Interpretable Deep Convolutional Fuzzy Networks

by

Mojtaba Yeganejou

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

In

Computer Engineering

Department of Electrical and Computer Engineering

University of Alberta

Abstract

While deep learning has proven to be a powerful new tool for modeling and predicting a wide variety of complex phenomena, those models remain incomprehensible black boxes. This is a critical impediment to the widespread deployment of deep learning technology, as decades of research have found that users simply will not trust (i.e. make decisions based on) a model whose solutions cannot be explained. Fuzzy systems, on the other hand, are by design much more easily understood. We propose to create more comprehensible deep networks by hybridizing them with fuzzy logic. Our proposed architecture first employs a convolutional neural network as an automated feature extractor, and then clusters datasets in that feature space using the Fuzzy C- Means (FCM) and Gustafson and Kessel (GK) clustering algorithms. After hardening the clusters, we employ a fuzzy version of Rocchio's algorithm to classify the data points. We will evaluate our system's performance on three well-known benchmark datasets (MNIST, Fashion MNIST, and CIFAR-10) by comparing our results with most recent published results (including state-of-the-art). Finally, we will demonstrate the interpretability of the hybrid system on the MNIST dataset.

"Do not pursue that of which you have no knowledge. Indeed, the hearing, the sight and the heart - about all those [one] will be questioned."

Quran 17:36

To

Muhammad:
The Messenger of God

Acknowledgements

I would like to thank my supervisor Professor Scott Dick for his great support and encouragement during my M.Sc. studies. I'm sincerely grateful of his guidance and supervision. God bless him.

I express my deepest appreciation to my wife, and my mother who are the greatest blessings that I have been gifted. Without their supports, love and prayers I would never be able to conduct this research and write this thesis.

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
NLP	**N**atural **L**anguage **P**rocessing
CNN	**C**onvolutional **N**eural **N**etwork
SGD	**S**tochastic **G**radient **D**escent
UMAP	**U**niform **M**anifold **A**pproximation and **P**rojection
FCM	**F**uzzy **C**-**M**eans
GK	**G**ustafson **K**essel
MNIST	**M**ixed **N**ational **I**nstitute of **S**tandards and **T**echnology
CIFAR	**C**anadian **I**nstitute **F**or **A**dvanced **R**esearch
MLP	**M**ulti-**L**ayer **P**erceptrons
ReLU	**Re**ctified **L**inear **U**nit
LRN	**L**ocal **R**esponse **N**ormalization
XAI	e**X**plainable **A**rtificial **I**ntelligence
ANFIS	**A**daptive **N**euro-**F**uzzy **I**nference **S**ystem
AHC	**A**gglomerative **H**ierarchical **C**lustering
w.r.t.	**w**ith **r**espect **t**o
KLT	**K**arhunen-**L**oeve **T**ransform
LRP	**L**ayer-wise **R**elevance **P**ropagation
FHV	**F**uzzy **H**yper**V**olume
ANFIS	**A**daptive **N**euro-**F**uzzy **I**nference **S**ystem
AHC	**A**gglomerative **H**ierarchical **C**lustering

Chapter 1

Introduction

1.1. Deep Learning

Machine learning studies algorithms that allow computers to inductively create models to predict, classify, and make decisions [1]. For a number of problems, e.g. object recognition, classical programming approaches do not lead to effective models but machine learning often does. The recent performance of machine learning algorithms has attracted great interest form industry and academia. Applications of machine learning include image recognition tasks [2] [3] [4] [5], Natural Language Processing (NLP) tasks [6] [7] [8] [9], speech recognition [10] [11] [12], potential drug molecules [13], analyzing particle accelerator data [14] [15], content filtering on social media, malware detection, weather forecasting, etc.

Deep learning as a famous sub-field of machine learning have become an extremely active area of research, as well as a hot topic in popular science. The term Deep Learning was introduced to the machine learning community by Rina Dechter in 1986, [16] and to Artificial Neural Networks (ANNs) by Igor Aizenberg and colleagues in 2000, in the context of Boolean threshold neurons [17]. The well-known success of AlphaGo in finally unseating champion human players in the ancient Chinese game [18] has seemingly become emblematic of deep learning's promise in the public eye. The demand for deep learning in the corporate world is so strong that NVIDIA (the leading maker of GPU platforms for deep learning) saw its stock price double in 2017 partly on the strength of that demand [19], valuing the company at over USD $154.5 billion in Nov. 2018 [20]. Applications of deep neural networks ranges from computer vision and speech recognition to natural language processing [21] [22].

However, deep neural networks are essentailly black boxes. A large-scale deep neural network is a densely-interconnected collection of thousands or tens of thousands of simple computational nodes, with all of its "learned knowledge" expressed as a distributed pattern of potentially millions of connection weights [23]. This model is completely uninterpretable for humans; a critical problem as research shows that users will not be willing to entrust such a model with critical decisions [24]. When the model is used for medical diagnosis or terrorism detection, for example, predictions cannot be acted upon in blind faith, as the consequences may be catastrophic [25]. The demand for an explanation mechanism is a crucial element of human-factors engineering for intelligent systems, and this is currently missing from deep learning approaches.

1.2. Interpretability

The interpretability problem is not new to the neural networks field; explanation mechanisms were demanded for earlier generations of shallow neural networks as well. One strand of this research focused on transforming the distributed knowledge in a network into an interpretable form (often if-then rules); see [26] for a discussion. There is a small amount of research continuing this line of investigation in deep networks. [27] and [28] investigate rule extraction, while [29] and [30] develop visualization tools. Finally, [31] suggests that the saliency of a given pixel in an input image can be determined by estimating the change in classifications resulting from its absence.

Another approach sought to create networks that were interpretable by design; fuzzy neural networks (e.g. [32]) and neuro-fuzzy systems (e.g. [33]) were prominent amongst these efforts [34]. It is our intention to extend this second approach to deep learning (we will henceforth refer to such architectures as "deep fuzzy systems.") Our overarching research problem is thus to create deep fuzzy systems which remain as accurate as existing models, while being substantially more interpretable. In this thesis, our objective is to design and evaluate one such deep fuzzy system as a classifier.

There is little current research in hybrids of fuzzy logic and deep learning, in spite of the fact that fuzzy neural networks and neuro-fuzzy systems have been major research topics for over 25 years [34].

Aviles et al. [35] used a combination of an ANFIS ensemble and a recurrent Long-Short Term Memory network to estimate contact forces on tissues during remote surgical procedures. Chen et al. report on the design of a fuzzy restricted Boltzmann machine in [36]. Zheng et al. extend this architecture to Pythagorean fuzzy sets in [37], while Shukla et al. instead extended it to interval type-2 fuzzy sets in [38]. A hybrid of Pythagorean fuzzy sets and stacked denoising autoencoders is reported in [39]. Zhou et al. [40], and Rajurkar and Verma [41] design stacked TSK fuzzy systems. Tan et al. use fuzzy logic to compress a trained CNN in [42]. Fuzzy c-means clustering was used to segment images before passing them to a CNN in [43]. However, other than [44] (which uses restricted Boltzmann machines and fuzzy modeling for performing probability-based clustering), the combination of deep learning and fuzzy clustering is currently unexplored.

Our contribution in this thesis is to design and evaluate a deep fuzzy classifier that combines two famous fuzzy clustering algorithms of Fuzzy C-Means (FCM) [45] and Gustafson-Kessel (GK) clustering [46] and a Convolutional Neural Network (CNN) for image classification tasks. The CNN is employed as an automated feature extraction engine [21], and FCM/GK is then used to identify fuzzy clusters in the derived feature space thus created. Those clusters are then hardened, and Rocchio's algorithm [47] is used to classify new observations. (Note that we could easily modify this classifier for regression and function approximation by replacing Rocchio's algorithm with e.g. the fuzzy nearest-prototypes algorithm [48].) To evaluate the system, we have chosen three benchmark datasets: MNIST [49], Fashion MNIST [50], and CIFAR-10 [51]. All these three datasets are computer vision datasets. MNIST includes handwritten digits gathered by NIST (National Institute of Standards and Technology). Fashion MNIST, which is a dataset for clothes classification, serve as a direct drop-in replacement of the original MNIST dataset for benchmarking machine learning algorithms. The dataset is labeled with ten classes: t-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots. CIFAR-10 includes images of objects. The dataset includes ten classes: airplane, automobile, bird, cat deer, dog, frog, horse, ship, and truck.

Chapter 2

Review

2.1. Representation Learning

In a broad sense, we may categorize learning in two ways. Learning with a teacher (supervised learning) and learning without. In supervised learning (the focus of this thesis), the system uses "learns" to assign labels to new examples mimicking the pattern of label assignments in known examples. This learning process involves updating a proposed solution in order to optimize an objective function (e.g. the distance between the desired targets and actual outputs of the system) [23].

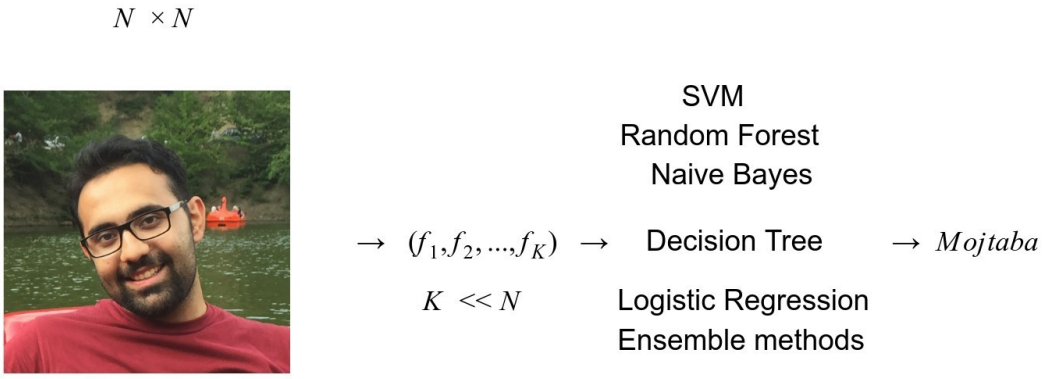


Figure 2.1: Conventional face recognition procedure.

Shallow machine learning systems were generally limited to directly learning a solution to the optimization problem; they were not able to determine what facets of the problem gave the most insight into a solution. This made manual feature extraction an essential part of a learning project. Figure 2.1

shows this conventional approach applied to face recognition. As shown this figure, experts needed to extract K features from an $N \times N$ input image, and then a classifier (e.g. SVM) performs the recognition task [52]. However, the conventional way had some defects. Different tasks require different experts. In addition, it's always time-consuming to find the most appropriate features. The experts also cannot assure us that the employed features are the best ones. Representation learning, on the other hand, provides us with methods of automatically discovering a better feature space for learning a given dataset [53].

Modern learning methods use representation-learning to extract new feature spaces, by applying nonlinear transforms to the raw data inputs. Given a set of training data that is not linearly separable, one can with high probability transform it into a set that is linearly separable by projecting it into a higher dimensional space via some non-linear transformation [54]. Indeed, these functions can make the system able to mimic the relation between input and output, i.e., automatic feature extraction [23]. Classification of extracted feature space from CNNs with (in essence) Multi-Layer Perceptrons (MLPs) is one well-known method of deep learning. We will discuss CNNs and MLPs in more detail later in this chapter.

2.2. Optimization via Gradient-Descent

Viewed as optimization problems, machine learning algorithms usually solve the optimization iteratively, rather than deriving a formula [55]. Optimization can be defined as minimizing or maximizing some function $f(w)$ by altering w. Such a function is called objective function. It's also called as cost function or loss function when we're minimizing it. (Note that maximization can be accomplished via a minimization algorithm by minimizing $-f(w)$.) Cauchy [56] invented the Gradient-Descent algorithm, which uses derivatives of the function in order to reach a local minimum. The gradient vector $\nabla f(x_0)$ is orthogonal (or perpendicular) to the curve $f(x) = y$ at the point (x_0). We interpret *y* as the utility of the solution (x_0) to the optimization problem, and thus $f(x)$ is a contour or *level curve* of constant utility for the optimization problem. Likewise, the gradient vector $\nabla f(x_0, y_0, z_0)$ is orthogonal to the level surface $f(x, y, z) = k$ at the point (x_0, y_0, z_0). The gradient vector shows the direction of greatest local increase of the optimization function at that specific point. Thus, each parameter of *f* could be updated in the direction of the negative gradient, and $f(w)$ thus moves closer to a local optimum:

$$w_{new} = w_{old} - \eta \nabla_w f(w) \quad (2\text{-}1)$$

where w is the system parameter. The optimization (or *cost*) function can be defined in various ways. One of the simplest is the sum of squared errors:

$$f(w) = \frac{1}{2n} \Sigma_x \left|\left| Y_{target}(\mathrm{x}) - Y(x, w) \right|\right|^2 \quad (2\text{-}2)$$

where x is the input sample, $Y\ (x, w)$ is model's prediction, and $Y_{target}(\mathrm{x})$ is the target label for input x.

2.2.1. Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) [57] is an extension of the Gradient Gescent algorithm. One problem in machine learning and especially deep learning is the number of required training samples for generalization. Deep learning methods require large training sets in order to generalize. However, this will make the model more computationally expensive. The cost function used by a machine learning algorithm often decomposes as a sum over training examples of some per-example loss function [55]. For example, the negative conditional log-likelihood of the training data can be written as:

$$J(w) = \frac{1}{m} \Sigma_{i=1}^{m} L\left(x^{(i)}, y^{(i)}, w\right) \quad (2\text{-}3)$$

where L is the per-example loss $L(x, y, w) = -logp(y|x; w)$. The required gradient of cost function can be calculated as:

$$\nabla_w J(w) = \frac{1}{m} \Sigma_{i=1}^{m} \nabla_w L\left(x^{(i)}, y^{(i)}, w\right) \quad (2\text{-}4)$$

Thus, as the training set size increases, the time required for the gradient step becomes prohibitively long. SGD instead chooses a *mini-batch* of examples $\mathrm{B} = \{x^{(1)}, \ldots, x^{(m')}\}$ to estimate the direction of the

gradient. The mini-batch size m' is a relatively small number of examples. The estimation of the gradient is then:

$$\nabla_w J(w) = \frac{1}{m} \Sigma_{i=1}^{m} \nabla_w L\left(x^{(i)}, y^{(i)}, w\right) \qquad 2\text{-}5$$

using only examples from the mini-batch B. The stochastic gradient descent algorithm then follows the negative estimated gradient for one update, then draws another m' examples from the dataset and repeats.

2.3. Artificial Neural Networks

2.3.1. Cortical neuron

The human brain includes about 10^{11} cortical neurons [58] and about 60 trillion connections [59]. These neurons connect to each other at specific parts of our brain and implement specific tasks. For example, the visual cortex is located at the back of our heads, while the auditory cortex is located at the center of brain [23]. By understanding these neurons and how they learn, we can develop bio-inspired algorithms [1].

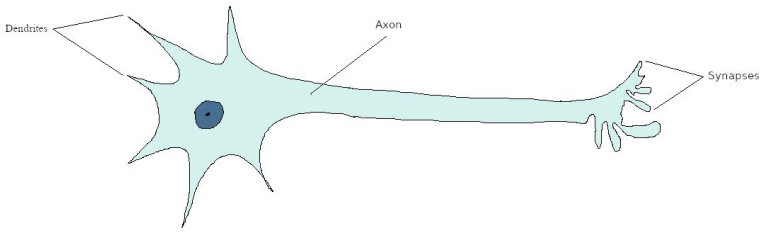


Figure 2.2: Cortical Neurons.

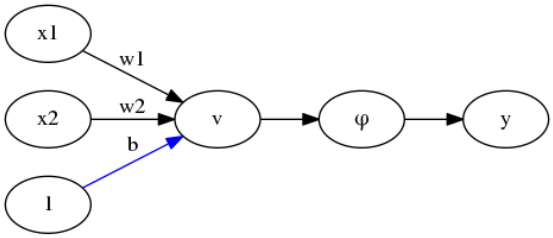


Figure 2.3: An Artificial Neuron.

Figure 2.2 illustrates the shape of a typical brain cell. It receives most of its inputs through dendritic spines leading to the cell body. When a neuron "fires" (generates an output pulse) is travels down the axon, and crosses to the dendrites of new neurons at synapses. The axon of a neuron is very long and is characterized by high electrical resistance and very large capacitance. Thus, the axon may be modeled as a resistance-capacitance (RC) transmission line, hence the common use of "cable equation" as the terminology for describing signal propagation along an axon. Analysis of this propagation mechanism reveals that when a voltage is applied at one end of the axon, it decays exponentially with distance, dropping to an insignificant level by the time it reaches the other end. Synapses, or nerve endings, are functional units that mediate the interactions between neurons. The most common kind of synapse is a chemical synapse, which operates as follows: A presynaptic process liberates a transmitter substance that diffuses across the synaptic junction between neurons and then acts on a postsynaptic process. A synapse converts a presynaptic electrical signal into a chemical signal and then back into a postsynaptic electrical signal. Thus, a neuron generates spikes when it receives enough charge from dendritic inputs and sends the spike via its axon. When the spike arrives in the axon, synapses send the information from one neuron to other neurons by converting the presynaptic electrical signal into a chemical signal and then back into a postsynaptic electrical signal in the recipient's dendritic tree [23].

2.3.2. Multi-Layer Perceptrons (MLPs)

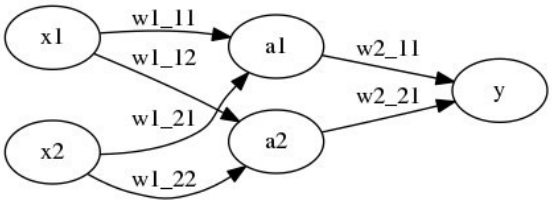


Figure 2.4: An artificial neural network.

Figure 2.3 shows one way of defining an artificial neuron which can act similar to the cortical neurons. As shown in this figure, each neuron receives some inputs and uses some weights in order to implement

impact level of each input (synapses). Then the overall value v passes a function φ. This function determines whether a spike should be generated or not. If yes, an output y will be created (performing the axon's job); hence, φ is often called the activation function [23]. These artificial neurons are called perceptrons and can be implemented mathematically as follows:

$$v = \Sigma_{i=1}^{n} x_i \times w_i \qquad 2\text{-}6$$

$$\varphi = \begin{cases} 1, & if\ v\ \geq \mathrm{b} \\ 0, & otherwise \end{cases} \qquad 2\text{-}7$$

The bias term b can be considered as a weight with its input = 1, and thus the φ function can be changed to the step function as follows [23]:

$$v = \Sigma_{i=0}^{n} x_i \times w_i \ ;\ w_0 = b\ ,\ x_0 = 1 \qquad 2\text{-}8$$

$$\varphi = \begin{cases} 1, & if\ v\ \geq 0 \\ 0, & otherwise \end{cases} \qquad 2\text{-}9$$

Figure 2.4 shows how to connect perceptrons together forming a Multi-Layer Perceptron (MLP). In a MLP all neurons of a layer are connected to all neurons of its subsequent layer. Thus, these systems are called fully connected networks. We have used $w[l]\_[i][j]$ to denote the weight connecting neuron i in layer l-1 to neuron j in layer l. The number of neurons in a layer is its "width," and the number of layers in the network is its "depth." Layers which do not directly connect to system inputs or outputs are called "hidden."

The problem with using step function as the activation function is that a small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip, say from 0 to 1. We can overcome this problem by introducing a sigmoidal activation function [60]. For example, Logistic neurons have the following activation function:

$$\varphi(v) = \frac{1}{1 + e^{-v}} \qquad 2\text{-}10$$

where v is defined in 2-8. There are other useful activation functions such as Rectified Linear Unit (ReLU), tanh, sign, etc. These activation functions can be defined as follows:

$$ReLU(v) = max(v, 0) \quad \text{2-11}$$

$$tanh(v) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad \text{2-12}$$

$$sign = \begin{cases} 1, & if\ v \geq 0 \\ -1, & otherwise \end{cases} \quad \text{2-13}$$

ReLU is the most commonly used activation function in modern deep networks, as it imposes a smaller computational burden. It also creates output values of zero, which is highly important in reducing overfitting [21]. We will describe overfitting in the **METHODOLOGY** chapter.

2.3.3. Back-Propagation

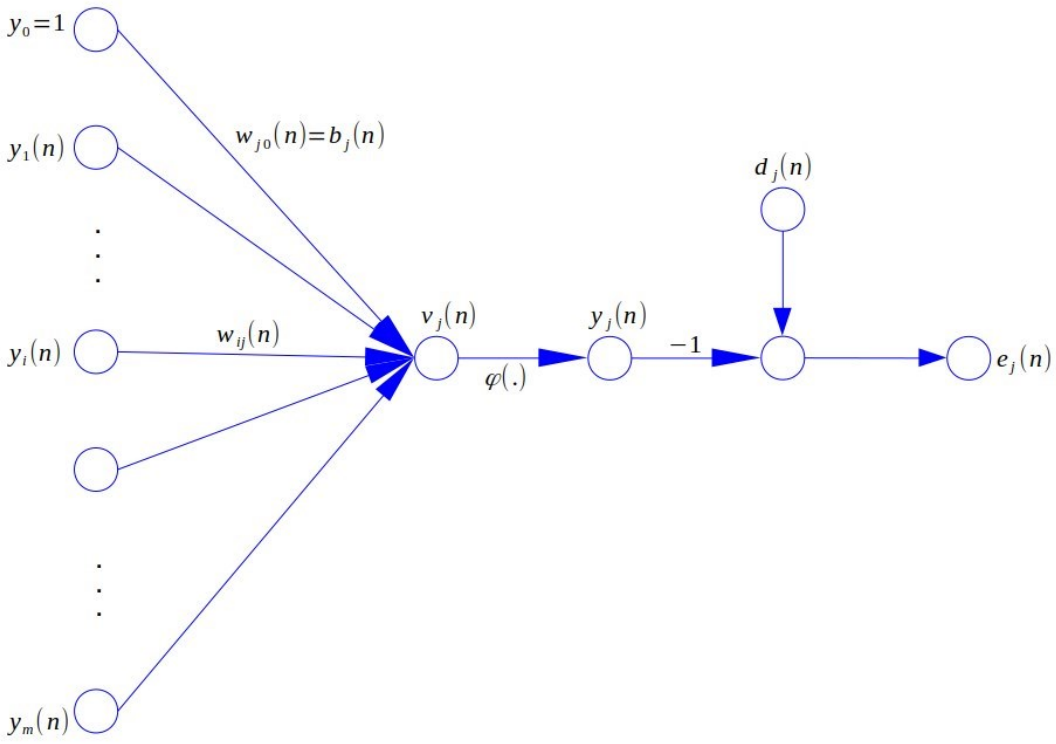


Figure 2.5: Signal-flow graph highlighting the details of output neuron j [23].

In this section Back-Propagation [23] method which is one kind of implementing Gradient Descent algorithm will be described. Figure 2.5 shows the Signal-flow graph of output neuron j. In order to tune the parameters with gradient descent we first we first define the network error as the squared prediction error of Eq. (2-2):

$$e_j(n) = d_j(n) - y_j(n) \quad \text{2-14}$$

$$\epsilon_n = \frac{1}{2}e_j^2(n) = \frac{1}{2}\Big(d_j(n) - y_j(n)\Big)^2 \quad \text{2-15}$$

Thus, the total error energy will be:

$$\epsilon(n) = \Sigma_{j=1}^{m}\epsilon_j(n) \quad \text{2-16}$$

In order to calculate the gradient for $w_{ji}(n)$ we use the chain rule:

$$\frac{\partial\epsilon(n)}{\partial w_{ji}(n)} = \frac{\partial\epsilon(n)}{\partial e_j(n)}.\frac{\partial e_j(n)}{\partial y_j(n)}.\frac{\partial y_j(n)}{\partial v_j(n)}.\frac{\partial v_j}{\partial w_{ji}(n)} \quad \text{2-17}$$

So:

$$\frac{\partial\epsilon(n)}{\partial\epsilon_j(n)} = e_j(n) \quad \text{2-18}$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)}\Big(d_j(n) - y_j(n)\Big) = -1 \quad \text{2-19}$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial\varphi\left(v_j(n)\right)}{\partial v_j(n)} = \varphi'\left(v_j(n)\right) \quad \text{2-20}$$

Note that $\varphi'\left(v_j(n)\right)$ depends on the activation function.

$$\frac{\partial v_j}{\partial w_{ji}(n)} = \frac{\partial}{\partial w_{ji}(n)}\Sigma_{(i=1)}^{m}w_{ji}(n)y_i(n) = y_i(n) \quad \text{2-21}$$

Thus:

$$\frac{\partial\epsilon(n)}{\partial w_{ji}(n)} = e_j(n).\varphi'\left(v_j(n)\right).y_i(n).(-1) \quad \text{2-22}$$

So:

$$\Delta w_{ji}(n) = -\eta.\frac{\partial\epsilon(n)}{\partial w_{ji}(n)} = \eta.e_j(n).\varphi'\left(v_j(n)\right).y_i(n) \quad \text{2-23}$$

We will define the error gradient at neuron j by:

$$\delta_j(n) = \frac{\partial \epsilon(n)}{\partial v_j(n)} = (-1).e_j(n).\varphi'\left(v_j(n)\right) \qquad 2\text{-}24$$

$$\Delta w_{ji}(n) = -\eta.\delta_j(n).y_i(n) \qquad 2\text{-}25$$

So, for an output neuron:

$$\delta_j(n) = \left[d_j(n) - y_j(n)\right].\varphi'\left(v_j(n)\right) \qquad 2\text{-}26$$

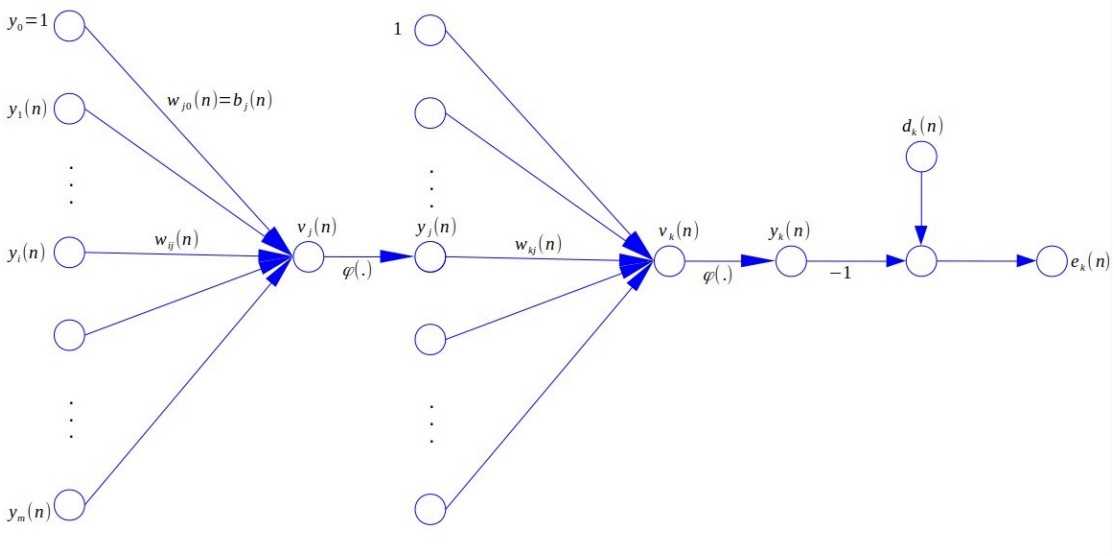


Figure 2.6: Signal-flow graph highlighting the details of output neuron k connected to hidden neuron j [23].

Figure 2.6 shows the signal-flow graph when there are hidden neurons. For a hidden neuron, we can follow the following steps:

$$\delta_j(n) = \frac{\partial \epsilon(n)}{\partial v_j(n)} = \frac{\partial \epsilon(n)}{\partial y_j(n)}.\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial \epsilon(n)}{\partial y_j(n)}.\varphi'\left(v_j(n)\right) \qquad 2\text{-}27$$

$$\frac{\partial \epsilon(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)}\left(\frac{1}{2}\Sigma_k\ e_k^2(n)\right) = \frac{\partial \epsilon(n)}{\partial e_k(n)}.\frac{\partial e_k(n)}{\partial y_j(n)} \quad 2\text{-}28$$

$$= \Sigma_k\ \left[e_k(n).\frac{\partial e_k(n)}{\partial y_j(n)}\right] = \Sigma_k\ \left[e_k(n).\frac{\partial e_k(n)}{\partial v_k(n)}.\frac{\partial v_k(n)}{\partial y_j(n)}\right] \quad 2\text{-}29$$

$$= \Sigma_k\ \left[e_k(n).(-1).\varphi'\big(v_k(n)\big).\frac{\partial v_k(n)}{\partial y_j(n)}\right] \quad 2\text{-}30$$

These equations will lead to:

$$\delta_j(n) = \Sigma_k\ e_k(n).(-1).\varphi'.w_{kj}(n) = \Sigma_k\ \delta_k(n)w_{kj}(n) \quad 2\text{-}31$$

$$\delta_j(n) = \varphi'\left(v_j(n)\right).\Sigma_k\ \delta_k w_{kj} \quad 2\text{-}32$$

Thus:

$$\Delta w_{ji}(n) = \eta.\delta_i(n).y_i(n) \quad 2\text{-}33$$

φ' depends on the activation function. For sigmoid activation function φ' will be calculated as follows:

$$\varphi'\left(v_j(n)\right) = \frac{exp\left(-v_j(n)\right)}{\left(1+exp\left(-v_j(n)\right)\right)^2} = \quad 2\text{-}34$$

$$\varphi\left(v_j(n)\right).\left[1-\varphi\left(v_j(n)\right)\right] = y_j(n)\left[1-y_j(n)\right]$$

2.3.4. Vanishing Gradients

According to 2-32, as the network becomes deeper there will be more terms of ϕ' in the error calculations. If we suppose that the activation function is sigmoid, φ' will follow Figure 2.7:

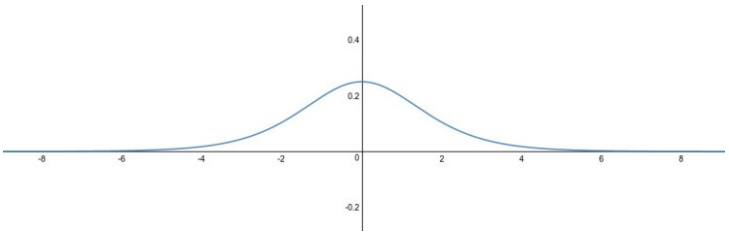


Figure 2.7: φ' plot.

As shown in Figure 2.7, the maximum of φ is 0.25. Now suppose the network has 4 hidden layers like Figure 2.8.

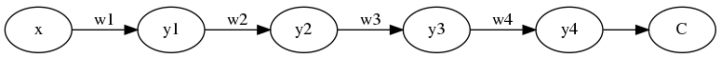


Figure 2.8: weight of a deep network.

In order to update $w1$, the error calculation term will include $\varphi'(y4) \cdot \varphi'(y3) \cdot \varphi'(y2) \cdot \varphi'(y1)$. Supposing values =0.25, the error term will be multiplied by 0.00390625 at most, driving the gradients to zero and thus preventing the weights of early layers from being updated [60].

2.4. Convolutional Neural Networks (CNNs)

In previous sections, we described MLPs and how to train them. We showed how fully-connected networks can find required features by utilization of non-linear functions. These neural networks have shown their good performance in lots of classification and prediction problems. However, these kinds of neural networks have some weaknesses. First is the number of weights. Fully Connected (F.C.) neural networks uses lots of parameters in order to connect a layer to its next layer. For example, consider the MNIST dataset which includes 28×28 = 784 input pixels. Assuming a hidden layer of 100 neurons, the number of weights between the input layer and first hidden layer will be 784 *100 = 78,400 weights. This continues when we add more layers. Having such a huge number of parameters will make training process slower due to the number of computations. Furthermore, such a large network requires more

training samples than a simple network in order not to overfit. Another important defect is that multi-layer perceptrons have no invariance with respect to local distortions. Thus, any permutation to the data will not affect network's performance. However, we know that there is important similarity between pixels which are near each other. For example, in recognition of objects such as a table in a picture, the pixels of a table are more similar to each other than non-table pixels. They have almost same color, and they're in a neighborhood. On the other hand, when there is a huge difference between pixels' intensities in a neighborhood, it means that those pixels can have important information like edges [61].

LeCun et al. [61] developed Convolutional Neural Networks (CNNs) for image recognition. The key insight behind the CNN is to take advantage of the N-dimensional (N=2, for 2D images) local structure surrounding every pixel. Quite obviously, the value (i.e. RGB, or whatever other color space is used) of a given pixel is only meaningful to the human eye in relation to the values of the surrounding pixels. There is in fact a body of literature indicating that human vision is tuned to recognize scenes having a relatively narrow band of spatial statistics (i.e. certain ranges of fractal dimensions); for a discussion see e.g. [62]. The CNN takes advantage of this fact by defining "convolutional" layers, in which groups of neurons perform distinct convolutions over their input (either an image at the first layer, or the outputs of another layer as signals progress deeper into the network) [61]. A convolution operation can be carried out by defining a collection of neurons whose spatial extents overlap and cover the entirety of the image, while replicating the operation in question. These groups are referred to as planes, and their outputs feature maps. Replication of the same operation is achieved simply by constraining the weight vectors for every neuron in a plane to be duplicates of each other. The resulting feature map is then the outputs of the given convolution, applied to the input image (e.g. edge detection). We expect there will be multiple planes for an image, yielding multiple feature maps that will be passed to the next layer. Convolutional layers are commonly followed by sub-sampling layers. This will reduce the resolution of the feature map and thesensitivity of the output to shifts and distortions [61]. After passing all convolutional operations, sub-sampling layers, and non-linearities, feature map values of the last convolutional layer are ready to be classified by e.g. an MLP.

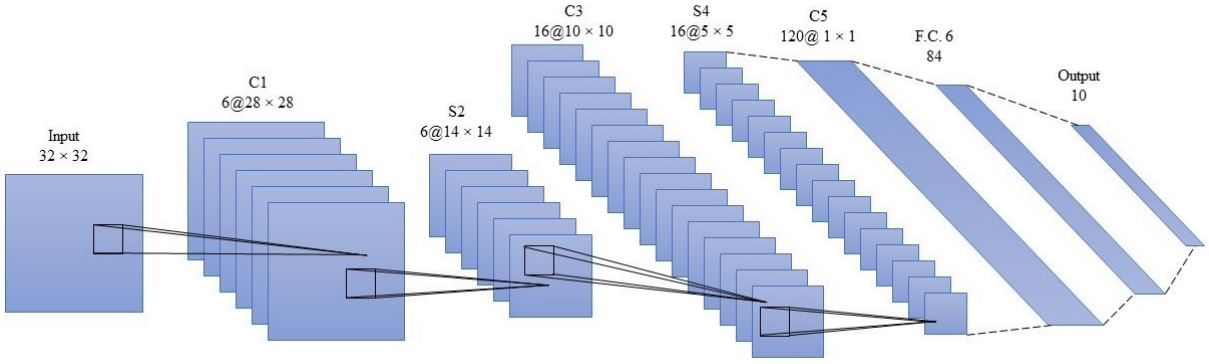


Figure 2.9: LeNet Architecture.

Figure 2.9 shows the architecture of LeNet [61]. Layer C1 in Figure 2.9 is the first convolutional layer, whose inputs are each individual pixel values from the input image. The convolutional layer computes the weighted sum of pixels in a window centering by a given pixel followed by a nonlinear function. The general form of the convolution operation is:

$$v_{ij}^{l} = \Sigma_{a=0}^{m-1}\Sigma_{b=0}^{m-1} w_{ab} X_{(i+a)(j+b)}^{l-1} \qquad 2\text{-}35$$

where the convolution window is of size $m \times m$, X_{ij} is the ij_{th} pixel in the current sub-image, and w_{ab} is the weight of the connection between that pixel and the current neuron. The nonlinear function is the Logistic function of Eq. (2-10)

Each convolutional layer can be followed by a sub-sampling ("pooling") layer, which maps small neighborhoods in each feature map output to single pixels. Two common subsampling methods are max-pooling and mean-pooling. Both these pooling layers consider a kernel size $(a \times b)$, and get their inputs through these kernels. In mean pooling (used in [61]) each unit computes the mean of its windowed inputs, multiplied by a trainable coefficient, and passing the result through the activation function. Max-pooling units compute the maximum of their inputs rather than the mean. 2-36 shows the max-pooling operation with kernel size of $a \times b$:

$$y_{ij}^{l} = \max_{a} \max_{b} \left(X_{(i+a)(j+b)}^{l-1} \right),\ a, b = (1, \dots, k) \qquad 2\text{-}36$$

2-37 shows the average pooling operation with kernel size of $c \times d$:

$$y_{ij}^{l} = average\left(X_{(i+c)(j+d)}^{l-1}\right).\omega_{ij} + b_{ij},\ c,d = (1,\dots,k) \qquad 2\text{-}37$$

where ω_{ij} is the trainable weight coefficient, and b_{ij} is the bias term of neuron.

LeNet (Figure 2.9) has three convolutional layers. LeCun et. al [61] add a zero padding to the inputs of MNIST and made the image 32 × 32. The first convolutional layer (C1) creates 6 feature maps of 28×28. Thus, the filters are of size 5×5 (as 32−5+1 = 28). This is called *Valid* padding. There's another padding named *Same*. If LeNet was using *Same* padding the output of C1 would be of size 32 × 32. Each of these paddings have their own advantages. *Same* padding let's network to get deeper and deeper while *Valid* padding reduces data dimension and prevents model to over-fit in comparison with *Same*. LeCun et. al, used same padding for only the first Layer (as we can see output of C1 is of size 28×28. Note that the inputs of MNIST are of size 28×28). The second layer is S2. This is a mean pooling layer which makes 6 feature maps of size 14 × 14. Layer C3 makes 16 feature maps. In this layer also, the filter size is 5 × 5 which results in having 16 feature maps of size 10 × 10. Note that this layer is using Valid padding. Then S4 performs another mean pooling and makes 16 feature maps of size 5 × 5. Layer C5 makes 120 feature maps by using filters of size 5 × 5. Here the architecture has 120 feature maps of size 1 × 1 which are similar to F.C. architectures. Thus, the convolutional layers can connect directly to a MLP. When the last convolutional layer is not of size 1×1, we will flatten and serialize the feature map values before passing them to the F.C. layers.

Variants on the CNN architecture often include layers of additional neuron types. Local Response Normalization (LRN) layers implement the concept of lateral inhibition from human vision [2]. Lateral inhibition is the capacity of an excited neuron to reduce the activity of its neighbors. Lateral inhibition disables the spreading of action potentials from excited neurons to neighboring neurons in the lateral direction. This creates a contrast in stimulation that allows increased sensory perception. It is also referred to as lateral antagonism and occurs primarily in visual processes, but also in tactile, auditory, and even olfactory processing [63]. In an artificial neural network, an LRN layer will make a neuron become more sensitive as compared to its neighbors. One form of lateral inhibition can be implemented as [2]:

$$b_{x,y}^{i} = \frac{a_{x,y}^{i}}{\left(k + \alpha \Sigma_{j=max\left(0,i-\frac{n}{2}\right)}^{min\left(N-1,i+\frac{n}{2}\right)} \left(a_{x,y}^{i}\right)^{2}\right)^{\beta}} \quad \text{2-38}$$

where $a_{x,y}^{i}$ is the activity of a neuron computed by applying kernel *i* at position (*x*, *y*) and then applying the ReLU non-linearity. Here *N* is the total number of kernels in the layer, and *n* is number of adjacent kernel maps at the same spatial position, while *k* (bias), α (scaling factor), β (exponent) are hyper-parameters.

2.5. Dimension Reduction

Many machine learning problems suffer from the curse of dimensionality; the number of possible solution configurations increases exponentially as the number of variables increases [55]. Dimension reduction techniques attempt to retain information from the dataset that is heuristically believed to be the most discriminative. They might be used for dataset compression, speeding up a learning algorithm machine learning algorithm, data visualization, etc. Dimension reduction also often helps remove noise from the data. Thus, it can help speed up learning by reducing the number of features, and promote generalization by preventing overfitting to noise.

2.5.1. Principle Component Analysis (PCA)

Principle Component Analysis (PCA) [64] is a well-known dimensionality reduction technique. It is also known under different names such as the discrete type of Karhunen-Loeve Transform (KLT) [65]. KLT uses Algorithm 1 for data dimension reduction.

Algorithm 1. KL Transform

`Start:`

`Set` $\underline{x} = [x_1, x_2, \dots, x_n]^T$

`Calculate` $\underline{m}_x = E\underline{x} = [m_1, m_2, \dots, m_n]^T = [E\{x_1\}, E\{x_2\}, \dots, E\{x_n\}]^T$

Calculate covariance matrix C $= E\{(\underline{x} - \underline{m_x})(\underline{x} - \underline{m_x})^T\}$
For large enough M: $\underline{m_x} = \frac{1}{M}\sum_{k=1}^{M} \underline{x_k}$
Form A from eigenvectors of matrix C.
Calculate $\underline{y} = A(\underline{x} - \underline{m_x})$
end

PCA projects the input data *x* into a lower dimensional data spanned by the space of eigenvectors of the covariance matrix for a dataset. To obtain k principal components $D \in \mathrm{R}^{k \times d}$ where d is number of features of the data, we obtain a singular value decomposition $X = U\Sigma V^T \in \mathrm{R}^{n \times d}$ giving $D = V_k^T \in \mathrm{R}^{k \times d}, H = U_k\Sigma_k \in \mathrm{R}^{n \times k}$, which consists of the top largest k singular values, and $U_k \in \mathrm{R}^{n \times k}$ *and* $V_k \in \mathrm{R}^{k \times d}$ are the corresponding singular vectors. The new representation for *X* is the *H*. This dimensionality reduction can be formulated as:

$$\min_{D \in \mathrm{R}^{k \times d},\ H \in \mathrm{R}^{n \times k}} ||X - HD||_2^2 \qquad 2\text{-}39$$

2.5.2. Uniform Manifold Approximation and Projection (UMAP)

UMAP (Uniform Manifold Approximation and Projection) [66] is a recent, well-regarded dimension reduction algorithm. UMAP uses local manifold approximations and patches together their local fuzzy simplicial set representations. UMAP is based on a Riemannian geometry and algebraic topology. It takes a topological representation of the high dimensional data. Then, in a similar process, the equivalent topological representation of a data can be developed for a low dimensional representation of data. UMAP minimizes the cross-entropy between the two topological representations. As far as we know, UMAP is the most recent and fastest dimension reduction technique [66].

We present Algorithm 2 and Algorithm 3, which summarize the computational pipeline for UMAP. In practice the fuzzy topological representation is not fully computed, but rather the objective is optimized via negative sampling on the edges of it, and then the corresponding memberships within it are calculated.

Algorithm 2: FuzzyTop - Fuzzy topological representation of a dataset

Start:
Data: Dataset $\mathcal{D} = \{x_i\}_{i=1}^{N} \subset \mathrm{R}^n$, number of neighbors $\tilde{n}$
Result: Fuzzy topological representation of $\mathcal{D}$ given by $K_{\mathcal{D}}$
for $i = 1, 2, \ldots, N$ **do**
compute $(\mathcal{D}, d_i) \in$ **FinEPMet**
$k_i = FinSing(\mathcal{D}, d_i) \in$ **sFuzz**
$K_{\mathcal{D}} = \perp_{i=1}^{N} K_i$
End

Algorithm 3. UMAP - Uniform Manifold Approximation and Projection

Start:
Data: Dataset $\mathcal{D} = \{x_i\}_{i=1}^{N} \subset \mathrm{R}^n$, number of neighbors $\tilde{n}$, embedding dimension d, maximum iterations i_{max}, learning rate α
Result: Low dimensional embedding of D given by $\mathcal{Z} = \{z_i\}_{i=1}^{N} \subset \mathrm{R}^d$
$K_{\mathcal{D}}$ = FuzzyTop$(\mathcal{D}, \tilde{n})$
Initialize $\mathcal{Z} \subset \mathrm{R}^d$
$K_{\mathcal{Z}_0}$ = FuzzyTop$(\mathcal{Z}_0)$
for $\tau = 1, \ldots, i_{max}$ **do**
$l = C\left(K_{\mathcal{D}}, K_{\mathcal{Z}_{\tau-1}}\right)$
$\mathcal{Z} = \mathcal{Z}_{\tau-1} - \alpha \nabla_{\mathcal{Z}_{\tau-1}} l$
$K_{\mathcal{Z}_\tau}$ = FuzzyTop$(\mathcal{Z}_\tau)$
end

In Algorithm 2 **FinEPMet**, **sFuzz**, and FinSing function relate to the Fuzzy topological representation defined at [66].

2.6. Fuzzy logic

Lotfi Zadeh introduced the concepts of Fuzzy sets [67] with the primary goal of developing a framework for representation of uncertain knowledge in 1965. Fuzzy logic [67] makes it possible to handle classes and structures with unsharp boundaries. In the second half of the 20th century, many scientific concepts, methods, and theories were "fuzzified". Fuzzification is a transformation that can be reconstructed and reflected upon in a scientific manner by appropriately expanding the framework of the structuralist view of scientific theories in the philosophy of science. The resulting fuzzy sets can then serve as a new modeling tool in scientific theory. Fuzzy sets can represent human perceptions that cannot be represented with the sharp boundaries of classical logic. It is able to translate knowledge-based

approaches to formal models that are easy to implement. In addition, it is able to work with uncertainties within the data by allowing objects to be mathematically modeled by imprecise propositions and analyzing them. It allows us to learn a knowledge-based representation of information inherent in the data.

Fuzzy sets deal with imprecise concepts in an exact way. Examples of imprecise expressions could be: “a polite person” or “a difficult exam”. Here it is the adjective that relates to something considered imprecise, but there are inexact expressions pertaining to other word classes as well such as “at this price, the house is a real bargain” [68]. The fuzzy perspective, which differs by introducing an infinite number of truth-values along a spectrum between perfect truth and perfect falsity, is one of the simplest ways of illustrating the imprecise expressions [69].

2.6.1. Fuzzy clustering algorithms

Fuzzy clustering analysis deals with discovery of structures and groupings within the data, which we can interpret by using extracted membership functions. Membership functions then show the relation between each sample and clusters. On the other hand, because there is always noise within the data and the noise cannot be completely removed, fuzzy analysis can provide us with better interpretability as it includes uncertainty in the model [70].

Clustering algorithms can be defined as algorithms that identify groups of objects that are more similar to each other. Literally, thousands of clustering algorithms has been developed [71]. Fuzzy clustering methods do not assign a datum to a specific cluster. There could be outliers, noise and boundary cases. These cases may belong to multiple groups and as a results data points do not neatly sort into highly compact and separate groupings. As with fuzzy sets, fuzzy clustering resolves this problem by defining membership functions for each data point, and the membership value can be a non-zero value for an arbitrary number of clusters [72].

2.6.1.1. Fuzzy C-Means (FCM)

The Fuzzy C-Means (FCM) algorithm [45] is one the most famous approaches in fuzzy clustering. Bezdek developed a fuzzification of the well-known K-Means algorithm in his Ph.D. thesis. FCM optimizes the objective function:

$$cost = \Sigma_{i=1}^{n}\Sigma_{j=1}^{c} u_{ij}^{m}\left\|x_i - c_j\right\|^2 \qquad 2\text{-}40$$

Where c_j is the center of cluster j, u_{ij} is the membership value of data point i belonging to cluster j, the norm is the Euclidean distance, and m is the fuzzifier constant which determines the level of cluster fuzziness [73]. A large m may result in small membership values, while small m can result in reaching a crisp partitioning. The default value for m is commonly set to 2. In Optimizing such an objective function can be done by iteratively changing centers and membership functions as follows with Alternating Optimization [74]:

$$v_i = \frac{\Sigma_{j=1}^{n} u_{i,j}^{m} x_j}{\Sigma_{j=1}^{n} u_{i,j}^{m}} \qquad 2\text{-}41$$

and

$$u_{ij} = \frac{1}{\Sigma_{k=1}^{c}\left(\frac{\left\|x_i - v_j\right\|}{\|x_i - v_k\|}\right)^{\frac{2}{m-1}}} \qquad 2\text{-}42$$

Algorithm 4: FCM algorithm

```
Start:
      Initialize membership functions
      Set ϵ (termination criterion), and τ (iteration threshold)
      while iteration < τ do
            Update centers using  2-41
            Update Membership functions using  2-42
            Set e = ‖u_new − u_old‖_2^2
            if e < ϵ then
```

```
                    break
end
```

2.6.1.2. Gustafson and Kessel (GK) method

FCM clusters are hyperspherical. By replacing the Euclidean distance by another metric, FCM can be extended to another clustering algorithm which is able to learn ellipsoidal clusters. Gustafson and Kessel (GK) [46] developed such an algorithm using the Mahalanobis distance. In the GK algorithm, each cluster is characterized by a covariance matrix A, which is necessarily symmetric and positive definite, along with the cluster centers. This matrix induces for each cluster a norm of its own $\|x\|_A := \sqrt{x^T A x}$. In order to avoid a minimization of the objective function by matrices with almost zero entries, a constant volume of clusters is required; det(A) = 1. GK algorithm will define a constant value ρ for each matrix A and uses $det(A) = \rho$. Thus, the objective function can be represented as follows:

$$cost = \Sigma_{j=1}^{n}\Sigma_{i=1}^{c} u_{ij}^{m}\left\|x_j - v_i\right\|_{A_i}^{2} - \Sigma_{i=1}^{c}\lambda_i(det(A_i) - 1) \qquad 2\text{-}43$$

where v_i is the center of the i_{th} cluster and can be updated by the following equation:

$$v_i = \frac{\Sigma_{j=1}^{n} u_{i,j}^{m} x_j}{\Sigma_{j=1}^{n} u_{i,j}^{m}} \qquad 2\text{-}44$$

The covariance matrices A can be also updated using the following equations:

$$A_i = \sqrt[p]{det(S_i)}S_i^{-1} \qquad 2\text{-}45$$

$$S_i = \Sigma_{j=1}^{n} u_{i,j}^{m}(x_j - v_i)(x_j - v_i)^T \qquad 2\text{-}46$$

where p is number of features of the dataset. Both GK and FCM algorithms continue iterations of learning until reaching a stopping conditions such as $\|u(k+1) - u(k)\| < \delta$ or $k < n$, where n is iteration threshold, and δ is termination criterion, and k is the current iteration [70].

2.6.1.3. Gath-Geva (GG) method

The algorithm by Gath and Geva (GG) [75] is an extension of GK algorithm. This algorithm takes size and density of clusters into account. Hence, it has better behaviors for irregular features [70]. The probabilistic interpretation of GG is shown by:

$$P(X|\eta) = \sum_{i=1}^{c} P(X, \eta_i) = \sum_{i=1}^{c} P(\eta_i) P(X, \eta_i) \quad \text{2-47}$$

Gath and Geva [75] assumed that the normal distribution N_i with expected value v_i and covariance matrix $A_i$is chosen for generating a datum with prior probability p_i

$$P(x_j|\eta_i) = \frac{p_i}{(2\pi)^{\frac{p}{2}}\sqrt{det(A_i)}} exp\left(-\frac{1}{2}(x_j - v_i)^T A_i^{-1}(x_j - v_i)\right) \quad \text{2-48}$$

where $= \{x_1, x_2, \dots, x_n\}$, $x_j \in R^p, j = 1,2, \dots, n$ is the data matrix. The covariance matrix of cluster i is $A_i \in R^{p \times p}$ where p is the dimension of data, c is the number of clusters, $d^2(x_j, v_i)$ for GG algorithm is chosen to be indirectly proportional to 2-48 which is the posterior probability likelihood function. A small distance means a high probability and a large distance means a low probability for membership. GG algorithm is based on minimization of the sum of weighted square distances between the data and the cluster centers of the objective function:

$$J_{GG}^m(U, A, \Sigma, X) = \sum_{i=1}^{c}\sum_{j=1}^{n} \mu_{ij}^m \, d^2(x_j, v_i) \quad \text{2-49}$$

Where the distance will be defined as follows:

$$d^2(x_j, v_i) = \frac{(2\pi)^{\frac{p}{2}}\sqrt{|A_i|_i}}{p_i} \exp[\frac{1}{2}(x_j - v_i)^T A_i^{-1}(x_j - v_i)] \quad \text{2-50}$$

This algorithm can be optimized by alternating optimization [74].

$$v_i = \frac{\sum_{j=1}^{n} u_{i,j}^m \; x_j}{\sum_{j=1}^{n} u_{i,j}^m} \quad \text{2-51}$$

$$A_i = \frac{\sum_{j=1}^{n} u_{i,j}^m (x_j - v_i)(x_j - v_i)^T}{\sum_{j=1}^{n} u_{i,j}^m} \quad \text{2-52}$$

$$p_i = \frac{\sum_{j=1}^{n} u_{i,j}^m}{\sum_{j=1}^{n}\sum_{l=1}^{c} u_{l,j}^m} \quad \text{2-53}$$

$$u_{ij}^{k} = \left[\sum_{l=1}^{c}\left[\frac{d^2(x_j, v_i)}{d^2(x_j, v_l)}\right]^{\frac{1}{m-1}}\right]^{-1} \qquad 2\text{-}54$$

Following algorithm shows the GG algorithm.

Algorithm 5: GG algorithm

```
Start:
      Initialize the parameters and memberships
      Set τ as iteration threshold and ϵ total change threshold
      while iteration < τ do
            Update parameters of  2-51, 2-52, 2-53, 2-54
            Set e = (1/c) Σ_{i=1}^{c} ||v_new − v_old||^2
            if e < ϵ then
                  break
end
```

2.6.1.4. Agglomerative Hierarchical Clustering

Agglomerative Hierarchical Clustering (AHC) methods have been employed in a variety of fields due to their usefulness. As usual, $X = \{x_1, \ldots, x_N\}$ is the set of objects. A measure of dissimilarity $D(x, x')$ is defined on an arbitrary pair $x, x' \in X$. And $G_1, \ldots, G_K$ are clusters that form the partition of X:

$$\bigcup_{i=1}^{K} G_i = X, \quad G_i \cap G_j = \emptyset, \qquad (\text{i} \neq \text{j}) \qquad 2\text{-}55$$

A set of clusters is defined by $\zeta = \{G_1, \ldots, G_k\}$. K is a given constant and varies as the algorithm proceeds. The algorithm starts with $K = N$, and in each iteration two clusters of the minimum dissimilarity are merged, and the number of clusters is reduced by 1: $K = K - 1$, and finally all clusters are merged into the trivial cluster of X: $K = 1$. Algorithm 6 presents the Agglomerative Hierarchical Clustering algorithm [76].

Algorithm 6: Agglomerative Hierarchical Clustering (AHC)

Start:

AHC1. Assume that initial clusters are given by
$\zeta = \{G_1, \dots, G_k\}, G_j = \{x_j\} \subset X$
Set K=N (K is the number of clusters).
Calculate $D(G, G')$ for all pairs $G, G' \in \zeta$ by $D(G, G') = D(x, x')$

AHC2. Search the pair of minimum dissimilarity:

$$(G_p, G_q) = arg \min_{G,G' \in \zeta} D(G, G')$$

let:

$$m_K = D(G_p, G_q) = \min_{G,G' \in \zeta} D(G, G')$$

Merge: $G_r = G_p \cap G_q$
Add G_r to ζ and delete G_p, G_q from ζ.
K = K-1
if K = 1 **then**
Go to **end.**

AHC3. Update dissimilarity $D(G_r, G'')$ for all $G'' \in \zeta$
Go to **AHC2.**

end

2.6.2. Clustering Validity

One of the major questions in clustering is how to evaluate the clustering result of a particular algorithm. The problem is called "cluster validity". Cluster validity is conducted to achieve three objectives [77]:

1. To compare the output of alternative clustering algorithms for a particular data set
2. To determine the best number of clusters for a particular data set (for example, the choice of parameter c for the FCM).
3. To determine if a given data set contains any structure (i.e., whether there exists a natural grouping of the data set).

Validity measures of a constrained fuzzy partition fall into three categories:

1. Membership-based validity measures which calculate certain properties of the membership functions in a constrained fuzzy partition.

2. Geometry-based validity measures which consider geometrical properties of a cluster as well as geometrical relationships between clusters.
3. Performance-based validity measures which evaluate a fuzzy partition based on its performance for a predefined goal.

The Partition Coefficient [77] measures the degree of fuzziness of clusters. The formula for this validity measure, denoted v_{pc}, is:

$$v_{pc} = \frac{1}{n}\sum_{i}\sum_{j} \mu_{Ci}(xj)\,,\ i = 1, \dots, c\ \ j = 1, \dots, n. \qquad 2\text{-}56$$

v_{pe} is another membership-based validity measure is the Partition entropy:

$$v_{pe} = \frac{-1}{n}\sum_{i}\sum_{j} \left[\mu_{Ci}(x_i) log_a\left(\mu_{Ci}(x_j)\right)\right],\ i = 1, \dots, c\ \ j = 1, \dots, n. \qquad 2\text{-}57$$

where $a \in (1, \infty)$ is the logarithmic base. The entropy measure increases as the fuzziness of the partition decreases [78]. The two membership-based validity measures are related in the following ways:

- $v_{pc} = 1 \Leftrightarrow v_{pe} = 0 \Leftrightarrow$ the partition is hard.
- $v_{pc} = 1/c \Leftrightarrow v_{pe} = \log_a(c) \Leftrightarrow \mu_{C_i}(x_j) = \frac{1}{c}$ for all i, j.

Xie and Beni introduced a validity measure that considers both the compactness of clusters as well as the separation between clusters. The basic idea of this validity measure is that the more compact the clusters are and the further the separation between clusters, the more desirable the partition. To achieve this measure, the Xie-Beni validity index [79] (denoted as v_xB) is defined formally as follows:

$$v_xB = \left[\sum_{i} \sigma_i /n\right]\left[1/d_{min}^2\right] \qquad 2\text{-}58$$

where σ_i is the variation of cluster C_i defined as follows:

$$\sigma_i = \sum_{j}\left[\mu_{Ci}(x_j)\right]\|x_j - v_i\|^2 \qquad 2\text{-}59$$

where n is the cardinality of the data set and d_{min} is the shortest distance between cluster centers defined as:

$$d_{min} = min\|v_j - v_i\|, i \neq j \qquad 2\text{-}60$$

The first term in 2-58 is a measure of non-compactness, and the second term is a measure of non-separation. Hence, the product of the two terms reflects the degree to which the clusters in the partition are not compact and not well separated. Thus, the lower the cluster index, the better the fuzzy partition is [78].

Other cluster validity measurement methods include: proportion exponent [80], uniform data functional [81], Davies–Bouldin (DB) index [82], Separation index (Dunn's index) [83], Fuzzy HyperVolume (FHV) [75], and hundreds of others.

2.6.3. Neuro-Fuzzy Systems

Neuro-fuzzy which refers to combination of artificial neural networks and fuzzy logic, results in a hybrid machine learning system that incorporates the human-like reasoning style of fuzzy systems using fuzzy sets and a linguistic model consisting of a set of IF-THEN fuzzy rules. The strength of neuro-fuzzy systems involves two important requirements in fuzzy modeling: interpretability and accuracy. ANFIS [33] (Adaptive Neuro-Fuzzy Inference System) is one the most well-known neuro-fuzzy systems [78].

2.6.3.1. ANFIS

2.6.3.1.1. Fuzzy Inference System

Fuzzy inference systems are also known as fuzzy-rule-based systems, fuzzy models, fuzzy associate memories (FAM), or fuzzy controllers when used as controllers. [33]. A fuzzy system is composed of five blocks [33]:

- A rule base containing a number of fuzzy if-then rules
- A database which defines the membership functions of the fuzzy sets used in the fuzzy rules
- A decision-making unit which performs the inference operations on the rules

- A fuzzification interface which transforms the crisp inputs into degrees of match with linguistic values
- A defuzzification interface which transform the fuzzy results of the inference into a crisp output.

Usually, the rule base and the database are jointly referred to the knowledge base. The steps of fuzzy reasoning (inference operations upon fuzzy if-then rules) performed by fuzzy inference systems are [5]:

1. Compare the input variables with the membership functions on the premise part to obtain the membership values (or compatibility measure) of each linguistic label.
2. Combine (through a specific T-norm operator, usually multiplication or min) the membership values on the premise part to get firing strength (weight) of each rule.
3. Generate the qualified consequent (either fuzzy or crisp) of each rule depending on the firing strength.
4. Aggregate the qualified consequents to produce a crisp output. (defuzzification step)

Most fuzzy inference systems can be classified into three types:

Type 1 The overall output is the weighted average of each rule's crisp output induced by the rule's firing strength and output membership functions. The output membership functions used in this scheme must be monotonic function [84].

Type 2 The overall fuzzy output is derived by applying a "max" operation to the qualified fuzzy outputs. Various schemes have been proposed to choose the final crisp output based on the overall fuzzy output; some of them are centroid of area, bisector of area, mean of maxima, maximum criterion, etc [85].

Type 3 Takagi and Sugeno's fuzzy if-then rules are used [86]. The output of each rule is a linear combination of input variables plus a constant term, and final output is the weighted average of each rule's output.

2.6.3.1.2. ANFIS architecture

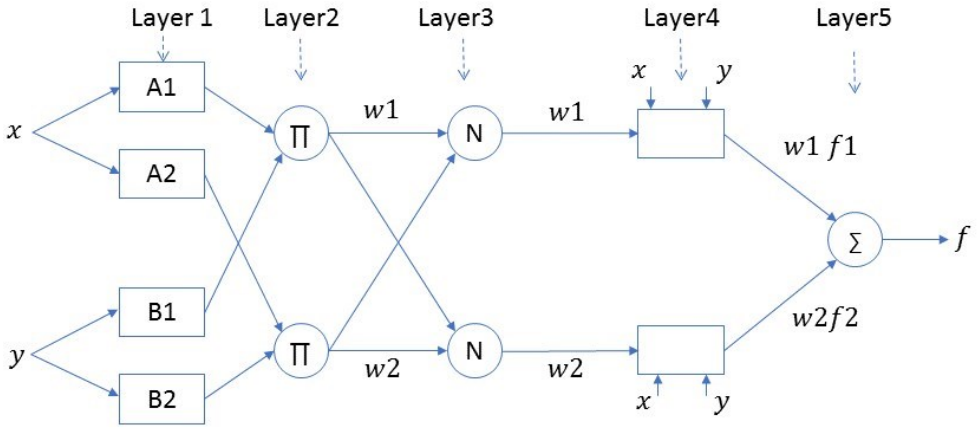


Figure 2.10: ANFIS architecture (adaptive nodes shown with a square and fixed node with a circle [33].

ANFIS was designed to be equivalent to a Takagi-Sugeno-Kang (TSK) fuzzy system. For example, assume that the system has two inputs x and y and one output z. For a first-order Sugeno fuzzy model [87], a rule set containing two fuzzy if-then rules could be the following:

$$Rule1: \ if\ x\ is\ A1\ and\ y\ is\ B1, then\ f1 = p1x + q1y + r1, \qquad 2\text{-}61$$

$$Rule2: \ if\ x\ is\ A2\ and\ y\ is\ B2, then\ f2 = p2x + q2y + r2, \qquad 2\text{-}62$$

The corresponding equivalent ANFIS architecture is as shown in Figure 2.10. In this figure, nodes of a layer have similar functions. Here we will be describing the layers:

Layer 1: Every node i in this layer is an adaptive node with a node function:

$$0_{1,i} = \mu_{A_i}(x) \qquad 2\text{-}63$$

In layer 1, there will be a subset of neurons, that respond to an input dimension. Each neuron implements a fuzzy membership function and the neuron in the subset together form a fuzzy partition of that input dimension. Here the membership function for A can be any appropriate parameterized membership function, such as the generalized bell function:

$$\mu_A(x) = \frac{1}{1 + |(x - c_i)/a_i|^{2b_i}} \quad 2\text{-}64$$

where a_i, b_i, c_i is the parameter set. As the values of these parameters change, the bell-shaped function varies accordingly, thus exhibiting various forms of membership functions for a fuzzy set A. Parameters in this layer are referred to as "premise parameters".

Thus, the architecture of Figure 2.10 will use the following equations for the first layer:

$$0_{1,i} = \mu_{A_i}(x)\ ,\ for\ i = 1,2, \quad 2\text{-}65$$

$$0_{1,i} = \mu_{B_{i-2}}(y)\ ,\ for\ i = 3,4 \quad 2\text{-}66$$

where x (or y) is the input to node i and A_i (or B_{i-2}) is a linguistic label (such as "small" or "large") associated with this node. In other words, $0_{1,i}$ is the membership grade of a fuzzy set A and it specifies the degree to which the given input x (or y) satisfies the quantifier A.

Layer 2: Every node in this layer is a fixed node labeled Π, whose output is the product of all incoming signals. The corresponding equations of Figure 2.10 for this layer will be:

$$0_{2,i} = \omega_i = \mu A_i(x)\ \mu_{B_i}(y), i = 1,2 \quad 2\text{-}67$$

Each node output represents the firing strength of a fuzzy rule.

Layer 3: Every node in this layer is a fixed node labeled N. The i_{th} node calculates the ratio of the i_{th} rule's firing strength to the sum of all rules' firing strengths:

$$0_{3,i} = \overline{\omega_i} = \omega_i/(\omega_1 + \omega_2), i = 1,2 \quad 2\text{-}68$$

For convenience, outputs of this layer are called "normalized firing strengths".

Layer 4: Every node i in this layer is an adaptive node with a node function

$$0_{4,i} = \overline{\omega_i} f_i = \overline{\omega_i}(\vec{p_i}x + \vec{q_i}y + \vec{r_i}) \quad 2\text{-}69$$

Layer 5: The single node in this layer is a fixed node labeled Σ, which computes the overall output as the summation of all incoming signals:

$$overall\ output = 0_{5,i} = \sum_i \overline{\omega_i} f_i = \frac{\sum_i \omega_i f_i}{\sum_i \omega_i} \quad 2\text{-}70$$

We note that the structure of this adaptive network is not unique; we can combine layers 3 and 4 to obtain an equivalent network with only four layers. In the extreme case, we can even shrink the whole network into a single adaptive node with the same parameter set. Obviously, the assignment of node functions and the network configuration are arbitrary, as long as each node and each layer perform meaningful and modular functionalities [33] [78].

2.7. Explainable Artificial Intelligence

Interpretability is defined as the degree to which a human can understand the cause of a decision or can consistently predict the model's result [88] [89]. Another definition of interpretability is Modern deep learning systems can match or even exceed human performance in some tasks. However, they are black boxes, and thus their decision-making process is sometimes incomprehensible to even human experts in the area. For example, consider the 37th move in the second game of the historic Go match between Lee Sedol, a top Go player, and AlphaGo [90] built by DeepMind. Another Go expert, Fan Hui, commented, *"It's not a human move. I've never seen a human play this move."*

2.7.1. Why eXplainable Artificial Intelligence (XAI)?

The simple fact is that all of the advances made in AI in the last two decades could be wasted if humans cannot *trust* those algorithms to make decisions at some level. Research has repeatedly shown that providing users with an explanation for decisions made by an algorithm is an essential precondition to winning user trust [24] [91]. In addition, explanation mechanisms enable other key system characteristics [92]:

- Verifiability: The AI model just infers from the data; any biases in the data are thus faithfully reproduced by the model [93]. It is also possible for the model to be incorrect; software is, after all, well-known for quality problems. Thus, verification of the system by domain experts is required. This, however, is only possible if those experts can *understand* the model.

- Comparative judgements: Interpreting a model will then allow us to compare and contrast different models. Distinct models may have the same classification performance, but completely different decision-making methods. It is possible that some of those models, while presently reasonable, might foreseeably lead to undesired outcomes in the future.
- Knowledge discovery: As with other data mining algorithms, deep learning can observe data patterns not visible to humans. However, in the KDD framework [94], such patterns must then be presented to an analyst and integrated into operations. Plainly, this means humans must understand them.
- Legal compliance: The European Union has issued regulations which implement a "right to explanation." Users can demand an explanation of an algorithmic decision impacting them [95].

2.7.2. Interpretability by Visualization Methods

In this section, we will describe some methods of adding interpretability to the system. The methodologies presented below are model-specific approaches that are often used for monitoring the learning algorithm, especially in neural networks. They have alsobeen found useful for interpreting predictions of neural networks.

- LRP: Layer-wise relevance propagation (LRP) redistributes the prediction backwards through the neural network from the observation using local redistribution rules until it finally assigns a relevance score to each input variable [96].
- Taylor Decomposition: Taylor decomposition is very similar to Layer-wise relevance propagation (LRP) except a different decomposition function (Taylor decomposition) is used for redistribution of the predicted value back to the input feature [96].
- Sensitivity Analysis: Sensitivity Analysis explains the model's prediction based on the model's locally evaluated gradient. Similar to partial dependence plots, sensitivity analysis quantifies the importance of each input variable (e.g., image pixel) while all other variables are fixed. Sensitivity analysis assumes that the most important input features are those to which the prediction is most sensitive [97].

- Guided backpropagation: Guided backpropagation is a non-conserving backward propagation technique. Similar to the regular backpropagation approach, but in case of guided backpropagation, the negative gradients are also suppressed. Negative gradients indicate that specific neuron has a negative influence on the class that we are trying to visualize. This way guided backpropagation puts weight only on the signals that contribute to the positive class scoring [98].
- DeconvNet: Like guided backpropagation, deconvolution is also a nonconserving backward propagation technique. Unlike guided backpropagation, however, deconvolution relies on max-pooling layers to direct the propagated signal to the appropriate locations in the image [30] [99] .

We will describe LRP, Taylor Decomposition, Guided Back-propagation, and DeconvNet methods in chapter of **PROPOSED ARCHITECTURE** in more detail.

Chapter 3

PROPOSED ARCHITECTURE

3.1. Deep Convolutional Neural Networks

3.1.1. CNNs Description

Figure 3.1 shows an example of our proposed architecture. In our design we have combined the convolutional layers from [61] with the proposed fuzzy classifier. Layer C1 computes six feature maps using a convolutional layer. Each neuron in this layer computes a convolution sum over a filter of size 5 × 5 following 2-35), using the same padding. Thus, if the input dimensionality is $D \times D$, the feature map will be of size $D \times D$. In Figure 3.1, D = 28 (as LeNet is tested on MNIST). Outputs of this layer then passes the ReLu activation function of 2-11). Layer S2 then applies the max-pooling operation of 2-36. Layer C3 computes 16 feature maps, followed again by ReLU 2-11.

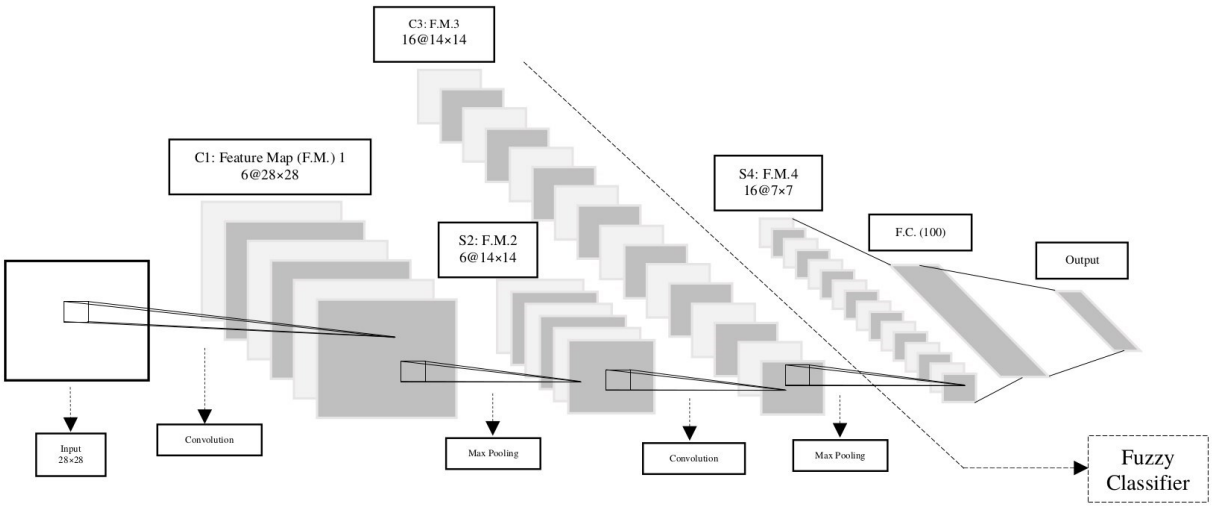


Figure 3.1: Deep Fuzzy architecture for MNIST and Fashion MNIST.

At this point, the data path will divide into two parts. For the Deep Fuzzy architecture, the feature maps of C3 will be the derived feature space passed to the inputs of the fuzzy classifier; thi sis depicted by the dashed line leading to the "Fuzzy Classifier" box. The fuzzy classifier performs fuzzy clustering and then applies Rocchio's algorithm to classify each sample. The other branch leads through a more traditional deep network output, which includes S4 as another sub-sampling layer and a fully-connected layer of perceptron neurons. Each one computes the weighted sum of its inputs:

$$v_j = \sum_i w_{ij} y_i + b_i \qquad 3\text{-}1$$

where y_i is the i_{th} output from Layer S4 (one pixel from one feature map), w_{ij} is the weight on the connection from that pixel to neuron j in the F.C. layer, and b_j is the bias for neuron j. This is followed by the ReLU activation function, producing the output of the j_{th} neuron in the F.C. layer. Finally, those outputs are passed to a fully connected layer of softmax neurons:

$$y_k = \frac{exp\left(\overrightarrow{w_k^T}.\overrightarrow{Y_{FC}}\right)}{\sum_{m=1}^{M} exp\left(\overrightarrow{w_m^T}.\overrightarrow{Y_{FC}}\right)} \qquad 3\text{-}2$$

where y_k is the k-th network output, $\overrightarrow{Y_{FC}}$ is the output vector from the previous F.C. layer, $\overrightarrow{w_k^T}$ is the transposed weight vector for the k-th softmax neuron, and M is the number of neurons in the softamx layer. The resulting network outputs are confined to [0,1], and sum to 1 for each network input.

To train the network, we employed the mini-batching form of stochastic gradient descent, with an added momentum term:

$$\Delta w_{ij}(n) = \eta.\delta_j(n).x_i(n) + \alpha.\Delta w_{ij}(n-1) \qquad 3\text{-}3$$

where $\Delta w_{ij}(n)$ is the update for the ij_{th} weight after observing the n_{th} mini-batch of input patterns, η is the learning rate, $\delta_j(n)$ is the error gradient for neuron j, $\Delta w_{ij}(n-1)$ was the update the ij_{th} weight after the previous mini-batch, and α is the momentum constant. Our loss function is the cross-entropy:

$$\epsilon = -\sum_{c=1}^{C} d_c log\big(p(y_c|\vec{x})\big) \quad 3\text{-}4$$

where d_c is a binary indicator of whether the ground-truth class for the current input $\vec{x}$ is c, and p is the probability that the output y_c will be predicted by the deep learner for input $\vec{x}$ [23]. In our experiments, we first train the full deep convolutional network (CNN and F.C. parts of Figure 3.1), and test it on our holdout data. Then, we copy the outputs of Layer C3 for each element of the training and test sets; this gives us the feature space and holdout test set we will use for our fuzzy classifier.

3.1.2. Capacity, Overfitting, Underfitting

The central challenge in machine learning is to induce a model that generalizes well to new, unseen data. We evaluate generalization by measuring performance on an out-of-sample test set; the sum of squared errors is a commonly used measure:

$$Error = \frac{1}{m^{test}} \Sigma_{x_{test}} \|Y_{test} - Y(x, w)\|^2 \quad 3\text{-}5$$

where Y_{test} is the target, $Y(x, w)$ is prediction of the network, m^{test} is number of testing samples. Achieving good generalization requires, from a practical perspective, accomplishing two related but distinct goals: minimizing training error, and minimizing the difference between training and testing error. The challenges in doing so can be dubbed underfitting and overfitting. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large [55]. In general, these are problems of capacity, defined as a model's ability to approximate a wide variety of input-output functions. Low capacity means that the model may not be capable of approximating a function well, while excessive capacity can overfit the data. One way of changing a model's capacity is changing number of input features and parameters. Regularization approaches that prune less-useful weights can reduce a parameter set [100], and feature selection or reduction usually reduce the number of input features [101].

Cross-validation [102] is a technique for assessing how the results of a statistical analysis will generalize to an independent data set. The goal of cross-validation is to test the model's ability to predict

new data that was not used, in order to provide an insight on how the model will generalize a dataset. One well-known implementation is k-fold cross-validation, which divides the available set of N examples into K subsets, where $K > 1$. $(K-1)$ subsets are used for training and the remaining subset will be used for validation [23]. The procedure repeats K times, with each subset used as the validation set exactly once. At the end, the population of validation results gives us a (hopefully robust) estimate of generalizability.

3.2. Rocchio's Algorithm

In this section we will describe the fuzzy version of Rocchio's algorithm [47]. This is the standard Rocchio's algorithm, preceded by the extra step of hardening fuzzy clusters into classes. Each data point in the training setis "assigned" to the cluster for which it has the highest membership. The hardened cluster is then labeled as the class shared by the largest number of examples assigned to it. An example is shown in Figure 3.2. In this figure, the data divides into the three classes of Square, Circle, and Triangle. 20 triangles, 6 squares, and 3 circles are found in *cluster*1. Because the most common data sample in *cluster*1 is triangle, this cluster will be of class Triangle. With the same procedure, *cluster*2 is of class Circle, and *cluster*3 is of class Square. Algorithm 6 summarizes classification with Rocchio's algorithm via FCM/GK clustering.

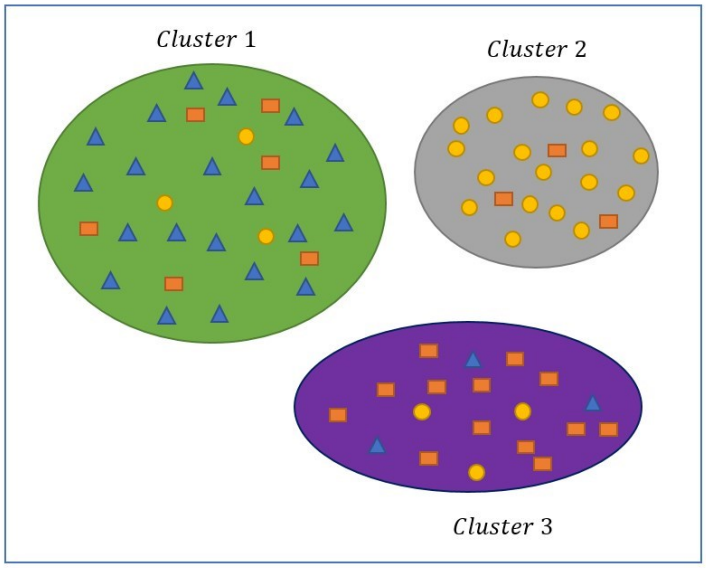


Figure 3.2: Fuzzy version of Rocchio's algorithm

Algorithm 7: Fuzzy Rocchio's Algorithm

```
Start:
   1. Calculate memberships via FCM/GK training
      Assign data points to clusters according to their membership
   2. Choose the most common label in each cluster as its class
   3. Calculate testing_Memberships via FCM/GK prediction
   4. Assign each data to a class according to its membership
End
```

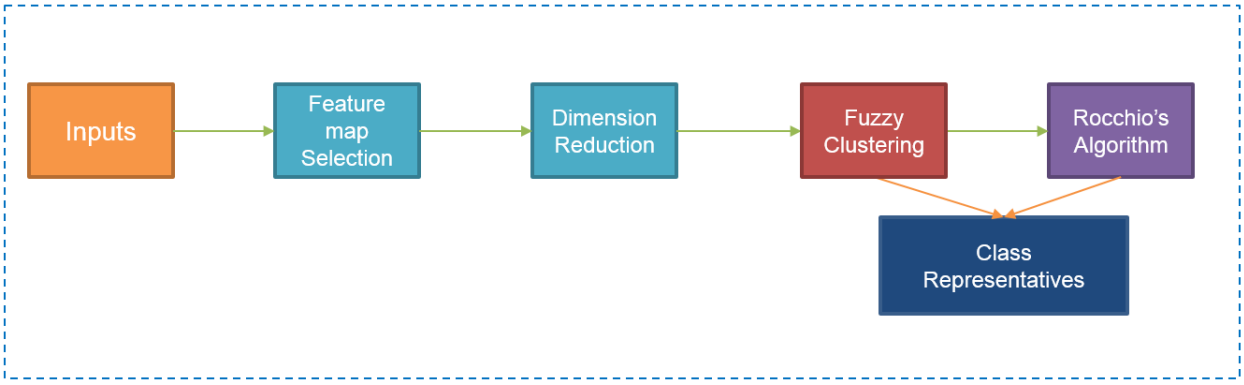


Figure 3.3: The fuzzy classifier unit.

3.3. Fuzzy Classifier

Figure 3.3 shows a block diagram of the fuzzy classifier unit. Feature map selection means that either all elements of a feature map are selected, or none are; this coherence is important as the classifier is intended to be interpretable. This stage omits redundant feature maps to improve classification accuracy. Dimension reduction based on PCA or UMAP is then employed as GK clustering requires a matrix inversion step; with each selected feature map yielding 196 features (a 14x14 image), clustering in the unreduced dataset could be prohibitively slow. The fuzzy clustering is executed, and Rocchio's algorithm used as the classifier.

The validation process for parameter tuning and testing procedure is presented in Algorithm 8. We use a cross-validation design for tuning parameters such as maximum number of iterations, the exponent constant, dimension of data or number of components, ρ (in GK), feature selection, etc. Algorithm 9 presents the fuzzy clustering component (an extension of Algorithm 4). The difference is the usage of Mahanabolis distance and co-variance matrix for stretching the axis while running GK.

Algorithm 8:Fuzzy classifier parameter Tuning with Cross-Validation

```
Start:
      trInp, trLabel, valInp, valLabel ← read input data
      selected features ←feature map selection
      input ←DimensionReduction(trInp[selected features])
      valData ←DimensionReduction(valInp[selected features])
      centers, memberships = FuzzyClustering (input)
      val_memberships = Fuzzyprediction (centers, valData)
      centerLabels ←Rocchio(trLabel, memberships, centers, input)
      accuracy ←Accuracy(valLabel, centerLabels, val_memberships)
      Tune parameters w.r.t. accuracy
End
```

Algorithm 9: Fuzzy clustering algorithms - Training and Prediction

```
Start:
      if training then
            Membership initialization
            while iteration < τ do
                  Calculate centers
```

```
                    Compute the distance
                    if GK then
                          Use Mahanabolis distance
                    else
                          Use Euclidean distance
                    Update memberships
                    if error termination criteria < ϵ then
                          go to end
      else
            Run the training phase without updating the centers.
end
```

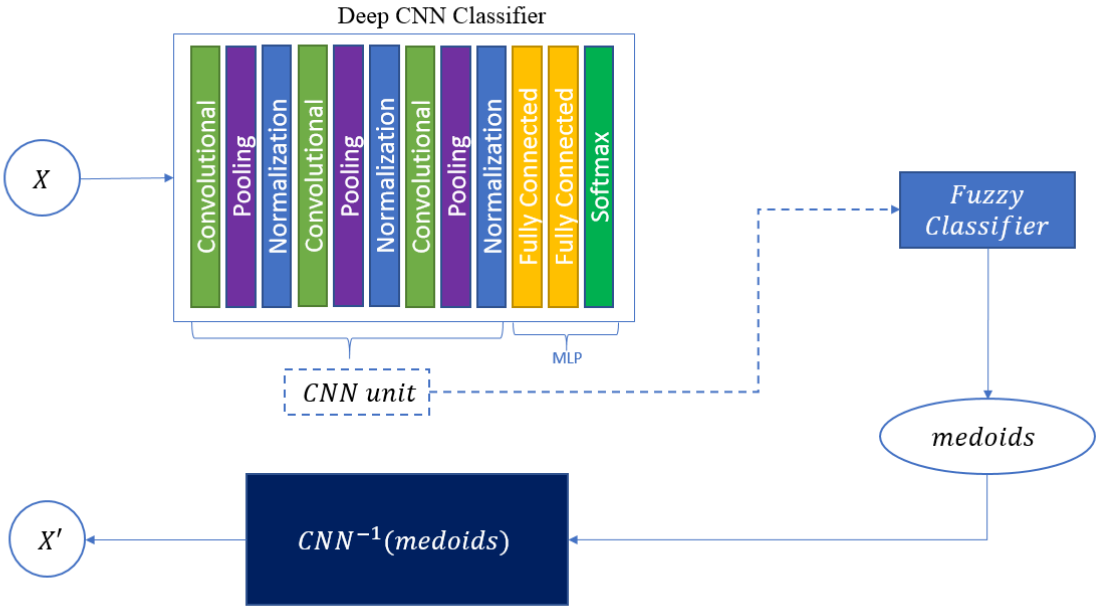


Figure 3.4: $CNN^{-1}_{medoids}$

Our explanation mechanism starts with the clustering; we identify the medoid element of each cluster, and treat it as the cluster representative [103]. We will then map this element back to the original image space, identifying the pixels that are most relevant in producing the classifier output. Plotting these as a heat map gives us our explanation for that whole cluster of images; we will henceforth refer to this as the *saliency map* for a given output. This process is summarized in Figure 3.4. We will discuss algorithms for computing $CNN^{-1}_{medoids}$ in the next section.

3.4. Techniques for Finding Saliency Maps

With the medoids of the fuzzy clusters identified, we now need to find how each pixel in the original feature space contributes to the final classification for each medoid. In this section we will describe three well-known methods for this task: Layer-wise Relevance Propagation (LRP), Taylor Decomposition, and Guided Backpropagation. I have used [104] toolbox for these methods.

3.4.1. Layer-Wise Relevance Propagation (LRP) [96]

The idea of pixel-wise decomposition is to find the contribution of a single pixel of an image x with V pixels to the prediction $f(x)$ made by a classifier f. The goal is to find to what extent each pixel contributes to the classification result (it could be positive or negative). Let the mapping $f: R^v \rightarrow R^1$, $f(x) > 0$, denote the learned structure. One approach is to decompose the prediction $f(x)$ as a sum of terms of the separate input dimensions x_d (i.e. pixels):

$$f(x) \approx \sum_{d=1}^{V} R_d \qquad 3\text{-}6$$

where R_d is a quantity called the *relevance* (discussed below). $R_d < 0$ contributes evidence against the presence of a structure to be classified while $R_d > 0$ contributes evidence for its presence. LRP in its general form assumes that the classifier can be decomposed into several layers of computation, with the first layer being the inputs and last layer is the real-valued prediction output f. The l_{th} layer is modeled as a vector $z = \left(z_d^{(1)}\right)_{d=1}^{v(l)}$ with dimensionality $v(l)$. Layer-wise relevance propagation assumes that we have a relevance score R_d^{l+1} for each dimension z_d of the vector z at layer $l+1$. The idea is to find a Relevance score of the previous layer $R_d^{(l)}$ for the vector z at layer l. This process continues iteratively until the relevances for the inputs are determined. The LRP method supposes that total amount of relevance is equal for all layers, and thus:

$$f(x) = \cdots = \sum_{d \in l+1} R_d^{(l+1)} = \sum_{d \in l} R_d^{(l)} = \sum_{d} R_d^{(1)} \qquad 3\text{-}7$$

The LRP method makes two assumptions. Firstly, it expresses the layer-wise relevance in terms of messages $R_{i\leftarrow j}^{(l,l+1)}$ between neurons i and j which can be sent along each connection. The messages are, however, directed from a neuron towards its input neurons (i.e. a backwards pass). Secondly, it defines the relevance of any neuron except an output neuron as the sum of incoming messages:

$$R_i^{(l)} = \sum_{k:\, i \text{ is input for neuron } k} R_{i\leftarrow k}^{(l,l+1)} \qquad 3\text{-}8$$

Note that the output neuron doesn't have any incoming message. Instead its relevance is defined as $R^{(l)} = f(x)$. Considering 3-7 and 3-8 the layer-wise relevance propagation is given by:

$$R_k^{(l+1)} = \sum_{i:\, i \text{ is input for neuron } k} R_{i\leftarrow k}^{(l,l+1)} \qquad 3\text{-}9$$

The difference from 3-8 is that 3-9 runs the summation over the input sources while 3-8 performs it on sinks. This can be demonstrated using 3-7 as follows:

$$\sum_k R_k^{(l+1)} = \sum_i \sum_{k:\, i \text{ is input for neuron } k} R_{i\leftarrow k}^{(l,l+1)}$$

$$= \sum_k \sum_{i:\, i \text{ is input for neuron } k} R_{i\leftarrow k}^{(l,l+1)} \qquad 3\text{-}10$$

$$= \sum_i R_i^{(l)}$$

One interpretation of 3-9 is that the messages $R_{i\leftarrow k}^{(l,l+1)}$ are used to distribute the relevance $R_k^{(l+1)}$ of neuron k onto its input neurons at layer l. Thus, in order to find the relevances the only required part is to be able to calculate the sent messages. These messages can be defined using the activation values and weights of the neurons as follows:

$$R_{i\leftarrow k}^{(l,l+1)} = R_k^{(l+1)} \frac{a_i w_{ik}}{\sum_h a_h w_{hk}} \qquad 3\text{-}11$$

As discussed above, in this equation, $R_{i\leftarrow k}^{(l,l+1)}$ is the sent message from neuron k to neuron i, and $R_k^{(l+1)}$is the relevance value for neuron k at layer l+1, a_i is the activation value, and w_{ik} is the weight between

neuron i and neuron k. If we suppose that the network has 5 layers. Similarly to the backpropagation algorithm, the relevance value for the last layer ($f(x)$) and 3-11 will be used to calculate the messages sent from the output neuron $R_k^{(5)}$ to the previous layer neurons. Then, the relevance of each neuron in layer 4 ($R_k^{(4)}$) can then be calculated using 3-8. Now that relevance values of layer 4 are available, we can follow the same procedure to find relevance values of layer 3. The procedure continues until we reach the input layer. At that point, the relevance values capture to what extent each input has contributed to the decision $f(x)$.

An important consideration is that neuron activations and weights in 3-11 can be zero or nearly so, possibly creating numerical instabilities. To prevent this, a *stabilizer* term $\in$ is added to the equation, yielding:

$$z_{ij} = a_i w_{ij} \tag{3-12}$$

$$z_j = \sum_i z_{ij} + b_j \tag{3-13}$$

$$R_{i\leftarrow j}^{(l,l+1)} = \begin{cases} \frac{z_{ij}}{z_j + \epsilon}.R_j^{(l+1)} & if\ z_j \geq 0 \\ \frac{z_{ij}}{z_j - \epsilon}.R_j^{(l+1)} & if\ z_j < 0 \end{cases} \tag{3-14}$$

By increasing ϵ value, the negative and positive relevance values will be more apparent in the resulting image [96].

Figure 3.5, and Figure 3.6 show a multilayer neural network annotated with the different variables and indices describing neurons and weight connections. Algorithm 10 also demonstrates the LRP algorithm. As shown in this algorithm, the activations and weights of a layer will be used to calculate the messages, and from these messages the relevance for each neuron will be calculated.

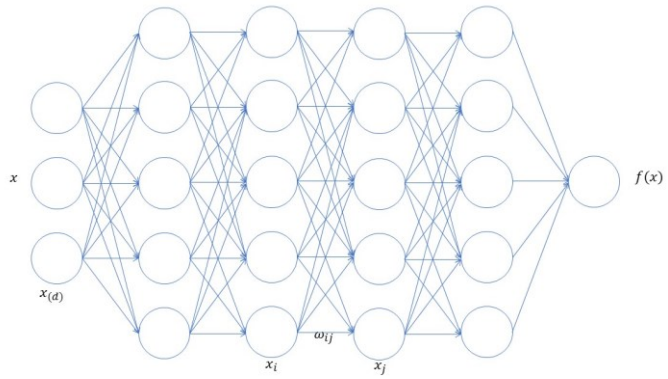


Figure 3.5: A multi-layer perceptron's forward pass [96].

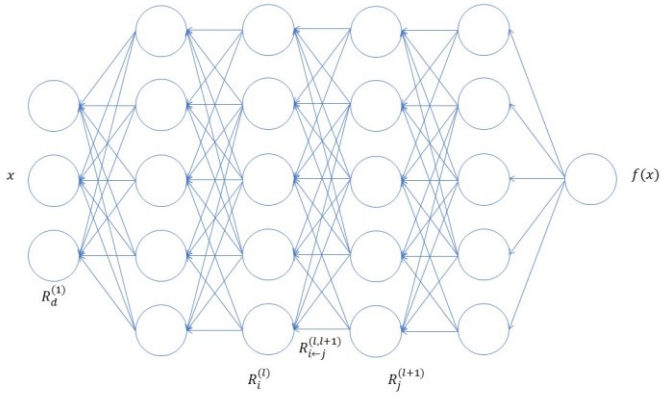


Figure 3.6: LRP algorithm's backward pass [96].

Algorithm 10: LRP algorithm

Start:
  Input: $R^{(L)} = f(x)$
    for $l \in L-1,\ldots,1$ **do**
      $R_{i\leftarrow j}^{(l,l+1)} \leftarrow$ as in 3-14
      $R_i^{(l)} = \sum_j R_{i\leftarrow j}^{(l,l+1)}$
  Output: $\forall d: R_d^{(1)}$
end

3.4.2. Taylor Decomposition [96]

A first order Taylor approximation can be used as an alternative to 3-6 for a general differentiable real-valued prediction function f:

$$f(x) \approx f(x_0) + \sum_{d=1}^{V} \frac{\partial f}{\partial x_{(d)}}(x_0)\ \left(x_{(d)} - x_{0(d)}\right) \qquad 3\text{-}15$$

Here $f(x)$ is the decision boundary function. In this setup, the choice of a Taylor base point x_0 is a free parameter. In case of classification the method looks for points with $f(x_0) = 0$, as the contribution of each pixel relative to the state of maximal uncertainty of the prediction, because $f(x) > 0$ denotes presence and $f(x) < 0$ absence of the learned structure. So, x_0 should be chosen to be a root of the predictor f. x_0 should furthermore be chosen to be close to x under the Euclidean norm, as Taylor approximation only holds in a neighborhood of x In case of multiple existing roots x_0 with minimal norm, they can be averaged or integrated in order to get an average over all these solutions. When $f(x_0) = 0$, the above simplifies to

$$f(x) \approx \sum_{d=1}^{V} \frac{\partial f}{\partial x_{(d)}}(x_0)\ \left(x_{(d)} - x_{0(d)}\right) \qquad 3\text{-}16$$

A first possible explanation of the classification decision $x \rightarrow f(x)$ can be obtained by Taylor expansion at a near root point x_0 of the decision function f:

$$R_d^{(1)} = (x - x_0)_{(d)} \cdot \left(\frac{\partial f}{\partial x_{(d)}}(x_0)\right) \qquad 3\text{-}17$$

where $R_d^{(1)}$is the relevance for d_{th} neuron of $1st$ layer (the input pixels). The derivatives of 3-17 can be efficiently calculated using the back-propagation algorithm. This method has been extended to Deep Taylor Decomposition in [105].

3.4.3. Guided Back-propagation [98]

3.4.3.1. Deconvnet [30]

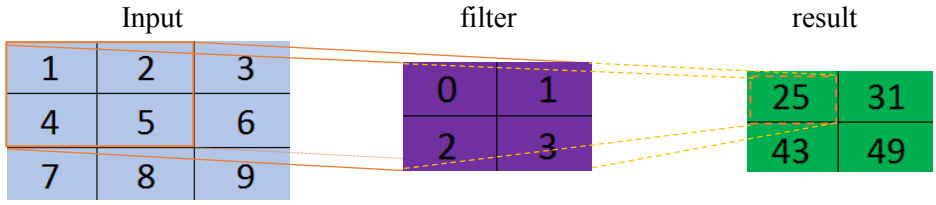
The deconvolutional network ("deconvnet") is a visualization technique learned by neurons in higher layers of a CNN. Given a high-level feature map, the "deconvnet" inverts the data flow of a CNN. Typically, a single neuron is left non-zero in the high-level feature map. Then the resulting reconstructed image shows the part of the input image that is most strongly activating this neuron (and hence the part that is most discriminative to it). In order to conduct a reconstruction through max-pooling layers this method first performs the forward pass calculations to compute "switches" – positions of maxima within each pooling region.

Deconvnet [30] [99] [106] uses unpooling and deconvolution layers as follows:

- Unpooling: The max pooling operation is non-invertible. However, we can obtain an approximate inverse by recording the locations of the maxima within each pooling region in a set of switch variables. In the deconvnet, the unpooling operation uses these switches to place the reconstructions from the layer above into appropriate locations, preserving the structure of the stimulus. Unpooling uses the same kernel of the max-pooling layer and the remaining activations are zeroed.
- Deconvolution: The convnet uses learned filters to convolve the feature maps from the previous layer. To invert this, the deconvnet uses transposed versions of the same filters, but applied to the rectified maps, not the output of the layer beneath. In practice this means flipping each filter vertically and horizontally Projecting down from higher layers uses the switch settings generated by the max pooling in the convnet on the way up. As these switch settings are peculiar to a given input image, the reconstruction obtained from a single activation thus resembles a small piece of the original input image, with structures weighted according to their contribution toward to the feature activation. Since the model is trained discriminatively, they implicitly show which parts of the input image are discriminative. Note that these projections are not samples from the model, since there is no generative process involved. A deconvolutional layer is just the transpose of its

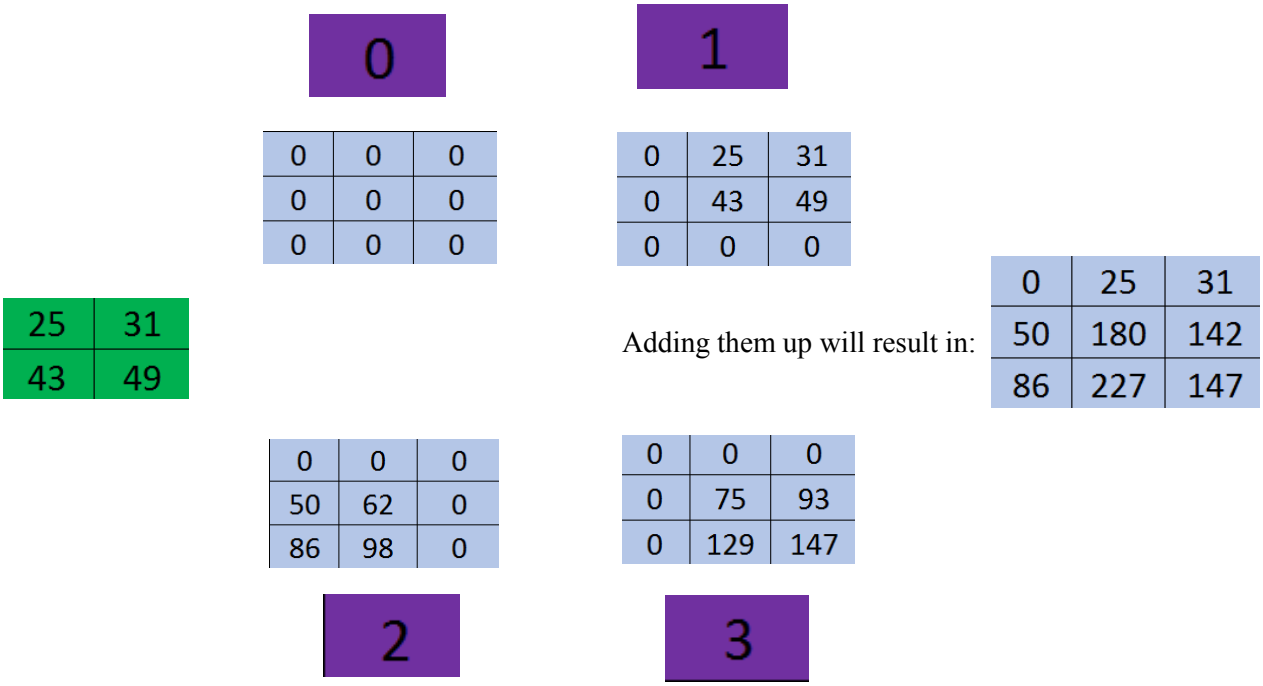
corresponding convolutional layer by adding paddings around the pixels and performing Deconvolution operations. We can find detailed arithmetic of deconvolution at [107].

In the following, we will describe how convolution operation works:



$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 2 + 8 + 15 = 25$$

The deconvolution operation uses the filter and the result to go backward. The simplest description is to multiply the output to each weight in the filter and placing the resulted matrix according the indices of the weight. Then the adding the results together will create the final result.



The following code can be used to check the results:

```
import tensorflow as tf
import numpy as np
x = tf.constant(np.array([[
    [[25], [31]],
    [[43], [49]]
]]), tf.float32)
myList = [[0,0,0,0],[0,1,0,0],[0,0,2,0],[0,0,0,3],[0,1,2,3]]
for l in myList:
    i,j,m,n = l
    f = tf.constant(np.array([
        [[[i]], [[j]]],
        [[[m]], [[n]]]
    ]), tf.float32)
    conv = tf.nn.conv2d_transpose(x, f, output_shape=(1, 3, 3, 1),
strides=[1, 1, 1, 1], padding='VALID')
    print(l)
    with tf.Session() as session:
        result = session.run(conv)
        for line in result[0]:
            for el in line:
                print(el[0], end=" ")
            print()
```

- Rectification: The convnet uses ReLU non-linearities, which rectify the feature maps thus ensuring the feature maps are always positive. To obtain valid feature reconstructions at each layer (which also should be positive), we pass the reconstructed signal through a ReLU non-linearity.

3.4.3.2. Extension of Deconvnet

Guided backpropagation extends the deconvolution approach by combining it with a simple technique visualizing the part of the image that most activates a given neuron using a backward pass of the activation of a single neuron after a forward pass through the network. Simonyan et al. [97] This is a visualization of firing strength of each pixel in order to produce a given result at one output neuron. The authors showed that this was similar to deconvolutional networks [30] except for the handling of the nonlinearity at rectified linear units (ReLUs). Springenberg et al., [98] combined these two approaches

into Guided Backpropagation, along with a rule to zero out the importance signal at a neuron if either the input or the importance signal are negative.

3.5. Research Questions for System Evaluation

In the following, we pose several research questions and design experiments to answer them. In general, our designs compare the performance of our deep fuzzy system against the same fuzzy classifier in the original image, against the full deep learner, and against recent results in the literature.

Any new learning algorithm must offer some improvement on the status quo; our first research question is thus *RQ1:Does the fuzzy clustering layer improve upon the accuracy of the deep learner it is based on? If not, what is the trade-off between the accuracy and interpretability?* We will evaluate RQ1 with a traditional comparison (via out-of-sample accuracy) of the deep fuzzy system against the deep learner it is based on. If (as we suspect) the deep fuzzy system is less accurate than the deep learner, this experiment also reveals the size of that difference.

The next question is whether the deep fuzzy system has in fact given us any advantage over the fuzzy system component itself. This yields *RQ2: Does the use of automated feature extraction lead to improve classification results versus training our fuzzy classifier in the original feature space?* RQ2 can again be evaluated by comparing the deep fuzzy system against the same fuzzy classifier trained in the original image space.

We then turn our attention to the wider literature on deep learning; there are many papers that investigate our benchmark datasets, and we also need to determine how our deep fuzzy system performs against them. This yields *RQ3: Is our proposed architecture comparably accurate to existing deep learning systems?* Again, this question is answered by comparing our deep fuzzy system to those other architectures, including the state-of-the-art (as best we can determine).

Our intent is that the deep fuzzy architecture of Figure 2 could be applied to any CNN, not just LeNet and its variations. Our next question is *RQ4: Can our deep fuzzy architecture be generalized to other CNNs?* We will examine this question by combining the fuzzy classifier with four different CNNs,

and repeating the experiments for RQ1 on each. These four networks include variants on LeNet and AlexNet, a CNN with four convolutional layers, and Residual networks.

We can also inquire what the impact of different clustering algorithms might be. FCM produces hyperspherical clusters, while GK clusters are hyperellipsoidal; is there a performance difference between them? (Note that these are only two of the thousands of fuzzy clustering algorithms that have been proposed.) This yields *RQ5: What is the impact of different clustering algorithms on the deep fuzzy system?* RQ5 can be evaluated by substituting different fuzzy clustering algorithms in Figure 3, and then comparing the resulting deep fuzzy systems using the same design as RQ1.

Finally, the prime intent of designing this deep fuzzy system is to obtain a more interpretable classifier. We thus pose research question *RQ6: Is the deep fuzzy system more interpretable than the original deep fuzzy system?* Formal experiments to measure a characteristic such as interpretability are designing in the psychology literature, and are beyond the scope of the current study. However, we can offer a demonstration of our proposed explanation mechanism, and explore how it accounts for both correct and erroneous classifications in our data.

Chapter 4

METHODOLOGY

4.1. Dataset Description

We have chosen three commonly-used benchmark datasets from the deep learning literature: MNIST [49], Fashion MNIST [50], and CIFAR-10 [51].

4.1.1. MNIST

MNIST [49] is a collection of grayscale images, each representing one handwritten digit and labeled with the correct value. The samples were gathered from roughly 250 individuals; one half are from from U.S. Census Bureau employees, while the other half come from high school students. MNIST is pre-partitioned into 60,000 training samples and 10,000 testing samples; the writers for the two groups are disjoint. Each MNIST image is a 28x28 pixel square, centered on the center of mass of the digit pixels. These images are constructed from a set of 20x20 pixel square images containing the actual digits, size-normalized to this bounding box. While the original samples were binary images, the anti-aliasing technique used in the normalization introduced gray levels. See [49] for more details.

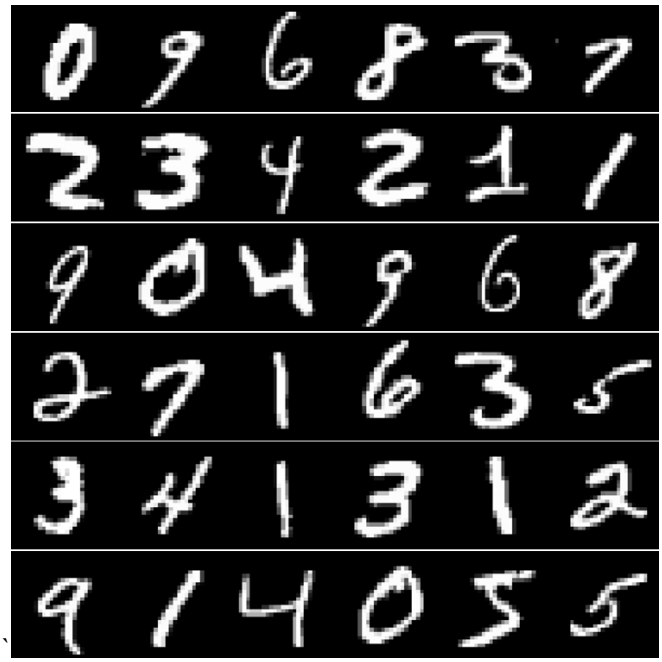


Figure 4.1: Some samples of MNIST hand writing digit dataset

4.1.2. Fashion MNIST

Fashion MNIST [50] is a dataset for clothes classification. Fashion MNIST serves as a direct drop-in replacement for the original MNIST dataset and shares the same image size and structure of training and testing splits with MNIST. Each Fashion MNIST sample is a 28x28 grayscale image, associated with a label from one of ten classes (t-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots). As discussed in [50] 1) MNIST is too easy to classify. 2) MNIST is overused. 3) MNIST cannot represent modern computer vision tasks. Thus, a more challenging and representative dataset was needed, and Fashion MNIST fills that need. Figure 4.2 shows some samples of Fashion MNIST dataset.

Figure 4.2: Some samples of Fashion MNIST dataset

4.1.3. CIFAR-10

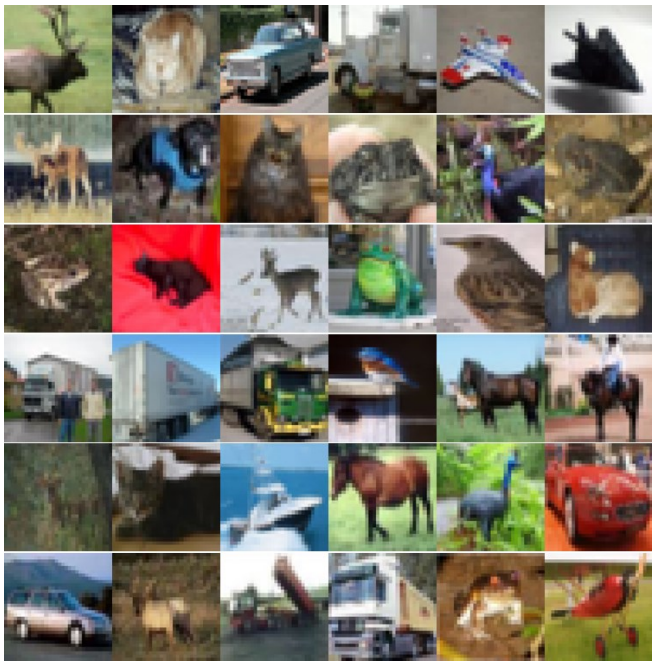


Figure 4.3: Some samples of CIFAR-10 dataset

CIFAR-10 [51] is an RGB image dataset, consisting of 60,000 images of size 32×32 in ten classes, with 6000 images per class. There are 50,000 training images and 10,000 testing images. The ten mutually exclusive classes are airplane, automobile, bird, cat deer, dog, frog, horse, ship, and truck. Figure 4.3 shows some samples from CIFAR-10 dataset.

4.2. Experimental Setup

The existing literature on all three datasets reports results based on a single-split design; for comparability we will follow this design, and use the same test sets of 10,000 images as our out-of-sample evaluation for all three datasets. Our training datasets will be 10,000 examples randomly selected from the predefined training sets, to reduce the computation time. For MNIST and Fashion MNIST we employ the CNN of Figure 3.1. However, preliminary experiments showed this architecture to perform poorly on CIFAR-10, and so we instead use a variant of the CNN in [2] for this dataset; the architecrture is depicted in Figure 4.4. Note that we have cropped each sample image to 24 × 24 image. That's a common technique to improve performance on CIFAR-10 [108].

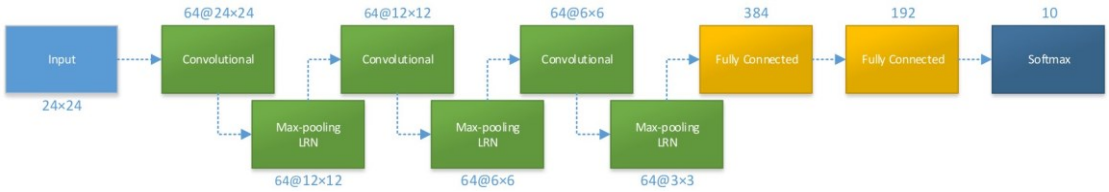


Figure 4.4: Trained CNN for CIFAR-10.

In Figure 4.4, the 32 × 32 input images come from the CIFAR10 dataset, and are cropped to 24 × 24. image. The first convolutional layer computes 64 feature maps of 24 × 24 (Same Padding). After a ReLU activation, C1's output passes through max-pooling and LRN, yielding 64 feature maps of 12 × 12. The network repeats this procedure twice more. Then two fully connected layers of perceptron neurons and a

softmax layer classify the data. The first F.C. layer has 384 neurons, and the second one includes 192 neurons. Note that this structure is itself essentially an MLP in itself.

For both Figure 3.1 and Figure 4.4, we have decided not to optimize the hyper-parameters of the CNN component of our design, in order to focus on the impact of the fuzzy component on our results. The CNNs are thus trained with a mini-batch size of 128, $\eta = 0.01$, $\alpha = 0.9$. While it may be the case that further optimization of the CNN might increase the overall performance of our deep fuzzy classifier, in this thesis we are more interested in the contribution of the fuzzy component and the interpretability. As discussed before, understanding and right usage of methods which prevent us from overfitting, can have a huge impact on system's performance. We will use L2 Regularization [100], feature map selection (feature selection [101]), and cross validation [102] for parameter tuning and improvement of system's performance.

As described in the question 2 and its answer, we will also get help from scatterplot of the data. All the scatterplots in this thesis are using PCA, and they're showing the 10,000 sample of the data. We have chosen this to reach a low-level insight of data shape and understand the difference between the data visually.

Chapter 5

EXPERIMENTAL RESULTS

In this section we will be describing our system's performance on MNIST, Fashion MNIST, and CIFAR-10, and discussing our findings on the six research questions we have defined.

5.1. Performance on MNIST

Our out-of-sample test accuracies on MNIST are reported in Table 1. This table divides into two sections, *Our experiments* and *Recent published results*. The first section demonstrates the performance of our deep fuzzy classifiers, deep CNN, and fuzzy classifiers. Note that the deep fuzzy classifier performs the classification in the extracted feature space while the conventional fuzzy (named as FCM/GK) will classify the data in the original feature space. The second section shows results of other recent methods including the state-of-the-art method.

The "Deep Fuzzy Classifier" (whether Deep FCM or Deep GK) and "Deep Convolutional Neural Network" (Deep CNN) entries in Table 1 represent the two data paths discussed in Figure 3.1. Note that for all experiments, our training dataset includes 10,000 samples. In the first branch which is training and testing the data with Deep CNN, the accuracy is 97.87. Deep FCM and Deep GK could reach 96.92, and 97.36 respectively. This means that the Fuzzy-based classifiers were not quite as accurate as the full deep neural network on MNIST. However, the difference is very small, and there does appear to be an important difference in interpretability; the deep fuzzy at this point generates reasonably-interpretable clusters in the derived feature space, while the deep network (see Figure 3.1) adds another max-pooling

layer, a fully-connected layer of ReLU neurons, and another layer of softmax neurons. These extra layers by themselves resemble a variation on the multi-layer perceptron architecture, which is well-known for being a black box. Thus, regarding question (1), the proposed architecture not only is performing well enough, but it also adds interpretability to MNIST classification.

Table 1: Comparison of our accuracy and most recent accuracies on MNIST.

Method	Number of Clusters	Accuracy
Our experiments		
DEEP FCM	100	96.92
DEEP FCM	10	87.63
DEEP GK	10	96.97
DEEP GK	100	97.36
DEEP CNN	-	97.87
FCM	100	84.48
GK	10	70.92
GK	100	81.61
Recent published results		
[109]	-	95.0
[110]	-	84.3
[111]	-	99.79
[112]	-	95.0

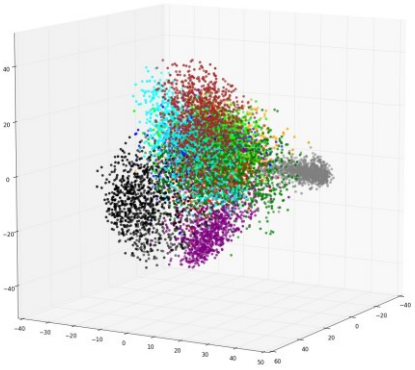
Regarding research question (2), we have tested our fuzzy clustering in the original feature space (pixels in the actual images). Three-dimensional scatterplots of the training data in the original feature space and the Layer C3 feature maps are depicted in Figure 5.1, respectively. These figures do suggest

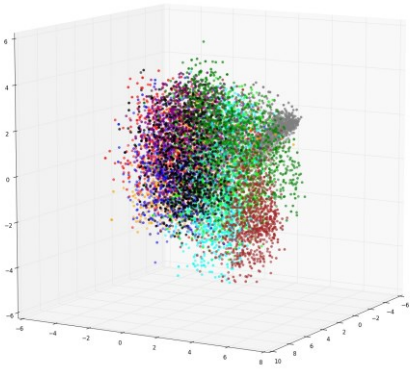
that the different classes become better separated in the derived feature space the CNN creates. As we can see from Table 1, there is also a substantial difference in accuracy when the fuzzy classifier unit classifies the data in original and derived feature spaces. We thus answer in the affirmative: automated feature extraction via the CNN did improve our classification accuracy. So, such an architecture will provide fuzzy logic with high level CNN's feature extraction mechanisms.

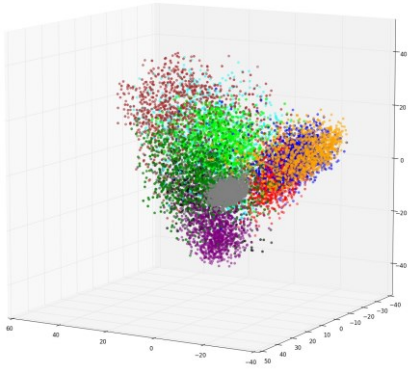
To evaluate research question (3), we have identified four recent publications in the deep learning field that evaluate their proposals on the MNIST dataset. Xie et al. [110] present a deep embedded clustering method that leans feature representations using deep neural networks. Diehl et al. [112] present a spiking neural network. Chen et al. [109] present a generative adversarial network that tries to maximize the mutual information between a small subset of the latent variables and the observation. Wan et al. [111] present a generalization of Dropout that randomly drops weights instead of activations; this is currently the best result on MNIST that we are aware of. [111] is known as DropConnect which is an extension of Dropout [113] introduced by Hinton et al. The difference between our result with [111] as the best available result is fairly small. Note that any research can define a specific model for a dataset, and train on it. We can also extend the proposed model to reach higher accuracy. These steps will be implemented later. On the other hand, we have showed [109] [110] which are fairly recent published results ranking behind our architecture. Thus, we can claim that deep fuzzy classifier is performing a good job in comparison with other recent methods.

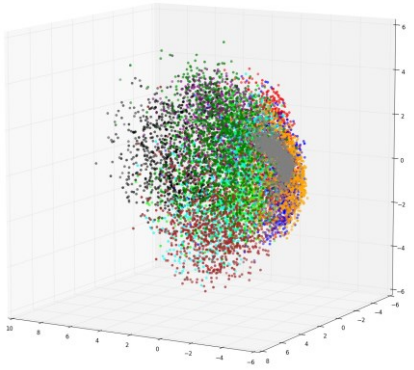
Regarding research question (5), we have provided results of Deep GK and Deep FCM. As shown in Table 1, DEEP GK could reach same classification accuracy with 10 clusters while DEEP FCM uses 100 clusters, and DEEP GK still has higher accuracy. This means that fuzzy clustering method has a high impact on the architecture's performance.

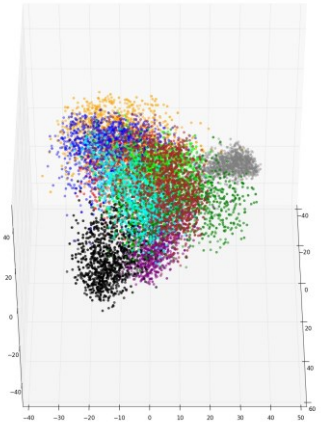
On the other hand, in the original feature space, GK and FCM classify the data with accuracy of 81.61, and 84.48 respectively (with 100 clusters). Considering DEEP GK's accuracy of 96.97 with only 10 clusters, and accuracy of 96.92 for DEEP FCM with 100 clusters, the Deep fuzzy classifier (specially GK) could play an important role on MNIST.











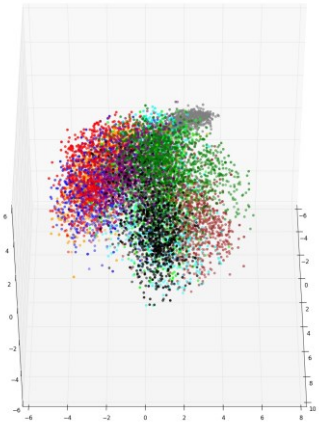


Figure 5.1: Scatter plots of MNIST dataset in CNN extracted feature space (column 1) and original feature space (column 2). (10,000 samples).

5.2. Performance on Fashion MNIST

In this section we will report the experimental results on Fashion MNIST dataset. Table 2 includes our out-of-sample test accuracies on this dataset. As shown in Table 2, Deep CNN classifies the data with the accuracy of 86.51 while Deep GK and Deep FCM classify the data with 79.60 and 81.71 respectively with 100 clusters. Deep CNN's accuracy is 4.8% greater than accuracy of Deep GK. Thus, regarding question (1), there's a trade-off between accuracy and interpretability.

Table 2: Comparison of our accuracy and most recent accuracies on Fashion MNIST.

Method	Number of Clusters	Accuracy
Our experiments		
DEEP FCM	100	79.69
DEEP FCM	10	64.93
DEEP GK	10	74.33
DEEP GK	100	81.71
DEEP CNN	-	86.51
FCM	10	58.06
FCM	100	74.24
GK	10	53.8
GK	100	73.1
Recent published results		
[114]	-	58.64
[115]	-	90.03

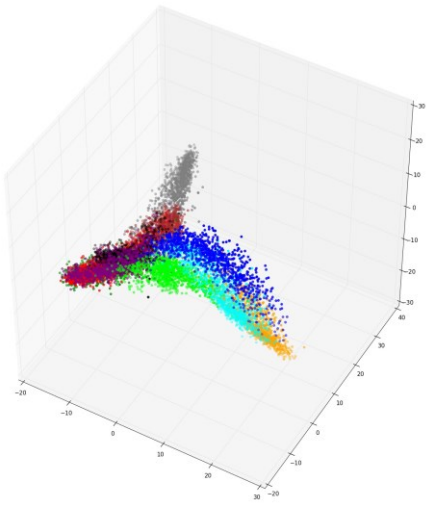
[2]	-	86.43
[116]	-	96.8

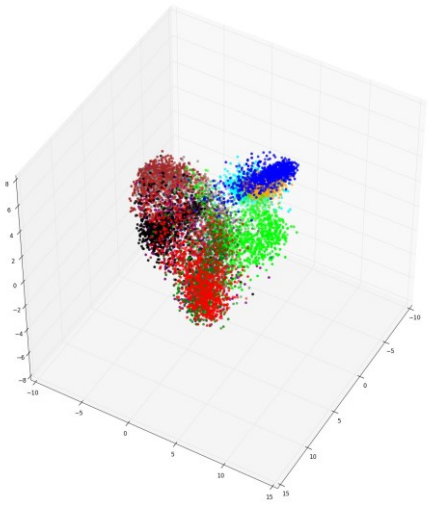
About question (2), comparing results of Deep FCM/GK with FCM/GK we find that the dataset has been classified in the extracted feature space better than in the original feature space. Figure 5.2 also shows that the data seems easier to classify in the extracted feature space than the original feature space, as data points in the original feature space are densely mixed and concentrated at one point, while the extracted feature space has been separated them. The difference between accuracies of FCM-based classifiers is 5.45, and 8.61 for GK-based classifiers. Thus, although the difference is not very big, we can claim that the fuzzy classifier is better to classify the data in extracted feature space than original feature space.

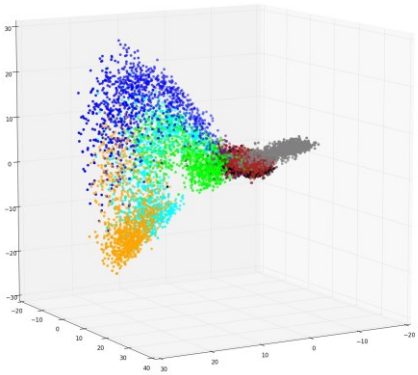
Question (3) asks whether the system's performance is comparable with other methods or not. We have gathered some recent published results on the Fashion MNIST dataset note that it was released in 2017 [50]. Zeng et al. [116] introduced a deep transfer learning method by integrating the features learned from the original images and features of extracted feature space. This paper [116] is currently the state-of-the-art method for Fashion MNIST classification as far as we know. This method has about 10% difference in terms of accuracy with the Deep CNN method, while the deep fuzzy classifier is about 15 percentage points behind the state-of-the-art. We suspect that substituting a different CNN (e.g. [116]) as the feature extractor could substantially increase our accuracy, but that will be a matter for future research. We can also explore other clustering methods such as Gath-Geva (GG) clustering [75]. Another idea is to substitute Rocchio's algorithm with another classifier.

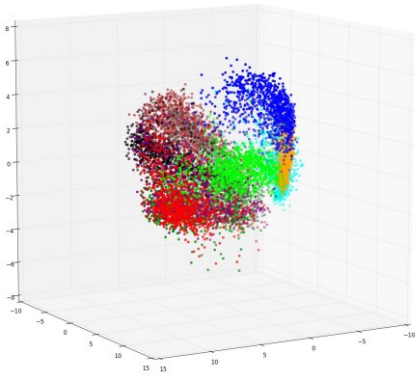
Regarding question (5), we can compare the FCM and GK results to each other. As shown in Table 2, Deep GK performs the classification with the accuracy of 74.33 with 10 clusters, while Deep FCM reaches 64.93 with the same number of clusters. This is about 10% difference in terms of accuracy. On the other hand, Deep GK and Deep FCM are performing similar to each other when we vary number of

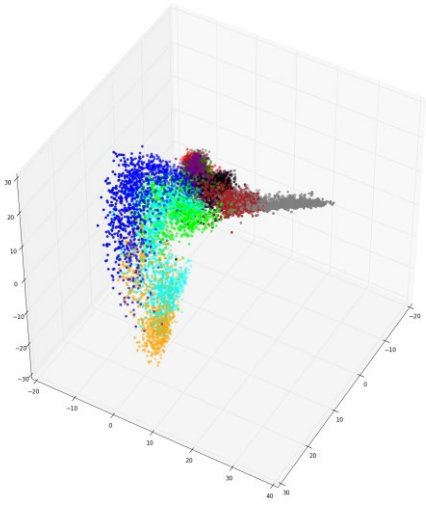
clusters to 100. However, we should note that interpretability does not come for free; there are often tradeoffs between how accurate a deep system works, and how interpretable it is.











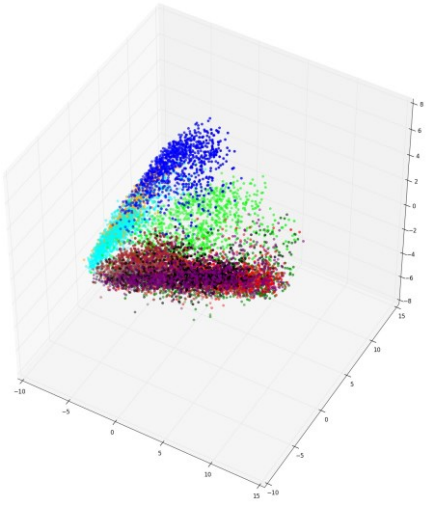


Figure 5.2: Scatter plots of Fashion MNIST dataset in CNN extracted feature space (column 1) and original feature space (column 2). (10,000 samples).

5.3. Performance on CIFAR-10

Our last dataset is CIFAR-10. CIFAR-10 which is an RGB dataset, includes pixels of 32 × 32, while MNIT and Fashion MNIST include pixels of size 28 × 28. However, when the data comes to the network the RGB image will be divided to 3 Red, Green, and Blue images. Thus, the input image is 3×32×32 = 3072. As discussed before, the data is cropped to 24 × 24 in order to reach better performance. Hence, the input data will be of size 3 × 24 × 24 = 1728.

Table 3, shows our out-of-sample test accuracies on CIFAR-10. Like the previous datasets, results of CIFAR-10 also includes two sections. One includes our experimental results while the other one includes recently published results. On the CIFAR-10 dataset, Cubuk et al. [117] could outperform other methods at Google Brain [118]. Previously we showed that [116] is state-of-the-art method in Fashion MNIST classification. [116] will classify CIFAR-10 with accuracy of 74.45%. The reason is Data Augmentation methods. [116] uses a Data Augmentation method in order to increase the accuracy for Fashion MNIST, and Cubuk et. al [117] introduce new data augmentation for CIFAR-10. These methods can be also applied in our method for increasing our accuracy. As shown in Table 3, the Deep CNN accuracy is 80.7. Thus, there's a huge difference between AlexNet and Cubuk et. al [117] performance (98.53%). Thus, the low accuracy is coming from CNN's accuracy, not fuzzy classifier. The difference between our experimental results on original feature space and the CNN's feature space in Table 3 and Figure 5.3 proves needs of using CNN's as feature extractors; question (2). As shown in the Table, DEEP GK has the accuracy of 67.98% while GK has 28.59% (difference = 39.39 percentage points). Regarding question (5), changing FCM to GK has helped the fuzzy classifier to increase its accuracy in both feature spaces. Comparing results of Table 3 with Table 1, and Table 2, changing the CNN will not affect the results, and the system is able to response for all type of CNNs (question (4)). However, dependent on the classification mechanism that a CNN is using, the accuracy of fuzzy system can fall behind it. Figure 4.4 shows the CNN architecture for CIFAR-10 dataset. As shown in the figure, the classifier unit consists of

two fully connected layers and one softmax layer. Such a classifier is complex enough to classify any dataset with high accuracy. Nevertheless, as shown in Table 3 our DEEP GK accuracy is not far behind the recent published accuracies. (question (3))

Table 3: Comparison of our accuracy and most recent accuracies on CIFAR-10.

Method	Number of Clusters	Accuracy
Our experiments		
DEEP FCM	100	59.29
DEEP FCM	10	49.04
DEEP GK	Max acc. @ 10	74.07
DEEP GK	100	72.05
DEEP CNN	-	80.7
FCM	10	22.66
FCM	Max acc. @ 18	31.31
GK	10	28.59
Recent published results		
[119]	-	79.7
[120]	-	96.53
[121]	-	82.18
[2]	-	89
[117]	-	98.53

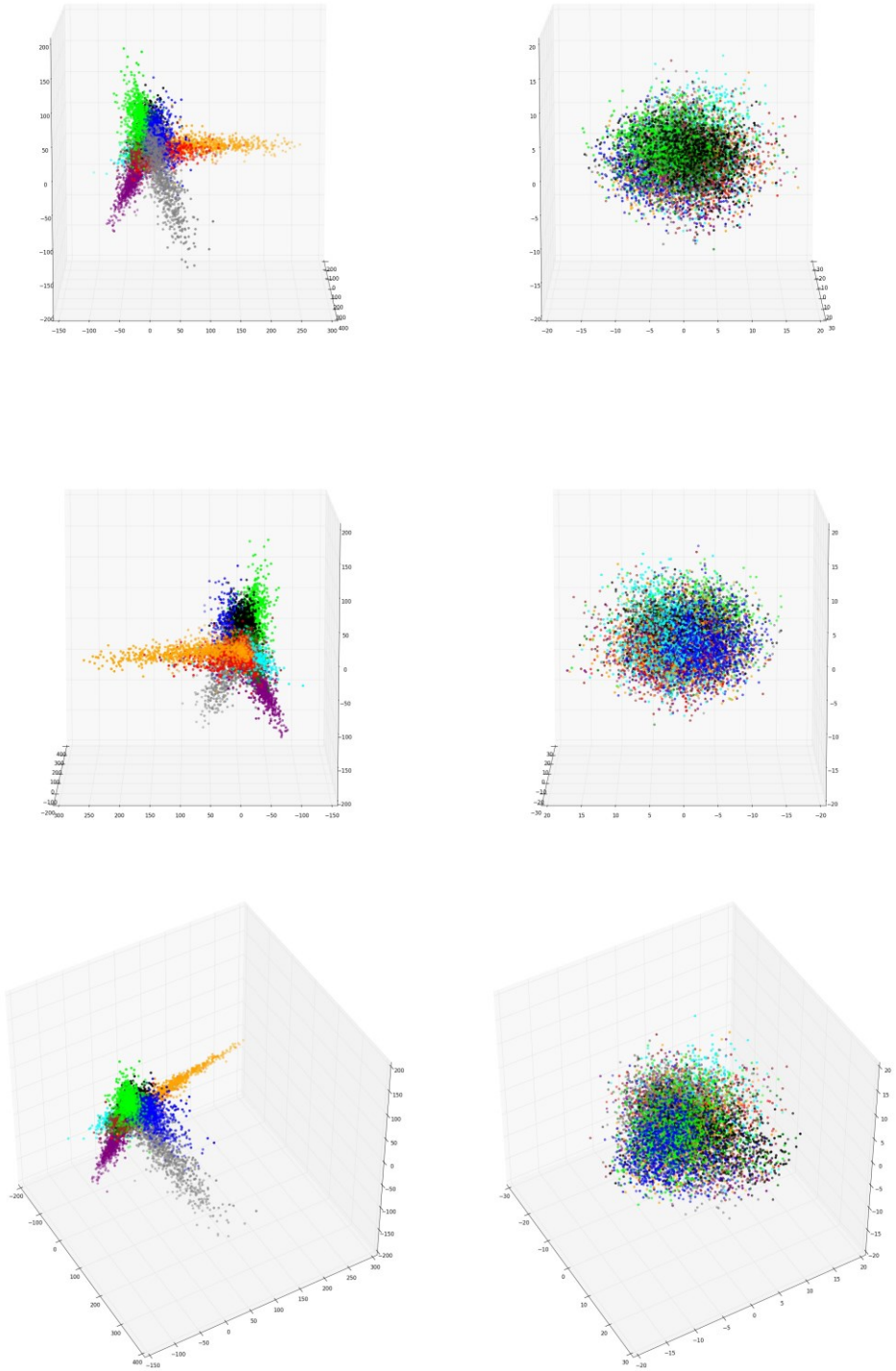


Figure 5.3: Scatter plots of CIFAR-10 dataset in CNN extracted feature space (row 1) and original feature space (row 2). (10,000 samples).

Chapter 6

Interpretability

6.1. Medoid Representatives

In this section we will try to answer research question 6: *Does using our approach yield a more interpretable classifier?* In the previous chapters, we talked about how to find representatives of each class and evaluated our system on three different datasets. We demonstrated that the Fuzzy clustering algorithm will provide us with some medoids for each class (Figure 3.3). The cluster medoid, which is an image drawn from the dataset, is the example whose average dissimilarity to all the objects in the cluster is minimal. In this chapter, we will explain how to use the saliency maps we have defined and the medoids together in order to interpret system's decision-making process. Figure 6.1 and Figure 6.2 are showing the medoid representatives of each class for MNIST dataset.

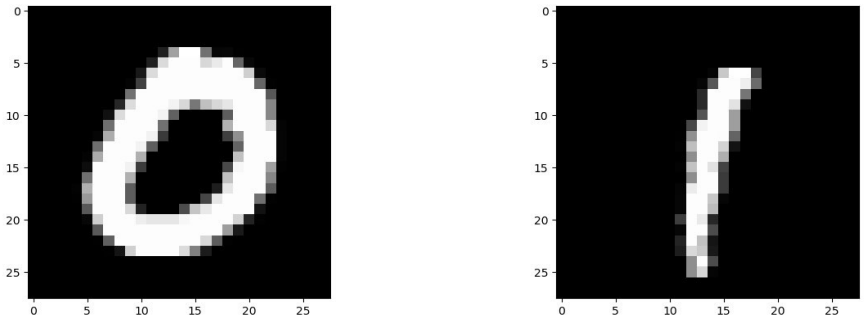


Figure 6.1: Medoid representatives of each digit

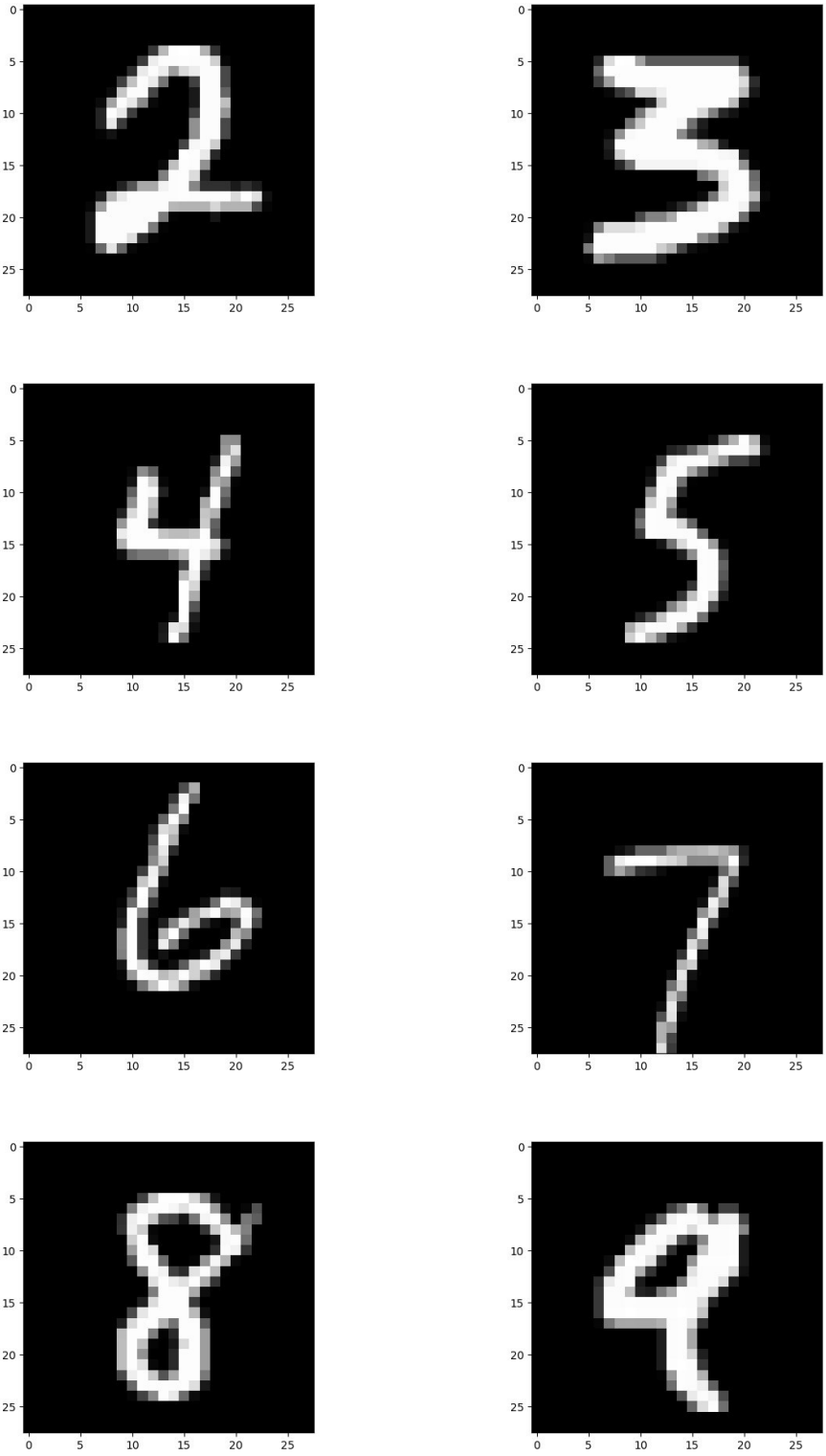


Figure 6.2: Medoid representatives of each digit (continued)

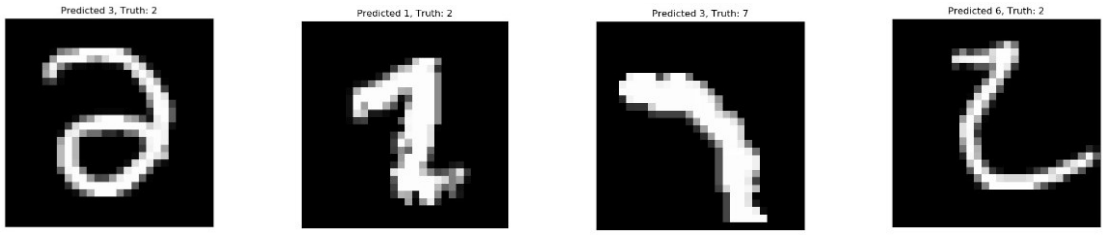


Figure 6.3: Some not understandable (hard to recognize) digits

As shown in Figure 6.1 and Figure 6.2, each medoid representative is completely recognizable for human beings. However, there are some samples shown in Figure 6.3 that are hard to recognize or not understandable for us. Our network has been misclassified them also. The network will classify any input, as it's forced to. However, all classifications will be performed according to the model induced by the learning algorithm. Understanding network's decision making process means not only understanding the reasons behind correct classifications, but also the reasons behind misclassifications.

In the following we will describe our interpretation results using Layer-wise Relevance Propagation (LRP), DeepTaylor decomposition, and Guided Back-propagation methods for each digit. Note that we have called these three methods as interpretation techniques. In Guided Back-propagation method, white and black pixels are showing the most effective pixels on classification. A completely white and a completely black pixel have same impact on classification, but different meanings. White ones are the pixels that need to be white in the original image, and black ones are the ones that need to be black in the original image. Gray pixels don't impact result of classification so much. However, LRP and Deep Taylor methods show both of these kind of pixels (black and white pixels in LRP and Guided Back-Propagation), in white. These methods just show to what extend the pixels have been helped the decision-making process. The whiter a pixel is, the more impact it has. Thus, the Guided Back-Propagation method may give us more detailed interpretations.

6.1.1. Digit 0

Figure 6.4 presents the saliency maps created by Guided Back-propagation, Deep Taylor decomposition, and LRP for the medoid of class "0." We highlight what we believe are the most

important pixels in those maps in green boxes, while Figure 6.5 presents the medoid itself. Our discussion will focus on comparing Figure 6.5 and Figure 6.4 (the same pattern will be followed for subsequent digits). We also provide the Logit and softmax output values of the CNN network to the left of the saliency maps (this will also be repeated for subsequent figures).

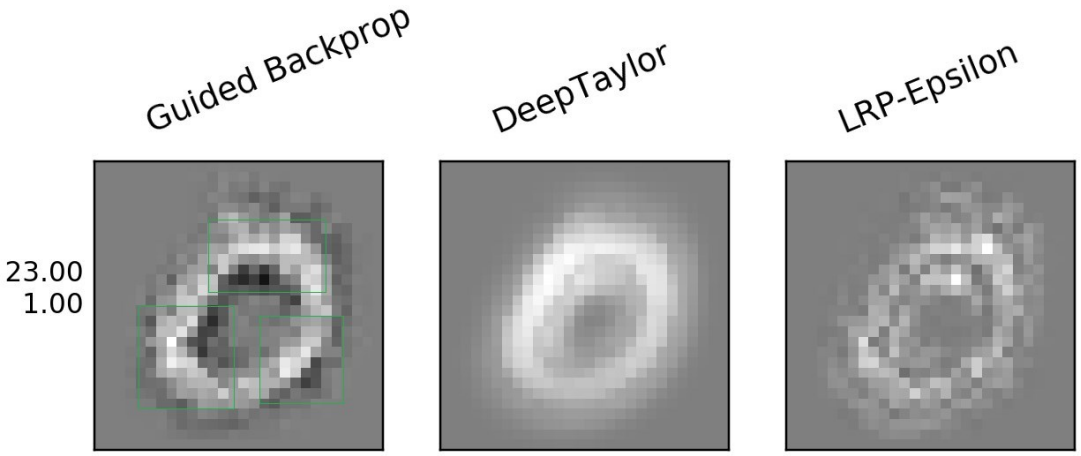


Figure 6.4: Saliency maps for medoid representative of class 0.

Figure 6.4 indicates the most important region in recognizing this input is the curved parts of the input. The LRP result shows that the curve part of the input placed at top right and bottom left are playing the most important rule. The figure also shows that the middle pixels are not important, but the network is extracting edges surrounding the input, i.e. the black pixels around the white ones. Deep Taylor method considers the top left curve as the most important part. It's also recognizing the edges around that curve. Guided Back propagation shows that the network thinks a digit 0 should have 3 important curves (showed in green windows) and recognizes their edges (the black pixels behind them).

Logit values, which are the values of the CNN F.C. layer before entering the softmax gate, provide additional evidence aiding the interpretation of our results. The logit value for medoid "0" is 23, mapping to a probability of 1.0 that the softmax gate produces an output of "0." This shows that the Fuzzy classifier has chosen a good representative for this class.

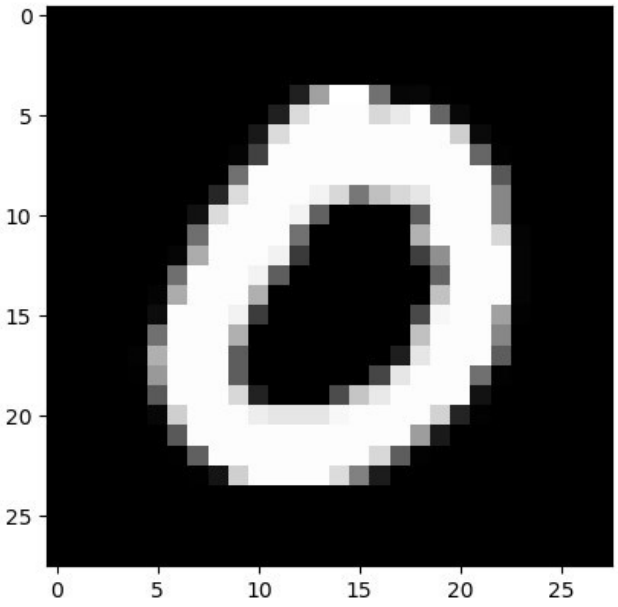


Figure 6.5: Medoid representative of class 0

6.1.2. Digit 1

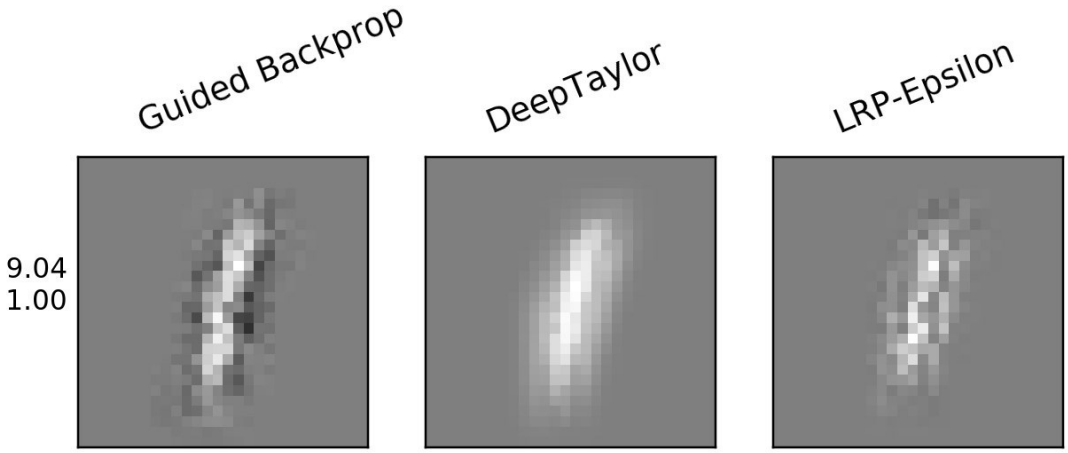


Figure 6.6: Saliency maps for medoid representative of class 1.

Figure 6.6 presents the saliency maps created by Guided Back-propagation, Deep Taylor decomposition, and LRP for medoid "1." Figure 6.7 presents the raw image of this medoid. As shown in Figure 6.6, the important features for recognizing digit 1 are a straight line and the surrounding edges. Note that in all methods, the high-importance pixels of the vertical line are almost connected to each other; we see this rarely in other digits. The difference between the Guided Backpropagation method and Deep Taylor is their focus. The Deep Taylor considers the middle of line as the most important part of image, while Guided Backpropagation (and LRP) finds two line at the top and bottom (connecting at the middle) of the vertical line. The logit value for this representative is 9.04, mapping to a probability of 1.0 of outputting class "1." which gives us almost 100% probability in classification of this representative of digit 1 after passing the logit to the softmax gate. This shows that Figure 6.7the fuzzy classifier again chose a very strong representative of class "1."

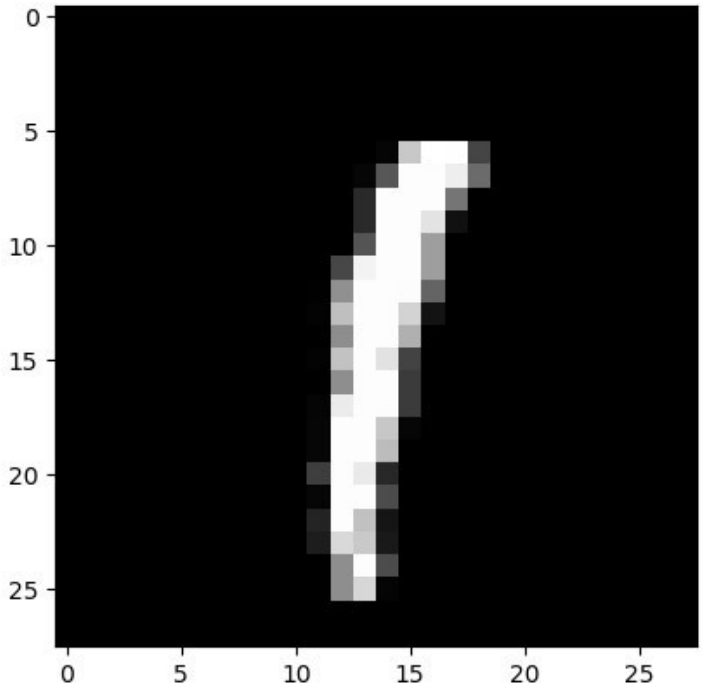


Figure 6.7: Medoid representative of class 1

6.1.3. Digit 2

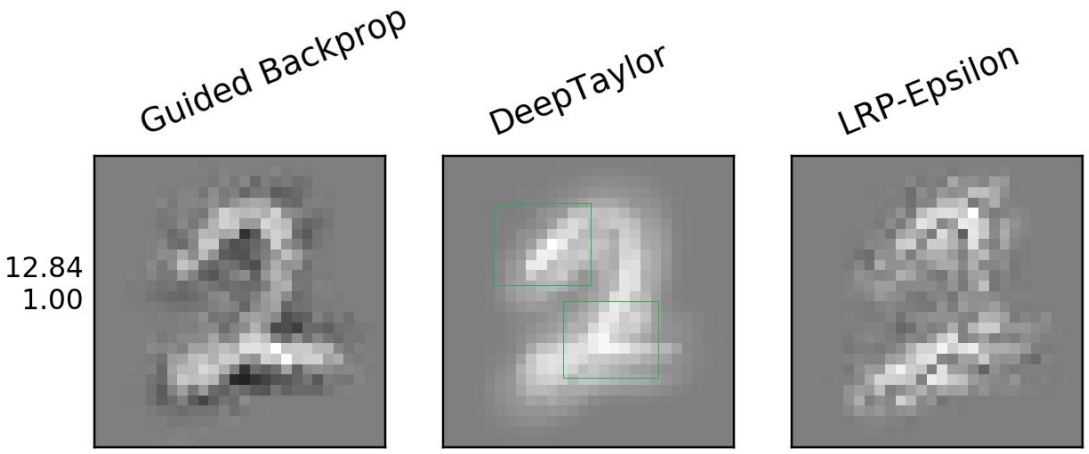


Figure 6.8: Saliency maps for medoid representative of class 2.

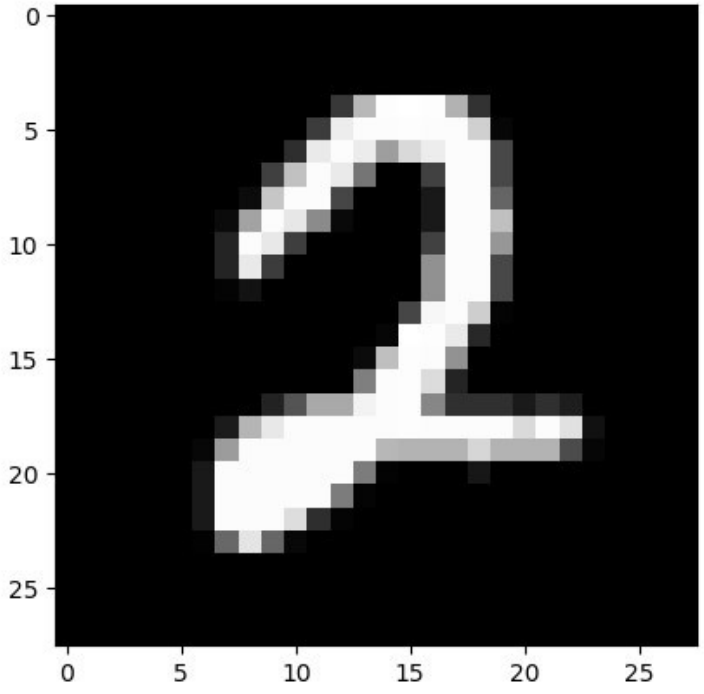


Figure 6.9: Medoid representative of class 2

Figure 6.8 presents the saliency maps for digit 2, and Figure 6.9 is the medoid image for the class. As shown in this figure, Guided back-propagation considers the horizontal tail as the most important region, followed by the upper curve, and their edges. Deep Taylor focuses on different regions, which we highlight in green boxes. Note that it includes the horizontal tail. The LRP method also shows the top right curve and bottom left tail as the most important regions. The logit value for this representative is 12.84, again meaning class "2" has a probability of 1.0.

6.1.4. Digit 3

We present our saliency maps and the medoid image for class "3" in Figure 6.10 and Figure 6.11, respectively. For Digit 3, all of the methods result in very similar images, although Guided back-propagation highlights the edges better than other methods (this seems to be the case for all digits). All methods consider the middle of image which interconnects bottom part and top part of digit 3 together, as the most important part, especially the curve that connects the bottom part to the middle. The results also show that although the entire top and bottom parts are not highly important, the tailing pixels have significant impact on recognizing digit 3 (especially the top part). Note that we may see elements of this curve part in digit 8, digit 5, and digit 6 as well. We will see how the network determines digit 3 and digit 8 (or digit 5 and 6) later; for now, we simply note that the important curve for digit 3 is on the right-hand side. The logit value for this medoid is also 17.81, meaning a probability of 1.0 for class "3."

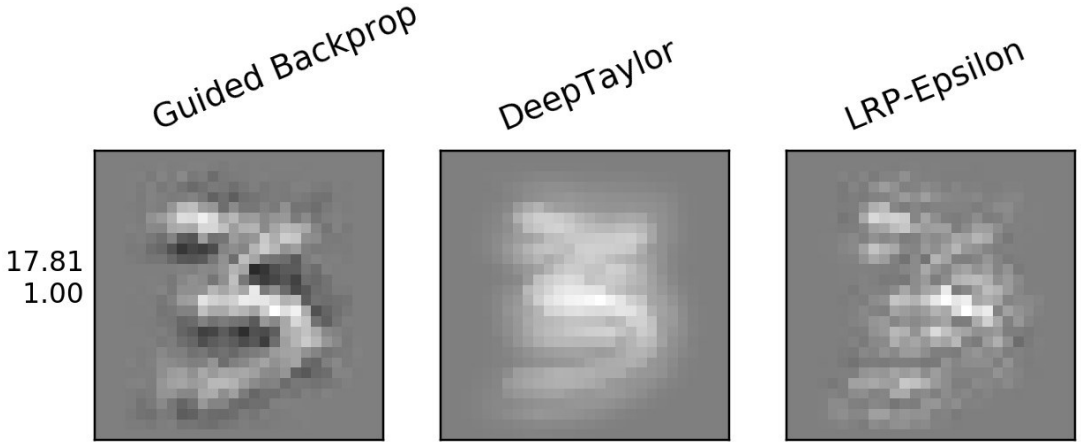


Figure 6.10: Saliency maps for medoid representative of class 3.

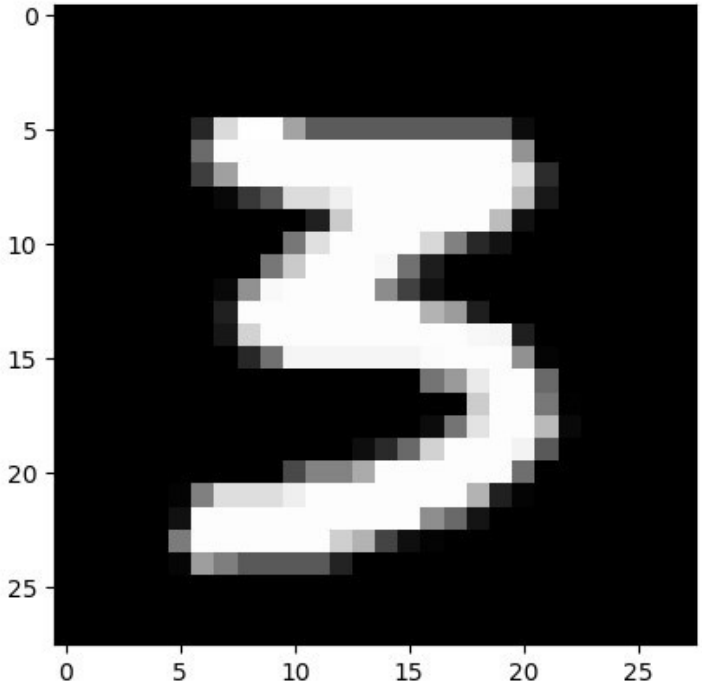


Figure 6.11: Medoid representative of class 3

6.1.5. Digit 4

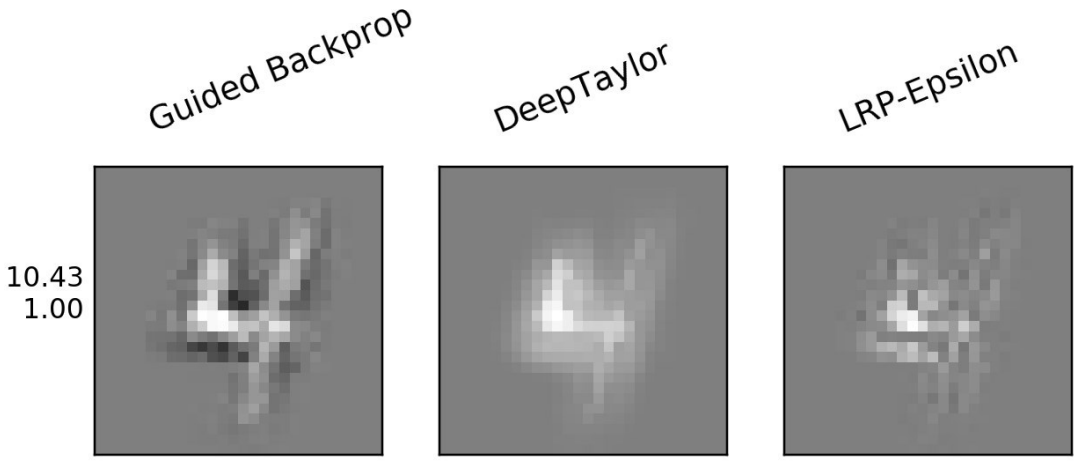


Figure 6.12: Saliency maps for medoid representative of class 4.

Figure 6.12 and Figure 6.13 present the saliency maps and medoid of digit 4 respectively. As shown in Figure 6.12, the Deep Taylor method focuses on the vertical line at the left part of the digit, while the LRP has focused on the horizontal line, while Guided Backpropagation combines these two lines together. Note that the right horizontal line is not considered as the important part of the image. This perhaps is how the network discriminates between digit 1 and digit 4. However, if we only focus on these two lines and their connection point, the network can mis-classify digit 5 (discussed in the next sub-section) and digit 4, as we can see a vertical and horizontal line connecting to each other at the middle left part of the image in digit 5 too. Here, the Guided backpropagation is showing some parts of right vertical line as intermediate important pixels. The Deep Taylor method also uses the vertical right line. But the LRP method has focused on the neighborhood of the interconnection point. The logit value for this medoid is 10.43, meaning the probability of class "4" is 1.0.

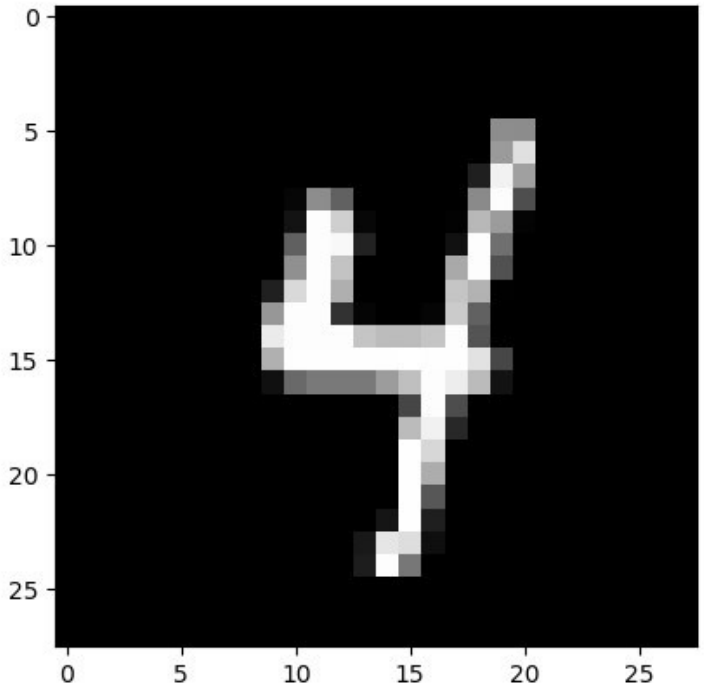


Figure 6.13: Medoid representative of class 4

6.1.6. Digit 5

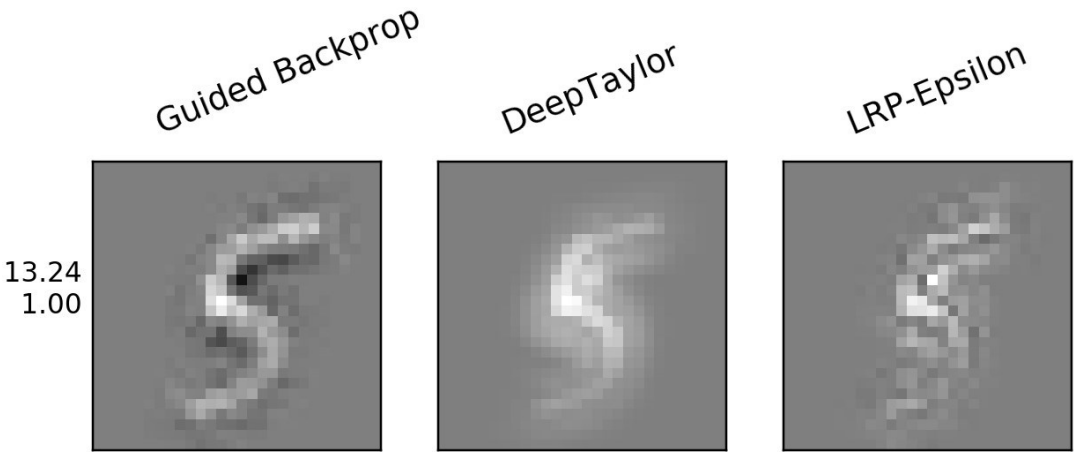


Figure 6.14: Saliency maps for medoid representative of class 5.

Regarding digit 5, as shown in Figure 6.14, all of the methods have very similar results, indicating that the network uses the top part of digit 5 for the classification. Note that the methods have been also extracted edges, black pixels in Guided Back-Propagation. As shown in Figure 6.14, it seems that the single most important pixel is the right edge of the right-concave curve at the top of the "5" (black pixel in Guided backpropagation, white in LRP), closely followed by the inflection point between the two curves. Let's also consider how Digit 3 could be misclassified as digit 5. As shown in the Figure 6.10, the top left tail is more important than the bottom left tail. It seems that the network has learned to determine the difference between digit 3 and digit 5 with those pixels in the top left part. The logit value for this medoid is 13.24, again giving a probability of 1.0 for class "5."

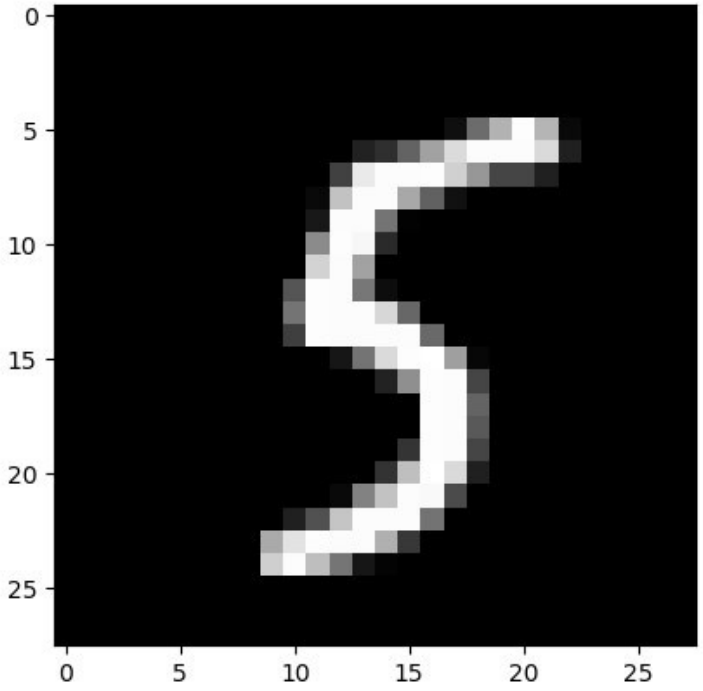


Figure 6.15: Medoid representative of class 5

6.1.7. Digit 6

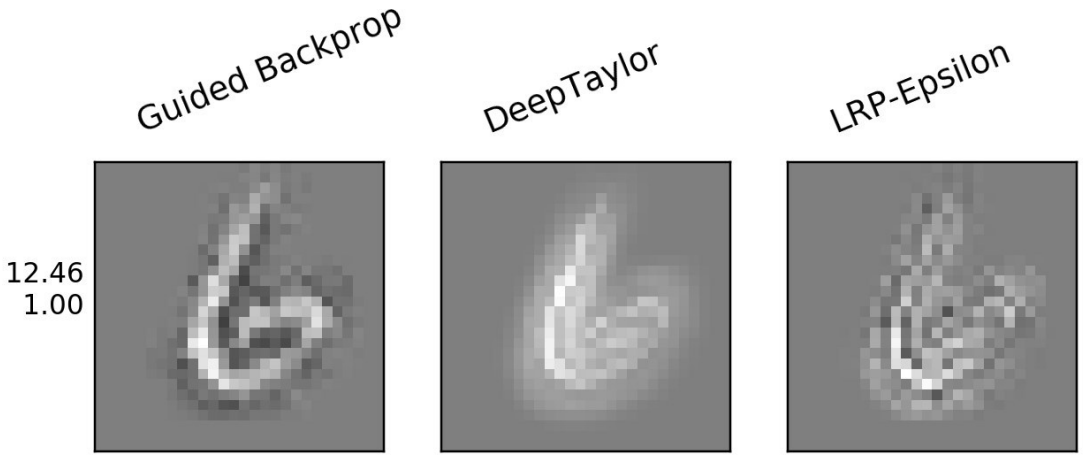


Figure 6.16: Saliency maps for medoid representative of class 6.

Figure 6.16 presents the saliency maps for digits 6, and Figure 6.17 the medoid image. Again, the three methods are very similar to each other. The only difference is their focus. Guided Backpropagation and Deep Taylor focus on finding the left line in the image, while LRP focuses more on the curved part. For this digit the logit value is 12.46, and so the probability of class “6” is 1.0.

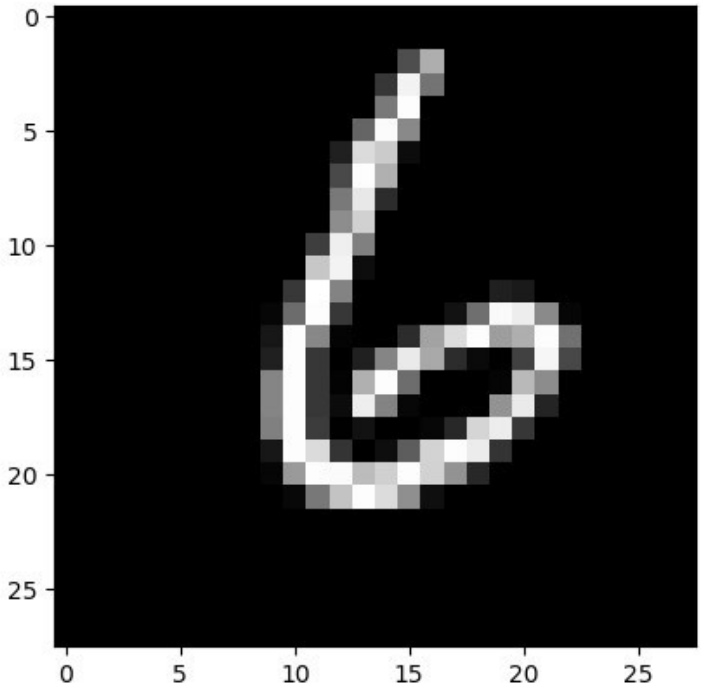


Figure 6.17: Medoid representative of class 6

6.1.8. Digit 7

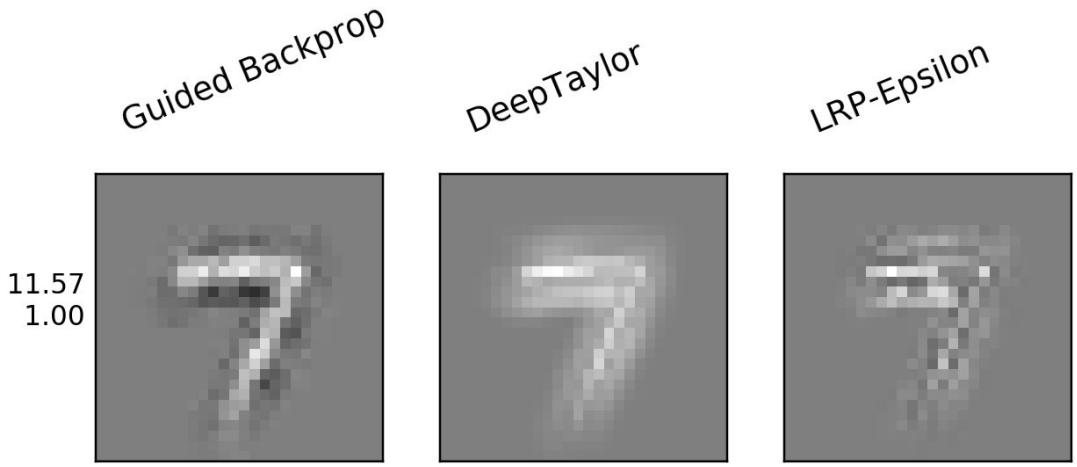


Figure 6.18: Saliency maps for medoid representative of class 7.

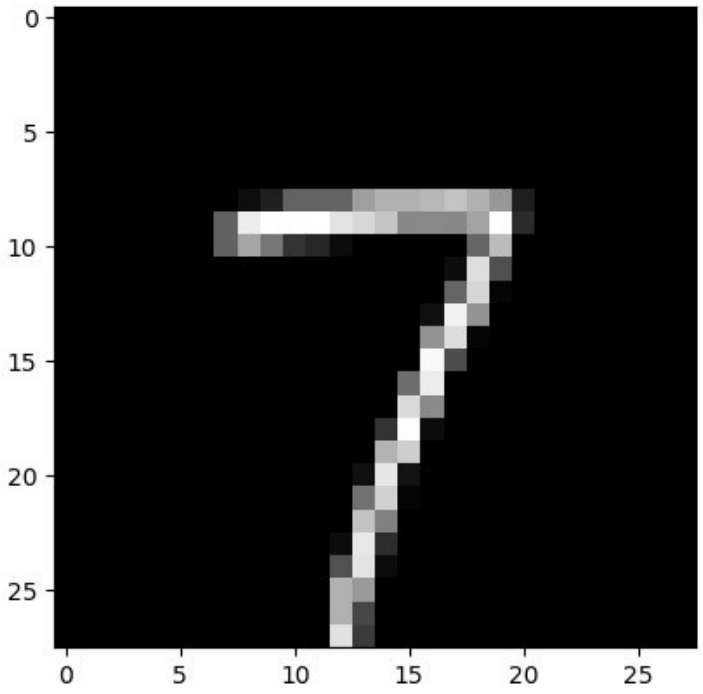


Figure 6.19: Medoid representative of class 7

Figure 6.18 presents our saliency maps for digit 7. All three methods are focusing on the connection between an approximately horizontal line and a straight line from top right to bottom left. Note that the edges are important for in the horizontal line. Thus, horizontal line not only seems whiter but it also includes important edge pixels. We also see that the "horizontal" line actually seems to be two line segments, offset from one another. Obviously this is a digitization artifact in the image (see Figure 6.19). The logit value is 11.57, giving a probability of 1.0 for class "7."

It is now instructive to consider a misclassified digit. Figure 6.20 is another sample from digit 7. This sample has been predicted as class "2," while belonging to class "7." This sample has a horizontal stroke in the middle of the descending line of the "7." Comparing Figure 6.8 and Figure 6.18, this is likely being confused with the lower horizontal tail in class "2." We will discuss this sample, and others, in greater detail in Section 6.2.

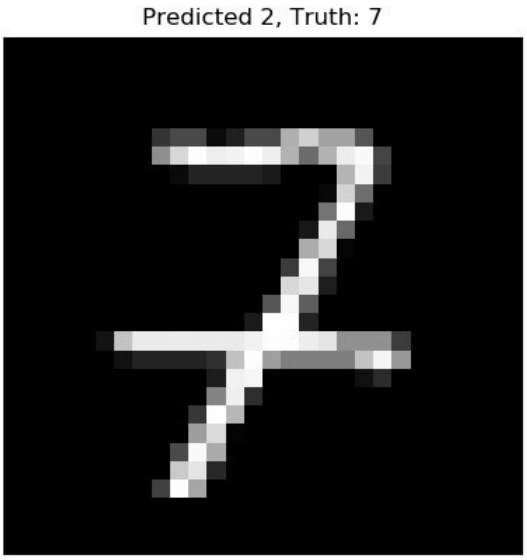


Figure 6.20: A sample of class 7

6.1.9. Digit 8

Figure 6.21 presents our saliency maps for digit 8, while the medoid image is presented in Figure 6.22. According to the methods (especially Guided Back-Propagation and LRP), two important regions are the

concave-right curves on the left side of the image, and their intersection at the middle of the digit. In addition, the empty regions within the two loops of the "8" are also seen as important – although the left half of those regions appear more so.

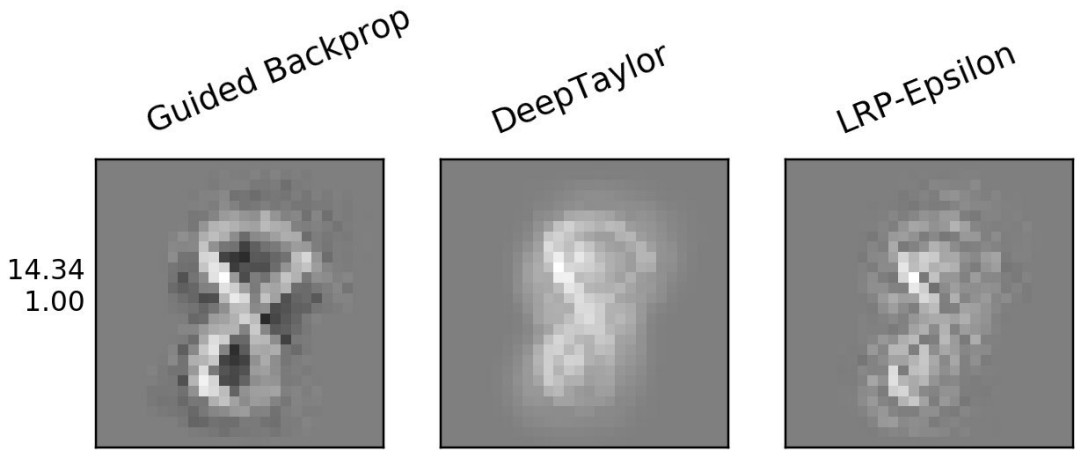


Figure 6.21: Saliency maps for medoid representative of class 8.

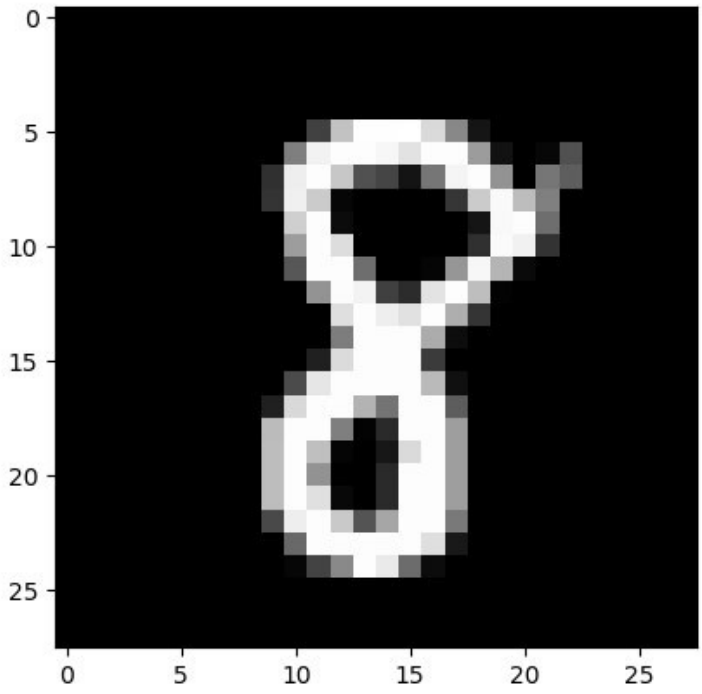


Figure 6.22: Medoid representative of class 8

In general, the network appears to consider the left half substantially more important than the right. Let us further consider this observation, in light of the saliency maps for other digits. The right-hand side of digit 8 includes two curved lines. The top right curve is common in digit 8, digit 2, digit 3, digit 7, digit 9, and maybe digit 6, 1, and 4. The bottom right curve is also common in digit 8, digit 3, digit 5, and digit 6. However, the top left part is only common between digit 8, digit 9, and maybe 5. The bottom left part is common between digit 6 and digit 8. It thus seems that the left-hand side of digit 8 is more discriminative, and that is what the network has picked up on. The logit value of the medoid is 14.34, indicating a probability of 1.0 for class "8."

6.1.10. Digit 9

Figure 6.23 presents the saliency maps for digit 9, and Figure 6.24 the medoid image. In contradiction with other digits, this medoid has been classified with logit value of 7.02, and probability of 0.88, indicating the CNN was less certain about this image than the other medoids (although it is still the highest probabilitiy) For this digit, Guided Back-Propagation focuses on a diagonal ellipse with an empty center. Deep Taylor and LRP also identify this shape, but they focus more on the left side of the image. Note that all three methods consider the horizontal line placed at the middle of image significantly important. This line appears to be the major difference between 8 and 9 in the top part of the image.

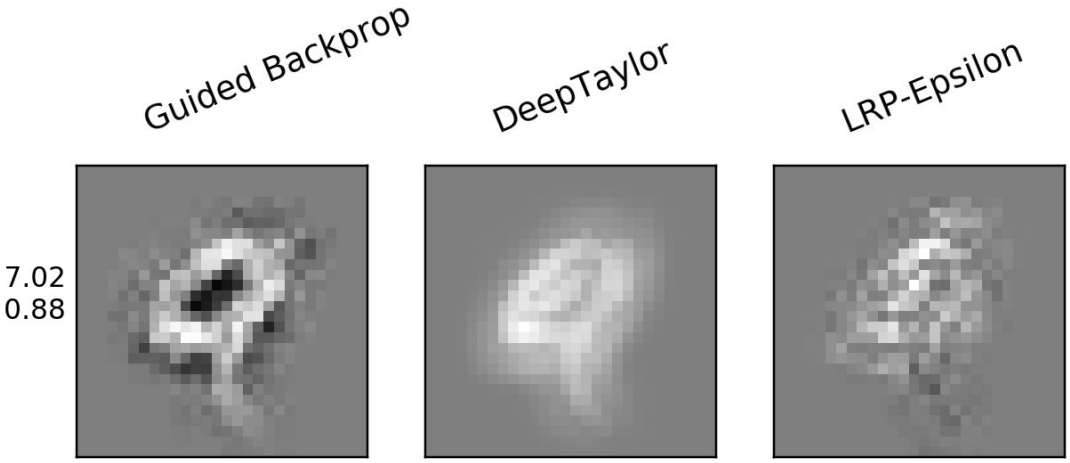


Figure 6.23: Results of interpretation techniques for medoid representative of class 9.

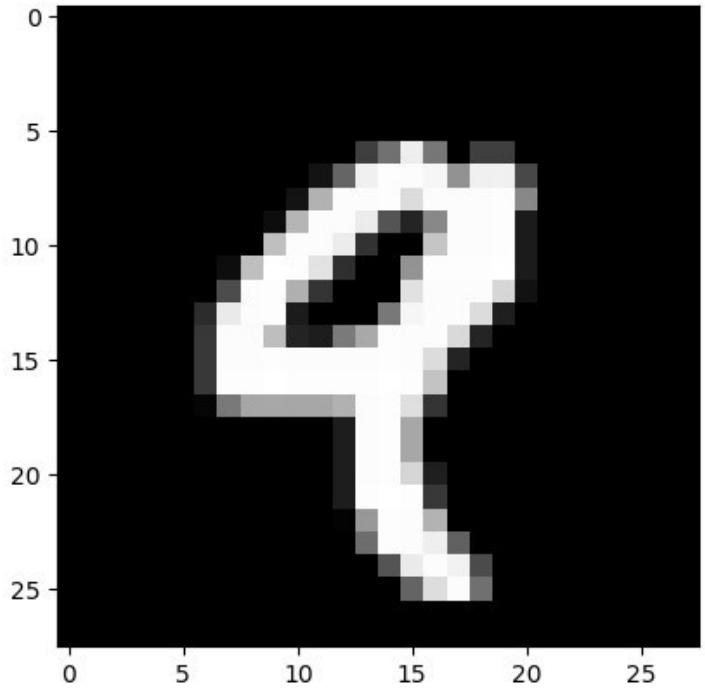


Figure 6.24: Medoid representative of class 9

6.1.11. Evidence from the CNN

In order to reach a deeper understanding of network's behavior, we can use the logit values of the softmax layer, shown in Table 4. In this table, rows represent a medoid, and columns represent an output neuron. Some cells in Table 4 has been colored as Green, Lime, Magenta, and gray. Green shows the highest logit value for each digit; unsurprisingly, these are the correct classifications. The lime cells have the highest logit after the green cells. We will discuss these results as follows:

- Digit 6 is the second-most most similar digit 0 after digit 0, but the gap is immense. The softmax function is based on a ratio between exponential functions, in which logit values are the exponents. Working from Table 4, $logit = 23 \rightarrow e^{23} = 9744803446.25$, $logit = 5.68 \rightarrow e^{5.68} = 292.949429923 \rightarrow \frac{9744803446.25}{9744803446.25+292.94942} > 0.999$. The contribution of the other digits to the denominator sum is negligible.
- Digit 8 is the most similar label to the representative of 1 after digit 1. Again, the difference in probabilities is extremely large.

Table 4: Confusion matrix of logit values for medoid representatives.

\ neuron	0	1	2	3	4	5	6	7	8	9
medoid 0	23	-3.36	3.67	0.83	-6.17	3.95	5.68	-1.2	0.95	1.25
medoid 1	-0.77	9.04	-1.73	-2.3	-2.33	-2.43	0.63	1.47	2.76	-1.26
medoid 2	1.29	2.77	12.84	4.54	-1.95	-6.81	-5.36	2.89	3.31	-1.33
medoid 3	0.4	-2.9	2.98	17.81	-5.58	10.97	-4.73	-0.19	0.18	-0.43
medoid 4	-5.79	-0.28	-2.67	1.85	10.43	1.63	-1.4	2.57	1.09	2.65
medoid 5	-1.52	-2.8	-2.82	5.74	-2.42	13.24	0.47	-3.02	3.57	2.61
medoid 6	2.69	-0.6	1.22	-0.78	0.3	-0.04	12.46	-5.6	-1.06	-5.9
medoid 7	1.44	2.11	3.87	2.79	-3.39	-1.43	-7.55	11.57	-0.56	3.76
medoid 8	0.61	0.24	3.28	2.93	-4.91	2.23	-2.2	-4.52	14.34	-0.08
medoid 9	0.7	-0.32	0.77	1.26	2.97	-3.64	-3.73	4.25	3.95	7.02

- Digit 3 is the most similar label to the representative of 2 after digit 2 which looks logical, according to the interpretations described before.

- Digit 5 is the most similar label to the representative of 3 after digit 3 which looks logical, according to the interpretations described before.
- Digit 9 is the most similar label to the representative of 4 after digit 4 which looks logical. This is because in some cases people will write the vertical lines of digit 4 close to each other, or don't write the left line vertically. It's diagonal in lots of cases.
- Digit 3 is the most similar label to the representative of 5 after digit 5 which looks logical, according to the interpretations described before.
- Digit 0 is the most similar label to the representative of 6 after digit 6 which looks logical.
- Digit 2 is the most similar label to the representative of 7 after digit 7. As described in interpretation of Figure 6.18 and Figure 6.20, the network considers the top part of digit 7 as the most important part of it. Then, the most important part was the leg of this digit. The middle line in some examples like Figure 6.20 will be considered as the tail of digit 2. In Figure 6.8 we saw that the top part is almost same with digit 3, and the bottom part can be also seen as the middle part of digit 3. Thus, the network considers 2 as the most similar digit to medoid of 7, while it considers 3 as the most similar digit to medoid of 2. So, the matrix is not symmetric, although in some cases (e.g. 5 and 3) the network considers both of them similar to each other. But this might not happen always as described.
- Digit 2 is the most similar label to the representative of 8 after digit 8. Although a human could say 3 is the most similar digit to this medoid, we see that because the left side of this medoid is more important than its right side, the network considers 2 as the most similar digit after 8.
- We can see four colors in the row of digit 9.
    - The logit value for digit 4 is 2.97 $\rightarrow exp(2.97) = 19.49191 \rightarrow softmax = 0.01536$
    - The logit value for digit 7 is 4.25 $\rightarrow exp(4.25) = 70.10541 \rightarrow softmax = 0.05525$
    - The logit value for digit 8 is 3.95 $\rightarrow exp(3.95) = 51.93536 \rightarrow softmax = 0.040932$
    - The logit value for digit 9 is 7.02 $\rightarrow exp(7.02) = 1118.7866 \rightarrow softmax = 0.88177$

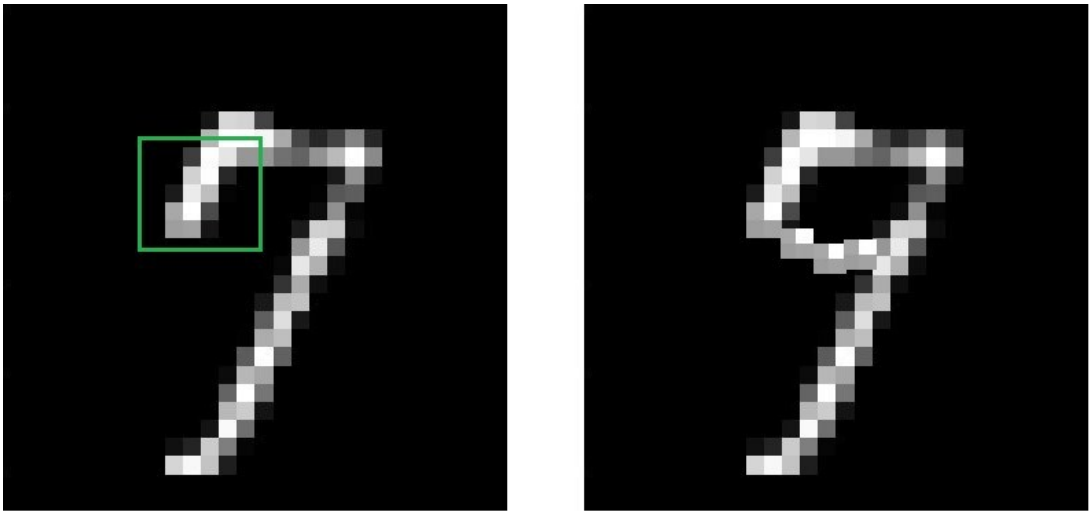


Figure 6.25: Left: A raw input for digit 7, Right: similarity of 7 to 9

As shown, the most similar digit is 7. Figure 6.25 is showing a raw input image of digit 7 on the left side. When then manually altered 12 pixels (out of 784) to produce the image on the right side – a recognizable digit 9. The network considers 7 as the most similar image to digit 9, as in the training inputs of digit 7, there are lots input images like Figure 6.25 which have the tail on left top part of the image (the green box). However, the softmax probabilities for Figure 6.24 are 5% digit 7, 4% digit 8, and about 2% digit 4. Because the network thinks the top side, especially the vertical line placed at the top right of Figure 6.24 is important, digit 4 has low probability here. As we described in Figure 6.12, the most important parts for digit 4 were placed on the left side, while Figure 6.23 shows the most important parts of digit 9 to be a circle including top and right curves.

6.2. Misclassified Examples

In the previous section we introduced medoid representatives of each digit and used our saliency maps to find the most important features for classifying each. These explanations should also help to explain the network's misclassifications; we will explore that question in this section. We will examine selected misclassifications, to see if our explanations in the previous section account for them as well. We will finally show the advantage of using our method in comparison with other methods.

6.2.1. Hard to recognize digits

As previously showed, there are some input samples which are hard to recognize or not understandable. Figure 6.3 shows some of these digits. These images are shown below again. Considering the saliency maps in the previous section, the predicted classes mostly make sense; the leftmost could be mistaken for a "3" if the lower loop was open to the left, the middle left has a strong vertical line in the center like a "1," and the rightmost does strongly resemble a "6." The right middle is the outlier, but it does intersect with the two regions of strong salience in Figure 6.10.

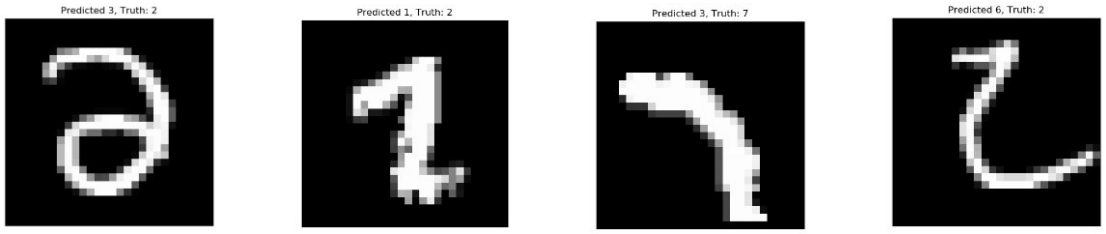


Figure 6.26: Some not understandable (hard to recognize) digits

6.2.2. Incomplete or Corrupted Digits

In some cases, the input digits are not written completely or are somehow corrupted. Figure 6.27 shows some examples of these digits. Let us again consider these in light of the saliency maps. For example, the left digit in Figure 6.27 is 1, while the network predicts it as 6. As discussed in Figure 6.16 the most important part for recognizing digit 6 is to find the curved line at the left part of image. We have showed this part in a yellow window. the right bottom gray pixels also make the network more confident that the input sample is digit 6 rather than digit 1. If we ask someone, he may also say it is a digit 6, and not 1. This input image seems to be corrupted. The digit next to it, is a digit 9 sample. However, the network classifies this digit as 7. In this image also, the input image is not complete. The digit has two important features of digit 7 which are showed in two yellow windows in the image (the horizontal line and leg of Figure 6.19 as discussed in Figure 6.18). The input has even the zigzag lines of digit 7. In addition, the digit doesn't have the important parts of recognizing digit 9 described in Figure 6.23, as two

important curves of the top circle are missing. Thus, it's completely explainable why network is classifying the sample as 7 instead of 9.

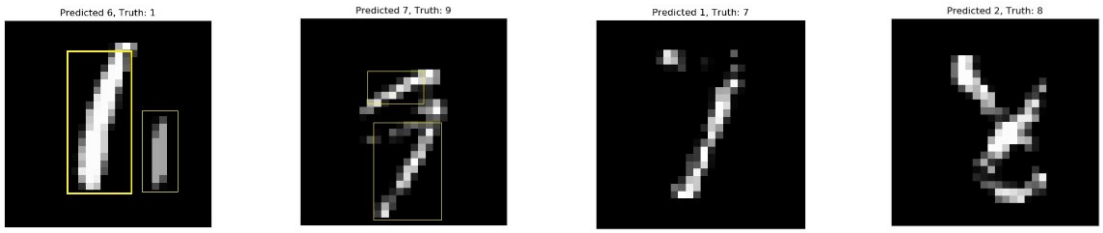


Figure 6.27: Not completely written digits and corrupted images

All these 4 images can be interpreted in the same wayOther misclassified digits can show interpretability of our system in more details. Thus, we will be discussing our results on some cases that are easy to recognize for human, but not for the trained network. This is another feature of our explanation system: it exposes some defects of the CNN model, and where it needs some improvements.

6.2.3. Misclassification process

In this section we will be interpreting some of the network's wrong decisions and the reasons behind them. Note that Deep CNN's system's accuracy is 97.87% which means 213 samples are misclassified. We cannot examine every misclassification in detail, and so will only discuss selected ones in this thesis. However, the interpretation procedure is similar for all digits and all misclassified samples. Fundamentally, we will first compute our three saliency maps, and then compare them with the saliency maps for the medoids of both the correct class and the erroneously predicted class. For some of the misclassifications, we will also superimpose the misclassified image over those two medoid images.

We first examine the digit in Figure 6.28; it was been classified as 2 while the ground truth is 7. Figure 6.29 presents our saliency maps for this image. The most important for the network is the middle of the image; this part of image seems more similar to digit 2 rather than digit 7. According to Figure 6.8, the network finds a horizontal tail for digit 2. However, for digit 7 (Figure 6.18) the most important parts are connecting a horizontal line (placed at top) to a straight line (from top right to bottom left). Thus, the network didn't learn to add the feature of having a horizontal line in the middle of image for digit 7.

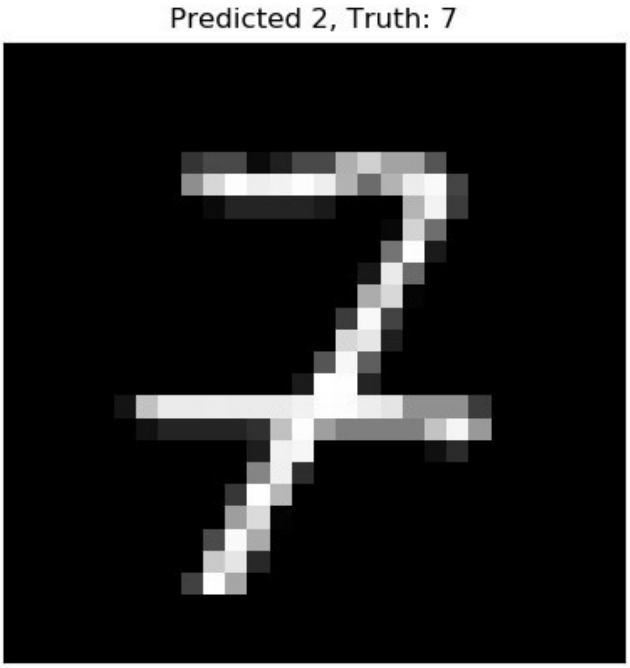


Figure 6.28: Misclassified Digit.

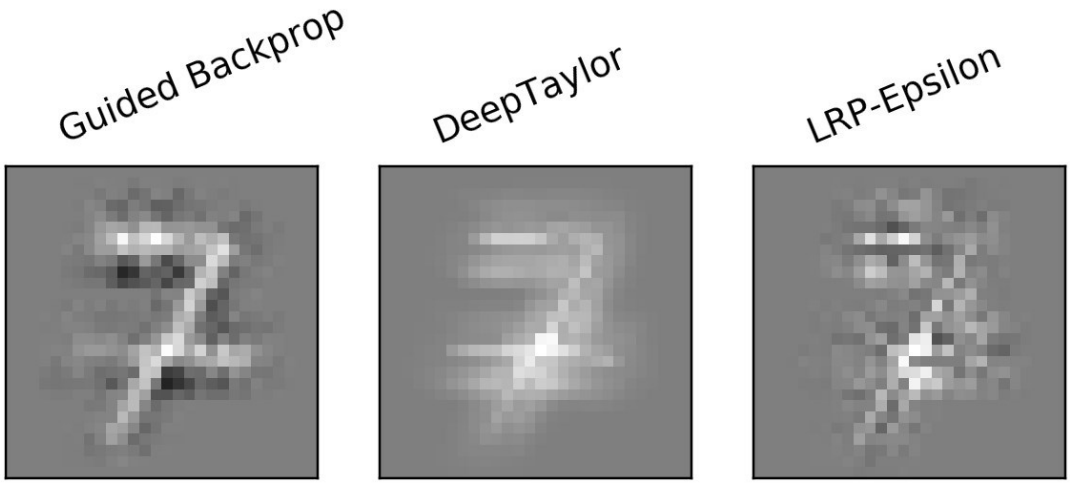


Figure 6.29: Saliency maps for Figure 6.28

Next, consider Figure 6.30, classified as 0 while the ground truth is 6. The saliency maps for it are presented in Figure 6.31, and show the network focusing on the bottom left part of the digit. However the

yellow parallelogram in Figure 6.30 represents the upper open curve of digit 6 from Figure 6.16, which is crucial for distinguishing 6 from 0 (form Table 4, neuron 0 does in fact have the second-highest logit value for the medoid of "6"). The small size of the highlighted upper curve in Figure 6.30, as well as its placement on the upper left of the image, seems to have confused the network, leaving only the central ellipse for the network to key in on; this is the defining feature of the "0" medoid in Figure 6.4. Finally, let us consider superimposing Figure 6.30 over Figures 6.5 and 6.17 (medoids of "0" and "6"). We use intensities of Figure 6.30, and assigned them exclusively to the red channel of a standard 24-bit RGB colour image; the green and blue channels are left null, and alpha-blending is set to make the image semi-transparent. This is then merged onto the existing greyscale images of Figure 6.5 and 6.17. The result is presented in Figure 6.32. Clearly, the misclassified sample intersects more with the "0" medoid than the "6" medoid.

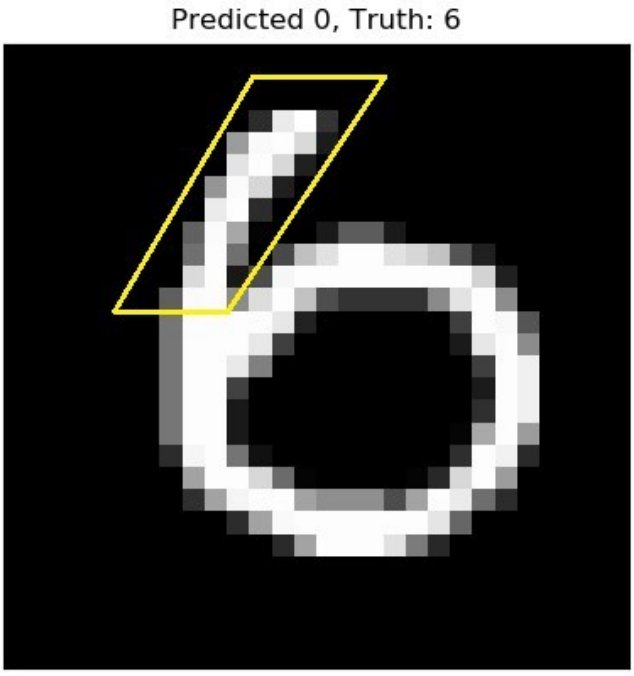


Figure 6.30: Misclassified Digit.

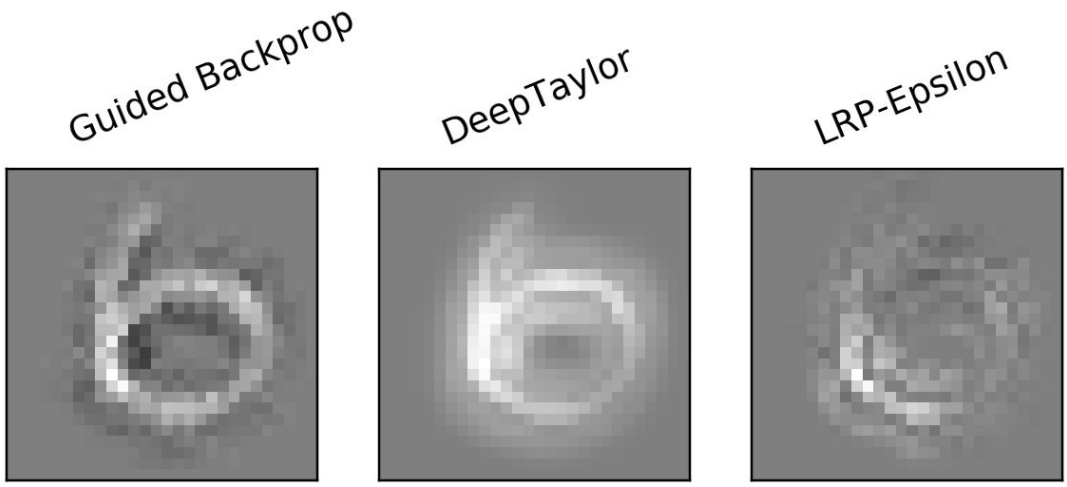


Figure 6.31: Saliency maps for Figure 6.30

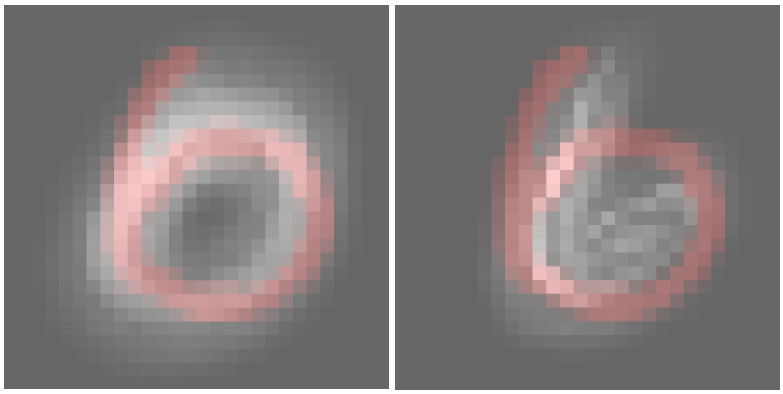


Figure 6.32: Results of superimposing Figure 6.30 on medoids of class 0 and 6.

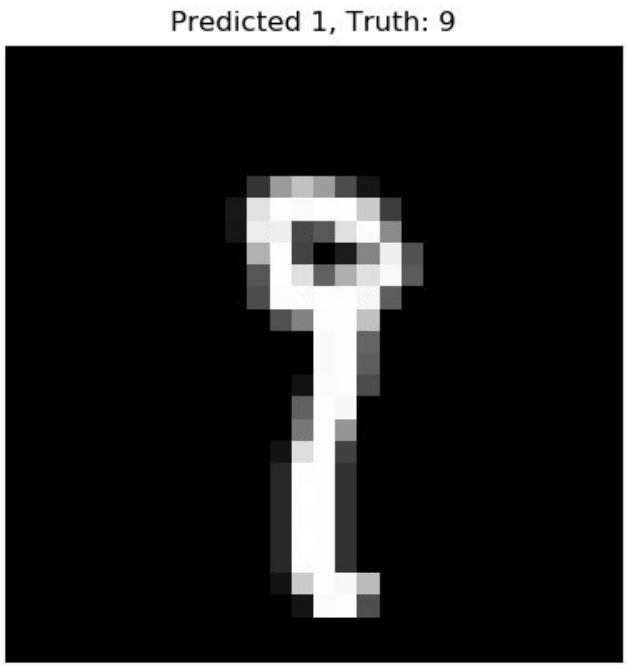


Figure 6.33: Misclassified Digit.

Figure 6.33 shows another misclassified sample. The written digit is a 9 that is misclassified as 1. Figure 6.34 presents the saliency maps for this digit. They are very similar to the maps for medoid "1" in Figure 6.6; a vertical line in the center of the image. Examining the logit values for the example, the next highest probabilities are for 7, 8, and 9, in that order. Considering those medoids, Figure 6.18 uses two straight lines for recognition of 7, and Figure 6.21 connects two curves at the left side of image. Figure 6.23 focuses on recognizing a big circle placed at top of the image and almost ignores the straight-line part. The input image Figure 6.33 shows some, but not all, key features of those three digits, and thus assigns these classes low probabilities. When we superimpose the image of Figure 6.33 on the medoids of "1" and "9," we obtain Figure 6.35, and see that the image does overlap better with the medoid of "1" than "9."

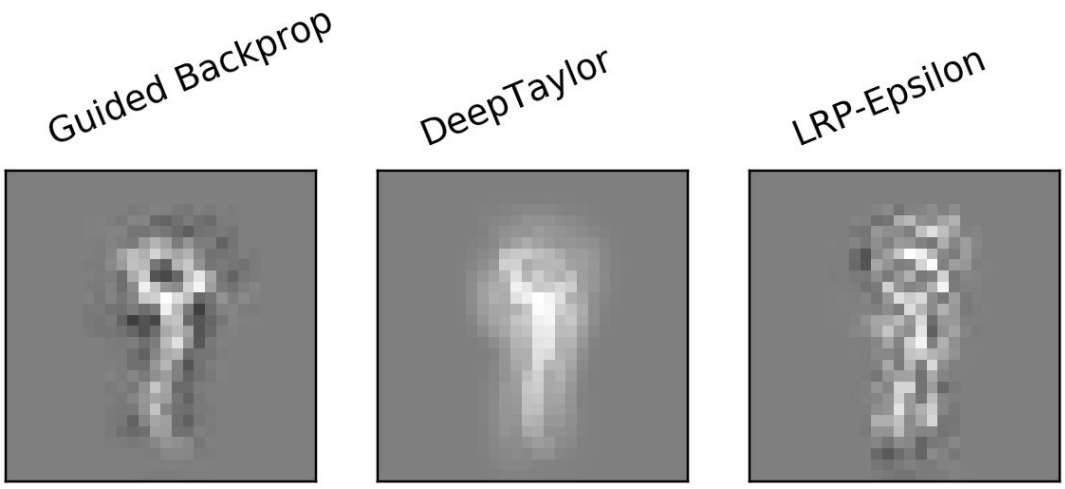


Figure 6.34: Saliency maps for Figure 6.33

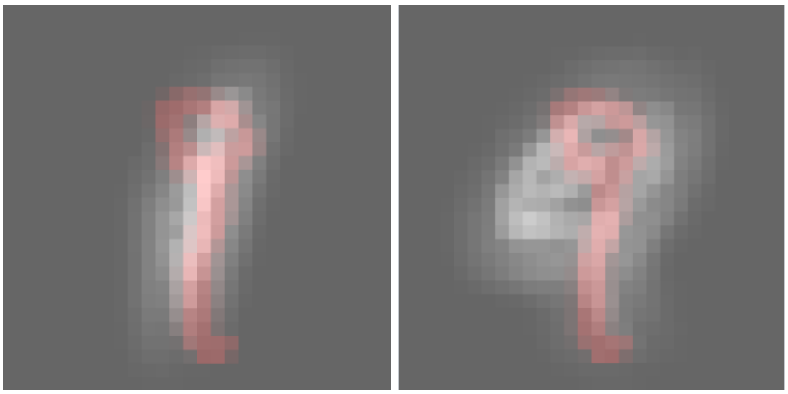


Figure 6.35: Results of superimposing Figure 6.33 on medoids of class 1 and 9.

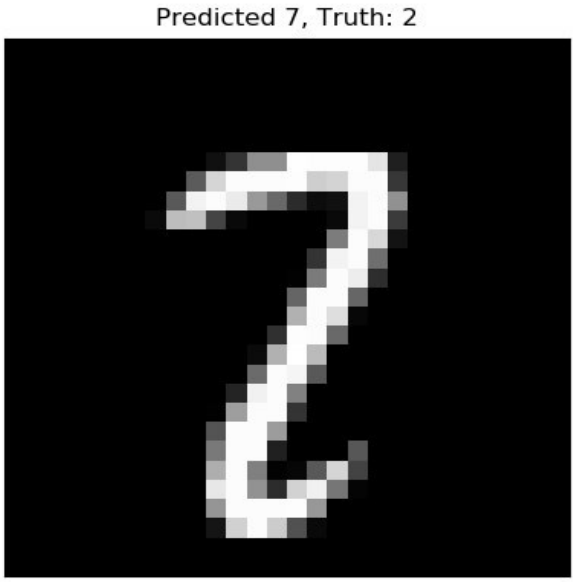


Figure 6.36: Misclassified Digit.

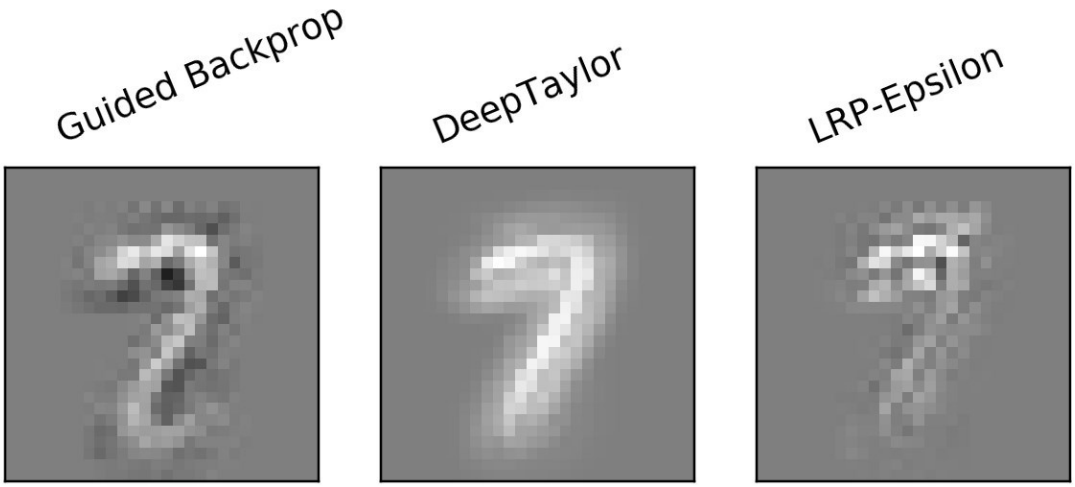


Figure 6.37: Interpretation techniques for Digit Figure 6.36

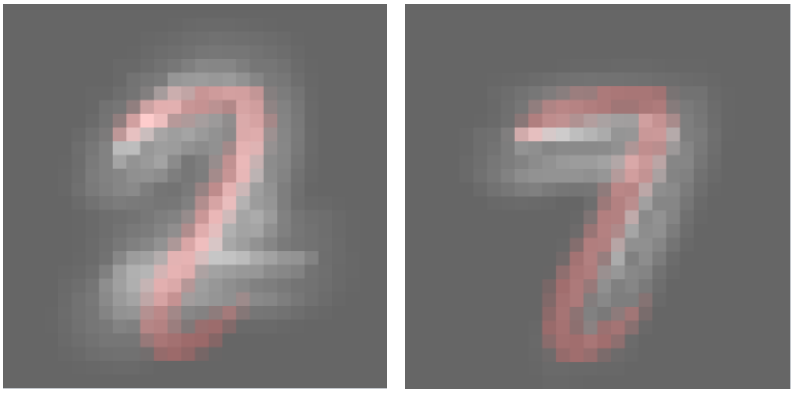


Figure 6.38: Results of superimposing Figure 6.36 on medoids of class 2 and 7.

Figures 6.36 to 6.38 present a digit 2 that has been misclassified as a 7. The saliency maps in Figure 6.37 resemble medoid 7 more than medoid 2 – in fact, the Deep Taylor and LRP maps are clearly not a recognizable "2" – we suspect the reader will agree these show a digit 7! Most critically, the crucial feature of a horizontal line at the bottom of the digit appears to be missing. The superimposition in Figure 6.38 also bears this out.

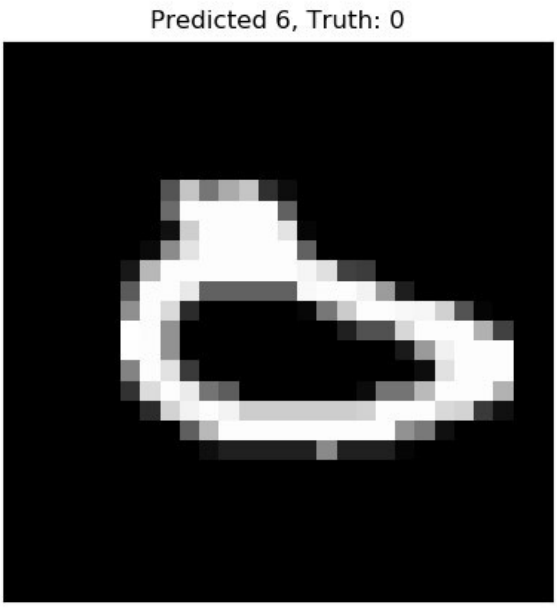


Figure 6.39: Misclassified Digit.

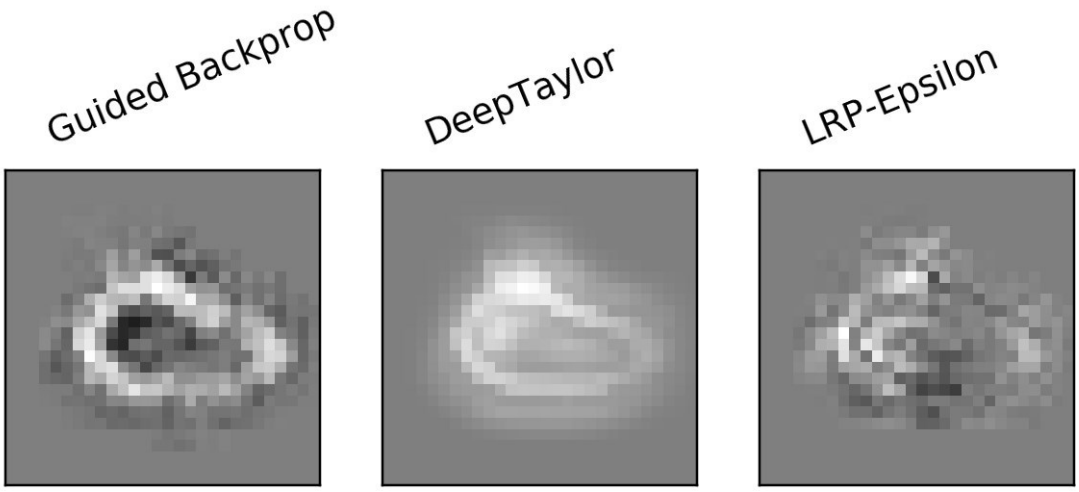


Figure 6.40: Saliency maps for Figure 6.39

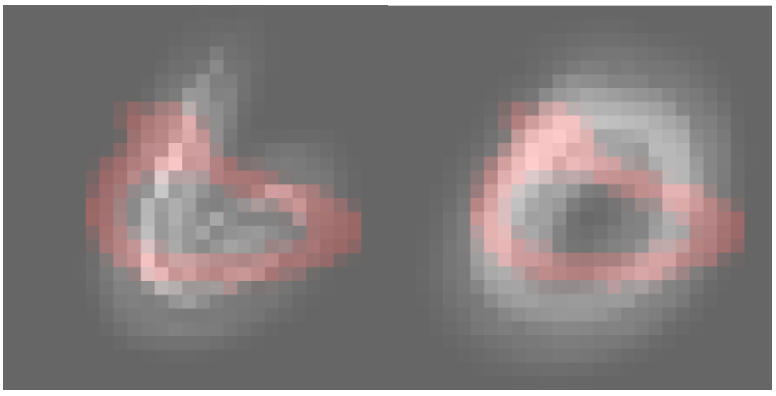


Figure 6.41: Results of superimposing Figure 6.39 on medoids of class 6 and 0.

Figure 6.39 – 6.41 present a digit 0 that is misclassified as 6. From the saliency maps, the fit to either 0 or 6 is not very good. However, the squashed shape of the left side of Figure 6.39 seems more similar to the left side of the "6" medoid. The superimposition of Figure 6.41 seems to support this contention.

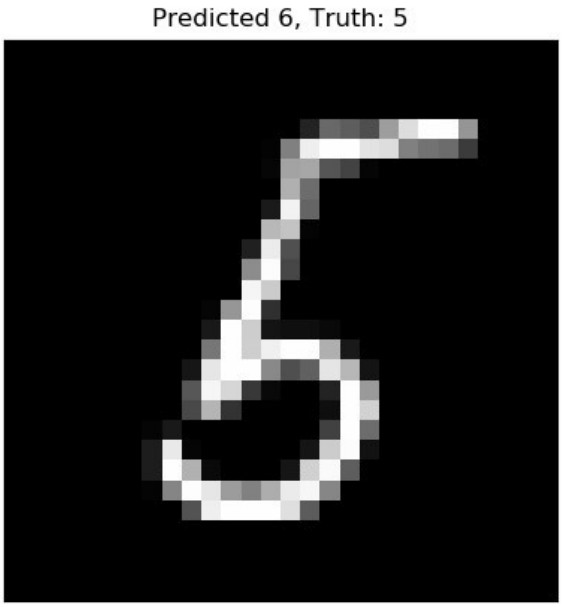


Figure 6.42: Misclassified Digit.

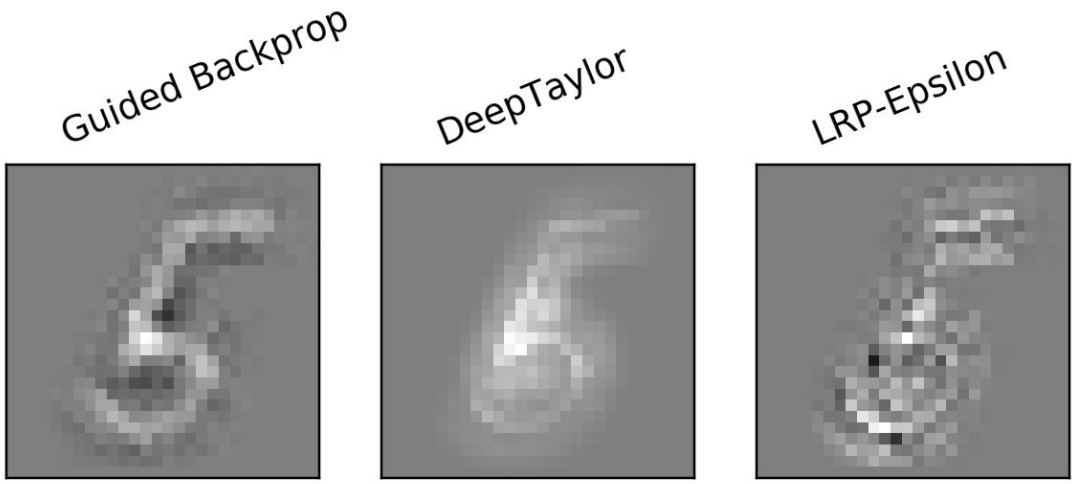


Figure 6.43: Interpretation techniques for Digit Figure 6.42

Figures 6.42 to 6.44 present a digit 5 that has been misclassified as a 6. The saliency maps in Figure 6.43 are quite similar to those of the "6" medoid, in particular the long curve at the top of the figure connecting to a bottom loop. Figure 6.44 confirms that the sample has covered the most important

features of medoid 6, specially the straight-line connecting bottom left to top right. However, this sample passes the important edges of digit 5. Note that edges (black pixels in Guided Back-Propagation) must be black and white pixels in those locations will have negative results on the classification.

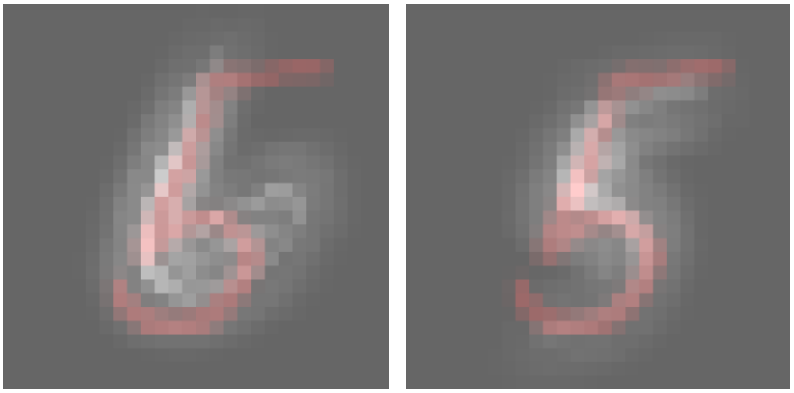


Figure 6.44: Results of superimposing Figure 6.42 on medoids of class 6 and 5.

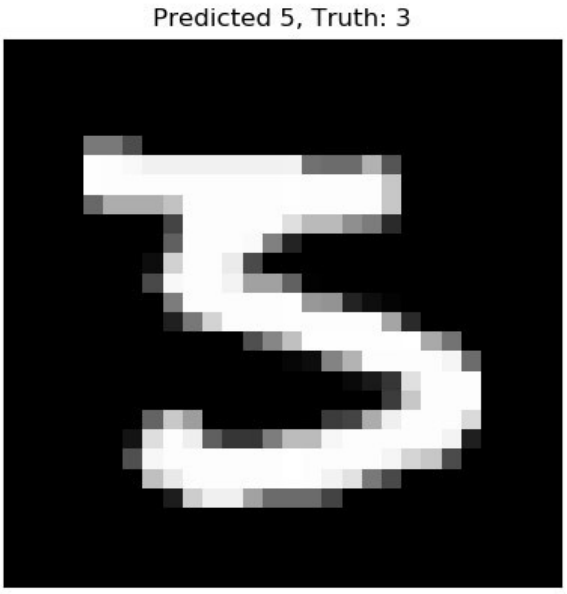


Figure 6.45: Misclassified Digit.

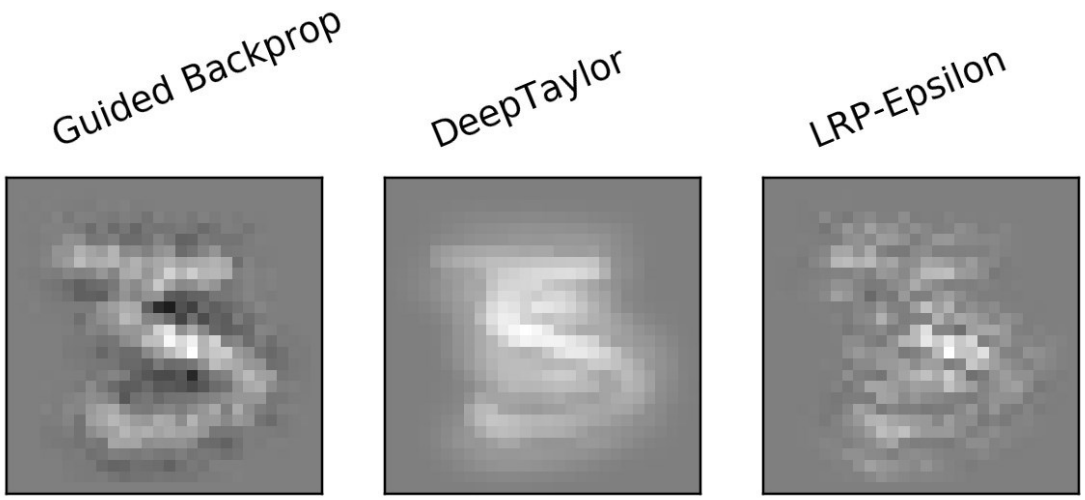


Figure 6.46: Interpretation techniques for Digit Figure 6.45

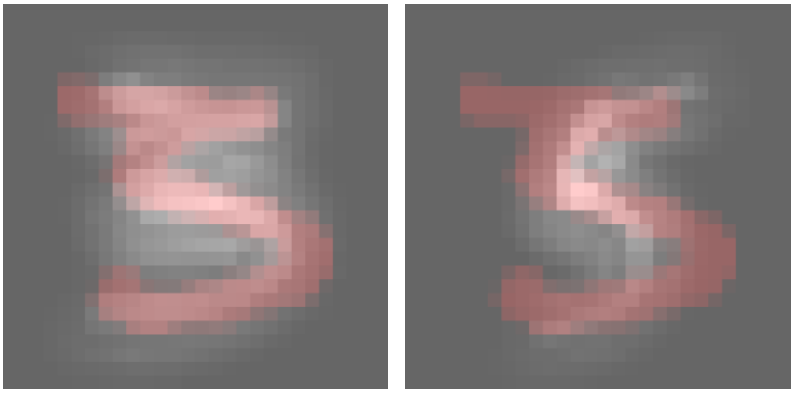


Figure 6.47: Results of superimposing Figure 6.45 on medoids of class 3, and 5.

Finally, consider Figures 6.45 to 6.47, showing a digit 3 misclassified as a 5. The saliency maps in Figure 6.446 do seem to resemble a 5 more than a 3; and in particular, the high-impact black pixels from Figure 6.14 match black pixels in this digit. Figure 6.47 shows that the raw input is covering both of the medoids reasonably well, but matches the highest-strength pixels better for medoid "5."

Chapter 7

Conclusion

In this thesis, we have designed and evaluated a novel deep fuzzy convolutional neural network. Our deep fuzzy system uses CNNs to automatically extract features from the input images, and then clusters the data in the derived feature space using the FCM and GK clustering algorithms. After hardening the clusters, we employ a fuzzy version of Rocchio's algorithm to classify the data points. We evaluated our system on three datasets (MNIST, Fashion MNIST, and CIFAR-10) and compared our results with other recently published results including state-of-the-art. We demonstrated there's a trade-off between interpretability and accuracy in the deep fuzzy system. We showed that the system is able to the medoid image of a class and map this representative backward to the original image space in order to interpret the network's decision-making process. Finally, we examined the network's decision-making process (describing logics behind both right and wrong classifications) using this explanation mechanism.

7.1. Future Directions

In the future, we intend to refine our accuracy on Fashion MNIST, and CIFAR-10 by changing the CNN feature extractor. We also aim to extend the classifier to regression and function approximation problems by replacing Rocchio's algorithm with other approaches. Another direction is to evaluate our system on larger computer vision datasets such as ImageNet, or datasets from other sub-fields of AI such as speech recognition, NLP, etc.

References

[1] G. Hinton, "Neural Networks for Machine Learning," [Online]. Available: https://www.coursera.org/learn/neural-networks.

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in Neural Information Processing Systems*, 2012, pp. 1097-1105.

[3] Farabet, Clement, Camille Couprie, Laurent Najman, and Yann LeCun. "Learning hierarchical features for scene labeling." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, no. 8, pp. 1915-1929, 2013.

[4] Tompson, Jonathan J., Arjun Jain, Yann LeCun, and Christoph Bregler. "Joint training of a convolutional network and a graphical model for human pose estimation." In *Advances in Neural Information Processing Systems*, 2014, pp. 1799-1807.

[5] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *IEEE conference on Computer Vision and Pattern Recognition*, 2015, pp. 1-9.

[6] Collobert, Ronan, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. "Natural language processing (almost) from scratch." *Journal of Machine Learning Research* 12, no. pp. 2493-2537, 2011.

[7] Bordes, Antoine, Sumit Chopra, and Jason Weston. "Question answering with subgraph embeddings." *arXiv preprint arXiv:1406.3676* (2014).

[8] Jean, Sébastien, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. "On using very large target vocabulary for neural machine translation." *arXiv preprint arXiv:1412.2007* (2014).

[9] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." In *Advances in Neural Information Processing Systems*, 2014, pp. 3104-3112.

[10] Mikolov, Tomáš, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černocký. "Strategies for training large scale neural network language models." In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011, pp. 196-201.

[11] Hinton, Geoffrey, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." *IEEE Signal Processing Magazine* 29, no. 6, pp. 82-97, 2012.

[12] Sainath, Tara N., Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. "Deep convolutional neural networks for LVCSR." In *IEEE international conference on Acoustics, speech and signal processing (ICASSP)*, 2013, pp. 8614-8618.

[13] Ma, Junshui, Robert P. Sheridan, Andy Liaw, George E. Dahl, and Vladimir Svetnik. "Deep neural nets as a method for quantitative structure–activity relationships." *Journal of Chemical Information and Modeling* 55, no. 2, pp. 263-274, 2015.

[14] Ciodaro, T., D. Deva, J. M. De Seixas, and D. Damazio. "Online particle detection with neural networks based on topological calorimetry information." In *Journal of physics:*

conference series, vol. 368, no. 1, p. 012030, 2012.

[15] Kaggle., "Higgs boson machine learning challenge," [Online]. Available: https://www.kaggle.com/c/higgs-boson.

[16] Dechter, Rina. "Learning while searching in constraint-satisfaction problems". In *Proceedings of the 5th national conference on Artificial Intelligence,* 1986, pp. 178-185.

[17] Aizenberg, Igor, Naum N. Aizenberg, and Joos PL Vandewalle. *Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications*. Springer Science & Business Media, 2013.

[18] Lee, Chang-Shing, Mei-Hui Wang, Shi-Jim Yen, Ting-Han Wei, I. Wu, Ping-Chiang Chou, Chun-Hsun Chou, Ming-Wan Wang, and Tai-Hsiung Yang. "Human vs. computer Go: Review and prospect." *arXiv preprint arXiv:1606.02032* (2016).

[19] K. Kam, "Nvidia Has Doubled Since March, But Ray Meyers Is Not Selling," [Online]. Available: https://www.forbes.com/sites/kenkam/2017/11/10/nvidia-has-doubled-since-march-but-ray-meyers-is-not-selling/#642cf6802b44*, 2017.*

[20] www.forbes.com, "Market Capitalization of NVIDIA," [Online]. Available: https://www.forbes.com/companies/nvidia/.

[21] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature 521*, no. 7553. pp. 436, 2015.

[22] Sarikaya, Ruhi, Paul A. Crook, Alex Marin, Minwoo Jeong, Jean-Philippe Robichaud, Asli Celikyilmaz, Young-Bum Kim et al. "An overview of end-to-end language understanding and dialog management for personal digital assistants." In *IEEE Workshop on Spoken Language Technology Workshop (SLT)*, 2016, pp. 391-397.

[23] Haykin, Simon, *Neural networks and learning machines*. 3rd ed. Upper Saddle River: New Jersey Pearson, 2009.

[24] Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. "Why should I trust you?: Explaining the predictions of any classifier." In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, 2016, pp. 1135-1144.

[25] Baehrens, David, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. "How to explain individual classification decisions." *Journal of Machine Learning Research* 11, no. pp. 1803-1831, 2010.

[26] Tickle, Alan B., Robert Andrews, Mostefa Golea, and Joachim Diederich. "The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks." *IEEE Transactions on Neural Networks* 9, no. 6. pp. 1057-1068, 1998.

[27] Bologna, Guido, and Yoichi Hayashi. "A rule extraction study on a neural network trained by deep learning." In *International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 668-675.

[28] Zilke, Jan Ruben, Eneldo Loza Mencía, and Frederik Janssen. "Deepred–rule extraction from deep neural networks." In *International Conference on Discovery Science*, 2016, pp. 457-473.

[29] Yosinski, Jason, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. "Understanding neural networks through deep visualization." *arXiv preprint arXiv:1506.06579* (2015).

[30] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." In *European conference on Computer Vision*, 2014, pp. 818-833.

[31] Zintgraf, Luisa M., Taco S. Cohen, Tameem Adel, and Max Welling. "Visualizing deep

neural network decisions: Prediction difference analysis." *arXiv preprint arXiv:1702.04595* (2017).

[32] Pal, Sankar K. "Multilayer Perceptron, Fuzzy Sets, and Classification." *IEEE Transactions on Neural Networks* 3, no. 5, pp. 683, 1992.

[33] Jang, J-SR. "ANFIS: adaptive-network-based fuzzy inference system." *IEEE Transactions on Systems, Man, and Cybernetics* 23, no. 3, pp. 665-685, 1993.

[34] Jang, Jyh-Shing Roger, Chuen-Tsai Sun, and Eiji Mizutani. "Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence [Book Review]." *IEEE Transactions on Automatic Control* 42, no. 10, pp. 1482-1484, 1997.

[35] Aviles, Angelica I., Samar M. Alsaleh, Eduard Montseny, Pilar Sobrevilla, and Alicia Casals. "A Deep-Neuro-Fuzzy approach for estimating the interaction forces in Robotic surgery." In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016, pp. 1113-1119.

[36] Chen, CL Philip, Chun-Yang Zhang, Long Chen, and Min Gan. "Fuzzy restricted Boltzmann machine for the enhancement of deep learning." *IEEE Transactions on Fuzzy Systems* 23, no. 6, 2015, pp. 2163-2173.

[37] Zheng, Yu-Jun, Wei-Guo Sheng, Xing-Ming Sun, and Sheng-Yong Chen. "Airline passenger profiling based on fuzzy deep machine learning." *IEEE Transactions on Neural Networks and Learning Systems 28*, no. 12, pp. 2911-2923, 2017.

[38] Shukla, Amit K., Taniya Seth, and Pranab K. Muhuri. "Interval type-2 fuzzy sets for enhanced learning in deep belief networks." In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017, pp. 1-6.

[39] Zheng, Yu-Jun, Sheng-Yong Chen, Yu Xue, and Jin-Yun Xue. "A Pythagorean-type fuzzy deep denoising autoencoder for industrial accident early warning." *IEEE Transactions on Fuzzy Systems* 25, no. 6, pp. 1561-1575, 2017.

[40] Zhou, Ta, Fu-Lai Chung, and Shitong Wang. "Deep TSK fuzzy classifier with stacked generalization and triplely concise interpretability guarantee for large data." *IEEE Transactions on Fuzzy Systems* 25, no. 5, pp.1207-1221, 2017.

[41] Rajurkar, Shreedharkumar, and Nishchal Kumar Verma. "Developing deep fuzzy network with takagi sugeno fuzzy inference system." In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017, pp. 1-6.

[42] Tan, Wei Ren, Chee Seng Chan, Hernán E. Aguirre, and Kiyoshi Tanaka. "Fuzzy qualitative deep compression network." *Neurocomputing 251*, pp. 1-15, 2017.

[43] John, Vijay, Seiichi Mita, Zheng Liu, and Bin Qi. "Pedestrian detection in thermal images using adaptive fuzzy C-means clustering and convolutional neural networks." In *14th IAPR International Conference on Machine Vision Applications (MVA)*, 2015, pp. 246-249.

[44] De la Rosa, Erick, and Wen Yu. "Data-Driven Fuzzy Modeling Using Deep Learning." *arXiv preprint arXiv:1702.07076* (2017).

[45] Bezdek, James Christian. "Fuzzy mathematics in pattern classification." *Ph. D. Dissertation, Applied Mathematics, Cornell University* (1973).

[46] Gustafson, Donald E., and William C. Kessel. "Fuzzy clustering with a fuzzy covariance matrix." In *IEEE Conference on Decision and Control including the 17th Symposium on Adaptive Processes,* 1979, pp. 761-766.

[47] Rocchio, Joseph John. “The SMART retrieval system: experiments in automatic document processing”, in *Relevance feedback in information retrieval,*Prentice-Hall Inc., New Jersey.

(1971): pp. 313-323.

[48] Keller, James M., Michael R. Gray, and James A. Givens. "A fuzzy k-nearest neighbor algorithm." *IEEE Transactions on Systems, Man, and Cybernetics* 4, pp. 580-585, 1985.

[49] "Dataset, MNIST," [Online]. Available: http://yann.lecun.com/exdb/mnist/.

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

[51] Krizhevsky, A. "Learning Multiple Layers of Features from Tiny Images." *Master's thesis, University of Tronto (2009)*.

[52] Intel, "Nervana," [Online]. Available: https://ai.intel.com/.

[53] Zhong, Guoqiang, Li-Na Wang, Xiao Ling, and Junyu Dong. "An overview on data representation learning: From traditional feature learning to recent deep learning." *The Journal of Finance and Data Science* 2, no. 4,pp. 265-278, 2016.

[54] Cover, Thomas M. "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition." *IEEE Transactions on Electronic Computers* 3, pp. 326-334, 1965.

[55] Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. Cambridge: MIT press, 2016.

[56] Cauchy, Augustin. "Méthode générale pour la résolution des systemes d'équations simultanées." *Comp. Rend. Sci. Paris* 25, no.pp. 536-538, 1847.

[57] Robbins, Herbert, and Sutton Monro. "A stochastic approximation method." In Herbert Robbins Selected Papers, pp. 102-109. Springer, New York, NY, 1985.

[58] Azevedo, Frederico AC, Ludmila RB Carvalho, Lea T. Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain." *Journal of Comparative Neurology* 513, no. 5, pp. 532-541, 2009.

[59] Gulati, Anil. "Understanding neurogenesis in the adult human brain." *Indian Journal of Pharmacology* 47, no. 6,pp. 583, 2015.

[60] M. A. Nielsen, "Neural Networks and Deep Learning," 2015. [Online]. Available: http://neuralnetworksanddeeplearning. com.

[61] LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." In *Proceedings of the IEEE* 86, no. 11, 1998, pp. 2278-2324.

[62] Chen, Teh-Chung, Torin Stepan, Scott Dick, and James Miller. "An investigation of implicit features in compression-based learning for comparing webpages." *Pattern Analysis and Applications* 19, no. 2, 2016, pp. 397-410.

[63] Yantis, Steven. *Sensation and perception*. Macmillan International Higher Education, 2013.

[64] Pearson, Karl. "LIII. On lines and planes of closest fit to systems of points in space." *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, no. 11, pp. 559-572, 1901.

[65] Stark, Henry, and John W. Woods. *Probability, random processes, and estimation theory for engineers*. Prentice-Hall, Inc., New Jersey, 1986.

[66] McInnes, Leland, and John Healy. "Umap: Uniform manifold approximation and projection for dimension reduction." *arXiv preprint arXiv:1802.03426* (2018).

[67] Zadeh, Lotfi A. "Fuzzy sets." *Information and control* 8, no. 3, pp. 338-353, 1965.

[68] H. Nurmi, "Probability and Fuzziness – Echoes from 30 Years Back," in *Views on fuzzy sets and systems from different perspectives: philosophy and logic, criticisms and applications*, Springer-Verlag Berlin Heidelberg, 2009, pp. 161-174.

[69] Machina, Kenton F. "Truth, belief, and vagueness." *Journal of Philosophical Logic* 5, no. 1, pp. 47-78, 1976.

[70] Evers, Frederick Thomas, Frank Höppner, Frank Klawonn, Rudolf Kruse, and Thomas Runkler. *Fuzzy cluster analysis: methods for classification, data analysis and image recognition*. John Wiley & Sons, Chichester ; New York, 1999.

[71] Stork, David G., Richard O. Duda, Peter E. Hart. "Pattern classification." 2nd ed., *Wiley-Interscience , New York ,* 2001.

[72] Klir, George, and Bo Yuan. *Fuzzy sets and fuzzy logic*. New Jersey: Prentice hall, 1995.

[73] Rojas, Raúl. *Neural networks: a systematic introduction*. Springer-Verlag Berlin Heidelberg, 1996.

[74] Bezdek, James C., and Richard J. Hathaway. "Some notes on alternating optimization." In *AFSS International Conference on Fuzzy Systems*, 2002, pp. 288-300.

[75] Gath, Isak, and Amir B. Geva. "Unsupervised optimal fuzzy clustering." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11, no. 7, pp. 773-780, 1989.

[76] Sadaaki, Miyamoto, Ichihashi Hidetomo, and Honda Katsuhiro, "Miscellanea.," in *Algorithms for fuzzy clustering: Methods in c-Means Clustering with Applications.*, Berlin: Springer, 2008, pp. 99-117.

[77] Yen, John, and Reza Langari. *Fuzzy logic: intelligence, control, and information*. Upper

Saddle River, NJ: Prentice Hall, 1999.

[78] Melin, Patricia, and Oscar Castillo. *Hybrid intelligent systems for pattern recognition using soft computing: An evolutionary approach for neural networks and fuzzy systems*. Springer-Verlag Berlin Heidelberg, 2005.

[79] Xie, Xuanli Lisa, and Gerardo Beni. "A validity measure for fuzzy clustering." *IEEE Transactions on Pattern Analysis & Machine Intelligence* 8, pp. 841-847, 1991.

[80] Windham, Michael P. "Cluster validity for fuzzy clustering algorithms." *Fuzzy Sets and Systems* 5, no. 2, pp. 177-185, 1981.

[81] Windham, Michael P. "Cluster validity for the Fuzzy c-means clustering algorithrm." *IEEE Transactions on Pattern Analysis & Machine Intelligence* 4, pp. 357-363, 1982.

[82] Davies, David L., and Donald W. Bouldin. "A cluster separation measure." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2, pp. 224-227, 1979.

[83] C. Dunn, Joseph. "A fuzzy relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". *Journal of Cybernetics and Systems*. vol 3, no. 3, pp. 32-57, 1973.

[84] Tsukamoto Y. An approach to reasoning method. Gupta M, Ragade RK, Yager RR, editors. *Advances in Fuzzy Set Theory and Applications*. Amsterdam: North-Holland, 1979.

[85] Lee, Chuen-Chien. "Fuzzy logic in control systems: fuzzy logic controller. I." *IEEE Transactions on Systems, Man, and Cybernetics* 20, no. 2, 404-418, 1990.

[86] Takagi, Tomohiro, and Michio Sugeno. "Derivation of fuzzy control rules from human operator's control actions." *IFAC Proceedings Volumes* 16, no. 13, pp. 55-60, 1983.

[87] Sugeno, Michio, and G. T. Kang. "Structure identification of fuzzy model." *Fuzzy Sets and Systems* 28, no. 1, pp. 15-33, 1988.

[88] Miller, Tim. "Explanation in artificial intelligence: Insights from the social sciences." *Artificial Intelligence 267,* 2018.

[89] Kim, Been, Rajiv Khanna, and Oluwasanmi O. Koyejo. "Examples are not enough, learn to criticize! criticism for interpretability." In *Advances in Neural Information Processing Systems*, 2016, pp. 2280-2288.

[90] Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser et al. "Mastering the game of Go with deep neural networks and tree search." *nature 529*, no. 7587, pp. 484, 2016.

[91] Caruana, Rich, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission." In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1721-1730.

[92] Samek, Wojciech, Thomas Wiegand, and Klaus-Robert Müller. "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models." *arXiv preprint arXiv:1708.08296* (2017).

[93] Hajian, Sara, Francesco Bonchi, and Carlos Castillo. "Algorithmic bias: From discrimination discovery to fairness-aware data mining." In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, 2016, pp. 2125-2126.

[94] Fayyad, Usama M., Gregory Piatetsky-Shapiro, and Padhraic Smyth. "Knowledge Discovery and Data Mining: Towards a Unifying Framework." In *Proceedings of AAAI*, pp.

82-88. 1996.

[95] Goodman, Bryce and Seth Flaxman. “European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation".” *AI Magazine 38*, pp. 50-57, 2017.

[96] Bach, Sebastian, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation." *PloS one* 10, no. 7, 2015.

[97] Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." *arXiv preprint arXiv:1312.6034* (2013).

[98] Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

[99] Zeiler, Matthew D., Graham W. Taylor, and Rob Fergus. "Adaptive deconvolutional networks for mid and high level feature learning." In IEEE International Conference on *Computer Vision (ICCV),* 2011, pp. 2018-2025.

[100] Bishop, Christopher M. "Pattern recognition and machine learning", 2nd ed, Cambridge, UK, 2006.

[101] Guyon, Isabelle, and André Elisseeff. "An introduction to variable and feature selection." *Journal of Machine Learning Research* 3, pp. 1157-1182, 2003.

[102] Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection." In *International Joint Conference on Artificial Intelligence, 1995,* pp. 1137-1145.

[103] Struyf, Anja, Mia Hubert, and Peter Rousseeuw. "Clustering in an object-oriented environment." *Journal of Statistical Software* 1, no. 4, pp. 1-30, 1997.

[104] Alber, Maximilian, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. "iNNvestigate neural networks!." *arXiv preprint arXiv:1808.04260* (2018).

[105] Montavon, Grégoire, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. "Explaining nonlinear classification decisions with deep taylor decomposition." *Pattern Recognition* 65, pp. 211-222, 2017.

[106] Zeiler, Matthew D. ; Dilip Krishnan; Graham W. Taylor; RobFergus. "Deconvolutional networks". *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010. pp. 2528-2535

[107] Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

[108] Xie, Lingxi, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. "Disturblabel: Regularizing cnn on the loss layer." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4753-4762. 2016.

[109] Chen, Xi, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets." In *Advances in Neural Information Processing Systems*, 2016, pp. 2172-2180.

[110] Xie, Junyuan, Ross Girshick, and Ali Farhadi. "Unsupervised deep embedding for clustering analysis." In *International Conference on Machine Learning*, 2016, pp. 478-487.

[111] Wan, Li, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. "Regularization of neural networks using dropconnect." In *International Conference on Machine Learning*, 2013, pp. 1058-1066.

[112] Diehl, Peter U., and Matthew Cook. "Unsupervised learning of digit recognition using spike-timing-dependent plasticity." *Frontiers in Computational Neuroscience* 9, pp. 99, 2015.

[113] Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580* (2012).

[114] Zhou, Yuqian, Kuangxiao Gu, and Thomas Huang. "Unsupervised Representation Adversarial Learning Network: from Reconstruction to Generation." *arXiv preprint arXiv:1804.07353* (2018).

[115] Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." In *Advances in Neural Information Processing Systems*, 2017, pp. 3856-3866.

[116] Zeng, Shaoning, Bob Zhang, and Jianping Gou. "Deep Collaborative Weight-based Classification*." arXiv preprint arXiv:1802.07589* (2018).

[117] Cubuk, Ekin D., Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. "AutoAugment: Learning Augmentation Policies from Data." *arXiv preprint arXiv:1805.09501* (2018).

[118] A. I. Google, "Google Brain Team," [Online]. Available: https://ai.google/research/teams/brain.

[119] Vinyals, Oriol, Yangqing Jia, Li Deng, and Trevor Darrell. "Learning with recursive perceptual representations." In *Advances in Neural Information Processing Systems*, 2012, pp. 2825-2833.

[120] Graham, B., 2014. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*.

[121] Mairal, Julien, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. "Convolutional kernel

networks." In *Advances in Neural Information Processing Systems*, 2014, pp. 2627-2635..