the harmonic mean of precision and recall, given as:

$$F1 = 2\frac{P \times R}{P + R}$$

Along with these metrics, the accuracy of the model is also given. This is given as:

$$Acc = \frac{tp + tn}{tp + tn + fp + fn}$$

MNIST Handwritten digits

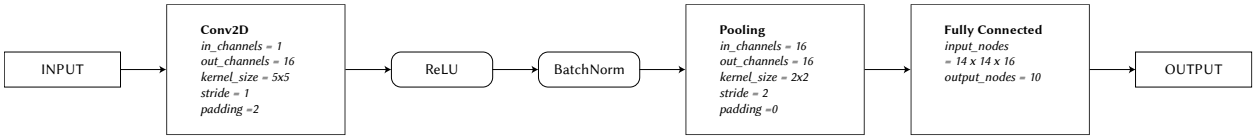

Figure 20: Baseline model for MNIST dataset

The baseline model for the MNIST dataset uses Max Pooling. This pooling layer was replaced by the fuzzy pooling for the experiment. While testing the performance of the fuzzy dilated convolution, the first convolution layer was replaced with a fuzzy dilated convolution layer with $\mu = 0.1$ and dilation = 2. All the models were trained for 5 epochs. The performance metrics are given below:

Table 2: Performance of the different variants of the CNN on the MNIST dataset

	Precision	*Recall*	*macro-F1*	*Accuracy (%)*
Baseline	0.9857	0.9585	0.9711	97.14
Fuzzy Convolution	0.9887	0.9847	0.9867	98.33
Fuzzy Pooling	0.9828	0.9769	0.9818	98.20

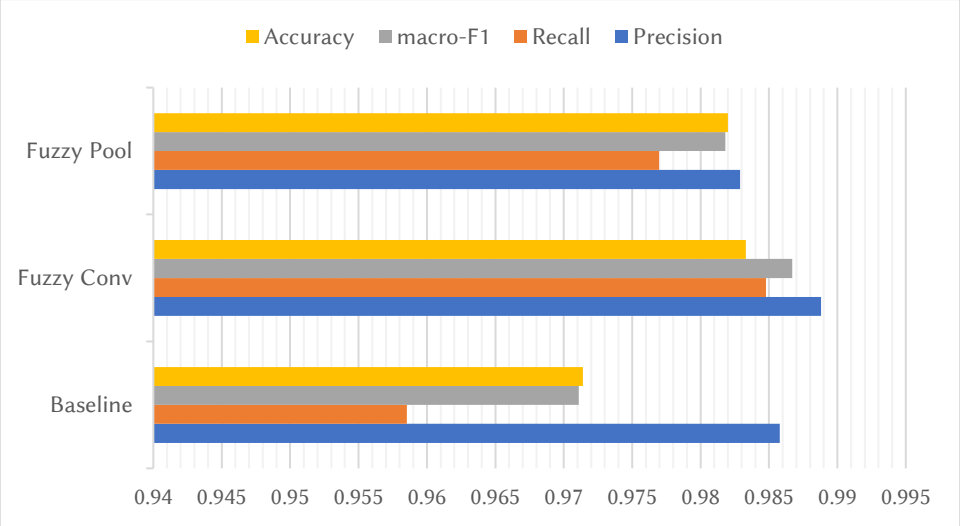

Figure 21. Comparison of performance of different variants of the CNN on the MNIST dataset

CIFAR-10

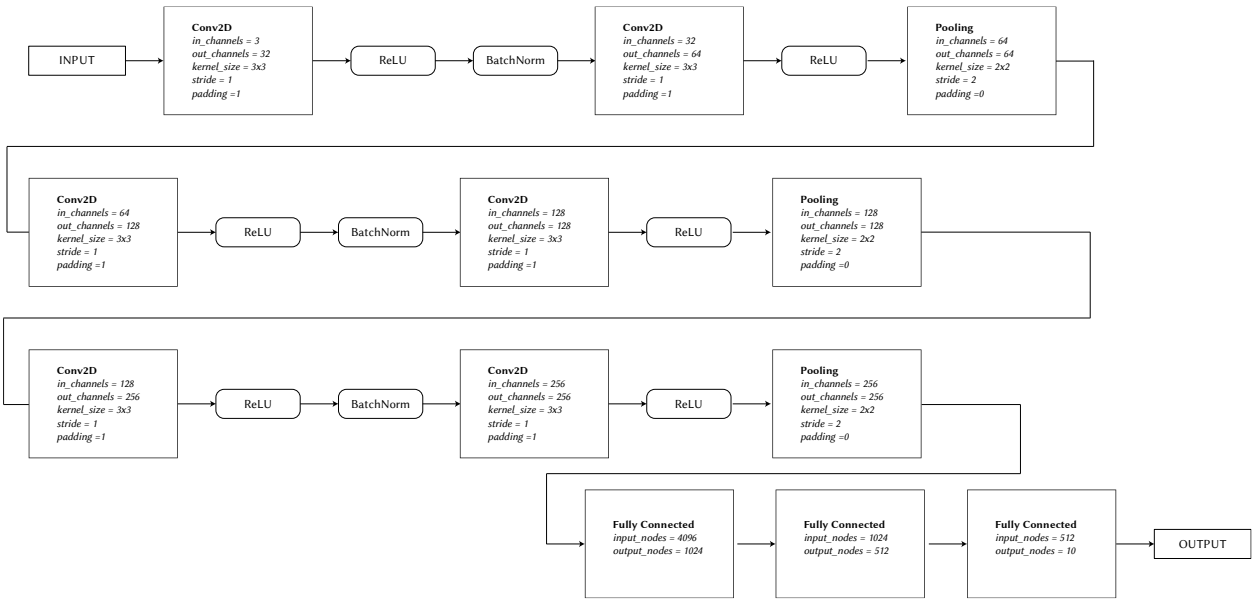

Figure 22. Baseline model for CIFAR-10 dataset

For the CIFAR-10 dataset, all the pooling layers are max pooling. For the model with fuzzy pooling, every odd numbered pooling layer from the beginning were converted to fuzzy while

the remaining were left unchanged. Similarly, for the fuzzy dilated convolution, the first, third and the fifth convolution layers were replaced with fuzzy dilated convolution layers with $\mu = 0.3$ and dilation = 2. All the models were trained for 10 epochs.

Table 3. Performance of the different variants of the CNN on the CIFAR-10 dataset

	Precision	*Recall*	*macro-F1*	*Accuracy (%)*
Baseline	0.9125	0.6854	0.7839	73.8
Fuzzy Convolution	0.816438	0.766067	0.790451	79.8
Fuzzy Pooling	0.894292	0.781885	0.83432	77.79

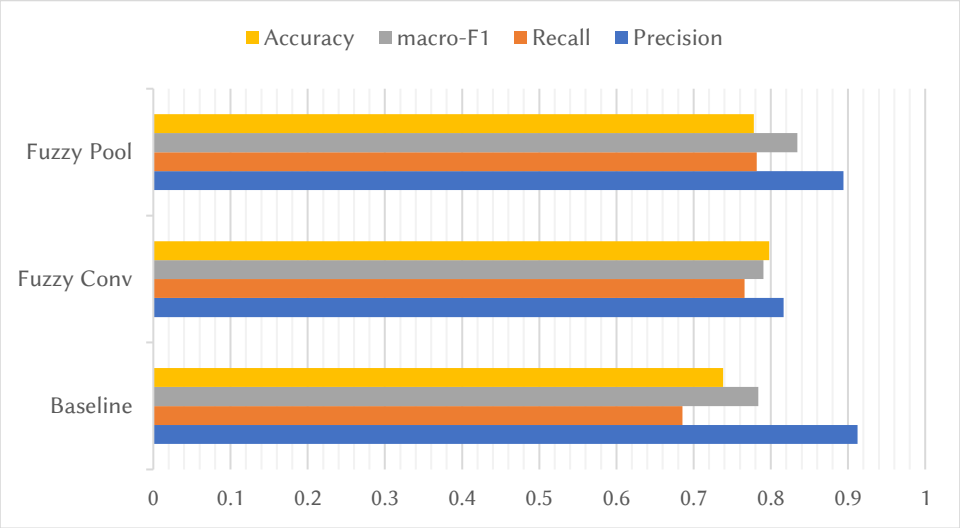

Figure 23. Comparison of performance of different variants of the CNN on the CIFAR-10 dataset

Breast Cancer Histopathology Images

The CNN for Breast Cancer Histopathology images is divided into five blocks, each containing convolution, batch normalization, Leaky Rectified Linear activation, and Max pooling in order. At the end of the five blocks, there is an average pooling layer with 8x8 window size. For testing the fuzzy dilated convolution and the fuzzy pooling layer, the first,

third and the fifth blocks were modified while the rest were left untouched. The fuzzy dilated convolution layers had with $\mu = 0.2$ and dilation = 2.

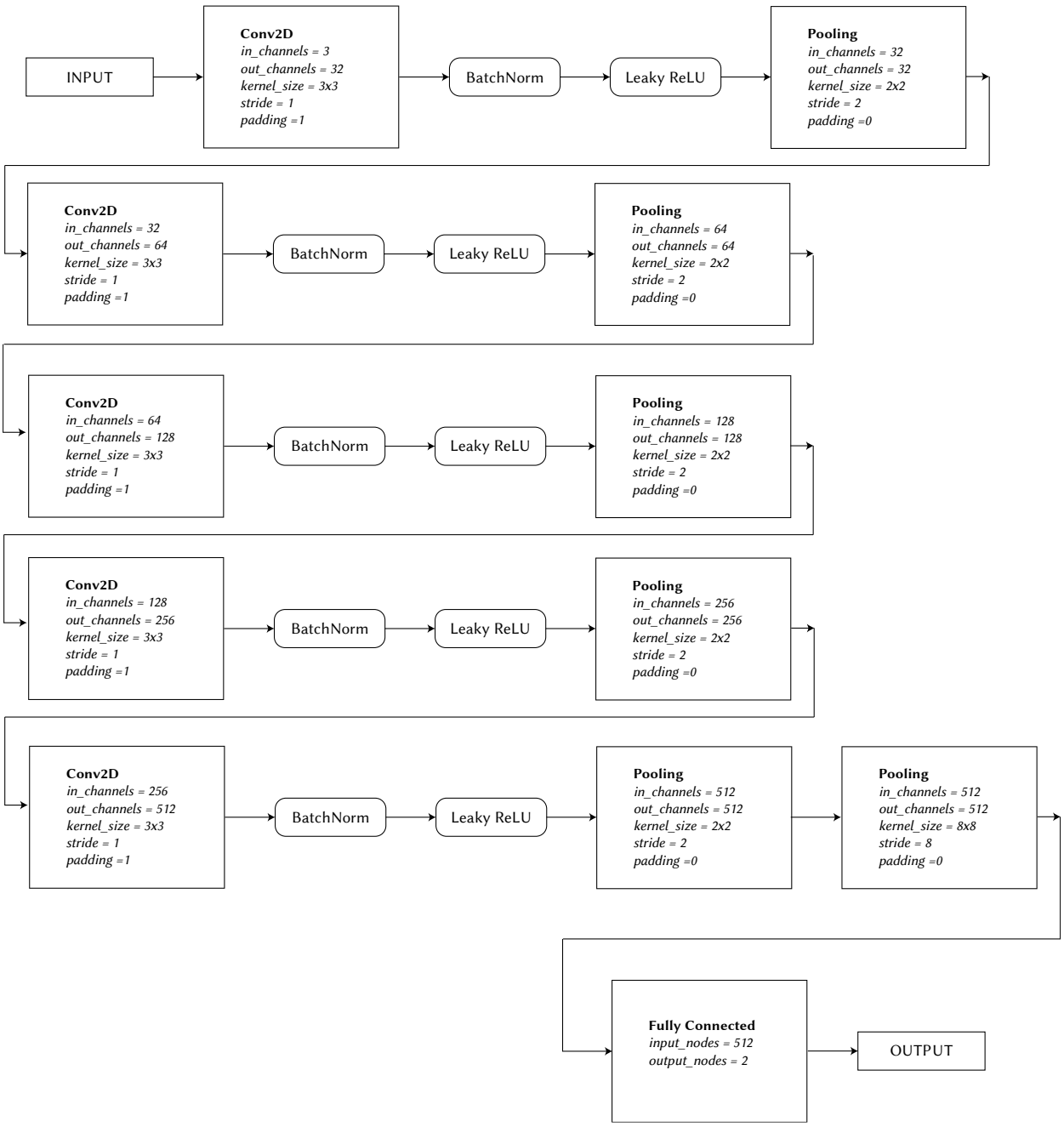

Figure 24. Baseline model for Breast Cancer Histopathology images dataset

Table 3. Performance of the different variants of the CNN on the Breast Cancer Histopathology images dataset

	Precision	*Recall*	*macro-F1*	*Accuracy (%)*
Baseline	0.932243	0.458621	0.614792	87
Fuzzy Conv	0.894292	0.781885	0.83432	87.5
Fuzzy Pool	0.894292	0.781885	0.83432	90.65

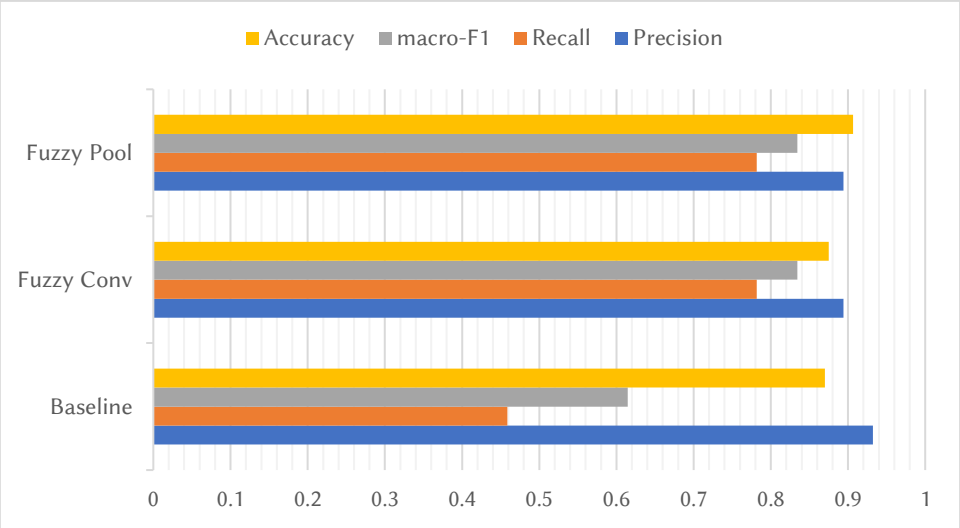

Figure 25. Comparison of performance of different variants of the CNN on the Breast Cancer Histopathology images dataset

Chapter 5:

Conclusion and Future Scope

Conclusion

From the results section, it is evident that incorporation of fuzzy pooling and fuzzy dilated convolution marginally improves performance. In the experiments, fuzzy convolution layers and the fuzzy pooling layers were used alongside standard convolution and max-pooling layers as it was seen that using the proposed fuzzy layers along with the standard layers improve the performance of the existing convolutional neural networks. In most of the cases, incorporation of the fuzzy layers improved the accuracy and the macro-F1 scores during testing.

Deep neural networks, especially convolutional neural networks are being extensively researched and the two proposed methods provide improved performance metrics on standard benchmarking datasets as well as a real-life digital pathology dataset.

Future Scope

For practical large-scale implementations, most deep learning models make use of software libraries that run on GPUs. Multiple software platforms are available, such as Nvidia CUDA, Intel Math Kernel Libraries and AMD ROCm that allow access to distributed parallel computing using GPUs. The implementation proposed in this thesis does not leverage GPU acceleration to the fullest. This made the experimentations time consuming. In future, implementation of the fuzzy dilated convolution as well as fuzzy pooling on GPUs will enable further experimentation using larger datasets.

Fuzzy dilated pooling has another scope of improvement where the membership value hyperparameter can be learned during the training. This may slightly improve the computational complexity of the backward pass, but the number of trainable parameters is still small since the kernel size is smaller. This can reduce the number of epochs needed to train the convolutional neural network,

References

1. LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature* 521.7553 (2015): 436.
2. Hubel, David H., and Torsten N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." *The Journal of physiology* 160.1 (1962): 106-154.
3. Greenspan, Hayit, Bram Van Ginneken, and Ronald M. Summers. "Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique." *IEEE Transactions on Medical Imaging* 35.5 (2016): 1153-1159.
4. Mitchell, Tom M. "Machine learning and data mining." *Communications of the ACM* 42.11 (1999).
5. Andrychowicz, Marcin, et al. "Learning to learn by gradient descent by gradient descent." *Advances in Neural Information Processing Systems*. 2016.
6. Mitra, Sushmita, and Yoichi Hayashi. "Neuro-fuzzy rule generation: survey in soft computing framework." *IEEE transactions on neural networks* 11.3 (2000): 748-768.
7. Maulik, Ujjwal, and Sanghamitra Bandyopadhyay. "Performance evaluation of some clustering algorithms and validity indices." *IEEE Transactions on pattern analysis and machine intelligence* 24.12 (2002): 1650-1654.
8. Yager, Ronald R., and Lotfi A. Zadeh, eds. *An introduction to fuzzy logic applications in intelligent systems*. Vol. 165. Springer Science & Business Media, 2012.
9. Litjens, Geert, et al. "A survey on deep learning in medical image analysis." *Medical image analysis* 42 (2017): 60-88.
10. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
11. Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747*(2016).
12. Robbins, Herbert, and Sutton Monro. "A stochastic approximation method." *The annals of mathematical statistics*(1951): 400-407.
13. Dauphin, Yann N., et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." *Advances in neural information processing systems*. 2014.
14. Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." *International conference on machine learning*. 2013.
15. Dozat, Timothy. "Incorporating nesterov momentum into adam." (2016).
16. Mukkamala, Mahesh Chandra, and Matthias Hein. "Variants of rmsprop and adagrad with logarithmic regret bounds." *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.
17. Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).

18. Tieleman, Tijmen, and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." *COURSERA: Neural networks for machine learning* 4.2 (2012): 26-31.
19. Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*(2014).
20. Botev, Aleksandar, Guy Lever, and David Barber. "Nesterov's accelerated gradient and momentum as approximations to regularised update descent." *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017.
21. Zhang, Ren, Furao Shen, and Jinxi Zhao. "A model with fuzzy granulation and deep belief networks for exchange rate forecasting." *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014.
22. Zheng, Yu-Jun, et al. "A Pythagorean-type fuzzy deep denoising autoencoder for industrial accident early warning." *IEEE Transactions on Fuzzy Systems* 25.6 (2017): 1561-1575.
23. Chen, CL Philip, et al. "Fuzzy restricted Boltzmann machine for the enhancement of deep learning." *IEEE Transactions on Fuzzy Systems* 23.6 (2015): 2163-2173.
24. Park, Seonyeong, et al. "Intra-and inter-fractional variation prediction of lung tumors using fuzzy deep learning." *IEEE journal of translational engineering in health and medicine* 4 (2016): 1-12.
25. Wang, Yani, Zhendong Wu, and Jianwu Zhang. "Damaged fingerprint classification by Deep Learning with fuzzy feature points." *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE, 2016.
26. Chopade, Heena A., and Meera Narvekar. "Hybrid auto text summarization using deep neural network and fuzzy logic system." *2017 International Conference on Inventive Computing and Informatics (ICICI)*. IEEE, 2017.
27. Zheng, Yu-Jun, et al. "Airline passenger profiling based on fuzzy deep machine learning." *IEEE transactions on neural networks and learning systems* 28.12 (2016): 2911-2923.
28. Deng, Yue, et al. "A hierarchical fused fuzzy deep neural network for data classification." *IEEE Transactions on Fuzzy Systems* 25.4 (2016): 1006-1012.
29. El Hatri, Chaimae, and Jaouad Boumhidi. "Fuzzy deep learning based urban traffic incident detection." *Cognitive Systems Research* 50 (2018): 206-213.
30. Tabrizi, Pooneh R., et al. "Automatic kidney segmentation in 3D pediatric ultrasound images using deep neural networks and weighted fuzzy active shape model." *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. IEEE, 2018.
31. Simon, Dan. "Biogeography-based optimization." *IEEE transactions on evolutionary computation* 12.6 (2008): 702-713.
32. Lodwick, Weldon A., and Janusz Kacprzyk, eds. *Fuzzy optimization: Recent advances and applications*. Vol. 254. Springer, 2010.
33. Karnik, Nilesh N., and Jerry M. Mendel. "Centroid of a type-2 fuzzy set." *Information Sciences* 132.1-4 (2001): 195-220.

34. Buckley, James J. *Fuzzy probabilities: new approach and applications*. Vol. 115. Springer Science & Business Media, 2005.
35. Salakhutdinov, Ruslan, and Geoffrey Hinton. "Deep boltzmann machines." *Artificial intelligence and statistics*. 2009.
36. Salakhutdinov, Ruslan, and Hugo Larochelle. "Efficient learning of deep Boltzmann machines." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.
37. Atanassov, Krassimir T. "Intuitionistic fuzzy sets." *Intuitionistic fuzzy sets*. Physica, Heidelberg, 1999. 1-137.
38. Yager, Ronald R. "Pythagorean membership grades in multicriteria decision making." *IEEE Transactions on Fuzzy Systems* 22.4 (2013): 958-965.
39. Ishibuchi, Hisao, Kouichi Morioka, and I. B. Turksen. "Learning by fuzzified neural networks." *International Journal of Approximate Reasoning* 13.4 (1995): 327-358.
40. Ishibuchi, Hisao, Hideo Tanaka, and Hideiko Okada. "Fuzzy neural networks with fuzzy weights and fuzzy biases." *IEEE international conference on neural networks*. IEEE, 1993.
41. Yao, Xin. "Evolutionary artificial neural networks." *International journal of neural systems* 4.03 (1993): 203-222.
42. Tettamanzi, Andrea, and Marco Tomassini. *Soft computing: integrating evolutionary, neural, and fuzzy systems*. Springer Science & Business Media, 2013.
43. Shinozaki, Takahiro, and Shinji Watanabe. "Structure discovery of deep neural network based on evolutionary algorithms." *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015.
44. Shinozaki, Takahiro, and Shinji Watanabe. "Structure discovery of deep neural network based on evolutionary algorithms." *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015.
45. Lee, Suk Jin, et al. "Customized prediction of respiratory motion with clustering from multiple patient interaction." *ACM Transactions on Intelligent Systems and Technology (TIST)*4.4 (2013): 69.
46. Lu, Wei, et al. "A semi-automatic method for peak and valley detection in free-breathing respiratory waveforms." *Medical physics* 33.10 (2006): 3634-3636.
47. Murphy, Martin J., and Damodar Pokhrel. "Optimization of an adaptive neural network to predict breathing." *Medical physics*36.1 (2009): 40-47.
48. Ngiam, Jiquan, et al. "Multimodal deep learning." *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011.
49. Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.
50. Nie, Laisen, et al. "Network traffic prediction based on deep belief network in wireless mesh backbone networks." *2017 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2017.

51. Chen, Weihong, et al. "A novel fuzzy deep-learning approach to traffic flow prediction with uncertain spatial–temporal data features." *Future Generation Computer Systems* 89 (2018): 78-88.
52. Xia, J. Y., X. Li, and Y. X. Liu. "Application of a New Restricted Boltzmann Machine to Radar Target Recognition." *2016 Progress in Electromagnetic Research Symposium (PIERS)*. IEEE, 2016.
53. Moreira, Mário WL, et al. "Evolutionary radial basis function network for gestational diabetes data analytics." *Journal of computational science* 27 (2018): 410-417.
54. Davoodi, Raheleh, and Mohammad Hassan Moradi. "Mortality prediction in intensive care units (ICUs) using a deep rule-based fuzzy classifier." *Journal of biomedical informatics* 79 (2018): 48-59.
55. Rahmat, Basuki, et al. "Vehicle License Plate Image Segmentation System Using Cellular Neural Network Optimized by Adaptive Fuzzy and Neuro-Fuzzy Algorithms." *International Journal of Multimedia and Ubiquitous Engineering* 11.12 (2016): 383-400.
56. Hernandez-Potiomkin, Yaroslav, et al. "Unsupervised Incident Detection Model in Urban and Freeway Networks." *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018.
57. Agana, Norbert A., and Abdollah Homaifar. "A deep learning based approach for long-term drought prediction." *SoutheastCon 2017*. IEEE, 2017.
58. Zhang, Feng, et al. "Deep-learning-based approach for prediction of algal blooms." *Sustainability* 8.10 (2016): 1060.
59. Pan, Guangyuan, Liping Fu, and Lalita Thakali. "Development of a global road safety performance function using deep neural networks." *International journal of transportation science and technology* 6.3 (2017): 159-173.
60. Li, Chuan, et al. "Multimodal deep support vector classification with homologous features and its application to gearbox fault diagnosis." *Neurocomputing* 168 (2015): 119-127.
61. Zhang, Meishan, Yue Zhang, and Guohong Fu. "Tweet sarcasm detection using deep neural network." *Proceedings of COLING 2016, The 26th International Conference on Computational Linguistics: Technical Papers*. 2016.
62. Fang, Hongqing, and Chen Hu. "Recognizing human activity in smart home using deep learning algorithm." *Proceedings of the 33rd Chinese Control Conference*. IEEE, 2014.
63. Hu, Ze, et al. "A deep learning approach for predicting the quality of online health expert question-answering services." *Journal of biomedical informatics* 71 (2017): 241-253.
64. He, Jibo, et al. "Detection of driver drowsiness using wearable devices: A feasibility study of the proximity sensor." *Applied ergonomics* 65 (2017): 473-480.
65. Al-Dmour, Hayat, and Ahmed Al-Ani. "A clustering fusion technique for MR brain tissue segmentation." *Neurocomputing*275 (2018): 546-559.

66. Ahmed, Sk Arif, et al. "Surveillance scene representation and trajectory abnormality detection using aggregation of multiple concepts." *Expert Systems with Applications* 101 (2018): 43-55.
67. Sutskever, Ilya, Geoffrey E. Hinton, and A. Krizhevsky. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* (2012): 1097-1105.
68. Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." *International journal of computer vision*115.3 (2015): 211-252.
69. Yu, Fisher, and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions." *arXiv preprint arXiv:1511.07122* (2015).
70. Yu, Dingjun, et al. "Mixed pooling for convolutional neural networks." *International Conference on Rough Sets and Knowledge Technology*. Springer, Cham, 2014.
71. "CS231n: Convolutional Neural Networks for Visual Recognition." *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition*, cs231n.stanford.edu/.
72. Yager, Ronald R. "On ordered weighted averaging aggregation operators in multicriteria decisionmaking." *IEEE Transactions on systems, Man, and Cybernetics* 18.1 (1988): 183-190.
73. Janowczyk, Andrew, and Anant Madabhushi. "Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases." *Journal of pathology informatics* 7 (2016).