INTEGRATING PART LEARNING WITH STRUCTURE LEARNING FOR HANDWRITTEN NUMERAL RECOGNITION

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

INFORMATION SYSTEMS

Submitted By:

JATIN MALHOTRA

2K18/ISY/04

Under the supervision of

DR. SEBA SUSAN

Department Of Information Technology

Delhi Technological University

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

JULY 2020

CANDIDATE'S DECLARATION

I Jatin Malhotra (2K18/ISY/04), student of M.Tech. (Information Technology), hereby declare that the project Dissertation titled "Integrating Part Learning with Structure Learning for Handwritten Numeral Recognition", which is submitted by me to the Department of Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree Master of Technology is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Date : July 13, 2020 Jatin Malhotra

Place : New Delhi

CERTIFICATE

I hereby certify that the Project Dissertation titled "Integrating Part Learning with Structure Learning for Handwritten Numeral Recognition", which is submitted by Jatin Malhotra, Roll Number 2K18/ISY/04, Department of Information Technology, Delhi Technological University in partial fulfilment of the requirements for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: New Delhi

Date:

Dr. Seba Susan
SUPERVISOR
Associate Professor
Department of Information Technology
Delhi Technological University

ACKNOWLEDGEMENT

I express my gratitude to my major project guide Dr. Seba Susan, Associate Professor, Department of Information Technology, Delhi Technological University, for the valuable support and guidance she provided in making this major project. It is my pleasure to record my sincere thanks to my respected guide for her constructive criticism and insight without which the project would not have shaped as it has.

I would also like to express my gratitude to the Delhi Technological University for providing me with the environment which allowed me to work without any obstructions. I humbly extend my words of gratitude to other faculty members, lab assistants, seniors and my classmates for providing their valuable help and time whenever it was required.

JATIN MALHOTRA

Roll No. 2k18/ISY/04

M.Tech. (Information Systems)

E-mail: jmalhotra009@gmail.com

ABSTRACT

Handwritten Numeral Recognition is the task of correctly identifying handwritten digits. It has an important use-case in digitising old script, image restoration. The challenges while performing handwritten digits recognition arise due to the fact that every person's handwriting can differ. This difference in handwriting can be due to different fonts, slant in letters. Bad quality of text or image from which digits also creates difficulty in recognition. Handwritten Numeral Recognition has been an active area of research for decades but still using part-learning to solve this problem has hardly been explored. Part-learning is a technique in which the original image is divided into parts (called patches), and these patches are learned. Focusing on patches instead of whole images help identify patterns which are sometimes missed. It makes the identification process resistant to background noise. Structure learning paradigm in which we train neural networks by leveraging structured learned from the neural network itself. Structure learned in structure learning can be vectors, graphs or even finite state machines. In this work, we propose a novel approach of integrating part-learning with structure learning for handwritten numeral recognition. Convolutional Neural Network has been used extensively used in the vision-related tasks. We have tested our approach with a multilayer perceptron, convnets and autoencoders. Comparison of the performance of handwritten numeral recognition on MNIST dataset between state-of-the-art techniques and our proposed method indicate the efficacy of our approach.

Keywords: Handwritten Numeral Recognition, Structure Learning, Part Learning, Convolutional Neural Network, Autoencoders

CONTENTS

1.	**Candidate's Declaration**	ii
2.	**Certificate**	iii
3.	**Acknowledgement**	iv
4.	**Abstract**	v
5.	**Contents**	vi
6.	**List of Figures**	vii
7.	**List of Tables**	viii
8.	**Chapter 1: INTRODUCTION**	1
9.	**Chapter 2: LITERATURE REVIEW**	3
	2.1 Handwritten Numeral Recognition	3
	2.2 Artificial Neural Network	3
	2.3 Convolutional Neural Network	5
	2.4 Autoencoders	7
	2.4.1 Convolutional Autoencoders	8
	2.5 Weight Initialisation	9
	2.6 Learning from Image Parts	10
	2.7 Structure Learning	10
10.	**Chapter 3: PROPOSED WORK**	11
	3.1 CNN Pre-Initialisation by Part Learning	11
	3.1.1 CNN architecture	11
	3.1.2 Proposed Methodology	12
	3.2 Integrating Part-Learning with Structure Learning MLP	13
	3.2.1 MLP architecture	13
	3.2.2 Proposed Methodology	13
	3.3 Integrating Part-Learning with Structure Learning CAE	14
	3.3.1 Convolutional Auto-Encoders Architecture	14
	3.3.2 Proposed Methodology	15

11.	**Chapter 4: RESULTS**	16
	4.1 Experimental Setup	16
	4.2 CNN Pre-Initialisation Implementation Results	17
	4.3 Part-Learning and Structure Learning MLP implementation Results	22
	4.4 Part-Learning and Structure Learning CAE implementation Results	33
	4.4.1 Ablation Study	39
	4.5 Application	44
12.	**Chapter 5: CONCLUSION**	45
13.	**REFERENCES**	47
14.	**LIST OF PUBLICATIONS**	53

LIST OF FIGURES

Figure No.	Description	Page No.
Fig. 2.1	Different layers of a Neural Network	4
Fig. 2.2	CNN Layers	5
Fig. 2.3	Convolution operation	6
Fig. 2.4	Max Pooling	7
Fig. 2.5	The general structure of an autoencoder	7
Fig. 2.6	Different layers of an Autoencoder	8
Fig. 2.7	Convolutional Autoencoder	8
Fig. 2.8	Training an RNN model, clustering the RNN's hidden states and outputting the final structured FSA.	10
Fig. 3.1	CNN architecture for our experiment	11
Fig. 3.2	Proposed methodology block diagram	13
Fig. 3.3 a)	The layout of the five image patches each covering a quarter area of the image	15
Fig. 3.3 b)	The five patches shown for the numeral '6'	15
Fig. 4.1	Sample of MNIST dataset	16
Fig. 4.2	Images prepared for initialising CNN	17
Fig. 4.3	Hidden state cluster of MLP	22
Fig. 4.4	The kmeans Elbow graphs for determining the interpretable hidden layer. Shown for the fifteen image patch layouts [(a)-(o)]	25-29
Fig. 4.5	Images of Left, Right, Top and Bottom patch of numeral '5'	35
Fig. 4.6	Sample of Devanagari handwritten character dataset	44
Fig. 4.7	Five patches created from character 'bha'	44
Fig. 4.8	Character 'bha'	44

LIST OF TABLES

Table No.	Description	Page No.
Table 1	Ordering of layers in proposed CNN Architecture	12
Table 2	Ordering of layers in proposed CAE Architecture	15
Table 3	Result of 5% cropped &resized image approach (average)	17
Table 4	Result of 5% cropped &resized image approach (maximum)	18
Table 5	Result of 5% masked image approach (average)	18
Table 6	Result of 5% masked image approach (maximum)	19
Table 7	Result of 10% cropped &resized image approach (average)	19
Table 8	Result of 10% cropped &resized image approach (maximum)	20
Table 9	Result of 10% masked image approach (average)	20
Table 10	Result of 10% masked image approach (maximum)	21
Table 11	Results summary	21
Table 12	MLP Net1 result for MNIST dataset	22
Table 13	SVM result on MLP Net1 hidden states for MNIST dataset	23
Table 14	KNN results on MLP Net1 hidden state for MNIST dataset	23
Table 15	MLP Net2 result for MNIST dataset	24
Table 16	Clustering Approach	30
Table 17	Learned Structure to SVM	31
Table 18	Learned Structure to KNN	31
Table 19	The output performance scores for MNIST dataset	32
Table 20	Features from original image to SVM Linear	33
Table 21	Features from original image to SVM RBF	33
Table 22	Fusing features from patch to SVM Linear	34
Table 23	Fusing features from patch to SVM RBF	34
Table 24	Concatenating predicted prob. from each patch to SVM Linear	35

Table 25	Average of predicted prob. from each patch to SVM Linear	35
Table 26	Maximum of predicted prob. from each patch to SVM Linear	36
Table 27	Concatenating predicted prob. from each patch to SVM RBF	36
Table 28	Concatenating predicted prob. from each patch to SVM RBF and from fused features to SVM RBF	37
Table 29	Concatenating predicted prob. from each patch to SVM RBF and from original image features to SVM RBF	37
Table 30	Concatenating predicted prob. from each patch to SVM RBF, from fused features and from original image features to SVM RBF	38
Table 31	Concatenating predicted prob. from each patch(excluding top patch) to SVM RBF and from fused features to SVM RBF	39
Table 32	Concatenating predicted prob. from each patch(excluding bottom patch) to SVM RBF and from fused features to SVM RBF	39
Table 33	Concatenating predicted prob. from each patch(excluding right patch) to SVM RBF and from fused features to SVM RBF	40
Table 34	Concatenating predicted prob. from each patch(excluding left patch) to SVM RBF and from fused features to SVM RBF	40
Table 35	Concatenating predicted prob. from left and right patch to SVM RBF and from fused features to SVM RBF	41
Table 36	Concatenating predicted prob. from top and bottom patch to SVM RBF and from fused features to SVM RBF	41
Table 37	Concatenating predicted prob. from top patch to SVM RBF and from fused features to SVM RBF	42
Table 38	Concatenating predicted prob. from bottom patch to SVM RBF and from fused features to SVM RBF	42
Table 39	Concatenating predicted prob. from right patch to SVM RBF and from fused features to SVM RBF	43
Table 40	Concatenating predicted prob. from left patch to SVM RBF and from fused features to SVM RBF	43

LIST OF ABBREVIATIONS

1	MLP	Multilayer Perceptron
2	CNN	Convolutional Numeral Network
3	AE	Autoencoder
4	CAE	Convolutional Autoencoder
5	1-D	One Dimensional
6	SVM	Support Vector Machine
7	RBF	Radial Basis Function

CHAPTER 1

INTRODUCTION

Handwritten Numeral Recognition is the task of correctly identifying handwritten digits. It has an important use-case in the digitisation of old documents. The challenges while performing handwritten digits recognition arise due to the fact that every person's handwriting can differ. This difference in handwriting can be due to different fonts, slant in letters. Bad quality of text or image from which digits needs to be recognised also creates difficulty.

In this work, we use structure learning and part learning to tackle the handwritten digits classification task. For part learning, we divide the image into various patches. Patch learning approach has not been extensively studied for handwritten digits recognition. Then structure learning is applied to individual patches. Patches are used to train the neural network, and then the structure is learned corresponding to each data sample in dataset. This structures learned from different patches of the same image are then combined to produce the vector that will represent the image instead of the image.

In the first part of our work, we have proposed a method that uses part-learning along with a novel CNN weight pre-initialisation strategy. In this two parts of images are created: top and bottom half. Some portion of training data is taken out and used to initialise weights of the two CNNs. The proposed method gets accuracy as high as compared to state-of-the-art methods.

In the second part, we proposed a method which used structure along with part-learning. In this five patch of the image was created: top, bottom, left, right and centre. These five patches were used to train five MLPs. Each MLP had the same architecture, which consists of three hidden layers. Having examined the data from each of the three hidden layers, it was found that the third hidden layer showed that it would be best suited to learn the structure from. After training the MLP, a structure was learned. This structure was used instead of original data for the classification task.

In the third part, the theme of the previous method was carried forward. The integration of part-learning with structure learning was explored with autoencoders. Learning structures from autoencoders were considered as autoencoders are naturally good at learning representation for a given data. Combining this with part-learning helped learn structure for different patches. Having different patches helped capture all regions of the image. Combining the structure learned from different patches and giving it to a classifier yielded accuracy comparable to state-of-the-art methods.

The approach of integrating part learning with structure learning for handwritten numeral recognition was extended for the task of handwritten character recognition. The experiment was done on Devanagari dataset. Five patches were created from the original images; Instead of MLP, a CNN was used to learning structures. Later the structures learned were combined and given to the SVM. The accuracy achieved was comparable to the highest accuracy reported on the dataset. Thus indicating the extendability of our proposed approach to different classification tasks.

CHAPTER 2

LITERATURE REVIEW

2.1 Handwritten Numeral Recognition

Handwritten Numeral Recognition is the task of correctly identifying handwritten digits. It has an important use-case in digitising old script, image restoration. The challenges while performing handwritten digits recognition arise due to the fact that every person's handwriting can differ. This variations in handwriting can be due to different fonts, slant in letters. Bad quality of text or image from which digits also creates difficulty in recognition.

Over the years, different methods to build handwritten digit recognition classifier have researched. [1] uses multi-layer perceptron to build a classifier,[2] uses a multi-stage process in which results of three MLPs are that are used is combined, [3] uses an approach based on a k-Nearest Neighbour[4] graph obtained with an image deformation model and [5] uses a different technique which involves partitioning the image into boxes and extracting a feature from these to classify.

2.2 Artificial Neural Networks

Computers outperform human in the speed of doing a calculation. A computation involving finding nth power(or root) of a number or multiplication of matrices is performed very quickly by a computer in comparison to humans. But human brain completely outshines computer when it comes to complex tasks which involve imagination, common sense and intuition. Inspired by the human brain structure, artificial neural networks are the network structure which helps computer behave and reason in a human-like manner. Artificial neural networks are used from the different task such as Image recognition[6], Natural language processing[7], Face recognition[8], Motion detection[9] etc.

An artificial neural network is an endeavour to re-enact neurons interconnection that make up a human brain. Neuron interconnection of the human mind is unpredictable and not very surely knew. Artificial neural networks are a more straightforward interconnection of neurons made by programming machines to carry on and settle on choices in a way like people. Artificial neural networks less unrivalled than human cerebrum; however, they are equipped for taking care of issues which are exceptionally perplexing or difficult to code.

Artificial neural networks are composed of processing layers which in turn are made of neurons. Neurons are the smallest unit of a neural network. The layout of the neuron is such that the output of one neuron is generally input to another neuron. A typical artificial neural network can have from few to several layers. Layers of an artificial neural network can be separated into three types. The input layer which gets different types of data. This layer goes about as the sensor artificial neural network. It gets information which this system means to process. Information layer passes the information to hidden layers through connections. These associations have some significance/weights joined to them, which assume a significant job in choosing what to search for and what to overlook in the information. Further handling prompts the output at the output layer. The quantities of neurons at the output layer are straightforwardly identified with the task for which the neural system is being utilised.

Three different layers in a neural network are as followed:-

1. Input Layer
2. Hidden Layers
3. Output Layer

Following is the manner in which these layers are laid

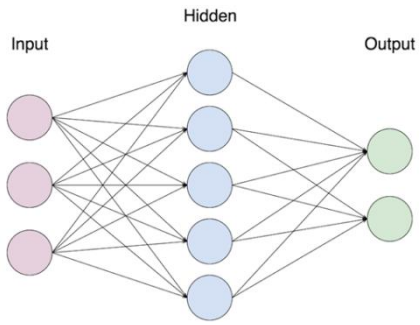


Figure 2.1 Different layers of a Neural Network[10]

2.3 Convolutional Neural Network

In deep learning, a Convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analysing visual imagery. CNN is not restricted to the visual application; they are also used in the field of speech and time-series data[41]. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organisation of the animal visual cortex.

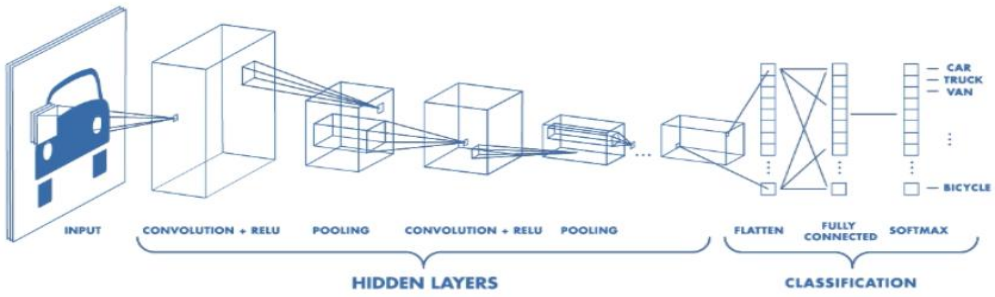


Fig. 2.2 CNN Layers[11]

CNN's have two components:

- Feature extraction

In this part, the network will perform a series of convolutions and pooling operations during which the features are detected. If you had a picture of a zebra, this is the part where the network would recognise its stripes, two ears, and four legs.

- The Classification part

Here, the fully connected layers will serve as a classifier on top of these extracted features. They will assign a probability for the object on the image being what the algorithm predicts it is.

Types of layers

1. Convolution layer where the convolution process happens.
2. Pooling layer is where the pooling operation is applied.
3. Normalisation layer where the activation (ReLU) process happens.
4. Fully Connected layer (Dense)

The Convolution layers forms the basis of CNN. The CNN preserves the spatial information of two dimensional image by taking it complete as input without transforming image into a 1-D array. The convolution layer takes image as input, apply convolution operation on it and gives a two dimensional output. Convolution operation involves dot product of various kernels with the input data. Initial the filters are random with they are learned by training the CNN. These filter learn very specific features and patterns. When a filter, let say, trained to detect edges detect an edge in image, the corresponding value in output of convolution layer becomes high in comparison to other parts. Convolutional layer activations constitute a powerful image representation[12]

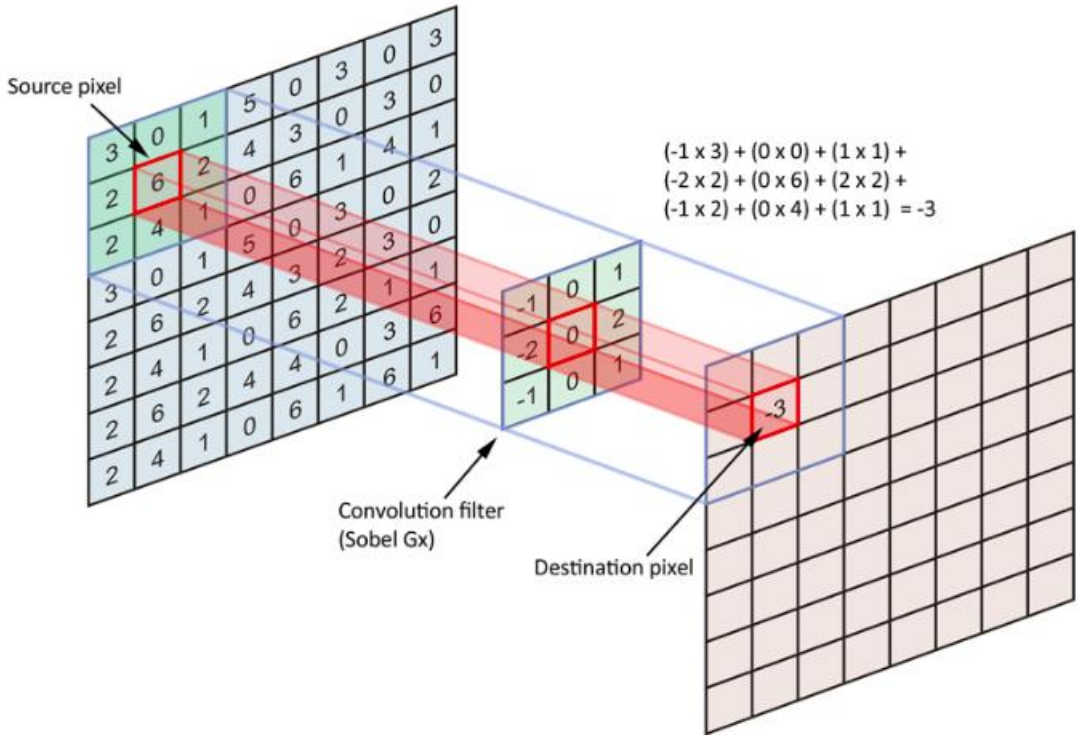


Fig 2.3 Convolution operation[13]

Different types of pooling methods exist as shown in[14], but generally, max pooling is found to be most effective[15]. In max-pooling, the maximum value lying in the filter region is chosen to represent that region. Pooling help reduces the computation by reducing the size of the image but at the same time retaining the essential features of the image. Pooling sometime can also lead to highlighting the features instead of just retaining them.

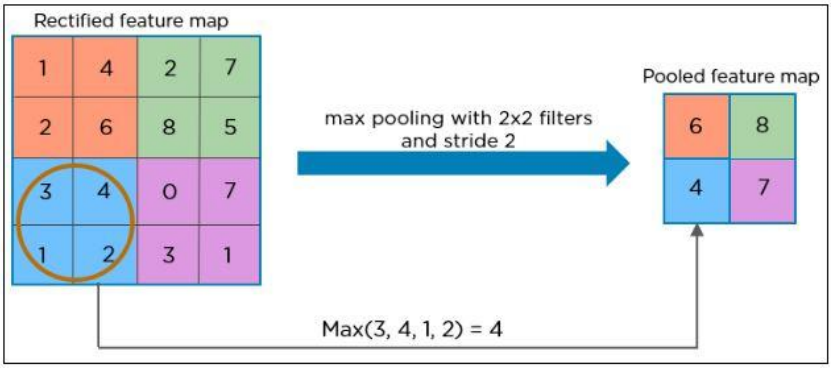


Fig. 2.4 Max Pooling[16]

2.4 Autoencoders

An autoencoder is a neural network that is used to learn representation from data. It works in an unsupervised way by trying to recreate its input as its output. In the process of trying to recreate its input, the features are learned. This learning generally takes place by forcing the input through a smaller hidden layer. The autoencoder can be viewed as having two components: encoder component and decoder component.

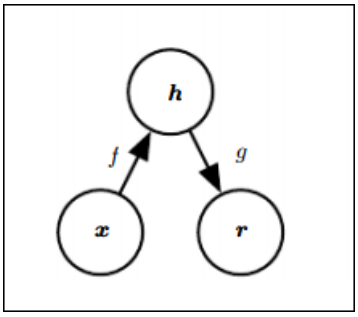


Fig. 2.5 The general structure of an Autoencoder[17]

Encoding function: $h = f(x)$

Decoding function: $r = g(h)$

hidden layer h describes a code used to represent the input. The encoder part encodes the input to that compact representation, and the decoder part reconstructs the original input from the compact representation.

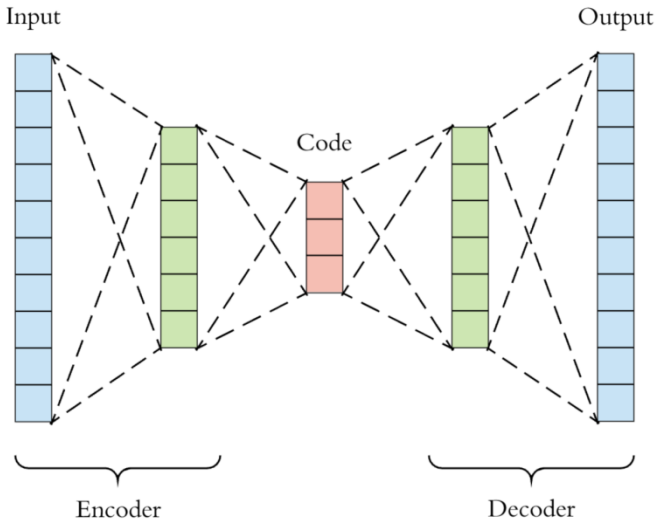


Fig. 2.6 Different layers of an Autoencoder[18]

2.4.1 Convolutional Autoencoders

Convolutional Autoencoders (CAE) are autoencoders which are mostly used to learn representation for image data. Like the Convolution Neural Network they take image as input. Thus the spatial information is preserved. Unlike CNN, CAE don't attempt to learn features for the purpose of classification, CAE don't have densely connected layers. Instead the CAE learns filter for the purpose of being able to reproduce the input image at output. CAE work in an unsupervised way. CAE can be thought of as composed of two parts: encoder and decoder. Encoders part tries to learn filters and generally along the way reduces the image size. Reducing the size leads to focusing on just important features and also help reduce computation. This is accomplished by pooling layer. While the decoder part tries to reconstruct the input and it increases it's input size back to input size of CAE. This is accomplished by up-sampling layer.

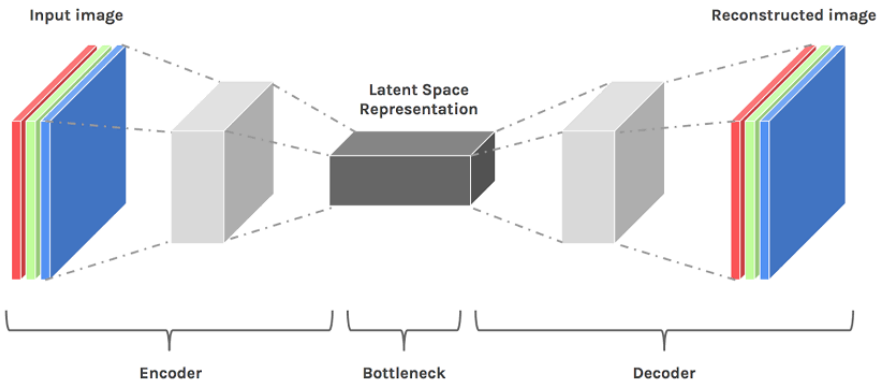


Fig. 2.7 Convolutional Autoencoder[19]

CAE offer a great improvement over the general AE when dealing with image data. The original dimension of image ensures no information is lost while trying to unroll image into 1-D vector. The latent representation learned by a CAE is smaller in size as compared to the original image input but still it can be used to reproduce the original image. Hence CAE are extremely useful in data compression. The advantage CAE offer over other compression method is it is not generic compression. Compression logic using CAE is data specific like other method which are pre-defined.

2.5 Weight Initialization

Initial weight plays a crucial role in determining the speed of convergence, generalisation and probability of convergence[44]. A derivative-based optimisation algorithm like gradient descent is heavily reliant on the initial state. Proper initialisation of the weights in a neural network is critical to its convergence. An optimum weight initialisation which strongly improves the performance of the backpropagation (BP) algorithm. Random initialisation is generally used as it often yields a favourable starting point for optimisation [20]. Various other weight initialization strategies([21] ,[22] , [23]).Part of the difficulty in training these models lies in determining the proper initialisation strategy for the parameters in the model. It is well known [24] that arbitrary initialisations can slow down or even completely stall the convergence process. The slowdown arises because arbitrary initialisations can result in the deeper layers receiving inputs with small variances, which in turn slows down backpropagation, and retards the overall convergence process. Weight initialisation is an area of active research, and numerous methods ([24], [25], [26] to state a few) have been proposed to deal with the problem of the shrinking variance in the deeper layers.

2.6 Learning from image parts

An improvement in learning is seen when subset or part of the image are learnt rather than the whole image itself[27][28]. Application of patch-based learning is not very well explored in the field of handwritten numeral recognition. Generally, in learning from image parts approach, patches are selected. These patches are individually operated upon, and the result of the individual patches is combined to produce the output. Patch from the image is cropped, processed and then learned by a suitable classifier. This approach, to an extent, is immune to background clutter. The major issue in the application of learning from parts approach is to be able to find a patch in the image which contains the desired information. Another difficulty with this approach is the lack of proper predefined area in the image. Along with that number of parts to be learnt is another important issue. As learning with fewer parts can miss important information and learn with too many parts can cause over-fitting and also a huge increase in complexity cost.

2.7 Structure Learning

Structure learning refers to the machine learning strategy where a structured object is learnt from the labelled training samples, instead of the class labels themselves [29]. Instances of 'structure' include vectors [30], trees [31], graphs [32], sequences [33] and finite state automata[34].

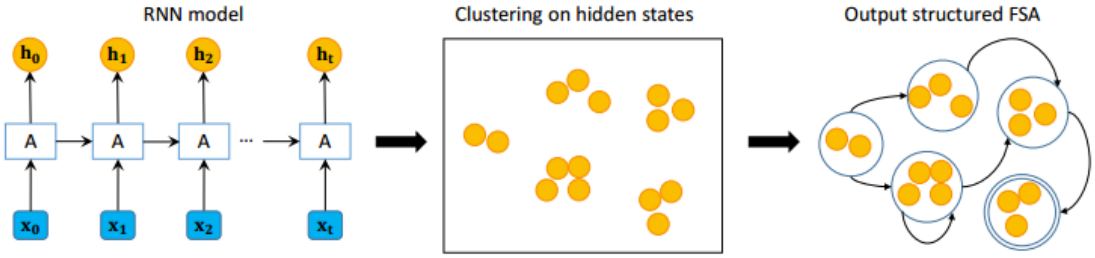


Fig 2.8 Training an RNN model, clustering the RNN's hidden states and outputting the final structured FSA. [34]

A structured perceptron is learnt in [35] for the tagging problem in natural language processing. A variant of the perceptron algorithm is suggested in [35] that learns the mapping between the input $x \epsilon X$ and output labels $y \epsilon Y$. The task is to learn the representation that maps the vector (x,y) to a d-dimensional feature vector. The derived feature serves as an indicator function that determines the tag. In the preceding literature, this task was termed- rule extraction from Artificial Neural Networks (ANNs) [36]. Rules were added and updated in ANNs, simultaneously increasing the symbolic knowledge derived from the network structure. A dynamically growing hidden layer is incorporated in the ANN in [37] that increments the size of the hidden layer on observing a higher entropy of the input distribution.

CHAPTER 3

PROPOSED WORK

3.1 CNN Pre-Initialisation by Part Learning

3.1.1 CNN architecture

The CNN used in the proposed method is shown in Fig. 3. Successive Convolution layer followed by the pooling layer forms the composition of CNN architecture. Filters of the size of size 3x3 are used for every convolution layer. 2x2 filters are used for all pooling layer, and max-pooling strategy is used for pooling. Rectified Linear Unit(ReLU) activation function is used in convolution layers, and Softmax activation function is used at the output layer to get the activations as probabilistic scores. Table 1 shows each layer in the proposed CNN. It also shows what the output shape of each layer is.

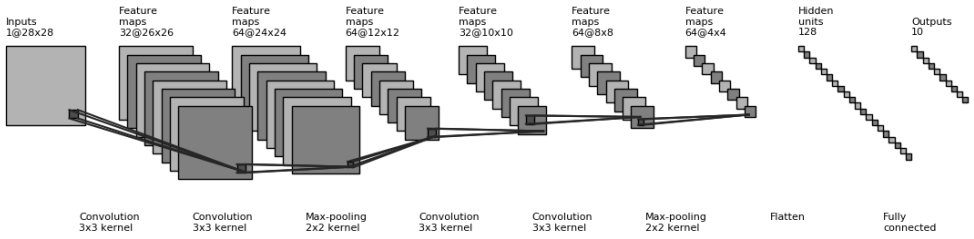


Fig. 3.1 CNN architecture for our experiment

Table 1. Ordering of layers proposed CNN Architecture

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
conv2d_3 (Conv2D)	(None, 10, 10, 32)	18464
conv2d_4 (Conv2D)	(None, 8, 8, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
dropout_2 (Dropout)	(None, 4, 4, 64)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_1 (Dense)	(None, 128)	131200
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

Total params: 188,266
Trainable params: 188,266
Non-trainable params: 0

3.1.2 Proposed Methodology

Inspired from how humans learn a complex task in phases, by starting with easy concept and then increasing the difficulty of the content after each phase of learning, a multi-phase learning approach is devised[46]. This proposed method comprises of two phases. In the first phase, simpler images of handwritten numerals are given to CNNs to learn from. These simpler images are created by masking either top or bottom half of normal images. An only small portion of images(5% or 10%) from the training set are used of phase 1 learning. After phase 1 learning the CNNs are pre-initialised with these less complex images. One CNN is trained with just images part of just the top portion, and another CNN is trained with just images part of just the bottom portion. In phase 2, both the CNNs are made to learn from the remaining images from the training set. After the training has completion, the testing is performed by combining the predictions of both CNNs to predict the output. The output layer softmax predicted probabilities are combined by either using the average or maximum, to get the class label.

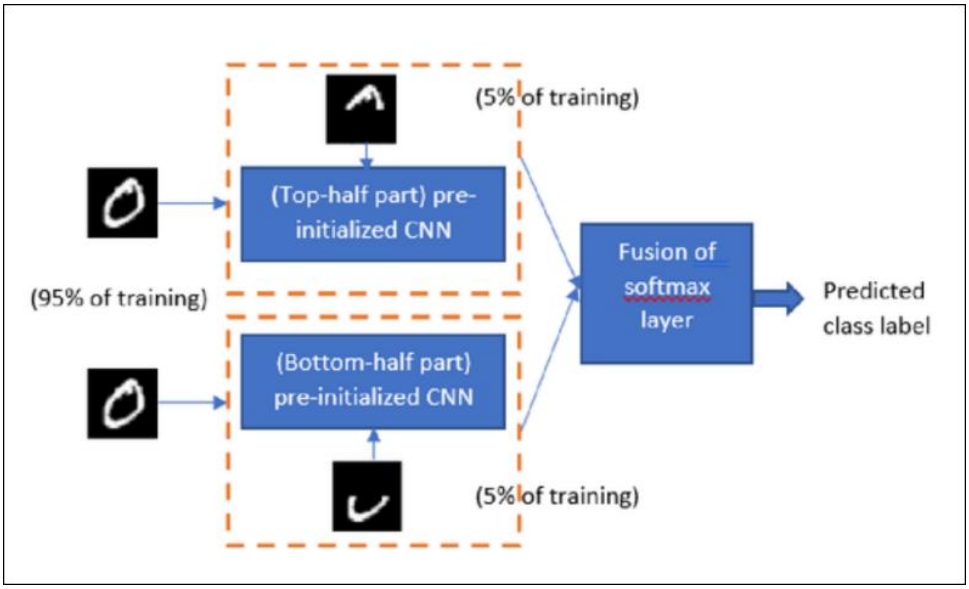


Figure 3.2 Proposed Methodology block diagram

3.2 Integrating Part-Learning with Structure Learning for ANN

3.2.1 MLP architecture

A 784-35-10 Multilayer Perceptron (MLP) network (**Net 1**) is the framework for the first phase of our experiments on the benchmark MNIST dataset of handwritten English numerals. The input images of 28*28 dimension are converted to a one-dimensional array of size 784*1. This 1-D array of 784*1 is then given as input to the neural network's input layer. Only one hidden layer of size 35 was used. The output layer contains ten neurons. Each representing the 10 different classes in the dataset namely 0,1,2,3,4,5,6,7,8,9.

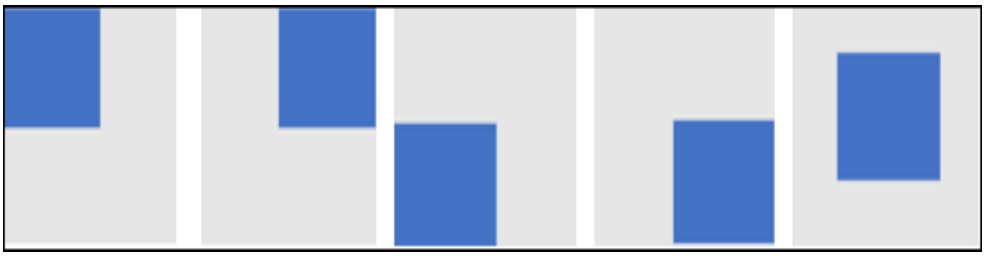
In the second phase of our experiments, a larger (deeper) network of size 196-35-35-35-10 (**Net 2**), with three hidden layers containing 35 neurons each, is used for training patches extracted from the MNIST images.

3.2.2 Proposed Method

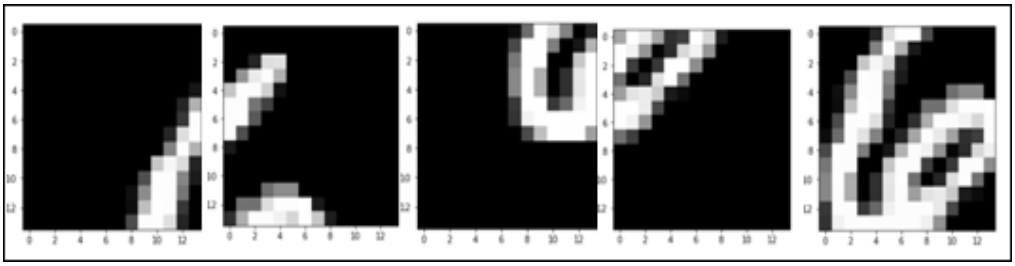
Inspired by the [34] in which the author shows that hidden states of RNN tend to form clusters and by the idea of learning structures from hidden states of RNN, this proposed work[47] aims to learn the structure from hidden states in an MLP.

In phase 1, the Net 1 is trained on the MNIST dataset. After the training is completed, the hidden state activation vector is extracted for both training and testing samples. This extracted feature vector (1x35) is further used for the purpose of classification rather than the original image. So these feature vector are given to KNN (k=1) for classification.

In the second phase, the patches created from the original image. Each patch is 14x14, and five such patches are used. Fig. 3 shows these patches for numeral '6'. All five patches are then trained on five different MLP (Net 2). After training the Net 2 , hidden state activation of all three hidden states is extracted for both training and testing data. To study which hidden layer is suitable for structure learning, K-elbow method is applied to data of all three hidden states. K-elbow method tells the clusterability of given data by trying to cluster data for various values of K and returns the optimal value of K. The hidden state thus selected is further used for classification. The hidden state activation vector of selected hidden state are given to classifier (KNN and SVM).



(a)



(b)

Fig. 3.3 (a) The layout of the five image patches each covering a quarter area of the image (b) The five patches shown for the numeral ‘6’

3.3 Integrating Part-Learning with Structure Learning for Convolutional Auto-Encoders

3.3.1 Convolutional Auto-Encoders Architecture

The Chronological layers of the proposed Convolution Auto-Encoders are shown in Table X below. Filters of the size 3x3 are used for every convolution layer. 2x2 filters are used for all pooling layer, and max-pooling strategy is used for pooling. Rectified Linear Unit(ReLU) activation function is used in convolution layers, and Softmax activation function is used at the output layer to get the activations as probabilistic scores. Table 1 shows each layer in the proposed CAE. It also shows what the output shape of each layer is.

Table 2. Ordering of layers proposed CAE Architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_1 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 16)	0
conv2d_2 (Conv2D)	(None, 14, 14, 8)	1160
max_pooling2d_2 (MaxPooling2	(None, 7, 7, 8)	0
conv2d_3 (Conv2D)	(None, 7, 7, 8)	584
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 8)	0
conv2d_4 (Conv2D)	(None, 4, 4, 8)	584
up_sampling2d_1 (UpSampling2	(None, 8, 8, 8)	0
conv2d_5 (Conv2D)	(None, 8, 8, 8)	584
up_sampling2d_2 (UpSampling2	(None, 16, 16, 8)	0
conv2d_6 (Conv2D)	(None, 14, 14, 16)	1168
up_sampling2d_3 (UpSampling2	(None, 28, 28, 16)	0
conv2d_7 (Conv2D)	(None, 28, 28, 1)	145

Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0

3.3.2 Proposed Methodology

Autoencoders are good at learning compact representation for a given data. With the aim of using this representation along with part-learning for handwritten numeral recognition, four patches are selected: top, bottom, right and left. Patches are created by masking a certain portion of images. Four CAEs with architecture, as shown in Table 2 are trained. Each CAE is trained with just one particular patch - CAE1 is trained with just top patch images, CAE2 is trained with just bottom patch images, CAE3 is trained with just right patch images, and CAE4 is trained with just left patch images. After training the CAEs, the output of the encoder part of autoencoder were extracted. These hidden layer activations are extracted for both training and testing data by just performing forward pass. These 1x128 dimensional features learned from CAE are further used in the experiment. Features learned from CAEs are given to SVM classifier. The predicted probabilities for each sample are used to represent the sample. These predicted probabilities are combined with predicted probabilities obtained by giving concatenated features learned from CAE to SVM. This works as newly learned features using both part and structure learning.

CHAPTER 4

RESULTS

4.1 Experimental Setup

The experiments are performed on handwritten numeral images from the MNIST dataset [42] for English handwritten numerals. There are 10 different classes of images, each corresponding to a digit in 0 to 9. Each image is of 28X28 dimension. Dataset consists of 70K images in total. Out of the 70K images, 60K images are for training and 10K are for testing purpose. The train and test segregation is done at the source itself. Example of the ten handwritten digits in the MNIST training set is shown in Fig. 4.1 below.

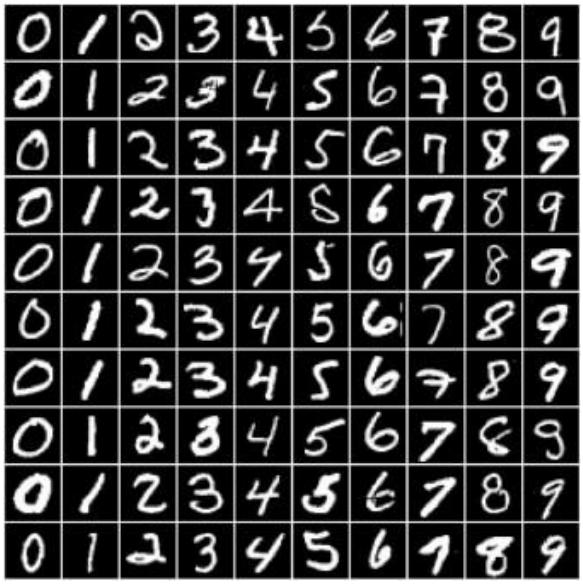


Fig. 4.1 Sample of MNIST dataset

Device Specifications

Processor: Intel(R) Core (TM) i5-5200U CPU @ 2.20GHz, Memory: RAM 8.00GB DDR3L-1600 SDRAM, Graphics card: NVIDIA GeForce 940M (2 GB DDR3L dedicated).

4.2 CNN Pre-Initialisation Implementation Results

The Fig.4.2 shows the images used for training the CNNs. The first row depicts 0-9 digits from MNIST dataset. Row2 is the bottom half masked images of the corresponding row1 images. Similarly, row3 shows the top half masked images of the corresponding row1 images.

Fig. 4.2 Images prepared for initialising CNN

CNN without pre-initialisation and without part learning reported accuracy of 99.44% on Architecture (a) for MNIST dataset. This method was taken as the base method and the accuracy of 99.44% as the baseline accuracy. All further attempts were made to improve on this accuracy.

Table 3 - Result of 5% cropped &resized image approach (average)

	precision	recall	f1-score	support
0	0.99	1	1	980
1	0.99	1	1	1135
2	1	1	1	1032
3	1	1	1	1010
4	0.99	1	1	982
5	0.99	0.99	0.99	892
6	1	0.99	0.99	958
7	0.99	1	0.99	1028
8	1	0.99	1	974
9	1	0.99	0.99	1009
micro avg	1	1	1	10000
macro avg	1	1	1	10000
weighted avg	1	1	1	10000

Table 4 - Result of 5% cropped &resized image approach (maximum)

	precision	recall	f1-score	support
0	0.99	1	1	980
1	0.99	1	1	1135
2	1	1	1	1032
3	1	1	1	1010
4	0.99	1	1	982
5	0.99	0.99	0.99	892
6	1	0.99	0.99	958
7	0.99	1	0.99	1028
8	1	0.99	1	974
9	1	0.99	0.99	1009
micro avg	1	1	1	10000
macro avg	1	1	1	10000
weighted avg	1	1	1	10000

Table 3 and Table 4 to represent the implementation result of using 5% resized images for pre-initialisation and combining the probabilistic scores of softmax output layer using averaging and choosing maximum criteria respectively. Accuracy of 99.52 and 99.51 was reported for the same. The results of both were higher in comparison to a straight forward baseline approach.

Table 5 - Result of 5% masked image approach (average)

	precision	recall	f1-score	support
0	0.99	1	1	980
1	0.99	1	1	1135
2	1	1	1	1032
3	0.99	1	1	1010
4	0.99	1	1	982
5	1	0.99	0.99	892
6	1	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	1	1	1	974
9	0.99	0.99	0.99	1009
micro avg	1	1	1	10000
macro avg	1	1	1	10000
weighted avg	1	1	1	10000

Table 6 - Result of 5% masked image approach (maximum)

	precision	recall	f1-score	support
0	1	1	1	980
1	0.99	1	1	1135
2	1	1	1	1032
3	0.99	1	1	1010
4	0.99	1	1	982
5	1	0.99	0.99	892
6	1	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	1	1	1	974
9	0.99	0.99	0.99	1009
micro avg	1	1	1	10000
macro avg	1	1	1	10000
weighted avg	1	1	1	10000

Table 5 and Table 6 represent the implementation result of using 5% masked images for pre-initialisation and combining the probabilistic scores of softmax output layer using averaging and choosing maximum criteria respectively. Accuracy of 99.53 and 99.56 was reported for the same. The results of both were higher in comparison to straight forward baseline approach and also a little better than that achieved by using a resized image of the top half and bottom half.

Table 7 - Result of 10% cropped &resized image approach (average)

	precision	recall	f1-score	support
0	0.99	1	1	980
1	1	1	1	1135
2	1	1	1	1032
3	1	1	1	1010
4	0.99	1	0.99	982
5	0.99	0.99	0.99	892
6	1	0.99	1	958
7	0.99	1	0.99	1028
8	1	0.99	1	974
9	0.99	0.99	0.99	1009
micro avg	0.99	0.99	0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	1	0.99	0.99	10000

Table 8 - Result of 10% cropped &resized image approach (maximum)

	precision	recall	f1-score	support
0	0.99	1	1	980
1	1	1	1	1135
2	1	1	1	1032
3	1	1	1	1010
4	0.99	1	0.99	982
5	0.99	0.99	0.99	892
6	1	0.99	1	958
7	0.99	1	0.99	1028
8	1	0.99	0.99	974
9	0.99	0.99	0.99	1009
micro avg	0.99	0.99	0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

Table 7 and Table 8 represent the implementation result of using 10% resized images for pre-initialisation and combining the probabilistic scores of softmax output layer using averaging and choosing maximum criteria, respectively. Accuracy of 99.5 and 99.49 was reported for the same. The results of both were higher in comparison to straight forward baseline approach but the results dropped a little in comparison to using just 5% of pre-initialisation.

Table 9 - Result of 10% masked image approach (average)

	precision	recall	f1-score	support
0	0.99	1	1	980
1	0.99	1	1	1135
2	1	1	1	1032
3	1	1	1	1010
4	0.99	1	1	982
5	0.99	0.99	0.99	892
6	1	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	1	1	1	974
9	1	0.99	0.99	1009
micro avg	1	1	1	10000
macro avg	1	1	1	10000
weighted avg	1	1	1	10000

Table 10 - Result of 10% masked image approach (maximum)

	precision	recall	f1-score	support
0	0.99	1	1	980
1	0.99	1	1	1135
2	1	1	1	1032
3	1	1	1	1010
4	0.99	1	1	982
5	0.99	0.99	0.99	892
6	1	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	1	1	1	974
9	1	0.99	0.99	1009
micro avg	1	1	1	10000
macro avg	1	1	1	10000
weighted avg	1	1	1	10000

Table 9 and Table 10 represent the implementation result of using 10% masked images for pre-initialisation and combining the probabilistic scores of softmax output layer using averaging and choosing maximum criteria, respectively. Accuracy of 99.55 and 99.53 was reported for the same. The results of both were higher in comparison to straight forward baseline.

Table 11. Summary

Method	Test Accuracy (5% Training for pre-initialization, 95% for fine tuning)	Test Accuracy (10% Training for pre-initialization, 90% for fine tuning)	Test Accuracy (Without pre-initialization and random initialization of weights)
Proposed CNN Model (Fusion of scores by Averaging) Cropped Resized Images	99.52	99.5	94.44
Proposed CNN Model (Fusion of scores by Maximum) Cropped Resized Images	99.51	99.49	94.44
Proposed CNN Model (Fusion of scores by Averaging) Masked Image	99.53	99.55	94.44
Proposed CNN Model (Fusion of scores by Maximum) Masked Image	99.56	99.53	94.44

4.3 Part-Learning and Structure Learning MLP implementation Results

Table 12 - MLP Net1 result for MNIST dataset

	precision	recall	f1-score	support
0	0.97	0.98	0.98	980
1	0.98	0.98	0.98	1135
2	0.96	0.94	0.95	1032
3	0.92	0.95	0.94	1010
4	0.95	0.93	0.94	982
5	0.96	0.91	0.93	892
6	0.95	0.96	0.95	958
7	0.97	0.93	0.95	1028
8	0.91	0.96	0.93	974
9	0.91	0.94	0.93	1009
Micro avg	0.95	0.95	0.95	10000
Macro avg	0.95	0.95	0.95	10000
Weighted avg	0.95	0.95	0.95	10000

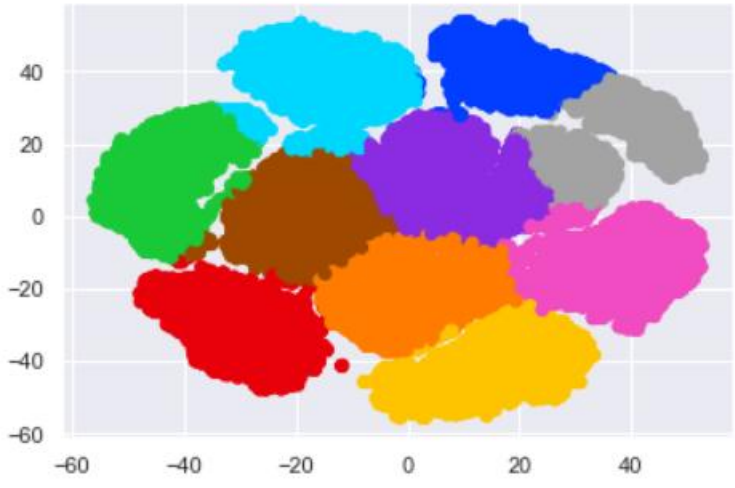


Fig. 4.3 Hidden state cluster of MLP

Table 13 - SVM result on MLP Net1 hidden states for MNIST dataset

	precision	recall	f1-score	support
0	1	0.83	0.9	980
1	1	0.94	0.97	1135
2	0.98	0.88	0.93	1032
3	0.99	0.82	0.9	1010
4	0.98	0.78	0.87	982
5	0.38	0.99	0.55	892
6	1	0.79	0.88	958
7	1	0.8	0.89	1028
8	0.99	0.81	0.89	974
9	0.99	0.83	0.91	1009
Micro avg	0.85	0.85	0.85	10000
Macro avg	0.93	0.85	0.87	10000
Weighted avg	0.94	0.85	0.87	10000

Table 14 - KNN result on MLP Net1 hidden states for MNIST dataset

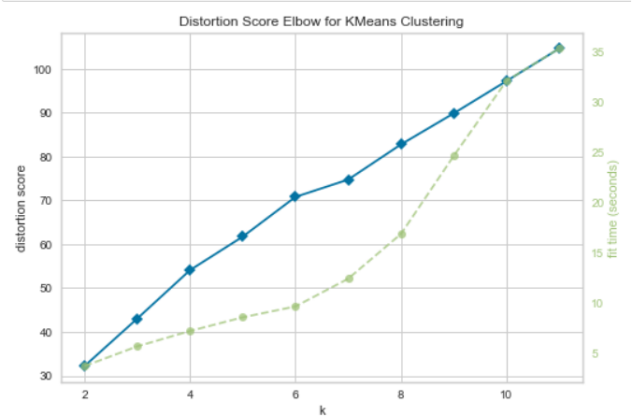
	precision	recall	f1-score	support
0	0.96	0.98	0.97	980
1	0.98	0.99	0.98	1135
2	0.97	0.96	0.96	1032
3	0.95	0.96	0.95	1010
4	0.95	0.95	0.95	982
5	0.97	0.92	0.94	892
6	0.96	0.97	0.97	958
7	0.97	0.96	0.96	1028
8	0.95	0.96	0.95	974
9	0.93	0.95	0.94	1009
Micro avg	0.96	0.96	0.96	10000
Macro avg	0.96	0.96	0.96	10000
Weighted avg	0.96	0.96	0.96	10000

The table 13 and table 14 show results for classification when Net1 hidden state is directly given to SVM and KNN classifier respectively. Train accuracy decreased in case of SVM (84.82), but there was a slight improvement (95.96) in case of train accuracy of KNN (K=5) classifier as compared to the baseline accuracy.

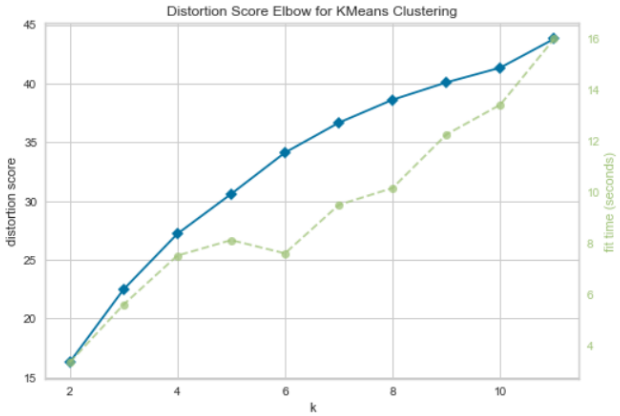
Table 15 - MLP Net2 result for MNIST dataset

	precision	recall	f1-score	support
0	0.96	0.98	0.97	980
1	0.99	0.98	0.98	1135
2	0.97	0.94	0.96	1032
3	0.96	0.97	0.96	1010
4	0.97	0.98	0.97	982
5	0.94	0.96	0.95	892
6	0.98	0.97	0.97	958
7	0.96	0.96	0.96	1028
8	0.95	0.95	0.95	974
9	0.97	0.95	0.96	1009
Micro avg	0.96	0.96	0.96	10000
Macro avg	0.96	0.96	0.96	10000
Weighted avg	0.96	0.96	0.96	10000

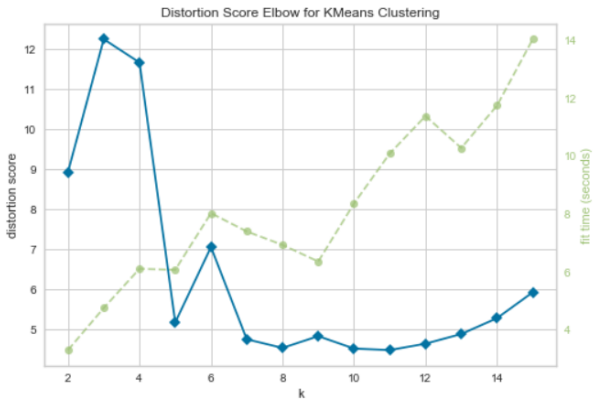
Above table 15 represent the MNIST dataset results for deep MLP (Net 2). Increasing the hidden layers helped model learn more features, and the accuracy increased to 96.46%. Further work on structure learning aimed to improve this accuracy.


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

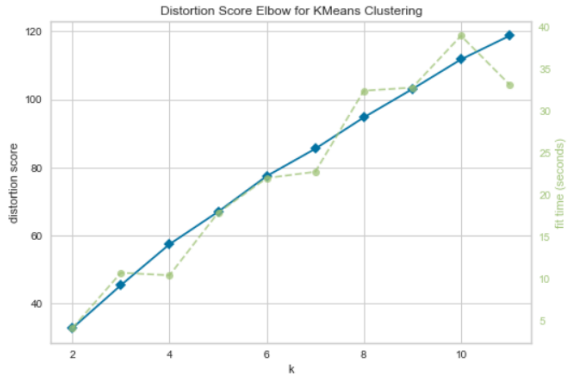
(a)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

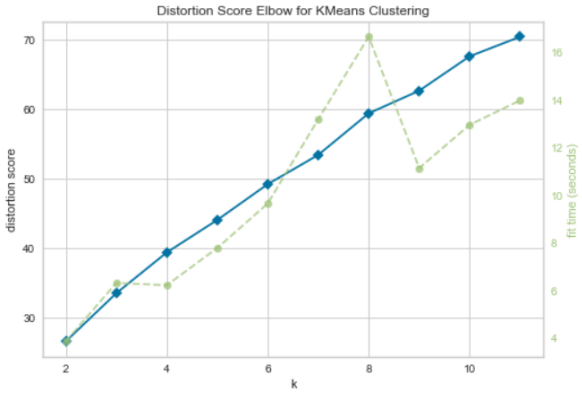
(b)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

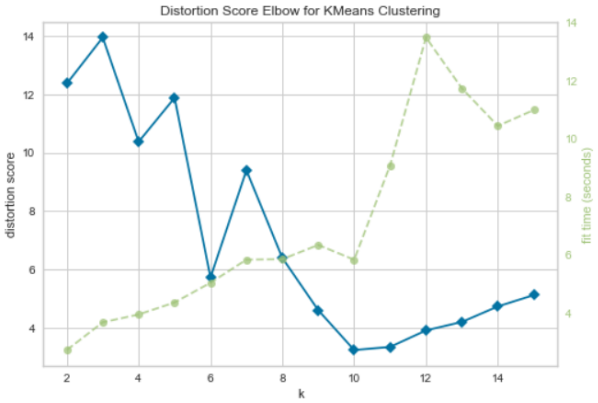
(c)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

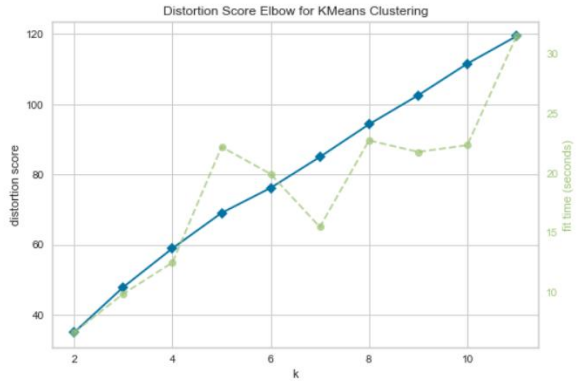
(d)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

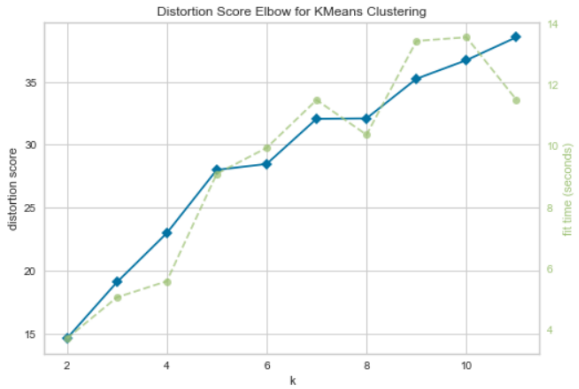
(e)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

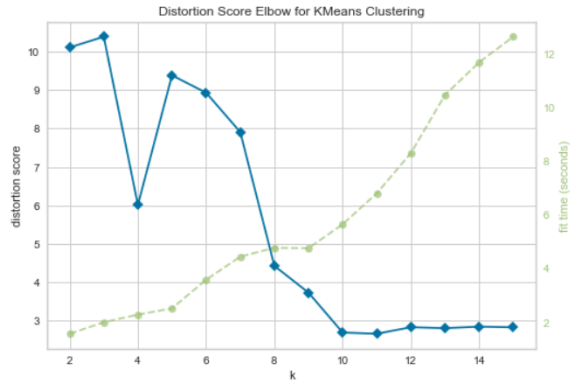
(f)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

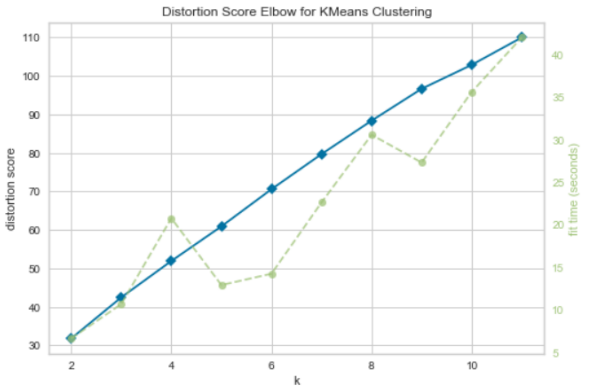
(g)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

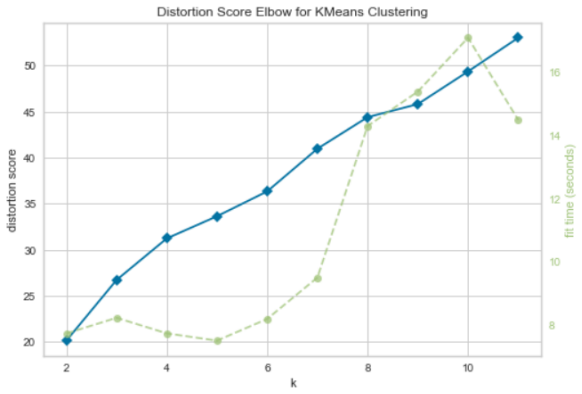
(h)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

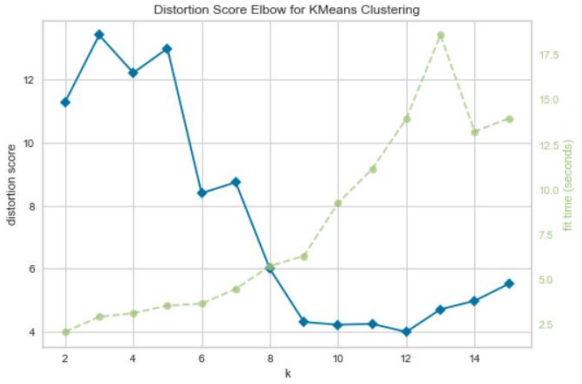
(i)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

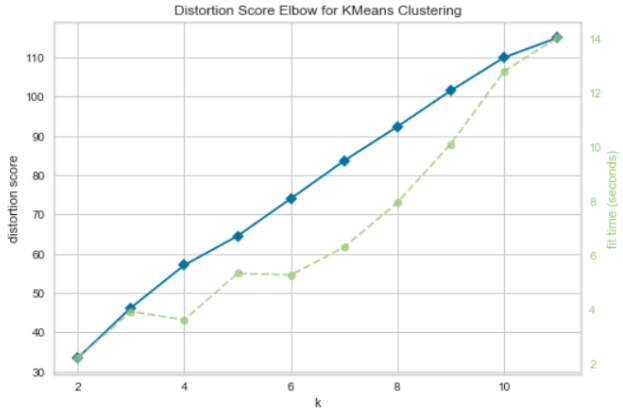
(j)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

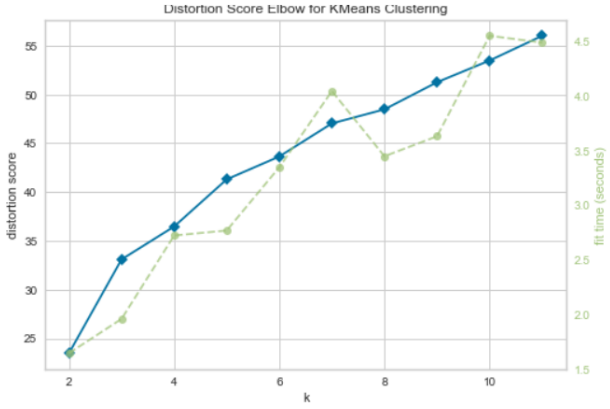
(k)


Distortion Score Elbow for KMeans Clustering
distortion score
fit time (seconds)
k

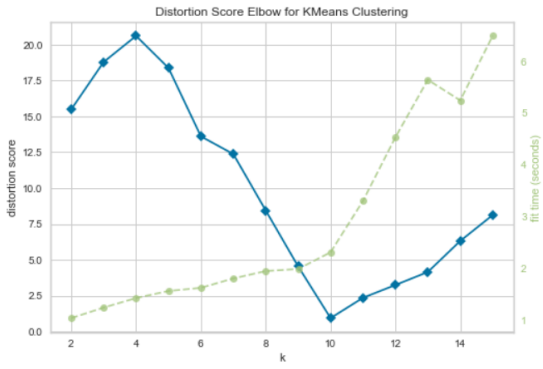
(l)



(m)



(n)



(o)

Fig. 4.4 The kmeans Elbow graphs for determining the interpretable hidden layer. Shown for the fifteen image patch layouts [(a)-(o)]

Analysis of figure 4.4 makes it clear that K-elbow graph of the first hidden layer and second hidden layer doesn't show any clusters. But the K-elbow graph of the third hidden layer showed that hidden state 3 had meaningful clustering possible. Further investigation was carried out on finding structure from 3rd hidden layer of all the patches.

Table 16 - Clustering Approach

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.98	0.99	0.98	1135
2	0.96	0.96	0.96	1032
3	0.96	0.97	0.96	1010
4	0.96	0.97	0.96	982
5	0.96	0.96	0.96	892
6	0.98	0.96	0.97	958
7	0.96	0.96	0.96	1028
8	0.96	0.95	0.96	974
9	0.96	0.94	0.95	1009
Micro avg	0.97	0.97	0.97	10000
Macro avg	0.96	0.96	0.96	10000
Weighted avg	0.97	0.97	0.97	10000

Above table. 16 shows the result of the clustering approach. All third hidden states vectors where clusters using K means. Value of K for each patch was found using K-elbow method. Cluster centre of all patches was stored. Later each training and testing data was assigned 5 cluster centre based on the cluster centre. The fused cluster centre was then used instead of the training and testing feature. The SVM classifier was used for further classification. There was an improved from Net2 result.

Table 17 - Learned Structure to SVM

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.98	0.99	0.98	1135
2	0.97	0.97	0.97	1032
3	0.96	0.98	0.97	1010
4	0.97	0.97	0.97	982
5	0.97	0.97	0.97	892
6	0.98	0.97	0.98	958
7	0.97	0.96	0.96	1028
8	0.97	0.95	0.96	974
9	0.97	0.96	0.97	1009
Micro avg	0.97	0.97	0.97	10000
Macro avg	0.97	0.97	0.97	10000
Weighted avg	0.97	0.97	0.97	10000

Table 18 - Learned Structure to KNN

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.98	0.99	0.98	1135
2	0.96	0.97	0.97	1032
3	0.96	0.97	0.96	1010
4	0.97	0.97	0.97	982
5	0.96	0.95	0.96	892
6	0.98	0.97	0.97	958
7	0.97	0.96	0.96	1028
8	0.97	0.96	0.96	974
9	0.96	0.95	0.96	1009
Micro avg	0.97	0.97	0.97	10000
Macro avg	0.97	0.97	0.97	10000
Weighted avg	0.97	0.97	0.97	10000

Table 17 and Table 18 represent accuracy when fused 3rd hidden state vectors are given to SVM and KNN classifier(K=7). Both improve the overall accuracy of Net 2 MLP (97.11&96.76). SVM gives the maximum accuracy of 97.11 found using the structure learning approach.

TABLE 19. THE OUTPUT PERFORMANCE SCORES FOR MNIST DATASET (PROPOSED STRUCTURE LEARNING METHOD- HIGHLIGHTED IN GREY)

Dataset	Method	F1-score
MNIST	Deep Structured Energy Based Models (Zhai *et al.,* 2016) [38]	0.9689
MNIST	Positive-Generative Adversarial Network (Chiaroni *et al.,* 2018) [39]	0.97
MNIST	Rank Pruning (Northcutt *et al.,* 2017) [40]	0.97
MNIST	Direct MLP classifier (**Net 1**)	0.9477
MNIST	Direct Naïve Bayes classifier	0.5658
MNIST	Hidden state clustering (**Net 1**) + distance from 10 centroids + SVM	0.95
MNIST	Hidden state vector (**Net 1**) + kNN (k=1)	0.96
MNIST	Patch-based Hidden state vector (**Net 2**) + kNN (k=7)	0.9676
MNIST	Patch-based Hidden state vector (**Net 2**) + SVM	0.9711

Table 19 shows a comparison of the proposed structure learning method to that of already available results on MNIST dataset. There is an improvement in the accuracy of the classification of Handwritten Numeral Recognition.

4.4 Part-Learning and Structure Learning CAE implementation Results

For all Convolutional Autoencoder experiments, configuration mentioned in Table 2 was used.

Table 20 - Features from original image to SVM Linear

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	1	0.99	1135
2	0.97	0.98	0.98	1032
3	0.97	0.98	0.97	1010
4	0.97	0.97	0.97	982
5	0.97	0.97	0.97	892
6	0.99	0.98	0.98	958
7	0.98	0.97	0.98	1028
8	0.97	0.96	0.97	974
9	0.97	0.96	0.96	1009
Micro avg	0.98	0.98	0.98	10000
Macro avg	0.98	0.98	0.98	10000
Weighted avg	0.98	0.98	0.98	10000

Table 21 - Features from original image to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.98	0.99	0.99	1032
3	0.98	0.99	0.98	1010
4	0.99	0.98	0.99	982
5	0.98	0.98	0.98	892
6	0.99	0.98	0.98	958
7	0.98	0.98	0.98	1028
8	0.98	0.98	0.98	974
9	0.98	0.97	0.97	1009
Micro avg	0.98	0.98	0.98	10000
Macro avg	0.98	0.98	0.98	10000
Weighted avg	0.98	0.98	0.98	10000

CAE was trained on MNIST dataset. After training the feature vector (Encoder output) were extracted for both training and test data. These feature vector of size 1x128 were them given to SVM for the numeral classification. An accuracy of 97.59 was achieved with linear kernel, and accuracy of 98.39 was achieved with RBF kernel, as shown in Table 20 and Table 21 respectively.

Table 22 - Fusing features from patches to SVM Linear

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.97	0.98	0.98	1032
3	0.97	0.99	0.98	1010
4	0.99	0.98	0.98	982
5	0.99	0.98	0.98	892
6	0.99	0.98	0.99	958
7	0.99	0.98	0.98	1028
8	0.99	0.97	0.98	974
9	0.98	0.97	0.98	1009
Micro avg	0.98	0.98	0.98	10000
Macro avg	0.98	0.98	0.98	10000
Weighted avg	0.98	0.98	0.98	10000

Table 23 - Fusing features from patches to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	0.99	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Part learning was introduced, and four images representing top, bottom, left, and right were created from original dataset by masking certain half of the image. CAEs were trained with just particular parts of images. Feature vectors were extracted all for CAEs and fused together. These fused features of 1x512 dimension were given to SVM classifier with linear and RBF kernel. Accuracy of 98.35 and 98.94% were reported as shown in above Table 22 and 23.

Fig. 4.5 Images of Left, Right, Top and Bottom patch of numeral '5'

Table 24 - Concatenating predicted prob. from each patch to SVM Linear

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.97	0.98	1032
3	0.97	0.99	0.98	1010
4	0.98	0.98	0.98	982
5	0.98	0.98	0.98	892
6	0.99	0.98	0.98	958
7	0.98	0.98	0.98	1028
8	0.98	0.98	0.98	974
9	0.98	0.96	0.97	1009
Micro avg	0.98	0.98	0.98	10000
Macro avg	0.98	0.98	0.98	10000
Weighted avg	0.98	0.98	0.98	10000

Table 25 - Average of predicted prob. from each patch to SVM Linear

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.97	0.97	0.97	1032
3	0.96	0.98	0.97	1010
4	0.98	0.98	0.98	982
5	0.98	0.97	0.98	892
6	0.99	0.98	0.98	958
7	0.97	0.97	0.97	1028
8	0.98	0.97	0.98	974
9	0.97	0.95	0.96	1009
Micro avg	0.98	0.98	0.98	10000
Macro avg	0.98	0.98	0.98	10000
Weighted avg	0.98	0.98	0.98	10000

Table 26 - Maximum of predicted prob. from each patch to SVM Linear

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.98	0.99	0.99	1135
2	0.96	0.97	0.97	1032
3	0.95	0.97	0.96	1010
4	0.97	0.98	0.98	982
5	0.96	0.97	0.97	892
6	0.98	0.97	0.98	958
7	0.97	0.96	0.96	1028
8	0.97	0.96	0.97	974
9	0.96	0.94	0.95	1009
Micro avg	0.97	0.97	0.97	10000
Macro avg	0.97	0.97	0.97	10000
Weighted avg	0.97	0.97	0.97	10000

The four CAEs which are trained on different parts of images were all given to SVM and the predicted probabilities of each sample was saved. These predicted probabilities were combined to create feature which was used to classify handwritten numeral. Probabilities were combined by concatenating, average and maximum value and given to SVM with linear kernel. Accuracies of 98.07, 97.71 and 96.99 was reported. Using SVM with RBF kernel for concatenated probabilities gave an accuracy of 98.65 which is shown in below table.

Table 27 - Concatenating predicted prob. from each patch to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.98	1010
4	0.99	0.99	0.99	982
5	0.98	0.99	0.98	892
6	0.99	0.99	0.99	958
7	0.98	0.99	0.98	1028
8	0.99	0.99	0.99	974
9	0.99	0.96	0.97	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Table 28 - Concatenating predicted prob. from each patch to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	1	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

The predicted probabilities four CAEs concatenated with predicted probabilities of fused features of all patches to SVM to SVM with RBF Kernel gave an accuracy of 99%. Same feature vector was given to MLP (architecture 50*512*512*10) and it gave an accuracy of 99%.

Table 29 - Concatenating predicted prob. from each patch to SVM RBF and from original image features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.99	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.98	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.97	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

The predicted probabilities four CAEs concatenated with predicted probabilities of features extracted from original image to SVM to SVM with RBF Kernel gave an accuracy of 98.88%.

Table 30 - Concatenating predicted prob. from each patch to SVM RBF, from fused features and from original image features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.99	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

The predicted probabilities of four CAEs concatenated with predicted probabilities of fused features of all patches and predicted probabilities of features extracted from original image to SVM with RBF Kernel gave an accuracy of 99%.

4.4.1 Ablation Study

Feature vector created by concatenating predicted probabilities of each from patch and predicted probabilities of fused features from each patch gave the maximum accuracy of 99%. Further study was carried out to find out what affect does diff. components in this feature vector have. This was done by systematically removing certain components from the feature vector and then checking its performance.

Table 31 - Concatenating predicted prob. from each patch(excluding top patch) to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Table 32 - Concatenating predicted prob. from each patch(excluding bottom patch) to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Table 33 - Concatenating predicted prob. from each patch(excluding right patch) to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Table 34 - Concatenating predicted prob. from each patch(excluding left patch) to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Above four tables show the result of removing just one patch features from the feature vector. Removing just top patch features, removing just bottom patch features, removing right top patch features and removing just left patch features are shown in Tables 31-34. In each case, the accuracy report was 98.97.

Table 35 - Concatenating predicted prob. from left and right patch to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Table 36 - Concatenating predicted prob. from top and bottom patch to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Removing top and bottom patch features resulted in an accuracy of 98.96 as shown in Table 35, and removing left and right patch features in an accuracy of 98.97, as shown in Table 36.

Table 37 - Concatenating predicted prob. from top patch to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Table 38 - Concatenating predicted prob. from bottom patch to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Table 39 - Concatenating predicted prob. from right patch to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Table 40 - Concatenating predicted prob. from left patch to SVM RBF and from fused features to SVM RBF

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	1	1	1135
2	0.99	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.99	0.99	0.99	974
9	0.99	0.98	0.98	1009
Micro avg	0.99	0.99	0.99	10000
Macro avg	0.99	0.99	0.99	10000
Weighted avg	0.99	0.99	0.99	10000

Having just predicted probabilities from one patch and predicted probabilities of fused features resulted in an accuracy of 98.96 as shown in Table 37-40(top, bottom, left ,right). And on testing classification accuracy on using just top half patch is 91.96, only bottom half patch acc is 89.54, only right half patch acc is 92.84, only left half patch acc is 0.9316, using the predicted probabilities of fused features the accuracy was 98.96. This indicated that predicted probabilities of fused features is the single most import component of the feature vector.

4.5 Application

Extending work to character recognition, the proposed method where applied to 'Devanagari handwritten character dataset (DHCD)[45]. The dataset contains 46 Devanagari characters: 36 letters and 10 digits. Figure 4.6 shows all 46 different characters present in dataset. The dataset contains in total of 92 thousand images. Each 46 of the class have 2 thousand images. 2000 images are divided into train and test set at the source itself. 1700 images of each class is used for training and remaining 300 images from each class is used for testing.

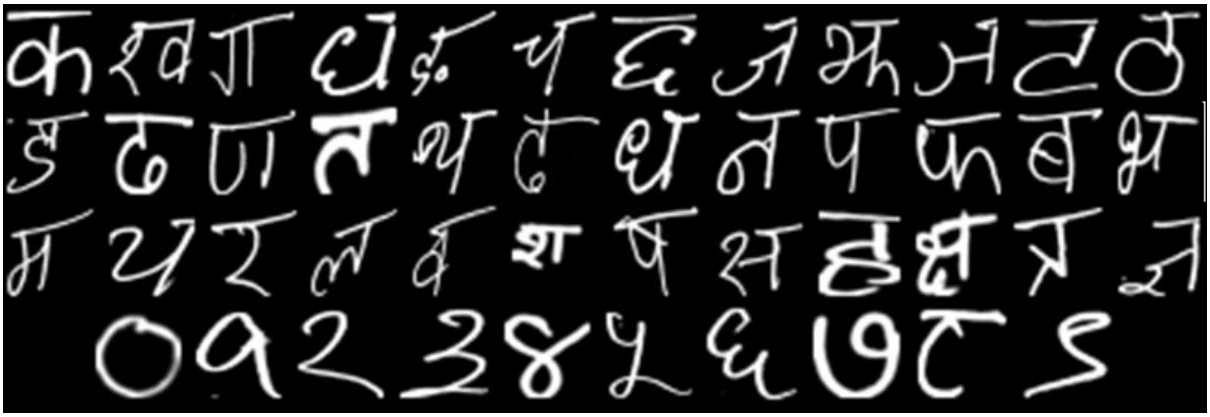


Figure 4.6 Sample of Devanagari handwritten character dataset

Patches were created was shown in Fig 4.7 (Figure 4.8 is the original image). The Deep CNN was used instead of MLP. Extracted features were given to deep neural network and SVM was used as the classifier. An accuracy of 98.39% was reported. Such high accuracy clearly indicates the proposed method can be extended to different recognition activities.

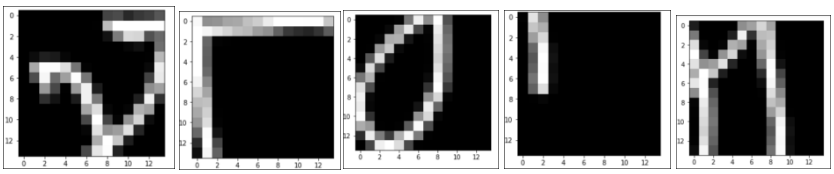


Figure 4.7 Five patches created from character 'bha'

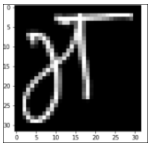


Figure 4.8 Character 'bha'

CHAPTER 5

CONCLUSION

A novel learning paradigm for handwritten numeral images is proposed in this work that initialises CNNs by learning parts in pre-initialisation phase. Two image parts: top-half and bottom-half are used in this model for initialising the weights. The CNNs are further fined-tuned on the remaining images. As it can be seen, the accuracy using CNN Pre-Initialization by Minimalistic Part-Learning has improved the classification accuracy for handwritten numeral digit recognition. Further, the improvement was seen in the proposed method clearly indicate the masking the image as a better effect on the classification task rather than the resizing of the image approach. High accuracies, comparable to the state-of-the-art, confirms the effectiveness of the proposed model.

The proposed model can further be applied for different classification tasks. The proposed model may be further divided to incorporate more stages. More than two parts of the image can be selected for pre-initialisation phase. Parts of an image that may contribute to maximum learning in pre-initialisation stage can be further researched.

After examining the accuracy using standard neural network method for MNIST dataset on two different architecture Net1 and Net2 , Structural Learning approach was explored. The hidden state vector as a feature vector method is applied to two different Multi-Layer Perceptron architecture. Improvement in accuracy was made in the classification of digits is made by clustering the hidden state vector of deep MLP. K-elbow method was used to determine the correct number of cluster. The K-elbow method runs the k-means clustering algorithm for a range of k values to determine the optimal value of k. Distortion score, which is sum of square distance between the sample and the cluster centre of cluster it is assigned, is used to determine goodness of cluster formed for each k value. Further improvement is seen in the model by using patch learning along with the efforts to learn the interpretable hidden state structure. It was found the in MLP having three hidden states, the third state had a meaningful clustering possible, and a structure(vector) could be learned . Giving this learned vector directly to k-Nearest Neighbor (kNN) classifier and the Support Vector Machine (SVM) with linear kernel yielded high accuracies.

Using autoencoder to learn the structure seems to improve the classification. This can be attribute that autoencoders are good at learning representation of given data. So the structures learned from CAE along with part-learning improves the accuracy, Future scope of this work lies in the area of learning different structures. Learning that targets how to select patch area from images could be explored.

REFERENCES

[1] Das, Nibaran, Ayatullah Faruk Mollah, Sudip Saha, and Syed Sahidul Haque. "Handwritten arabic numeral recognition using a multi layer perceptron." *arXiv preprint arXiv:1003.1891* (2010).

[2] Bhattacharya, Ujjwal, and Bidyut Baran Chaudhuri. "Handwritten numeral databases of Indian scripts and multi-stage recognition of mixed numerals." *IEEE transactions on pattern analysis and machine intelligence* 31, no. 3 (2008): 444-457.

[3] Cecotti, Hubert. "Active graph based semi-supervised learning using image matching: application to handwritten digit recognition." *Pattern Recognition Letters* 73 (2016): 76-82.

[4] Dudani, Sahibsingh A. "The distance-weighted k-nearest-neighbor rule." *IEEE Transactions on Systems, Man, and Cybernetics* 4 (1976): 325-327.

[5] Susan, Seba, and Veerendra Singh. "On the discriminative power of different feature subsets for handwritten numeral recognition using the box-partitioning method." In *2011 Annual IEEE India Conference*, pp. 1-5. IEEE, 2011.

[6] Quraishi, Md Iqbal, J. Pal Choudhury, and Mallika De. "Image recognition and processing using Artificial Neural Network." In *2012 1st International Conference on Recent Advances in Information Technology (RAIT)*, pp. 95-100. IEEE, 2012.

[7] Lopez, Marc Moreno, and Jugal Kalita. "Deep Learning applied to NLP." *arXiv preprint arXiv:1703.03091* (2017).

[8] Nazeer, Shahrin Azuan, Nazaruddin Omar, and Marzuki Khalid. "Face recognition system using artificial neural networks approach." In *2007 International Conference on Signal Processing, Communications and Networking*, pp. 420-425. IEEE, 2007.

[9] Courellis, Spiros H., and Vasilis Z. Marmarelis. "An artificial neural network for motion detection and speed estimation." In *1990 IJCNN International Joint Conference on Neural Networks*, pp. 407-421. IEEE, 1990

[10] J. Dacombe, An introduction to Artificial Neural Networks, 2017. [Online] . Available: https://medium.com/@jamesdacombe/an-introduction-to-artificial-neural-networks-with-example-ad459bb6941b. [Accessed: 07- Jul- 2020]

[11] S. Saha, A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, 2018. [Online] . Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 07- Jul- 2020]

[12] Liu, Lingqiao, Chunhua Shen, and Anton van den Hengel. "The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4749-4757. 2015.

[13] A. Biswal, Convolutional Neural Network Tutorial, 2020. [Online] . Available: https://www.simplilearn.com/convolutional-neural-network-tutorial-article. [Accessed: 07- Jul- 2020]

[14] Pala, Tuba, Uğur Güvenç, Hamdi Tolga Kahraman, İbrahim Yücedağ, and Yusuf Sönmez. "Comparison of Pooling Methods for Handwritten Digit Recognition Problem." In *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, pp. 1-5. IEEE, 2018.

[15] Scherer, D., Müller, A. and Behnke, S., 2010, September. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks* (pp. 92-101). Springer, Berlin, Heidelberg.

[16] A. Biswal, Convolutional Neural Network Tutorial, 2020. [Online] . Available: https://www.simplilearn.com/convolutional-neural-network-tutorial-article. [Accessed: 07- Jul- 2020]

[17] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[18] A. Gad, Image Compression using Autoencoders in Keras,2020. [Online] . Available: https://blog.paperspace.com/autoencoder-image-compression-keras/. [Accessed: 07- Jul- 2020]

[19] Larrue, Tara, Yunchuan Li, Xiaoxu Meng, and Chang-Mu Han. "Denoising Videos with Convolutional Autoencoders." (2018).

[20] Daniely, Amit, Roy Frostig, and Yoram Singer. "Toward deeper understanding of neural networks: The power of initialisation and a dual view on expressivity." In *Advances In Neural Information Processing Systems*, pp. 2253-2261. 2016.

[21] Drago, Gian Paolo, and Sandro Ridella. "Statistically controlled activation weight initialisation (SCAWI)." *IEEE Transactions on Neural Networks* 3, no. 4 (1992): 627-631.

[22] Yam, Jim YF, and Tommy WS Chow. "A weight initialisation method for improving training speed in feedforward neural network." *Neurocomputing* 30, no. 1-4 (2000): 219-232.

[23] Nienhold, Dino, Kilian Schwab, Rolf Dornberger, and Thomas Hanne. "Effects of weight initialisation in a feedforward neural network for classification using a modified genetic algorithm." In *2015 3rd International Symposium on Computational and Business Intelligence (ISCBI)*, pp. 6-12. IEEE, 2015.

[24] Mishkin, Dmytro, and Jiri Matas. "All you need is a good init." *arXiv preprint arXiv:1511.06422* (2015).

[25] Saxe, Andrew M., James L. McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks." *arXiv preprint arXiv:1312.6120* (2013).

[26] Sussillo, David, and L. F. Abbott. "Random walk initialisation for training very deep feedforward networks." *arXiv preprint arXiv:1412.6558* (2014).

[27] Susan, Seba, Rohit Ranjan, Udyant Taluja, Shivang Rai, and Pranav Agarwal. "Neural net optimisation by weight-entropy monitoring." In *Computational Intelligence: Theories, Applications and Future Directions-Volume II*, pp. 201-213. Springer, Singapore, 2019.

[28] Susan, Seba, Nandini Aggarwal, Shefali Chand, and Ayush Gupta. "Image coding based on maximum entropy partitioning for identifying improbable intensities related to facial expressions." *Sādhanā* 41, no. 12 (2016): 1393-1406.

[29] BakIr, Gökhan, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, and Ben Taskar, eds. *Predicting structured data*. MIT press, 2007.

[30] Liu, Weiwei, Ivor W. Tsang, and Klaus-Robert Müller. "An easy-to-hard learning paradigm for multiple classes and multiple labels." *The Journal of Machine Learning Research* 18, no. 1 (2017): 3300-3337.

[31] Wang, Kai, Zhaoyan Ming, and Tat-Seng Chua. "A syntactic tree matching approach to finding similar questions in community-based qa services." In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 187-194. ACM, 2009.

[32] Chen, Liang-Chieh, Alexander Schwing, Alan Yuille, and Raquel Urtasun. "Learning deep structured models." In *International Conference on Machine Learning*, pp. 1785-1794. 2015.

[33] Liu, Weiwei, and Ivor Tsang. "On the optimality of classifier chain for multi-label classification." In *Advances in Neural Information Processing Systems*, pp. 712-720. 2015.

[34] Hou, Bo-Jian, and Zhi-Hua Zhou. "Learning with interpretable structure from rnn." *arXiv preprint arXiv:1810.10708* (2018).

[35] Collins, Michael. "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms." In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pp. 1-8. Association for Computational Linguistics, 2002.

[36] 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 (1998): 1057-1068.

[37] Susan, Seba, and Mayank Dwivedi. "Dynamic growth of hidden-layer neurons using the non-extensive entropy." In *2014 Fourth International Conference on Communication Systems and Network Technologies*, pp. 491-495. IEEE, 2014.

[38] Zhai, Shuangfei, Yu Cheng, Weining Lu, and Zhongfei Zhang. "Deep Structured Energy Based Models for Anomaly Detection." In *International Conference on Machine Learning*, pp. 1100-1109. 2016.

[39] Chiaroni, Florent, Mohamed-Cherif Rahal, Nicolas Hueber, and Frédéric Dufaux. "Learning with a generative adversarial network from a positive unlabeled dataset for image classification." In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 1368-1372. IEEE, 2018.

[40] Northcutt, Curtis G., Tailin Wu, and Isaac L. Chuang. "Learning with confident examples: Rank pruning for robust classification with noisy labels." *arXiv preprint arXiv:1705.01936* (2017).

[41] LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." *The handbook of brain theory and neural networks* 3361, no. 10 (1995): 1995.

[42] LeCun, Yann, Corinna Cortes, and Christopher JC Burges. "The MNIST database of handwritten digits, 1998." *URL http://yann. lecun. com/exdb/mnist* 10 (1998): 34. Maaten, Laurens van der, and Geoffrey Hinton.

[43] Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialisation and momentum in deep learning." In *International conference on machine learning*, pp. 1139-1147. 2013.

[44] Kim, Y. K., and J. B. Ra. "Weight value initialisation for improving training speed in the backpropagation network." In *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pp. 2396-2401. IEEE, 1991.

[45] S. Acharya, A.K. Pant and P.K. Gyawali â€œDeep Learning Based Large Scale Handwritten Devanagari Character Recognitionâ€ • ,In Proceedings of the 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA), pp. 121-126, 2015.

[46] Susan, Seba, and Jatin Malhotra. "CNN Pre-Initialization by Minimalistic Part-Learning for Handwritten Numeral Recognition." In *International Conference on Mining Intelligence and Knowledge Exploration*. Springer, Cham, 2019.

[47] Susan, Seba, and Jatin Malhotra. "Learning Interpretable Hidden State Structures for Handwritten Numeral Recognition." In *2020 4th International Conference on Computational Intelligence and Networks (CINE)*, pp. 1-6. IEEE, 2020.

LIST OF PUBLICATIONS

1. Susan, Seba, and Jatin Malhotra. "CNN Pre-Initialization by Minimalistic Part-Learning for Handwritten Numeral Recognition." In *International Conference on Mining Intelligence and Knowledge Exploration*. Springer, Cham, 2019.

2. Susan, Seba, and Jatin Malhotra. "Learning Interpretable Hidden State Structures for Handwritten Numeral Recognition." In *2020 4th International Conference on Computational Intelligence and Networks (CINE)*, pp. 1-6. IEEE, 2020.